Web services: WSDL
Web Service Description Language

Adapted from:
"Web Services: Principles and Technology" by Michael P. Papazoglou
Pearson - Prentice Hall, July 2007
Outline

- Web Service Description Language
- Structure of WSDL documents
  - WSDL interface definition part
  - WSDL interface implementation part
- WSDL Message Exchange Patterns
- WSDL Usage
Problems to solve

- How to make the service invocation part of the language in a more or less transparent manner.
  - Don’t forget this important aspect: whatever you design, others will have to program and use

- How to exchange data between machines that might use different representations for different data types. This involves two aspects:
  - data type formats (e.g., byte orders in different architectures)
  - data structures (need to be flattened and the reconstructed)

- How to find the service one actually wants among a potentially large collection of services and servers.
  - The goal is that the client does not necessarily need to know where the server resides or even which server provides the service.

- How to deal with errors (QoS) in the service invocation in a more or less elegant manner:
  - server is down,
  - communication is down,
  - server busy,
  - duplicated requests ...
Why are Service Descriptions Needed?

- **Web Services need to be defined in a consistent manner so that they can be discovered and used by other web services and applications.** Web service consumers must determine the precise XML interface of a web service:
  - XML schema can partially fill this need as it allows developers to describe the structure of XML messages understood by web services.
  - XML Schema alone cannot describe important additional details involved in communicating with a web service.

- **Service description** is a key to making the service oriented architecture loosely coupled and reducing the amount of required common understanding and custom programming and integration

- **WSDL Service description + SOAP infrastructure** sufficiently isolates all technical details, e.g., machine- and implementation language-specific elements, away from the service requester’s application and the service provider’s web service.
Web Services Description Language

To use SOAP with a particular web service would require, at least:

- some documentation explaining the **structure of SOAP messages**
- which **protocol** will be employed (HTTP or SMTP, for instance)
- **operations exposed** along with their parameters in a machine-understandable standard format
- the **Internet address of the web service** in question.

The **Web Services Description Language (WSDL)** is the service representation language used to describe the details of the complete interfaces exposed by web services and thus is the means to accessing a web service.
A web service description in WSDL is an XML document that describes the mechanics of interacting with a particular web service.

**WSDL represents a “contract” between the service requester and the service provider**
- in much the same way that an interface in an object-oriented programming language, e.g., Java, represents a contract between client code and the actual object itself.

- WSDL is platform and language-independent and is used primarily (but not exclusively) to describe SOAP-enabled services. Essentially, WSDL is used to describe precisely:
  - **What** a service does, i.e., the operations the service provides,
  - **Where** it resides, i.e., details of the protocol-specific address, e.g., a URL, and
  - **How** to invoke it, i.e., details of the data formats and protocols necessary to access the service’s operations.
Structure of WSDL documents

- WSDL documents can be separated into distinct sections.
  - The **service-interface definition** describes the general web service interface structure. This contains all the operations supported by the service, the operation parameters and abstract data types.
  - The **service implementation part** binds the abstract interface to a concrete network address, to a specific protocol and to concrete data structures. A web service client may bind to such an implementation and invoke the service in question.

- This enables each part to be defined separately and independently, and **reused** by other parts.

- The combination of these two parts contains **sufficient information** to describe to the service requester how to invoke and interact with the web service at a provider’s site.
Web Service Interface Definition

- WSDL specifies a grammar and syntax that describes web services as a collection of communicating endpoints.

- The data being exchanged between the endpoints are specified as part of messages and every kind of processing activity allowed at an endpoint is considered as an operation.

- WSDL is layered top of the XML schema and provides the means to group messages into operations and operations into interfaces.
The web service interface definition is considered as an **abstract definition** of a web service in that it does not carry any deployment-specific details and is used to describe a specific type of service.

- A service interface definition is an abstract service description that can be instantiated and referenced by multiple concrete service implementations.

