Web service architecture / General Concept of Web Services

- Web services evolved from previous technologies that served the same purpose such as RPC, ORPC (DCOM, CORBA and JAVA RMI).
- Web Services were intended to solve three main problems:
  - Interoperability
  - Firewall traversal
  - To hide Complexity

- Definition:
  - A Web service is not a Web site that a human reads, but for reading from other processes running on different machines.
  - A Web service is an interface that describes a collection of operations that are network accessible through standardized XML messaging.

- A web service that:
  - Communicates via open protocols (HTTP, etc.)
  - Processes XML messages framed using SOAP

The Web Service Model / Architecture (or) Basic SOA architecture:

- An early model of SOA is inspired by the initial set of Web services standards, its architecture modeled around three basic components:
  - **Service provider** - Publish operations
  - **Service registry** - Bind operations
  - **Service requestor** - Find operation

- The relationship between **relationship between service, requester and provider** is described in the following figure:

- It is also called **Service-Oriented Architecture (SOA)**, which is an architectural style aimed at achieving loose coupling, and thereby permitting the reuse of interacting software agents / components.
Service Provider
- This is the provider of the web service. The service provider implements the service and makes it available on the Internet. Service provider role includes the following:
  o Creates the web service using specific programming language such as c#.net, or java
  o Generate WSDL file by compiling the web service
  o Publish the web service with UDDI registry

Service Registry
- This is a logically centralized directory of services. 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.
- Service provider use service registry to publish their services
- Service requestor use service registry to locate the desired service

Service Requestor
- This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.
- Service requestor does the following task:
  o Sends the desired service search query to the service registry
  o The service registry is then return all matching service descriptions to the service requester
  o The service requester is then pick the best suitable service by comparing features of all given services
  o The service requester is then send the service request message to the service provider using SOAP protocol

First generation web services standards / Components
- The major standards that you are likely to encounter in your Web services development: SOAP, XML, XML Schema (XSD), HTTP, and WSDL.
  • XML – eXtensible Markup Language – A uniform data representation and exchange mechanism.
  • SOAP – Simple Object Access Protocol – A standard way for communication.
  • UDDI – Universal Description, Discovery and Integration specification – A mechanism to register and locate WS based application.
  • WSDL – Web Services Description Language – A standard meta language to described the services offered.

The Web services framework
- The framework established by Web services is comprised of architectures, technologies, concepts, models, and even sub-frameworks
- The Web services framework is characterized by:
  • an abstract (vendor-neutral) existence defined by standards organizations and implemented by (proprietary) technology platforms
• core building blocks that include Web services, service descriptions, and messages
• a communications agreement centered around service descriptions based on WSDL
• a messaging framework comprised of SOAP technology and concepts
• a service description registration and discovery architecture sometimes realized through UDDI
• a well-defined architecture that supports messaging patterns and compositions
• a second generation of Web services extensions (also known as the WS-* specifications) continually broadening its underlying feature-set

Services (as Web services)
- A web service encapsulates various extents of logic and provides interface to other programs to invoke web services
- Web services can be designed to duplicate the behavior and functionality found in proprietary distributed systems, or they can be designed to be fully SOA-compliant.
- This flexibility has allowed Web services to become part of many existing application environments and one of the reasons behind their popularity

Basic Web services design concepts / web service design classification
- Web services can be labeled using temporary and permanent classifications
- Fundamentally, every Web service can be associated with:
  (i) Temporary classification (based on service roles)
  (ii) Permanent classification (based on service models)
- We explore both of these design classifications in the following two sections

Temporary classifications
- Temporary classifications relate to roles assumed by a service at runtime.
- A temporary classification based on the service roles it assumes during the runtime processing of a message

Service roles:
- A Web service is capable of assuming different roles, depending on the context within which it is used. For example, a service can act as the initiator, relayer, or the recipient of a message
- A service is therefore a unit of software capable of altering its role, depending on its processing responsibility in a given scenario
- A service is capable of altering its role, depending on its processing responsibility in a given scenario.
- So The Web service can able to change its role more than once within a given business task
- The service can assume any one of the following fundamental service roles:
• Service provider
• Service requester
• Intermediaries
• Initial sender and ultimate receiver
• Service compositions / service controller

Service provider
- The web service can assume is role as service provider under the following conditions:
  - If the Web service is invoked via an external source, such as a service requestor then the web service said to be service provider as shown in the figure:

- If the web service receives any request message, then ther Web service is classified as a service provider.
- The Web service provides a published service description offering information about its features and behavior
- On successful execution of web service request, the service provider may reply response message
- The service provider role is synonymous with the server role in the classic client-server architecture
- The following key term to be remembered with respect to service provider:
  - **service provider entity** - refers to the the organization or individual providing the Web service
  - **service provider agent** - refers to the Web service itself, which acting as an agent on behalf of its owner(organization)
Service requestor
- Any unit of processing logic or program that is capable of issuing a request message to the service provider is classified as a service requestor.
- A Web service is always a service provider but also can act as a service requestor.
- A Web service assume the service requestor role under the following circumstances:
  - The Web service invokes a service provider by sending it a request message as shown in the Figure:

- A service requestor is best viewed as a software program that initiates a conversation with a service provider
- The service requestor is comparable to the client in a typical client-server environment.
- Service requestor frequently referred as service consumer
- The following key term to be remembered with respect to service requester:
  - Service requestor entity - It refers to the organization or individual requesting the Web service
  - Service requestor agent - It refers to the Web service itself, acting as an agent on behalf of its owner

Intermediaries
- Web services and service agents that route and process a message after it is initially sent and before it arrives at its ultimate destination are referred to as intermediaries or intermediary services
- Because an intermediary receives and submits messages, it always changeover its role through service provider to service requestor as shown in the following figure:

There are two types of intermediaries. They are
(i) **Passive intermediaries**
(ii) **Active intermediaries**

**Passive intermediaries**
- It is typically responsible for routing messages to a subsequent location as shown in the figure.
Passive intermediaries may use information in the SOAP message header to determine the routing path, but, it does not alter the content of the message.

