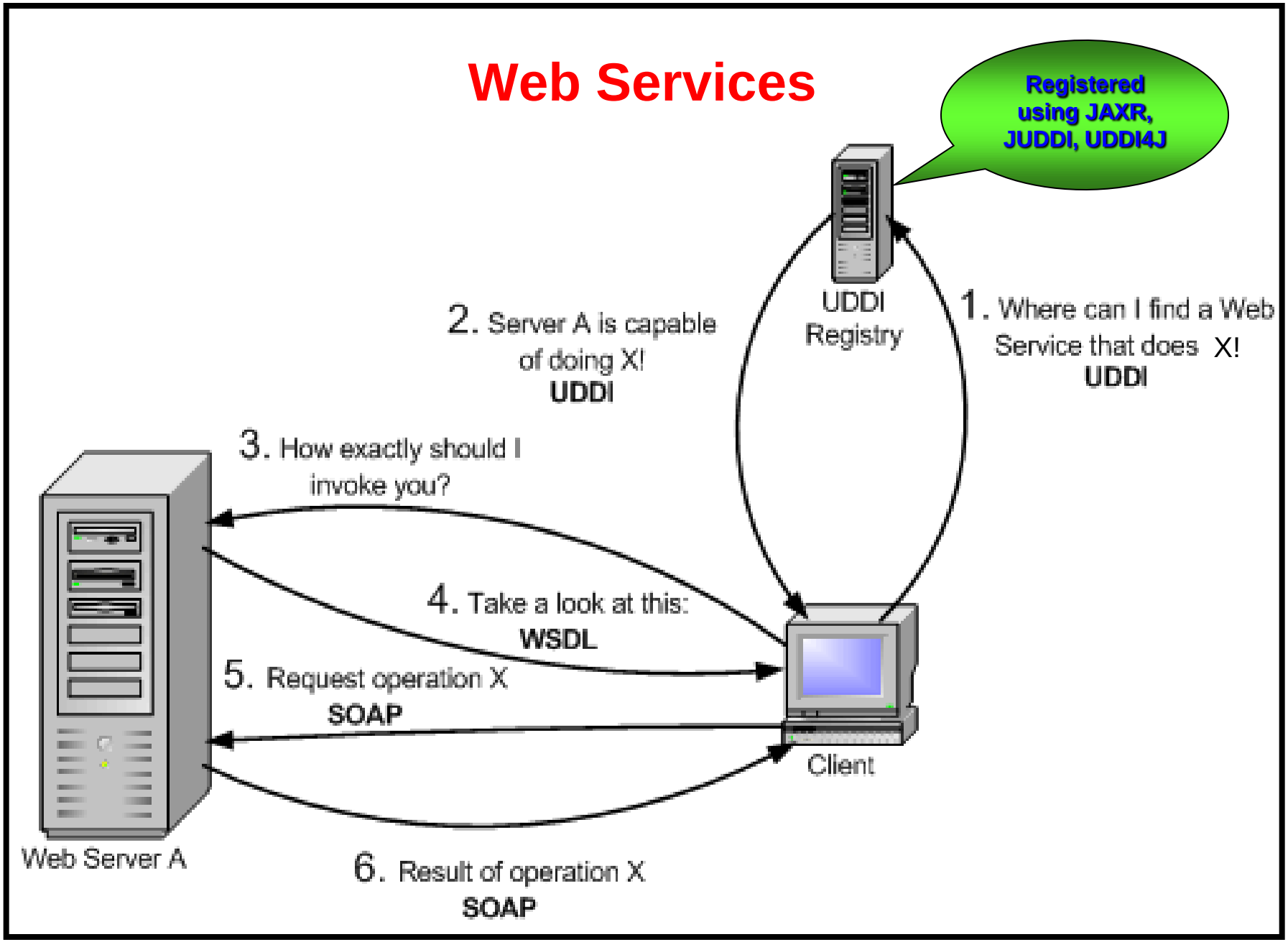
Web Services

Web Services
Part II

Web Services



Client-Service Implementation

- Suppose we have found the service and have its WSDL description, i.e. got past step 4.
- In the implementation, it is convenient to use stubs - java classes suitable for web services defined with WSDL.