- The service interface contains the WSDL elements that comprise the reusable portion of the service description, these include:
  - the `<portType>`, `<operation>`, `<message>`, `<part>` and `<types>` elements, among others

- These are described in the following and are shown in the figure, which describes the relationship between the WSDL data structures in UML.
WSDL elements hierarchy

- Part
  - Message
    - Operation
      - PortType
        - Binding
          - Port
            - Service
              - abstract endpoint type
                (no transport or encoding)
              - concrete endpoint type
                (with transport or encoding)
              - endpoint instance
                (with network address)
              - service
                (related endpoint instances)
Listing 1: example WSDL interface definition

```xml
<definitions name="PurchaseOrderService"
    targetNamespace="http://supply.com/PurchaseService/wSDL"
    xmlns:tns=http://supply.com/PurchaseService/wSDL
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"

    <!-- WSDL SOAP binding & WSDL namespace follow -->
    xmlns:soap=":http://schemas.xmlsoap.org/soap/
    xmlns:wSDL=":http://schemas.xmlsoap.org/wSDL"
>

<wSDL:types>
    <xsd:schema
        targetNamespace="http://supply.com/PurchaseService/wSDL"
        xmlns:complexType="tns:POType">
        <xsd:sequence>
            <xsd:element name="PONumber" type="xsd:integer"/>
            <xsd:element name="PODate" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="InvoiceType">
        <xsd:all>
            <xsd:element name="InvPrice" type="xsd:float"/>
            <xsd:element name="InvDate" type="xsd:string"/>
        </xsd:all>
    </xsd:complexType>
</xsd:schema>
</wSDL:types>

<wSDL:message name="POMessage">
    <wSDL:part name="PurchaseOrder" type="tns:POType"/>
    <wSDL:part name="CustomerInfo" type="tns:CustomerInfoType"/>
</wSDL:message>

<wSDL:message name="InvMessage">
    <wSDL:part name="Invoice" type="tns:InvoiceType"/>
</wSDL:message>

<wSDL:portType name="PurchaseOrderPortType">
    <wSDL:operation name="SendPurchase">
        <wSDL:input message="tns:POMessage"/>
        <wSDL:output message="tns:InvMessage"/>
    </wSDL:operation>
</wSDL:portType>
```

---

**Port type with one operation**

An operation with request (input) & response (output) message

**Data that is sent**

**Data that is returned**

**Abstract data type definitions**

**Definitions**
The `<definitions>` element as a root element in WSDL usually contains several XML namespace definitions.

- The first attribute in the `<definitions>` element is `name`, which is used to name the entire WSDL document.
- The `<definitions>` element also declares an attribute called `targetNamespace`, which identifies the logical namespace for elements defined within the WSDL document and characterise the service, and is usually chosen to be unique to the individual service (a URL set to the name of the original WSDL file). This helps clients differentiate between web services and prevents name clashes when importing other WSDL files. These namespaces are simply unique strings – they usually do not point to a page on the Web.
- The `xmlns:tns` (sometimes referred to as `this namespace`) attribute is set to the value of `targetNamespace` and is used to qualify (scope) properties of this service definition.
- The namespace definitions `xmlns:soapbind` and `xmlns:xsd` are used for specifying SOAP-binding specific information as well as data types, respectively.
- The final statement defines `xmlns:wsdl` as the default namespace for all WSDL elements defined in a WSDL specification such as messages, operations, and portTypes. The `wsdl:types` definition encapsulates schema definitions of all types using XML XSD.
The WSDL `<types>` element serves as a container that contains all abstract data types that define a web service interface.

A `<type>` element in WSDL is comparable to a data type in Java or C++.

- WSDL uses a few primitive data types that XML Schema Definition (XSD) defines, such as `int`, `float`, `long`, `short`, `string`, `boolean and so on`, and allows developers to either use them directly or build complex data types based on those primitive ones before using them in messages.