**Active intermediaries:**
- The active intermediaries also route messages to a forwarding destination. However, prior to transmitting a message, these services actively process and alter the message contents as shown in the following figure:

**Initial sender and ultimate receiver**
- Initial senders are simply service requestors that initiate the transmission of a message. Therefore, the initial sender is always the first Web service in a message path.
- Ultimate receivers are simply service providers that exist as the last Web service along a message's path.
- This feature is described in the following figure:
Service compositions
- This particular term does not apply to a single Web service, but to a composite relationship between collections of services. Any service can enlist one or more additional services to complete a given task. Further, any of the enlisted services can call other services to complete a given sub-task.
- Therefore, each service that participates in a composition assumes an individual role of service composition member, which is described in the figure: A service composition consisting of four members.

![Diagram of service composition](image)

- Service compositions also are referred to as service assemblies.
- The concept of service composability is very important to service-oriented environments.
- In fact, service composition is frequently governed by WS-* composition extensions, such as **WS-BPEL** and **WS-CDL**.

Permanent classifications
- Permanent classification of web services done based on service models.
- The manner in which services are being utilized in the real world has led to a classification based on the nature of the application logic they provide. These classifications are known as service models.

Service model
- Service models refer to permanent classifications that represent the logic housed by the service, as well as its role within the overall solution.
- The basic set of common service models listed below:
  - Business service model
  - Utility service model
  - Controller service model
Business service model
- Within an SOA, the business service represents the most fundamental building block. It encapsulates a distinct set of business logic within a well-defined functional boundary.
- It is fully autonomous but not limited to executing in isolation, as web service they frequently expected to participate in service compositions.
- Business services are used within SOAs as follows:
  - as fundamental building blocks for the representation of business logic
  - to represent a corporate entity or information set
  - to represent business process logic
  - as service composition members

Utility service model
- Any generic Web service or service agent designed for potential reuse can be classified as a utility service.
- Utility service models are functionally generic and non-application specific in nature.
- Utility services are used within SOAs for the following reasons:
  - To enable the characteristic of reuse within SOA
  - To promote the interoperability characteristic of SOA
  - To provide unbelievable intermediary solution services
  - To provides a services with the highest degree of autonomy

- When working with the service abstraction, a utility service is most commonly associated with the application service layer. As a result, a utility service can be referred to as a utility application service.

Controller service model
- Service compositions are comprised of a set of independent services that each contribute to the execution of the overall business task. The assembly and coordination of these services is often a task in itself and one that can be assigned as the primary function of a dedicated service or as the secondary function of a service that is fully capable of executing a business task independently.
- The controller service controls and manage the execution of the overall business task, acting as the parent service to service composition members.
- The Controller services are used within SOAs as follows:
  - to support and implement the principle of composability
  - to influence reuse opportunities
  - to support autonomy in other services

- Sometime the controller services themselves can become subordinate service composition members of other services. In this case the composition is composed into a larger composition. In this situation there may be a master controller
service that acts as the parent to the entire service composition, as well as a sub-controller, responsible for coordinating a portion of the composition as described in the following figure:

- The figure shows a service composition consisting of a master controller, a sub-controller, four business services, and one utility service.

- The controller service model is used frequently when building SOA with specialized service abstraction layers
- Example of some services are listed below:
  - **Accounts Payable Service** = business service
  - **Internal Policy Service** = utility service
  - **Invoice Submission Service** = business service
  - **Purchase Order Service** = business service
  - **Load Balancing Service** = utility service
  - **Authentication Service** = utility service

Service descriptions with WSDL
- SOA provides WSDL for establishing a loosely coupled form of communication between web services
- To provide machine readable description of the web service, WSDL used as primary service description document
The following figure describes how WSDL definitions enable loose coupling between services:

*Service endpoints and service descriptions*

Service endpoints
- A WSDL describes the point of contact for a service provider, also known as the service endpoint or just endpoint.
- It provides a **formal definition of the endpoint interface** and also establishes the **physical location (address) of the service**.
- A WSDL service description also known as **WSDL service definition** or just **WSDL definition**.
- A WSDL service description can be separated into two categories:
  - abstract description
  - concrete description
- The following Figure shows the structure of WSDL document consisting of abstract and concrete parts that collectively describe a service endpoint.
Abstract description
- An abstract description establishes the interface characteristics of the Web service without any reference to the technology used to host or enable a Web service to transmit messages
- By separating this information, the integrity of the service description can be preserved regardless of whatever changes might occur to the underlying technology platform
- An abstract description consists of three main parts: portType, operation, and message

portType
- The portType section provides a high-level view of the service interface by sorting the messages that a service can process into groups of functions known as operations
- The term "portType" is being renamed to "interface" in version 2.0 of the WSDL specification

Operation
- Each operation represents a specific action performed by the service. A service operation is comparable to a public method used by components in traditional distributed applications.

message
- Each operations can have input and output parameters. In web service, these parameters are defined using message.
- Therefore, an operation consists of a set of input and output messages

Concrete description
- This section maps or connects the abstract description of web service to some real, implemented technology such as .Net web services
- Because the execution of service application logic always involves communication, the abstract Web service interface needs to be connected to a physical transport protocol. This connection is defined in the concrete description portion of the WSDL file, which consists of three related parts: binding, port, and service

Binding
- A binding represents one possible transport technology the service can use to communicate
- SOAP is the most common form of binding, but others also are supported. A binding can apply to an entire interface or just a specific operation.

port
- The port represents the physical address at which a service can be accessed with a specific protocol
- The term "port" is being renamed "endpoint" in version 2.0 of the WSDL specification.

**service**
- Within the WSDL language, the term service is used to refer to a group of related endpoints

**Metadata and service contracts**

**Meta data**
- Meta data provides the information about service. There are three separate documents that each describe an aspect of a service:
  - **WSDL definition**
  - **XSD schema**
  - **policy**

Each of these three service description documents can be classified as service metadata, as each provides information about the service.

**WSDL definition**
- provides machine readable description of web services

**XSD Schema**
- WSDL definitions frequently rely on XSD schemas to validate the structure of incoming and outgoing messages.

**Policy**
- Policies can provide rules, preferences, and processing details above and beyond what is expressed through the WSDL and XSD schema documents.

**service contracts**
Service description documents can be collectively viewed as establishing a service contract, which is a set of conditions that must be accepted by a potential service requestor to enable successful communication with service provider. The above figure describes a service contract comprised of a collection of service descriptions and possibly additional documents.