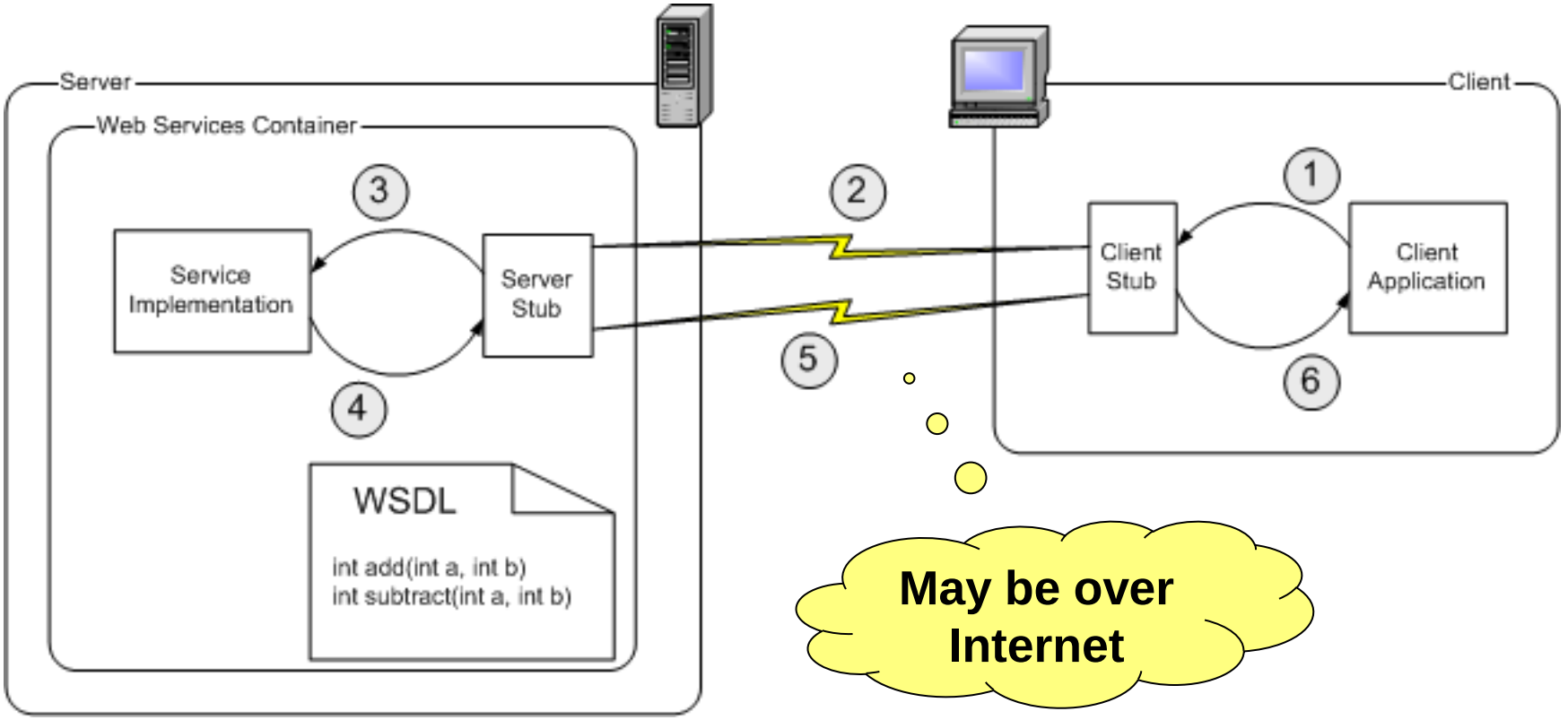
Client Stub

- Between the client code and the network is a client stub, sometimes called client proxy.
- The client stub is responsible for taking a request from the client and converting the request into a SOAP request on the network - marshalling.
- Also responsible for receiving SOAP responses on network and converting to a suitable form for client.