- The data types and elements defined in the `<types>` element are used by message definitions when declaring the parts (payloads) of messages. Any complex data type that the service uses must be defined in an optional `<types>` section immediately before the `<message>` section.

Listing 1 illustrates two complex types that have been defined in its `<types>` section: `POType` and `InvoiceType`.

- These two complex types are assigned the `xmlns:tns` namespace by the `targetNamespace` attribute. The elements `<sequence>` and `<all>` are standard XSD elements. The construct `<sequence>` requires that the content model follows the element sequence defined, while the construct `<all>` denotes that all the elements that are declared in the `<complexType>` statement must appear in an instance document.

- The WSDL document must be aware of all namespaces used in the document.
The `<message>` element describes the payload of a message used by a web service. A message consists of `<part>` elements, which are linked to `<types>` elements.

A `<message>` element describes the payload of outgoing and incoming messages, i.e., messages that are directly sent to or received from a web service. In addition, the message element can describe contents of SOAP header blocks and fault detail elements.

While a `<message>` represents the overall data type formation of an operation, parts may further structure this formation.

- Part elements may select individual type definitions contained in a `<types>` element. The chosen type definition is directly attached to `<part>` elements.

In Listing 1 the PurchaseOrder service defines two `<message>` elements to describe the parameters and return values of that service.

- The message called POMessage describes the input parameters of the service, while
- the message InvMessage represents the return (output) parameters.
The central element externalising a service interface description is the `<portType>` element. A `<portType>` element defines an abstract type and its operations but not an implementation.

- Therefore, a `<portType>` element describes the kinds of operations that a web service supports – the messaging mode and payloads – without specifying the Internet protocol or physical address used. This element contains all named operations supported by a service.
- The `<portType>` element is central to a WSDL description; the rest of the elements in the definition are essentially details that the `<portType>` element depends upon.

A `<portType>` element is simply a logical grouping of `<operations>`. Operations in WSDL are the equivalent of method signatures in programming languages. Operations in WSDL represent the various methods being exposed by the service: they include the name of the method and the input and output parameters. Each `<operation>` element declared by a `<portType>` uses one or more message definitions to define its input, output, and faults.

- A typical operation defines the input and output parameters or exceptions (faults) of an operation. A WSDL `<portType>` element may have one or more `<operation>` elements, each of which defines an RPC-style or document-style web service method. Each `<operation>` element is composed of at most one `<input>` or `<output>` element and any number of `<fault>` elements.

The WSDL example in Listing 1 defines a web service that contains a `<portType>` named PurchaseOrderPortType that supports a single `<operation>`, which is called SendPurchase.
The syntax and the examples used above were specified in terms of WSDL 1.1.

The recent WSDL 1.2 definition has introduced some changes among which `<portType>` elements are renamed to `<interface>` elements.

The WSDL 1.2 definition also supports a useful new feature in the form of the `<extends>` attribute, which allows multiple `<interface>` declarations to be aggregated together and further extended to produce a completely new `<interface>` element.

For example, consider the situation where the basic purchase order web service needs to be extended to support inventory-checking activities. Using the `<extends>` mechanism, a new interface could be created that contains all of the operations from each `<interface>` element, i.e., purchase order and inventory-checking, and extends them appropriately by defining any additional operations required by the web services application developer.
WSDL Implementation

- The purpose of WSDL is to specify a web service abstractly and then to define how the WSDL developer will reach the implementation of these services.
- The service implementation describes
  - where the service is located, or more precisely,
  - to which network address the message must be sent in order to invoke the web service.
  - a WSDL service element. A service implementation document can contain references to more than one service interface document by means of <import> elements.
- The service implementation part of WSDL contains the elements <binding> (although sometimes this element is considered as part of the service definition) <port> and <service> and describes how a particular service interface is implemented by a given service provider.
WSDL elements hierarchy