Note:
- Sometime, the service contract can also refer to additional documents or agreements that are not expressed by service descriptions.
- For example, a Service Level Agreement (SLA) agreed upon by the respective owners of a service provider and its requestor can be considered part of an overall service contract

Semantic descriptions
- The most challenging part of providing a complete description of a Web service is in communicating its semantic qualities.
- Examples of service semantics include:
  - how a service behaves under certain conditions
  - how a service will respond to a specific condition
  - what specific tasks the service is most suited for
- Most of the time service semantics are assessed by humans, by comparing the qualities of services offered by various organizations, or by reading supplementary documentation published alongside service descriptions.
- The ultimate goal of semantic description is to provide sufficient semantic information in a structured manner. This helps service requestors program to evaluate and choose suitable service providers independently

Service description advertisement and discovery
- As the amount of services increases within and outside of organizations, mechanisms for advertising and discovering service descriptions may become necessary.
- For example, central directories and registries become an option to keep track of the many service descriptions that become available. These repositories allow humans (and even service requestors) to:
  - locate the latest versions of known service descriptions
  - discover new Web services that meet certain criteria

UDDI (Universal Description, Discovery, and Integration)
- This is why UDDI formed as part of the first generation of Web services standards. UDDI provides us with a registry model
- 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 has two sections:
  o A registry of all web service's metadata, including a pointer to the WSDL description of a service.
  o A set of WSDL port type definitions for manipulating and searching that registry

**Private and public registries**
- UDDI specifies a standard for structuring registries that keep track of service descriptions as shown in the Figure. These registries can be searched manually and accessed programmatically via a standardized API

- There are two types of registries. They are
  o **Public registry**
  o **Private registry**

**Public registry**
- Public registries accept registrations from any organizations. Once signed up, organizations acting as service provider entities can register their services.

**Private registry**
- Private registries can be implemented within organization boundaries to provide a central repository for descriptions of all services the organization develops, leases, or purchases

**Components of UDDI**
- Following are descriptions of the primary parts that comprise UDDI registry records.

**Business entities and business services**

**Business entities**
- Each public registry record consists of a business entity containing basic profile information about the organization
business services
- This record consists of one or more business service areas, each of which provides a description of the services offered by the business entity
- 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.

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

tModel
- tModel stands for technical model.
- tModel section of UDDI records provides pointer to actual service description
- The basic structure of a UDDI business entity record shown in the following figure:

- A business or a company can register three types of information into a UDDI registry. This information is contained in three elements of UDDI. These three elements are:
  - White Pages,
  - Yellow Pages, and
  - Green Pages.

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

**Yellow Pages**
- Yellow pages contain more details about the company. They include descriptions of the kind of electronic capabilities the company can offer to anyone who wants to do business with it.
- Yellow pages uses commonly accepted industrial categorization schemes, industry codes, product codes, business identification codes and the like to make it easier for companies to search through the listings and find exactly what they want.

**Green Pages**
Green pages contains technical information about a web service. A green page allows someone to bind to a Web service after it's been found. It includes:
- The various interfaces
- The URL locations
- Discovery information and similar data required to find and run the Web service.

**Messaging with SOAP (Simple Object Access Protocol)**
- The SOAP specification has been accepted as the standard transport protocol for messages processed by Web services.
- Since its initial release, SOAP has been further revised to accommodate more sophisticated message structures in support of enterprise distributed applications and enterprise SOAs.

**Messages**
- The SOAP specification's main purpose is to define a standard message format. The structure of SOAP format is quite simple.
- Now we take a closer look at the details of the SOAP message format. SOAP message consist of the following components:
  - SOAP envelope
  - SOAP Header
  - SOAP Body
  - SOAP Fault
  - SOAP Attachment

**Envelope**
- Every SOAP message is packaged into a container known as an envelope. The envelope is responsible for housing all parts of the message as shown in the Figure that shows “The basic structure of a SOAP message”
Header
- Each message can contain a header, an area dedicated to hosting meta information
- This header section is a vital part of the overall architecture, but optional, it is rarely omitted.
- Through the use of header blocks, SOAP messages are capable of containing a large variety of supplemental information related to the delivery and processing of message contents.
- It is the header blocks through which numerous extensions can be implemented
- The header blocks can include the following informations:
  - **processing instructions** that may be executed by service intermediaries or the ultimate receiver
  - **routing information or workflow information** associated with the message
  - **security measures** implemented in the message
  - **reliability rules** related to the delivery of the message
  - **context and transaction management** information

Body
- The actual message contents are hosted by the message body, which typically consists of XML formatted data.
- The contents of a message body are often referred to as the message payload.

Message styles
- SOAP message format support two message styles. They are
  - **RPC Style message**
  - **Document Style message**
RPC Style Message
- The SOAP specification was originally designed to replace proprietary RPC protocols
- Here, the actual SOAP message is mapped to remote method/operation call. The root element of the SOAP message is mapped to name of method/operation
- The child elements of SOAP message is name of the formal parameters. The value of the child elements are actual parameter values for method call

Document-Style Messages
- SOA relies on document-style messages to enable larger payloads and reduced message transmission volumes between services.
- Here, the actual SOAP message is mapped to interface or implementation classes.

Attachments
- To facilitate the delivery of non-textual data that is not easily formatted into an XML document, the use of SOAP attachment technologies exist.
- SOAP attachments are commonly employed to transport binary files, such as images.
- Each provides a different encoding mechanism used to bundle data in its native format with a SOAP message.