Server Stub

- Between the service and the network is a server stub, sometimes called a skeleton.
- Responsible for receiving a SOAP request from the client stub and converting it into a suitable form for the service –un-marshalling.
- Also converts the response from the service into a SOAP message for the client stub.

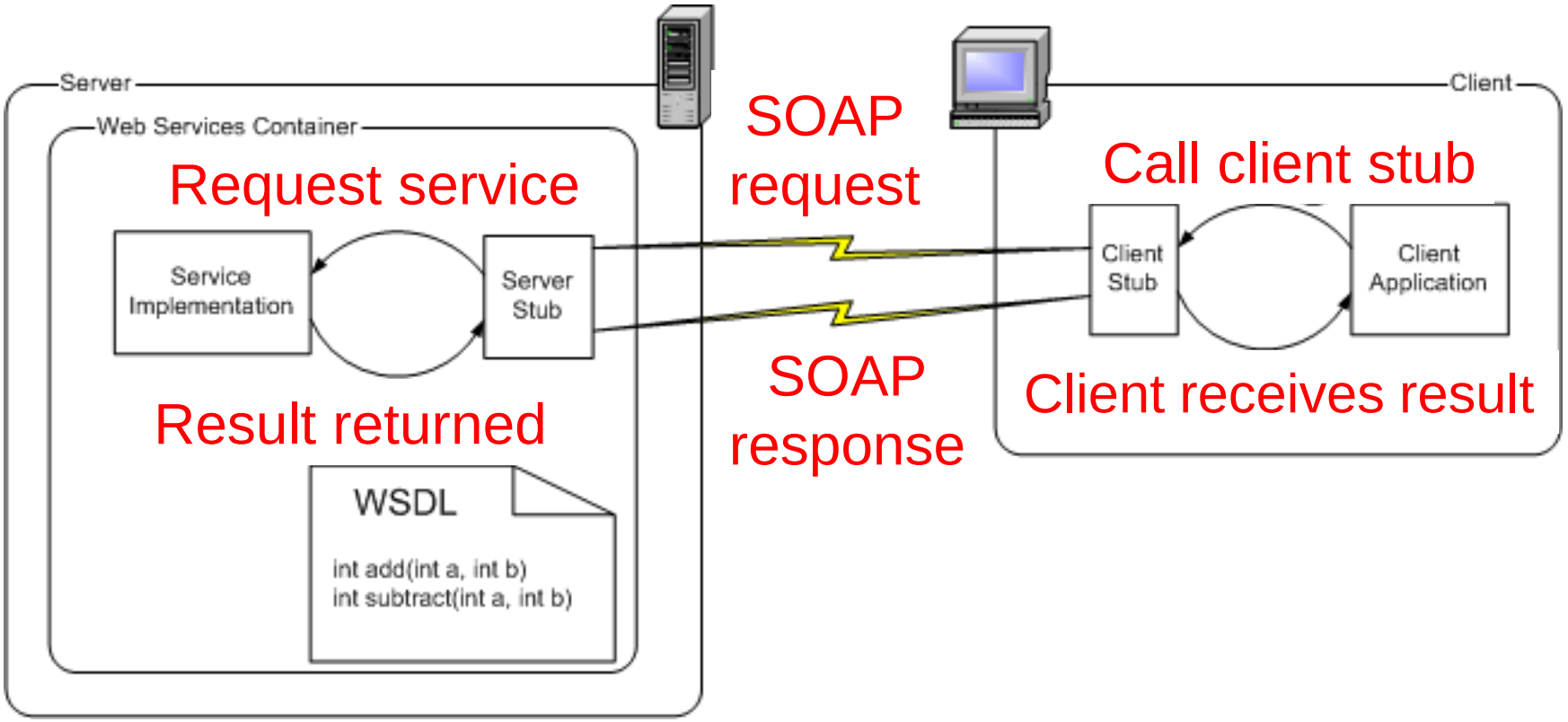
Web Service Application



Steps

- Client calls client stub.
- SOAP request sent across network (LAN/WAN)
- Server stub receives request and sends request to service
- Service send result to server stub
- Server stub sends result across network to client stub.
- Client stub sends result to client.