- Service
  - Port
    - Binding
      - PortType
        - Message
          - Operation
            - Part
              - Message
                - Part
                  - Message
                    - Part
                      - Message
                        - Part
                          - Message
Listing 2 WSDL implementation description

```xml
<definitions> . . .
  <import namespace="http://www.supply.com/PurchaseService-interface"
    location="http://www.supply.com/wSDL/PurchaseService-interface.wsdl"/>

<!-- wsdl:binding states a serialisation protocol for this service -->
<!-- type attribute must match name of portType element in Listing 1 -->
<wsdl:binding name="PurchaseOrderSOAPBinding"
  type="tns: PurchaseOrderPortType">

<!-- leverage off soapbind:binding synchronous style -->
<soapbind:binding style="rpc"
  transport="http://schemas.xmlsoap.org/soap/http"/>
<wsdl:operation name="SendPurchase">
</wsdl:operation>

<!-- again bind to SOAP -->
<soapbind:operation style="rpc"
  soapAction="http://supply.com/PurchaseService/wSDL/SendPurchase"/>

<!-- further specify that the messages in the wsdl:operation "" use SOAP -->
<wsdl:input>
  <soapbind:body use="literal"
    namespace="http://supply.com/PurchaseOrderService/wSDL"/>
</wsdl:input>

<wsdl:output>
  <soapbind:body use="literal"
    namespace="http://supply.com/PurchaseOrderService/wSDL"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>

<wsdl:service name="PurchaseOrderService">
  <wsdl:port name="PurchaseOrderPort" binding="tns: PurchaseOrderSOAPBinding">
    <!-- give the binding a network endpoint address or URI of service -->
    <soapbind:address location="http://supply.com:8080/PurchaseOrderService"/>
  </wsdl:port>
</wsdl:service>
</definitions>
```
The central element of the implementation description is the `<binding>` element. The `<binding>` element specifies how the client and web service should exchange messages. The client uses this information to access the web service.

A `<binding>` element contains information of how the elements in an abstract service interface ( `<portType>` element) are converted into concrete representation in a particular combination of:
- concrete protocols, e.g., SOAP or HTTP,
- messaging styles, e.g., RPC or documents styles, and
- formatting (encoding) styles, e.g., literal or SOAP encoding.

The WSDL `<binding>` element, and its sub-elements, are used in combination with protocol specific elements.
- Each type of protocol, e.g., MIME, SOAP or HTTP GET or POST, has its own set of protocol specific elements and its own namespace.

A `<port>` element defines the location of a web service and we can think of it as the URL where the service can be found. A `<port>` associates an endpoint, for instance, a network address location or URL, with a specific WSDL `<binding>` element.
- It is even possible for two or more `<port>` elements to assign different URLs to the same `<binding>` element. This might be, for instance, useful for load balancing or fail-over purposes.

A `<service>` element contains a collection (usually one) of WSDL `<port>` elements. Each `<service>` element is named, and each name must be unique among all services in a WSDL document.

The previous figure illustrates how the elements of the WSDL interface and implementation are related to each other.
The structure of the `<binding>` element resembles that of the `<portType>` element. This is no coincidence as the binding must map an abstract port type description to a concrete implementation.

- The `<type>` attribute identifies which `<portType>` element this binding describes. The `<binding>` element declared in the listing of Listing 2 is actually composed of two different namespaces. There are elements that are members of the WSDL 1.1 namespace `http://schemas.xmlsoap.org/wsdl/`, which is declared in Listing 1 and is the default namespace of the WSDL document.

- The WSDL generic binding elements are as shown in this listing: `<operation>`, `<input>` and `<output>`. The sub-elements of the `<binding>` element ( `<operation>`, `<input>` and `<output>`) map directly to corresponding children of the `<portType>` element.

- The `soapbind:binding`, `soapbind:operation`, and `soapbind:body` elements, on the other hand, are protocol specific. They are SOAP-specific elements which are members of the namespace for the SOAP-WSDL binding `http://schemas.xmlsoap.org/wsdl/soap/` that is also declared in Listing 1.
the `<binding>` element named POMessageSOAPBinding links the `<portType>` element named PurchaseOrderPortType (refer to Listing 1) to the `<port>` element named PurchasePort.