Faults
- SOAP messages offer the ability to add exception handling logic by providing an optional fault section that can reside within the body area.
- The use of this section is to store a simple error message that is used to deliver error condition information when an exception occurs

SOAP Nodes
- In abstract, the programs that services use to transmit and receive SOAP messages are referred to as SOAP nodes as shown in the following Figure:
- In figure, a SOAP node transmitting a SOAP message received by the service logic
Node types
- Based on the service that use them, every SOAP nodes are given label that identify their type, depending on what form of processing they are involved with in a given message processing scenario
- A list of type labels associated with SOAP nodes listed below. Notice that these names are very similar to the Web service roles
  - **SOAP sender**: A SOAP node that transmits a message
  - **SOAP receiver**: A SOAP node that receives a message
  - **SOAP intermediary**: SOAP node that receives and transmits a message, and optionally processes the message prior to transmission
  - **initial SOAP sender**: the first SOAP node to transmit a message
  - **ultimate SOAP receiver**: The last SOAP node to receive a message

**SOAP intermediaries**
- SOAP intermediary nodes transition its role through SOAP receiver and SOAP sender types when processing a message as shown in the figure:

**Figure:** Different types of SOAP nodes involved with processing a message.

Type of intermediaries:
SOAP nodes acting as intermediaries can be classified as
- **Forwarding intermediaries**: it is responsible for relaying the contents of a message to a subsequent SOAP node, without modifying the content of SOAP message
- **Active intermediaries**: the intermediary will often process and alter header block information relating to the forwarding logic it is executing. It can alter existing header blocks, insert new ones, and execute a variety of supporting actions. For example, it will remove a header block it has processed
Message paths
- A message path refers to the route taken by a message from when it is first sent until it arrives at its ultimate destination.
- Therefore, a message path consists of at least one initial sender, one ultimate receiver, and zero or more intermediaries as shown in the figure:

Figure: A message path consisting of three Web services.

- Note also that a message path is sometimes not predetermined. The use of header blocks processed by intermediaries can dynamically determine the path of a message. This may be the result of routing logic, workflow logic, or environmental conditions

Figure: A message path determined at runtime.
SOA and WS-* Extensions

- The term "WS-*" that refers to the second-generation Web services specifications. These are extensions to the basic Web services framework established by first-generation standards represented by WSDL, SOAP, and UDDI

Web Services and Contemporary SOA

- The term "WS-*" became popular because the majority of titles given to second-generation Web services specifications have been prefixed with "WS-". Some of the example of WS-* includes the following:
  - Message exchange patterns
  - Service activity
  - Coordination
  - Atomic transactions
  - Business activities
  - Orchestration
  - Choreography
- The messaging model used for inter-service communication is simple in nature

Message exchange patterns (MEPs)

- MEP define the way that SOAP messages exchanged between two services
- An MEP is a generic interaction pattern that defines the message exchange between two services
- Message exchange patterns (MEPs) represent a set of templates that provide a group of pre-defined mapped out sequences for the exchange of messages. The most common example is a request and response pattern.
- MEPs can be composed to support the creation of larger, more complex patterns

Types of MEPS

- Many MEPs have been developed, each addressing a common message exchange requirement. It is useful to have a basic understanding of some of the more important MEPs.
- Basically there are two categories of MEPS. They are
  - Primitive MEPs
    - Request response
    - Fire-and-forget
  - Complex MEPS
    - publish-and-subscribe model

Primitive MEPS

- Before the arrival of contemporary SOA, set of well defined messaging frameworks were already used common set of primitive MEPs. They are listed below:
Request-response
- The request-response MEP establishes a simple exchange in which a message is first transmitted from a source (service requestor) to a destination (service provider).
- Upon receiving the message, the destination (service provider) then responds with a message back to the source (service requestor).

Figure: The request-response MEP.

- This is the most popular MEP in use among distributed application environments and the one pattern that defines synchronous communication.

Fire-and-forget
- This simple asynchronous pattern is based on the unidirectional transmission of messages from a source to one or more destinations.
- The fire-and-forget MEP is shown in the following figure:

- A number of variations of the fire-and-forget MEP exist, including:
  - **The single-destination pattern:** where a source sends a message to one destination only.
  - **The multi-cast pattern:** where a source sends messages to a predefined set of destinations.
  - **The broadcast pattern:** this is similar to the multi-cast pattern, except that the message is sent out to a broader range of recipient destinations.
- The fundamental characteristic of the fire-and-forget pattern is that a response to a transmitted message is not expected.
Complex MEPs
- Primitive MEPs can be assembled in various configurations to create different types of messaging models, sometimes called complex MEPs.
- A classic example is the publish-and-subscribe model. The publish-and-subscribe messaging model is a composite of two primitive MEPs.
- It involves two steps:

**Step 1.** The subscriber sends a message to notify the publisher that it wants to receive messages on a particular topic and publisher sends acknowledgement to the subscriber on successful registration. This could be implemented by a request-response MEP, where the subscriber's request message, indicating that it wants to subscribe to a topic, is responded to by a message from the publisher, confirming that the subscription succeeded or failed.

**Step 2.** Upon the availability of the requested information, the publisher broadcasts messages on the particular topic to all of that topic's subscribers. This could be implemented by the fire-and-forget patterns, allowing the publisher to broadcast a series of unidirectional messages to subscribers.

- The process of publish-and-subscribe model is described in the following figure:

WS-* specifications that incorporate this messaging model include:
- **WS-BaseNotification**
- **WS-BrokeredNotification**
- **WS-Topics**
- **WS-Eventing**
**MEPs and SOAP**
- The extensible nature of SOAP allows countless messaging characteristics and behaviors to be implemented via SOAP header blocks.
- The SOAP language also provides an optional parameter that can be set to identify the MEP associated with a message.
- The WSDL and SOAP specifications support specific variations of common MEPs.

**MEPs and WSDL**
- MEPs play a larger role in WSDL service descriptions as they can coordinate the input and output messages associated with an operation.
- WSDL operations support different configurations of incoming, outgoing, and fault messages. These configurations are equivalent to message exchange patterns.

**Four MEPs supported by WSDL 1.1**
- Release 1.1 of the WSDL specification provides support for four message exchange patterns.
- These patterns are applied to service operations from the perspective of a service provider or endpoint. In WSDL 1.1 terms, they are represented as follows:
  - **Request-response operation**: Upon receiving a message, the service must respond with a standard message or a fault message.
  - **Solicit-response operation**: Upon submitting a message to a service requestor, the service expects a standard response message or a fault message.
  - **One-way operation**: The service expects a single message and is not obligated to respond.
  - **Notification operation**: The service sends a message and expects no response.
- Figure shows the four basic patterns supported by WSDL 1.1.

![Diagram of four MEPs](image)

- Among these four, only the request-response operation and one-way operation MEPs are recommended by the WS-I Basic Profile.