Web Service Application



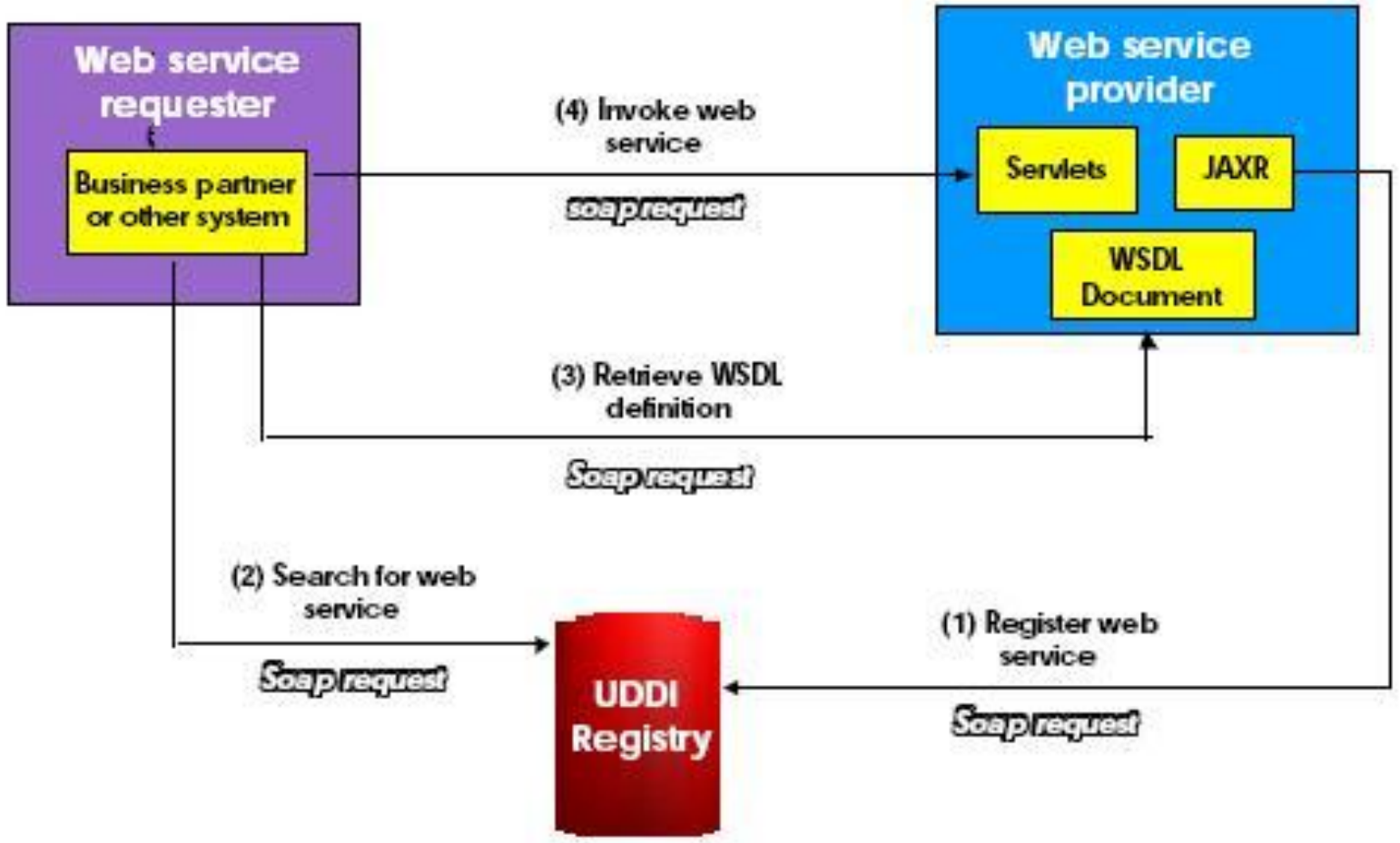
WSDL

Web Service Description

- Need a way of formally describing a service, what is does, how it is accessed, etc.
- An Interface Description Language (IDL)

Web Service Description

- Where does it fit?