- This is affected through the binding name POMessageSOAPBinding as can be seen from the dotted arrow in Listing 2.
- Several bindings may represent various implementations of the same `<portType>` element. If a service supports more than one protocol, then the WSDL `<portType>` element should include a `<binding>` for each protocol it supports.
- For a given `<portType>` element, a `<binding>` element can describe how to invoke operations using a single messaging/transport protocol, e.g., e.g., SOAP over HTTP, SOAP over SMTP or a simple HTTP POST operation, or any other valid combination of networking and messaging protocol standards.

It must be noted that a binding does not contain any programming language or service implementation-specific details. How a service is implemented is an issue completely external to WSDL.
In Listing 2 the `<binding>` element is shown to contain a `<soapbind:binding>` element that specifies the protocol by which clients access the web service.

More specifically, the purpose of the SOAP binding element `<soapbind:binding>` is to signify that the messaging style is RPC and that the SOAP protocol format is going to be used as a binding and transport service.

This declaration applies to the entire binding. It signifies that all operations of the PurchaseOrderPortType are defined in this binding as SOAP messages. It then becomes the responsibility of SOAP to take the client from the abstract WSDL specification to its implementation.

- Since SOAP is used for this purpose, SOAP’s namespace must also be used.
- A WSDL implementation allows the use of other protocols, such as HTTP without using SOAP and MIME. If either of these protocols needs to be used, the `<soapbind:binding>` element must be declared using the namespace prefixes associated with it, i.e., HTTP or MIME.
Listing 2 indicates that the `transport` attribute specifies HTTP as the lower-level transport service that this binding will use.

The style attribute defines the type of default operations within this binding, which is "rpc". The transport and style attributes are part of the SOAP binding element `<soapbind:binding>` (not to be confused with the WDSL `<binding>` element).

The abstract operation SendPurchase together with its input and output messages from the abstract service interface description (see Listing 1) is mapped to SOAP messages. The data types of these messages that are abstractly described by means of XSD in the service interface description should be SOAP encoded for the transfer.
In Listing 2 the `<soapbind:operation>` element is used to indicate the binding of a specific operation, e.g., SendPurchase, to a specific SOAP implementation.

The SOAPAction attribute in the `<soap:operation>` element is an attribute that a SOAP client will use to make a SOAP request.

The SOAPAction attribute of the `<soapbind:operation>` is used to specify the HTTP SOAPAction header, which in turn can be used by SOAP servers as an indication of the action that should be taken by the receipt of a message at runtime.

- This usually captures the name of a method to invoke in a service implementation.

- The SOAPAction attribute is a server specific URI used to indicate the intent of request. It can contain a message routing parameter or value that helps the SOAP runtime system dispatch the message to the appropriate service.

- The value specified in this attribute must also be specified in the SOAPAction attribute in the HTTP header of the SOAP request. The purpose of this is to achieve interoperability between client and service provider applications. The SOAP client will read the SOAP structure from the WSDL file and coordinate with a SOAP server on the other end.
The style of messaging has a direct impact on how the body of the SOAP message is constructed, thus declaring the correct style, “rpc” or “document” is important.

- When RPC-style messaging is used then the Body of the SOAP message will contain an element that represents the operation that needs to be performed.
- This element gets its name from the <operation> defined in the <portType> element, in our example SendPurchase.
- The <operation> element will contain parameter and response elements that are directly derived from the <input> and <output> parts of its message definition. Next slide illustrates how the <operation> defined in the <portType> element and its message parts are mapped to an RPC-style SOAP Message.
<soapbind:body> element

The <soapbind:body> element enables applications to specify the details of the input and output messages and enable the mapping from the abstract WSDL description to the concrete protocol description. Consider for example, the <input> elements for the SendPurchase operation.