**Eight MEPs supported by WSDL 2.0**
- Release 2.0 of the WSDL specification extends MEP support to eight patterns as follows:
- **The in-out pattern**: This is comparable to the request-response MEP and equivalent to the WSDL 1.1 request-response operation. Here service requester always initiates the exchange by transmitting the request message.
- **The out-in pattern**: This is the reverse of the previous pattern, where the service provider initiates the exchange by transmitting the request. (Equivalent to the WSDL 1.1 solicit-response operation)
- **The in-only pattern**: This operation supports the standard fire-and-forget MEP. (Equivalent to the WSDL 1.1 one-way operation)
- **The out-only pattern**: This operation is the reverse of the in-only pattern. It is used primarily in support of event notification. (Equivalent to the WSDL 1.1 notification operation.)
- **The robust in-only pattern**: It is a variation of the in-only pattern that provides the option of launching a fault response message as a result of a transmission error or processing error.
- **The robust out-only pattern**: It is like the out-only pattern, has an outbound message initiating the transmission. The difference here is that a fault message can be issued in response to the receipt of this message.
- **The in-optional-out pattern**: This is similar to the in-out pattern with one exception, that the delivery of a response message is optional and should therefore, the service requestor does not be expect response message. This pattern also supports the generation of a fault message.
- **The out-optional-in pattern**: This operation is the reverse of the in-optional-out pattern, where the incoming message is optional. This pattern also supports the generation of a fault message.

**MEPs and SOA**
- MEPs are highly generic and abstract in nature. They simply relate to an interaction between two services. Their relevance to SOA is equal to their relevance to the abstract Web services framework. They are therefore a fundamental and essential part of any Web services-based environment, including SOA.

**Service activity**
- The interaction of a group of services working together to complete a task can be referred to as a service activity
- In an activity, multiple Web services collaborate to do a specific piece of work, as shown in the following figure:

![Service Activity Diagram](image-url)
- A logical unit of work completed by a collection of services is referred to as service activity.

- For example consider the task of washing a car. This activity includes series of following small tasks:
  1. Locate bucket.
  2. Locate sponge.
  3. Locate hose.
  4. Fill bucket with warm water.
  5. Add soap to water.
  7. Rub sponge on car.

- ...and so on.

- The steps that comprise this more complex task could be summarized into a series of simple (or primitive) tasks, as follows:
  1. Gather required equipment.
  2. Prepare water.
  3. Wash car.

- Individually, simple tasks do not accomplish anything of relevance, but it is only when they are assembled into a complex task that they represent a useful unit of work.

- Similarly, most business tasks automated by service-oriented applications consist of a complex activity that requires the involvement of multiple services that each complete a subset of the work.

**Primitive and complex service activities**

**Primitive service activities:**

- A simple or primitive activity is typified by synchronous communication and therefore often consists of two services exchanging information using a standard request-response MEP as shown in the following figure:

  **Figure**: A primitive service activity consisting of a simple MEP.
- Primitive activities are almost always short-lived; The scope of primitive activities can be limited to the completion of simple MEPs.

**Complex Activity:**
- Complex activities, on the other hand, can involve many services (and MEPs) that collaborate to complete multiple processing steps over a long period of time as shown in the following figure:

![Complex Activity Diagram](image)

**Figure:** A complex activity involving four services.

- Complex activities are common within SOAs and exist as part of any non-trivial service-oriented application.

**Service activities and SOA**
- The underlying application logic of each Web service is generally not mapped as part of a service activity. Complex activities are commonplace in larger service-oriented solutions and can involve numerous participating services.

**Coordination**
- Coordination establish a framework that provides a means for context information in complex activities.
- The context information to be managed, preserved, updated, and distributed to activity participants by the coordination services, which is described in the above figure

**Figure:** Coordination provides services that introduce controlled structure into activities.
Coordinator composition
- WS-Coordination establishes a framework that introduces a generic service based on the coordinator service model as shown in the following Figure, in that a service controls a composition of three other services that each plays a specific part in the management of context data.

Figure: The coordinator service composition.
The coordinator composition consists of the following services:

- **Activation service**: it is responsible for the creation of a new context and for associating this context to a particular activity.
- **Registration service**: It allows participating services to use context information received from the activation service to register for a supported context protocol.
- **Protocol-specific services**: These services represent the protocols supported by the coordinator's coordination type.
- **Coordinator**: The controller service of this composition, also known as the coordination service.

**Coordination types and coordination protocols**

**Coordination types:**
- The WS-Coordination framework is extensible and can be utilized by different coordination types
- The two coordination types most commonly associated with WS-Coordination are: [WS-AtomicTransaction](#) and [WS-BusinessActivity](#).

**Coordination protocols:**
- Coordination protocol is best viewed as a set of rules that are imposed on activities that all registered participants must follow

**Coordination contexts and coordination participants**

**Coordination contexts**
- The context created by the activation service is referred to as a **coordination context**. It contains a collection of information that represents the activity and various supplementary data.
- Examples of the type of data held within a coordination context include:
  - a unique identifier that represents the activity
  - an expiration value
  - coordination type information

**Coordination participants**
- A service that wants to take part in an activity managed by WS-Coordination must request the coordination context from the activation service.
- It then can use this context information to register for one or more coordination protocols.
- A service that has received a context and has completed registration is considered a **participant** in the coordinated activity.

**The activation and registration process**
- The coordination service composition is instantiated when an application service contacts the activation service as shown in the following figure

**Figure**: The WS-Coordination registration process.
- Through the creation of a `CreateCoordinationContext` request message, application service asks the activation service to generate a set of new context data.
- After receiving the `ReturnContext` message, the application service now can invite other services to participate in the coordination. This invitation consists of the context information the application service originally received from the activation service.
- Any Web service in possession of this context information may issue a registration request to the registration service. Upon a successful registration, a service is officially a participant.
- The registration service passes the service the location of the coordinator service, with which all participants are required to interact. At this time, the coordination service is also sent the address of the new participant.

**The completion process**
- The application service can request that a coordination be completed by issuing a completion request message to the coordination service.
- The coordinator, in turn, then issues its own completion request messages to all coordination participants.
Each participant service responds with a completion acknowledgement message as shown in the following figure:

Figure: The WS-Coordination completion process.