Web Service Definition Language (WSDL)

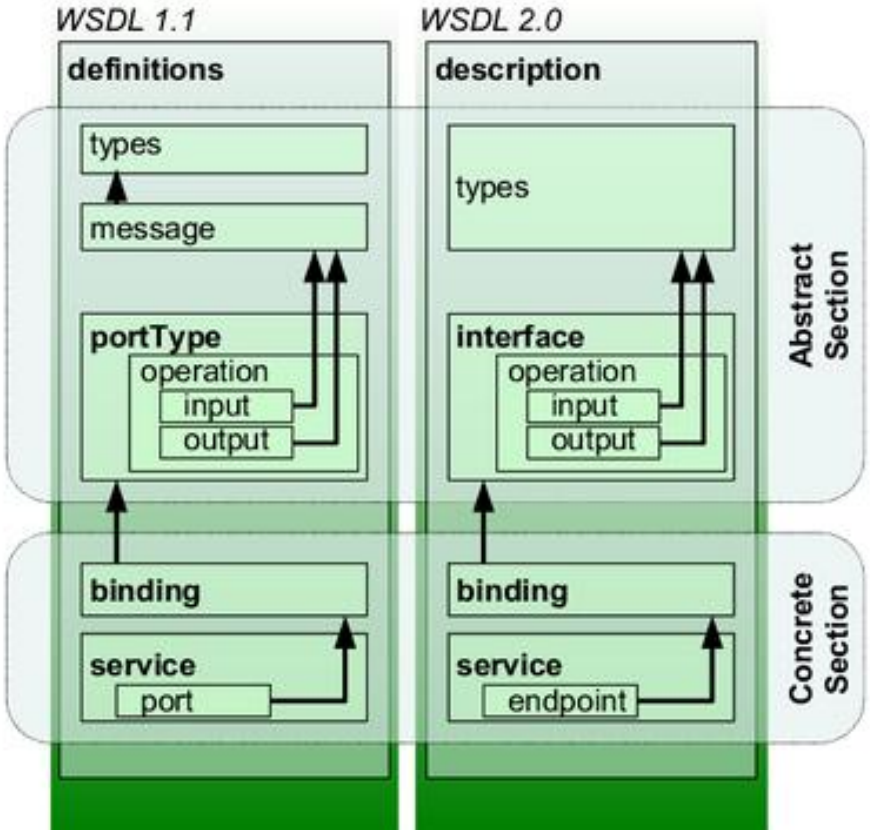
A W3C standard XML document that describes three fundamental properties of a service:

- *What* it is - operations (methods) it provides.
- *How* it is accessed - data format, protocols.
- *Where* it is located - protocol specific network address.

Web Service Definition Language (WSDL)

- XML language for describing web services
- Web service is described as
  - A set of communication endpoints (ports)
- Endpoint is made of two parts
  - Abstract definitions of operations and messages
  - Concrete binding to networking protocol (and corresponding endpoint address) and message encoding (ex. SOAP over HTTP)
- Why this separation?
  - Enhance reusability

Web Service Definition Language (WSDL)



Web Service Definition Language (WSDL)

WSDL Document Example

- Simple service providing stock quotes
- A single operation called GetLastTradePrice
- Deployed using SOAP 1.1 over HTTP
- Request takes a ticker symbol of type string
- Response returns price as a float