- The entire POMessage message from the <portType> declaration for the SendPurchase operation is declared to be abstract. This is indicated by the use="literal" attribute. This means that the XML defining the input message and its parts are in fact abstract, and the real, concrete representation of the data is to be derived.

- The purpose of the use attribute within the <soapbind:body> element enables applications to specify how the parts of the message are defined.

- Use = "literal" encoding indicates that the resulting SOAP message contains data formatted exactly as specified in the abstract definitions (<type>, <message>, and <portType> sections). Consequently, the datatype that the message part references will be serialised according to its exact representation in the type definition section, i.e., according the XML schema.

- Use = “encoded”, then each message part references an abstract type using the type attribute. These abstract types are used to produce a concrete message by applying an encoding specified by the encodingStyle attribute. The part names, types and value of the namespace attribute are all inputs to the encoding, although the namespace attribute only applies to content not explicitly defined by the abstract types.

```xml
<soap:body use="encoded" namespace="http://example.com/stockquote" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>`
"
The `<service>` element finally binds the web service to a specific network-addressable location. It takes the bindings that were declared previously and ties them to one or more `<port>` elements, each of which represents a web service.

- **A `<service>` is modelled as a collection of related ports - where a `<port>` element is a single endpoint defined as a combination of binding and a network address - at which a service is made available.**

- A web service exchanges messages in a defined format through a `<port>` element. More precisely, the `<port>` element is a single protocol-specific address to an individual binding element. Here, a mandatory location URI must be provided to denote the physical endpoint that service requesters must use to connect to the service.

- The listing in Listing 2 contains only one web service, viz. the PurchaseOrderService, thus only the `<port>` element named PurchasePort is used to reveal the service location.

- However, the service PurchaseOrderService could, for instance, contain three ports all of which use the PurchaseOrderPortType but are bound to SOAP over HTTP, SOAP over SMTP, and HTTP GET/POST, respectively:
  - For instance, a PC-desktop application may use SOAP over HTTP,
  - while a WAP application designed to run on a cellular phone may use HTTP GET/POST, since an XML parser is typically not available in a WAP application.
  - All three services are semantically equivalent in that they all take a purchase order number, a date and customer details as input and return an associated invoice. By employing different bindings, the service is more readily accessible on a wider range of platforms.
Elements of the WSDL as part of requester-service interaction

Each message part is defined by some type, either custom defined or XSD provided.

- **Type-a custom defined**
- **int XSD built-in**

Input & output messages form an operation. A set of operations forms a port-type.

A port exposes the service using a specific binding.

**Client #B**

- SOAP/HTTP request message
- SOAP/HTTP response message

**Client #B**

- HTTP GET request message
- HTTP response message

**service**

- port#1
- port#2

A binding specifies how the operations are invoked using a specific protocol, e.g., SOAP.

A service is a collection of related endpoints (ports) that the client wishes to invoke.

Java implementation
Elements of the WSDL as part of requester-service interaction

- Previous Figure summarises several of the constructs explained in the previous slides by illustrating the various WSDL elements involved in a client-service interaction.
- This figure shows one client invoking a web service by means of SOAP over HTTP and another client invoking the same service by means of HTTP.
- The figure depicts a single service that contains multiple ports. The service may contain more than one ports, which are bound to binding elements, and that binding elements are associated with a <portType> by means of a type relationship.
- Service providers all have different bindings and/or addresses. Port addresses are specified by the <saopbind:address> element of <port>, as already explained.
- In the case of multiple ports, these ports are alternatives, so that the client of the service can choose which protocol they want to use to communicate with the service (possibly programmatically by using the namespaces in the bindings), or which address is closest.
WSDL

- Structure of WSDL documents
  - WSDL interface definition part
  - WSDL interface implementation part

- WSDL Message Exchange Patterns

- Using WSDL
WSDL Message Exchange Patterns

- WSDL interfaces support **four types of operations**.
- These operations represent the most common interaction patterns for web services. Since each operation defined by WSDL can have an input and/or an output, the four WSDL interaction patterns represent possible combinations of input and output messages.