Coordination and SOA
- A coordinator-based context management framework introduces a layer of composition control to SOAs as shown in the figure:

Figure: Coordination as it relates to other parts of SOA.
- It standardizes the management and interchange of context information within a variety of key business protocols.
- By introducing an activity management layer to SOA, coordination promotes service statelessness and supports the controlled composition of complex activities.

Atomic transactions
- When managing certain types of corporate data, the need to wrap a series of changes into a single action is fundamental requirement for many business processes.
- Atomic transactions implement the commit and rollback features to enable cross-service transaction support as shown in the figure:

  Figure: Atomic transactions apply an all-or-nothing requirement to work performed as part of an activity.

ACID transactions
- The protocols provided by the WS-AtomicTransaction specification enable cross-service transaction functionality comparable to the ACID-compliant transaction features found in most distributed application platforms.
- The term "ACID" representing the following four required characteristics of a traditional transaction:
  - **Atomic**: Either all of the changes within the scope of the transaction succeed, or none of them succeed.
• **Consistent**: None of the data changes made in the current transaction should violate the validity of any associated data models. Any violations result in a rollback of the transaction.

• **Isolated**: If multiple transactions occur concurrently, they may not interfere with each other.

• **Durable**: Upon the completion of a successful transaction, changes made as a result of the transaction must be made permanent.

**Atomic transaction protocols**

- WS-AtomicTransaction is a coordination type, meaning that it is an extension created for use with the **WS-Coordination context management framework**.

- To participate in an atomic transaction, a service first receives a coordination context from the activation service.

- After receiving coordination context, the service can register for available atomic transaction protocols.

- The following primary transaction protocols are provided:
  - A **Completion protocol**: It is used to initiate the commit or abort states of the transaction.
  - The **Durable 2PC protocol**: Only the services that representing permanent data repositories can register with this protocol.
  - The **Volatile 2PC protocol**: Only the services that managing non-persistent (temporary) data can register with this protocol.

- Most often these protocols are used to enable a **two-phase commit (2PC)** that manages an atomic transaction across multiple service participants.

**The atomic transaction coordinator**

![Diagram](image.png)

**Figure**: The atomic transaction coordinator service model
- When WS-AtomicTransaction protocols are used, the coordinator controller service can be referred to as an atomic transaction coordinator, which plays a key role in managing the participants of the transaction process and in deciding the transaction's ultimate outcome.
- An atomic transaction coordinator shown in the above Figure

**The atomic transaction process**
- The atomic transaction coordinator is responsible for deciding the outcome of a transaction. It decides whether commit or rollback the entire transaction, based on feedback it receives from all of the transaction participants.
- The collection of this feedback is separated into two phases: Prepare phase and Commit phase.

**Prepare phase:**
- During the prepare phase, all participants are notified by the coordinator, and each is asked to prepare and then issue a vote as shown in the following figure:

![Prepare phase diagram]

*Figure: The coordinator requesting that transaction participants prepare to vote.*

- Then each participant's vote consists of either a "commit" or "abort" request as shown in the following figure:

![Commit phase diagram]

*Figure: The transaction participants voting on the outcome of the atomic transaction.*
Commit phase:
- After the votes are collected, the atomic transaction coordinator enters the commit phase. It now reviews all votes and decides whether to commit or rollback the transaction.
- if all participants voted to “commit”, then the coordinator declares the transaction successful, all the changes are made permanent by invoking commit
- if any one vote requests an abort, or if any of the participants fail to respond, then the transaction is aborted, and all changes are rolled back

Figure: The coordinator aborting the transaction and notifying participants to rollback all changes.

Atomic transactions and SOA
- An atomic transaction therefore plays an important role in ensuring quality of service; also promote interoperability when extended into integrated environments. Following figure illustrates how atomic transactions support these aspects of SOA.

Figure: Atomic transaction relating to other parts of SOA.

Summary:
- WS-AtomicTransaction is a coordination type that supplies three coordination protocols that can be used to achieve two-phase commit transactions across multiple service participants.
- The atomic transaction coordinator makes the ultimate decision to commit or rollback a transaction. This decision is based on votes collected from participants.

**Business activities**

- Business activities mananges complex, long-running service activities, which can take hours, days, or even weeks to complete a task.
- The business activity primarily differ from atomic transactions. For instance, business activity protocols do not offer rollback capabilities. This is because it would not be possible to provide ACID-type transaction functionality for long-running business activities.
- Instead, business activities provide an optional compensation process that, much like a "plan B," can be invoked when exception conditions are encountered as shown in the following Figure:

  **Figure: A business activity controls the integrity of a service activity by providing participants with a "plan B" (a compensation).**

  ![Diagram](image)

  "Let's try plan A. If that doesn't work, we'll try plan B."

**Business activity protocols**

- WS-BusinessActivity builds on the WS-Coordination context management framework by providing two protocols for which activity participants can register.
- Participants and the business activity coordinator progress through a series of states during the lifespan of a business activity. State transition is accomplished through the exchange of notification messages.
- The two protocols listed below, each of which dictates how a participant may behave within the overall business activity:
  - **The BusinessAgreementWithParticipantCompletion protocol**: this protocol allows a participant to determine when it has completed its part in the business activity.
  - **The BusinessAgreementWithCoordinatorCompletion protocol**: in this protocol, participants rely on the business activity coordinator to notify it that it has no further processing responsibilities.
- Business activity participants interact with the standard WS-Coordination coordinator composition to register for a protocol.
The business activity coordinator
- When business activity protocols are used with WS-Coordination, the WS-Coordination controller service assumes a role of business activity coordinator as shown in the figure:

![Diagram of business activity coordinator](image)

- BusinessAgreementWithParticipantCompletion
- BusinessAgreementWithCoordinatorCompletion

Business activities and atomic transactions
Figure: Two atomic transactions residing within the scope of a business activity.
- It is important to note that the use of a business activity does not exclude the use of atomic transactions. In fact, it is likely that a long-running business activity will encompass the execution of several atomic transactions during its lifetime as shown in the above figure.