Elements of a WSDL document

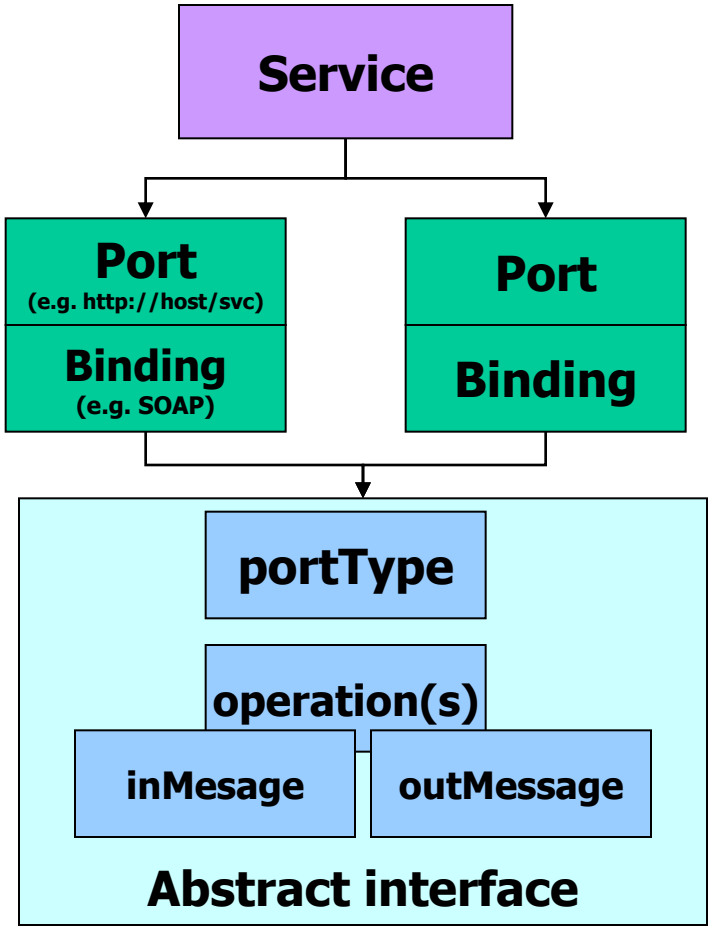
The WSDL Document Structure

A WSDL document describes a web service using these **major** elements:

- **<portType>** The operations performed by the web service
- **<message>** The messages used by the web service
- **<types>** The data types used by the web service
- **<binding>** The communication protocols used by the web service
- Other are Operation, port and service

WSDL Structure

```
<definitions>
    <types>
        definition of types.....
    </types>
    <message>
        definition of a message....
    </message>
    <portType>
        definition of port.....
    </portType>
    <binding>
        definition of a binding....
    </binding>
</definitions>
```



Types

- The **<types>** element defines the data types that are used by the web service.
- For maximum platform neutrality, WSDL uses XML Schema syntax to define data types.

Type Example

```
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
xmlns:tns="http://example.com/stockquote.wsdl"
xmlns:xsd1="http://example.com/stockquote.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/”>
<types>
    <schema targetNamespace="http://example.com/stockquote.xsd"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="TradePriceRequest">
          <complexType>
                   <all>
                            <element name=”tickerSymbol" type="string"/>
                   </all>
          </complexType>
    </element>
          <element name="TradePrice">
          <complexType>
                   <all>
                            <element name="price" type="float"/>
                   </all>
          </complexType>
    </element>
    </schema>
</types>
```

message

- The **<message>** element defines the data elements of an operation.
- Each message can consist of one or more parts. The parts can be compared to the parameters of a function call in a traditional programming language.
- Abstract, typed definitions of data being exchanged

portType/interface

The **<portType>** element is the most important WSDL element.

- It describes a web service, the operations that can be performed, and the messages that are involved.
- The <portType> element can be compared to a function library (or a module, or a class) in a traditional programming language.