- The WSDL operations correspond to the incoming and outgoing versions of **two basic operation types**:
  - an incoming single message passing operation and its outgoing counterpart ("one-way" and "notification" operations),
  - the incoming and outgoing versions of a synchronous two-way message exchange ("request-response" and "solicit response").
- Any combination of incoming and outgoing operations can be included in a single WSDL interface.
  - As a result, the four types of operations presented above provide support for both push and pull interaction models at the interface level.
  - Outgoing operations in WSDL support loosely coupled peer-to-peer interactions between services.
A one-way operation is an operation in which the service endpoint receives a message, but does not send a response.

- An example of a one-way operation might be an operation representing the submission of an order to a purchasing system. Once the order is sent, no immediate response is expected.
- This message exchange pattern is typically thought of as asynchronous messaging. In an RPC environment, a one-way operation represents a procedure call to which no return value is assigned.
- A one-way message defines only an input message. It requires no output message and no fault. Next to the request/response message exchange pattern, this is the most popular message exchange pattern employed today.
Notification operation

- A notification operation is an operation in which the service endpoint sends a message to a client, but it does not expect to receive a response.

- This type of messaging is used by services that need to notify clients of events. A web service that uses the notification-messaging pattern follows the push model of distributed computing.
  - The assumption is that the client (subscriber) has registered with the web service to receive messages (notifications) about an event. Notification is when a <portType> element contains an <output> tag but no <input> message definitions.
  - An example of this could be a service model in which events are reported to the service and where the endpoint periodically reports its status. No response is required in this case, as most likely the status data is assembled and logged and not acted upon immediately.
A request/response operation is an operation in which the service end point receives a message and returns a message in response. In the request/response message pattern a client requests that some action is taken from the service provider.

If an <operation> element is declared with a single <input> element followed by a single <output> element, it defines a request/response operation. By listing the <input> tag first, the <operation> indicates that the web service receives a message that is sent by the client. Listing the <output> tag second indicates that the web service should respond to the message.

An example of this is the SendPurchase operation as an example of the request/response-messaging pattern. The SendPurchase operation receives as input message a containing purchase order (order number and date) and customer details and responds with a message containing an invoice.

In an RPC environment, this is equivalent to a procedure call, which takes a list of input arguments and returns a value.
Solicit/response operation

- A solicit/response operation is an operation in which the service endpoint sends a message and expects to receive an answering message in response.
- This is the opposite of the request/response operation since the service endpoint is initiating the operation (soliciting the client), rather than responding to a request.
- Solicit/response is similar to notification messaging, except that the client is expected to respond to the web service. As with notification messaging, clients of the solicit/response web services must subscribe to the service in order to receive messages. With this type of messaging the <portType> element first declares an <output> tag and then a <input> message definition – exactly the reverse of a request/response operation.
  - An example of this operation might be a service that sends out order status to a client and receives back a receipt.
WSDL

- Structure of WSDL documents
  - WSDL interface definition part
  - WSDL interface implementation part

- WSDL Message Exchange Patterns

- Using WSDL
WSDL and proxy class - 1

How to use WSDL?

- Developers can implement web services logic within their applications by incorporating available web services.
- The mechanism that makes this possible is called the proxy class.
  - Proxy classes enable developers to reference remote web-services and use their functionality within a local application, as if the data the services return were generated locally.
  - The application developer communicates with any remote objects by sending messages to these local objects, which are commonly known as *proxy* objects.
The proxy classes (or *stub* classes) are client-side images of the remote (provider) object classes that implement the web services.

- The server-side are commonly known as *skeletons* in the distributed computing systems domain.