**Business activities and SOA**

- Business activities fully complement the composable nature of SOA by tracking and regulating complex activities while also allowing them to carry on for long periods of time.
- Through the use of the compensation process, business activities increase SOA’s quality of service by providing built-in fault handling logic.
- The following figure describes the business activity relating to other parts of SOA:

- The support of the WS-BusinessActivity extension by multiple solutions promotes inherent interoperability and can greatly simplify integration architectures. Business activities take this a few steps further, though, by allowing the scope of the activity to include interaction with outside business partners.

**Orchestration**

- Orchestration is one form of composition in which, one service act as controller service that control sequence of execution of other services within complex activity.
- In these systems, a centrally controlled set of workflow logic facilitates interoperability between two or more different applications.
- A common implementation of orchestration is the **hub-and-spoke model** that allows multiple external participants to interface with a central orchestration engine.
- With orchestration, different processes can be connected without having to redevelop the solutions.
- Orchestration bridges this gap by introducing new workflow logic. Further, the use of orchestration can significantly reduce the complexity of solution environments.
- Orchestration can represent and express business logic in a standardized, services-based venue.
- Orchestration further leverages the interoperability sought by service designs by providing potential integration endpoints into processes.
- Figure: An orchestration controls almost every facet of a complex activity.

Business protocols and process definition
- The workflow logic that comprises an orchestration can consist of numerous business rules, conditions, and events, which collectively establish a business protocol that defines how participants can interoperate to achieve the completion of a business task.
- The details of the workflow logic encapsulated by an orchestration are contained within a process definition

Process services and partner services
Process services:
- A process service is a service that helps the partner services to interact with each other in a composition, which consist of complex activities
- Figure: A process service coordinating and exposing functionality from three partner services.
Partner service:
- Other services allowed to interact with the process service are identified as **partner services** or **partner links**.
- Figure: The process service, after first being invoked by a partner service, then invokes another partner service.

**WS-BPEL (Web services Business Process Execution Language)**
- WS-BPEL is a primary industry specification that standardizes orchestration.
- WS-BPEL is the most recent name given to this specification, which also is known as BPEL4WS and just BPEL.

**Basic activities and structured activities**
- WS-BPEL breaks down workflow logic into a series of predefined primitive activities. They are
  - Basic activities (receive, invoke, reply, throw, wait)
  - Structured activities (sequence, switch, while, flow, pick)

**Basic Activities**
- Basic activities represent fundamental workflow actions, which includes the following: receive, invoke, reply, throw, wait

**Structured activities**
- Basic activities can be assembled using the logic supplied by structured activities, which includes the following: Sequence, switch, while, flow, pick
- These activities can be used to express actual business process logic
- Basic and structured activities can be organized so that the order in which they execute is predefined

**Sequences, flows, and links**

**Sequence**
- A sequence aligns groups of related activities into a list that determines a sequential execution order.
- Sequences are useful when one piece of application logic is dependent on the outcome of another.

**Flows**
- Flows contain groups of related activities that can execute concurrently within a flow
- Activities running within a **flows** need not required to wait for another activities to finish their task.
- However, the flow itself does not finish until all encapsulated activities have completed processing.

**Links**
- Links are used to establish formal dependencies between activities that are part of flows.
- Rules provided by links are also referred to as synchronization dependencies.
- Before any linked activity can begin, requirements contained within any incoming links first must be satisfied.
- Similarly, before an activity fully can complete, it must ensure that any requirements established in outgoing links first are met.

**Orchestration and coordination**
- Orchestration, as represented by WS-BPEL, can fully utilize the WS-Coordination context management framework by incorporating the WS-BusinessActivity coordination type.
- This specification defines coordination protocols designed to support complex, long-running activities.

**Orchestration and SOA**
- Orchestration provides an automation model where process logic is centralized yet still extensible and composable as shown in the following figure.

![Figure: Orchestration relating to other parts of SOA.](image)

These qualities lead to increased organizational agility because:
- The workflow logic encapsulated by an orchestration can be modified or extended in a central location.
- By establishing potentially large-scale service-oriented integration architectures, orchestration, on a fundamental level, can support the evolution of a diversely federated enterprise.

**Drawback:** An orchestration expresses a body of business process logic that is typically owned by a single organization, but not for multiple organization.
Choreography

- Orchestration helps to control business process flow of complex activities within the organization, but does not support collaboration between services running in different organization.
- Choreography allows collaboration and integration services running in different organizations that needs to work together to achieve a common goal.
- The Web Services Choreography Description Language (WS-CDL) is one of several specifications that attempts to organize information exchange between multiple organizations (or even multiple applications within organizations).
- This increases public collaboration as shown in the following figure.

![Figure: A choreography enables collaboration between its participants.](image)

Collaboration

- Choreographies are intended for public message exchanges, which supports B2B (Business-to-Business) interactions.
- The goal is to establish a kind of organized collaboration between services representing different service entities in that no single organization can control the collaboration logic.
- Choreographies therefore provide the potential for establishing universal interoperability patterns for common inter-organization business tasks.

Roles and participants

- Web services running within choreography can assume any one of predefined roles.
- This determines what the service does and what the service can do within the context of a particular business task.
- Roles can be bound to WSDL definitions are categorized as participants (services).

Relationships and channels

**Relationship:** Each potential exchange between two roles in choreography is defined individually as a relationship. Every relationship consequently consists of exactly two roles.
Channels
- Now that we've defined who can talk with each other, we require a means of establishing the nature of the conversation.
- Channels define the characteristics of the message exchange between two specific roles.
- Channels establish the nature of the conversation between two roles(services)

Interactions and work units
interactions
- The actual logic behind a message exchange is encapsulated within an interaction.
- Interactions are the fundamental building blocks of choreographies, because the completion of an interaction represents actual progress within a choreography

Work units
- Related to interactions are work units. These impose rules and constraints that must be adhered to for an interaction to successfully complete.

Reusability, composability, and modularity
Reusability
- Each choreography can be designed in a reusable manner, allowing it to be applied to different business tasks

composability
- Using an import facility, a choreography can be assembled from independent modules. These modules can represent distinct sub-tasks and can be reused by numerous different parent choreographies as shown in the following figure:

- Choreographies themselves can be assembled into larger compositions.
Orchestrations VS choreographies

- Even though both represent complex message interchange patterns, there is a common distinction that separates the terms "orchestration" and "choreography."