WSDL <portType> operation types

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

TYPE	DEFINTION
One-Way	**The operation can receive a message but will not return a response**
Request-response	**The operation can receive a request and will return a response**
Solicit-response	**The operation can send a request and will wait for a response**
Notification	**The operation can send a message but will not wait for a response**

port, service, operation

- Describes "**where**" service is.
- **port** - describes how a binding is deployed at the endpoint of a network
  - Defines a single communication endpoint
  - Endpoint address for binding
  - URL for HTTP, email address for SMTP
- **service** - a named collection of ports
- **operation**- Abstract description of an action
  - Refers to an input and/or output messages

Ex. Messages, Operation, Port type

```
<message name="GetLastTradePriceInput">
   <part name="body" element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput">
   <part name="body" element="xsd1:TradePrice"/>
</message>
<portType name="StockQuotePortType">
   <operation name="GetLastTradePrice">
        <input message="tns:GetLastTradePriceInput"/>
        <output message="tns:GetLastTradePriceOutput"/>
   </operation>
<!-- More operations -->
</portType>
```

Binding

Describes “how” the elements in abstract interface (portType) are converted in actual data representations and protocols e.g. SOAP over HTTP.

Ex. Binding, Port, Service

```
<binding name="StockQuoteSoapBinding"
   type="tns:StockQuotePortType">
   <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
   <operation name="GetLastTradePrice">
        <soap:operation
            soapAction="http://example.com/GetLastTradePrice"/>
                <input> <soap:body use="literal" />
                </input>
                <output> <soap:body use="literal" />
                </output>
   </operation>
</binding>
<service name="StockQuoteService">
   <documentation>My first service</documentation>
   <port name="StockQuotePort"
        binding="tns:StockQuoteSoapBinding">
        <soap:address location="http://example.com/stockquote"/>
   </port>
</service>
```

Can be RPC

Can be **"Encoded"**, literal means that the type definitions literally follow an XML schema defn.

Binding

- The **binding** element has two attributes - the name attribute and the type attribute.
- The name attribute (you can use any name you want) defines the name of the binding, and the type attribute points to the port for the binding, in this case the "***StockQuotePortType***" port.
- The **soap:binding** element has two attributes - the style attribute and the transport attribute.
- The style attribute can be "rpc" or "document". In this case we use document. The transport attribute defines the SOAP protocol to use on HTTP.
- The **operation** element defines each operation that the port exposes.
- For each operation the corresponding SOAP action has to be defined. One must also specify how the input and output are encoded. In this case we use "literal".

Binding

- The **binding** element has two attributes - the name attribute and the type attribute.
- The name attribute (you can use any name you want) defines the name of the binding, and the type attribute points to the port for the binding, in this case the "***StockQuotePortType***" port.

```
<binding name="StockQuoteSoapBinding"
   type="tns:StockQuotePortType">
   :
   :
   :
   :    :
<service name="StockQuoteService">
   <documentation>My first service</documentation>
   <port name="StockQuotePort"
        binding="tns:StockQuoteSoapBinding">
        <soap:address location="http://example.com/stockquote"/>
   </port>
</service>
```

Binding

- The **soap:binding** element has two attributes - the style attribute and the transport attribute.
- The style attribute can be "rpc" or "document". In this case we use document. The transport attribute defines the SOAP protocol to use on HTTP.

```
:
:
: <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http"/>
:
:
:
```

Binding

- The **operation** element defines each operation that the port exposes.
- For each operation the corresponding SOAP action has to be defined. One must also specify how the input and output are encoded. In this case we use "literal".

:

```
<operation name="GetLastTradePrice">
      <soap:operation

   soapAction="http://example.com/GetLastTradePrice"/>
            <input> <soap:body use="literal" />
            </input>
            <output> <soap:body use="literal" />
            </output>
</operation>
```

:

Binding

- The **binding** element has two attributes - the name attribute and the type attribute.
- The name attribute (you can use any name you want) defines the name of the binding, and the type attribute points to the port for the binding, in this case the "***StockQuotePortType***" port.
- The **soap:binding** element has two attributes - the style attribute and the transport attribute.
- The style attribute can be **"rpc" or "document"**. In this case we use document. The transport attribute defines the SOAP protocol to use on HTTP.
- The **operation** element defines each operation that the port exposes.
- For each operation the corresponding SOAP action has to be defined. One must also specify how the input and output are encoded. In this case we use "literal".

Binding

- **Document:** the content of <soap:Body> is specified by XML Schema defined in the <wsdl:type> section. It does not need to follow specific SOAP conventions. In short, the SOAP message is sent as one "document" in the <soap:Body> element without additional formatting rules having to be considered. Document style is the default choice.