- The proxy object is simply a local object with methods that are merely a pass-through to the web service it is representing.

- The role of the proxy class is to act as the local representative for the remote object and basically is, to the client, the remote reference. However, instead of executing an invocation, the proxy forwards it in a message to a remote object.
WSDL code generators

- WSDL is well suited (XML expressiveness) for code generators that can read WSDL definitions and generate a programming interface for accessing a web service.
  - For instance, a JAX-RPC provider may use WDSL 1.1 to generate Java RMI interfaces and network stubs, which can be used to exchange messages with a web service interface.
- WSDL code generator tools allow automatic creation of web services, automatic generation of WSDL files and invocation of web services.
- These toolkits speed the creation of web-services
  - by generating the service implementation template code from the WSDL specifications, leaving only the application-specific implementation details to the developer.
  - They also simplify the development of client applications by generating service proxy code from the WSDL specification.
- Several code generators can generate interfaces and network stubs from WSDL documents. These include amongst others IBM Websphere Studio Application Developer, Microsoft .NET Studio, and Apache Axis.
Generating proxies from WSDL code generators (example Java)
WSDL State-of-the-art

The World Wide Web Consortium (W3C) has issued Web Services Description Language (WSDL) 1.2 and WSDL 1.2 Bindings as W3C Public Working Drafts.

WSDL version 1.2 enhancements to version 1.1 include the following:

- **Language clarifications** that make it easier for developers to understand and use WSDL.
- **Support** for XML Schemas and XML Information Set.
- **Better definition** for the HTTP 1.1 binding and a binding for SOAP 1.2, which allows description of services using the most current version of SOAP.
- Adopting a conceptual **framework approach** to define a description model of WSDL. This **conceptual model** comprises a set of components with attached properties, which collectively describe a web service. Each subsection in this conceptual model describes a different type of component, its defined properties, and its representation as an XML Information Set.
WSDL Outlook

- WSDL is in its current version an extension of the IDL model to support interaction through the Internet:
  - XML as syntax and type system
  - possibility of grouping operations into a service
  - different options for accessing the service (addresses and protocols)
- This is its great advantage ...
  - it is straightforward to adapt existing middleware platforms to use or support WSDL
  - automatic translation from existing IDLs to WSDL is trivial
- ... but also the disadvantage
  - electronic commerce and B2B interactions are not single service calls
  - WSDL does not reflect the structure of the procedures to follow to correctly interact with a service (conversations)
    - business protocol = set of valid conversations
- Without a business protocol, most of the development work is still manual
ebXML shows here again what could be a possible evolution path for WSDL (or the type of technology that is being built on top of WSDL)

- ebXML does not consider a client/server model but an interaction between partners (peer-to-peer)

Consequently, the service description model for ebXML is the description of how two business processes interact with each other:

- partners publish their processes (an external view over them)
- a collaboration agreement is drawn based on those processes
- the collaboration agreement describes the business protocol between those partners
Flow Models are used for the specification of complex service interactions.

The flow model describes:
- how activities (here implemented as web services operations) are combined,
- the order that these steps are executed,
- the decision points where steps may or may not have to be performed and
- the passing of data items between the steps involved.
Automatic development

- The ultimate goal of WSDL is to provide support for automating as much as possible for the development process for Web services:
  - given the WSDL description, a WSDL compiler generates the stubs or skeletons necessary to develop clients that can interact with the service
  - for that purpose, WSDL must rely on a standard protocol so that generic stubs can be created, this is where SOAP comes into the picture
  - WSDL is meant as a bridge between internal services and external (Web) services

- Similarly, the ultimate goal in ebXML is to automate the process of developing a collaboration agreement, deploying it and enforcing its rules:
  - given a collaboration agreement (possibly a standard one), one should be able to automatically generate a stub or skeleton for the individual business processes at the ends of the agreement
  - partners need only to extend the stub process with their own internal logic
  - this is why ebXML needs more than SOAP as the agreement is used to control and direct the flow of messages between partners at the platform level