<table>
<thead>
<tr>
<th>Orchestration</th>
<th>Choreography</th>
</tr>
</thead>
<tbody>
<tr>
<td>An orchestration expresses organization-specific business workflow</td>
<td>A choreography, on the other hand, is not necessarily owned by a single organization</td>
</tr>
<tr>
<td>A single organization owns and controls the logic behind an orchestration</td>
<td>No single organization owns or controls the logic behind choreography</td>
</tr>
</tbody>
</table>

- Choreography acts as a community interchange pattern used for collaborative purposes by services from different provider entities (organizations) as shown in the figure.

Figure: A choreography enabling collaboration between two different orchestrations.

Choreography and SOA

Figure: Choreography relating to other parts of SOA
- Two services belonging to different organizations, each exposing functionality from entire enterprise business solutions can interact via a basic choreography to complete a more complex task.
- Choreography therefore can assist in the realization of SOA across organization boundaries.
- While it natively supports composability, reusability, and extensibility, choreography also can increase organizational agility and discovery.

Two mark question?

1. Differentiate document-style SOAP messages and document-centric XML documents?
   - Document-centric XML documents are generally refers to published documents represented by XML
   - Where as document-style SOAP messages contain application data

2. What is the difference service agent and service provider agent?
   - A service provider agent is web service that provides services on behalf of its organization
   - Where as a service agent is a small runtime program used to perform generic processing tasks in support of Web services

3. Compare Orchestrations and activities?

<table>
<thead>
<tr>
<th>Activity</th>
<th>Orchestration</th>
</tr>
</thead>
<tbody>
<tr>
<td>An activity is a generic term that can be applied to any logical unit of work completed by a service-oriented solution</td>
<td>The scope of a single orchestration, can be classified as a complex, and most likely, long-running activity</td>
</tr>
</tbody>
</table>

4. What is the difference between orchestration and choreography?

<table>
<thead>
<tr>
<th>Orchestration</th>
<th>Choreography</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orchestration helps to control business process flow of complex activities within the organization, but does not support collaboration between services running in different organization</td>
<td>Choreography allows collaboration and integration of multiple services from different organizations that need to work together to achieve a common goal.</td>
</tr>
<tr>
<td>An orchestration expresses organization-specific business workflow</td>
<td>A choreography, on the other hand, is not necessarily owned by a single organization</td>
</tr>
<tr>
<td>A single organization owns and controls the logic behind an orchestration</td>
<td>No single organization owns or controls the logic behind choreography</td>
</tr>
</tbody>
</table>

5. List the types of intermediaries?

There are two types of intermediaries they are:

- **Forwarding/Passive intermediaries**: it is responsible for relaying the contents of a message to a subsequent SOAP node, without modifying the content of SOAP message.
Active intermediaries: the intermediary will often process and alter header block information relating to the forwarding logic it is executing. It can alter existing header blocks, insert new ones, and execute a variety of supporting actions. For example, it will remove a header block it has processed.

6. What are the two SOAP message style?
SOAP message format supports two message styles. They are
- **RPC Style message**
  - It was originally designed to replace proprietary RPC protocols
  - It maps root element of SOAP message with the name of the function call
- **Document Style message**
  - SOA relies on document-style messages to enable larger payloads and reduced message transmission volumes between services
  - It maps root element of SOAP message with the name of the class or interface

7. What is WS-CDL?
- **WS-CDL** stands for Web Service Choreography description language
- **WS-CDL** is one of several specifications that attempts to organize information exchange between multiple organizations
- **WS-CDL** is a primary industry specification that standardizes choreography

8. When to use Business Activities and when to use Atomic Transactions?
- If transaction time for your WS-Coordination is about a few seconds to a few minutes, then use Atomic Transactions.
- If transaction time for your WS-Coordination is about an hour, days, or even weeks to complete a task, then use Business Activities.

<table>
<thead>
<tr>
<th>Atomic Transactions</th>
<th>Business Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic Transactions manages complex short-running business activities, it transaction can take about a few seconds to a few minutes</td>
<td>Business activities manages complex, long-running service activities, which can take hours, days, or even weeks to complete a task</td>
</tr>
<tr>
<td>It offers rollback facilities for the unsuccessful transaction</td>
<td>It does not offer rollback facilities for the unsuccessful transaction, instead it offers an alternate compensation plan, in case of failure</td>
</tr>
</tbody>
</table>

9. What is the role of UDDI in SOA?
- **UDDI** stands for Universal Description, Discovery, and Integration
- **UDDI** is a specification for a distributed registry of web services
- **UDDI** offers a registry of directory that is used by both service provider and service requester
  - Service provider uses service registry to publish their services
  - Service requestor uses service registry to locate the desired service
10. Draw the diagram of first generation web service architecture?

![Diagram of first generation web service architecture]

11. What are two web service design classifications?
   - Fundamentally, every Web service can be associated with:
     (i) **Temporary classification** (based on service roles)
         - Temporary classifications relate to roles assumed by a service at runtime
     (ii) **Permanent classification** (based on service models)
         - Permanent classification of web services done based on service models

12. List all fundamental service roles?
   - **SOAP sender**: A SOAP node that transmits a message
   - **SOAP receiver**: A SOAP node that receives a message
   - **SOAP intermediary**: SOAP node that receives and transmits a message, and optionally processes the message prior to transmission
   - **Initial SOAP sender**: The first SOAP node to transmit a message
   - **Ultimate SOAP receiver**: The last SOAP node to receive a message

13. What is Service activity?
   - The interaction of a group of services working together to complete a task can be referred to as a service activity

14. List the protocol supported by WS-Transactions?
   Atomic transaction supports the following protocols:
   - **A Completion protocol**: It is used to initiate the commit or abort states of the transaction
   - **The Durable 2PC protocol**: Only the services that representing permanent data repositories can register with this protocol
   - **The Volatile 2PC protocol**: Only the service that deals with non-persistant data can register with this protocol.

15. List the protocol supported by Business Activities?
   - The two protocols listed below, each of which dictates how a participant may behave within the overall business activity are
     a. **The BusinessAgreementWithParticipantCompletion protocol**
     b. **The BusinessAgreementWithCoordinatorCompletion protocol**