- **RPC:** The structure of an RPC style <soap:Body> element needs to comply with the rules specified in detail in Section 7 of the SOAP 1.1 specification. According to these rules, <soap:Body> may contain only one element that is named after the operation, and all parameters must be represented as sub-elements of this wrapper element.

SOAP Binding (revisit)

- A WSDL document describes a web service. A WSDL binding describes how the service is bound to a messaging protocol, particularly the SOAP.
- A WSDL SOAP binding can be either a RPC style or a document style binding. A SOAP binding can also have an encoded use or a literal use. This gives us four style/use models:
  - RPC/encoded
  - RPC/literal
  - Document/encoded
  - Document/literal

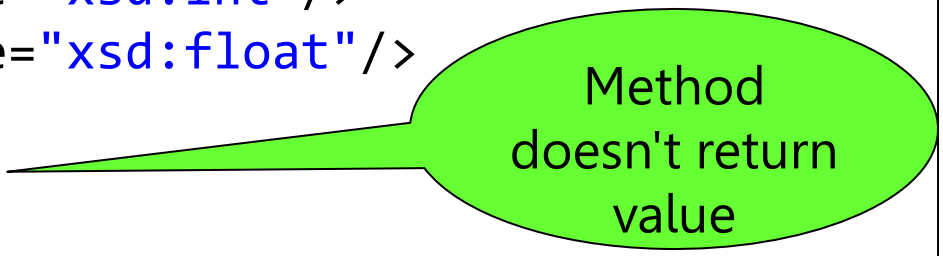
RPC/encoded WSDL for myMethod

- **public void myMethod(int x, float y);**

```
<message name="myMethodRequest">
     <part name="x" type="xsd:int"/>
     <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
     <operation name="myMethod">
          <input message="myMethodRequest"/>
          <output message="empty"/>
     </operation>
</portType>

<binding .../>
```



RPC/encoded SOAP message for myMethod

- `public void myMethod(int x, float y);`

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x xsi:type="xsd:int">5</x>
            <y xsi:type="xsd:float">5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

Method name appears in the message

overhead which degrades throughput

https://www.ibm.com/developerworks/library/ws-whichwsdl/

RPC/encoded SOAP message for myMethod

- Strengths
  - Straightforward WSDL.
  - The operation name appears in the message, so the receiver has an easy time dispatching this message to the implementation of the operation.
- Weaknesses
  - The type encoding info (such as xsi:type="xsd:int") is usually just overhead which degrades throughput performance.

RPC/encoded SOAP message for myMethod

- One cannot easily validate this message since only the <x ...>5</x> and <y ...>5.0</y> lines contain things defined in a schema; the rest of the soap:body contents comes from WSDL definitions.
- Although it is legal WSDL, RPC/encoded is not WS-I compliant.

RPC/literal WSDL for myMethod

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
```

RPC/literal SOAP message for myMethod

```
<soap:envelope>
    <soap:body>
        <myMethod>
          <x>5</x>
          <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

RPC/literal SOAP message for myMethod

- **Strengths**
  - The WSDL is still straightforward.
  - The operation name still appears in the message.
  - The type encoding info is eliminated.
  - **RPC/literal is WS-I compliant.**
- **Weaknesses**
  - You still cannot easily validate this message since only the <x ...>5</x> and <y ...>5.0</y> lines contain things defined in a schema; the rest of the soap:body contents comes from WSDL definitions.

Document/encoded

- Nobody follows this style. It is not WS-I compliant.

Document/literal WSDL for myMethod

```
<strong><types>
    <schema>
        <element name="xElement" type="xsd:int"/>
        <element name="yElement" type="xsd:float"/>
    </schema>
</types></strong>

<message name="myMethodRequest">
    <part name="x" <strong>element="xElement"</strong>/>
    <part name="y" <strong>element="yElement"</strong>/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
```

Document/literal SOAP for myMethod

```
<soap:envelope>
    <soap:body>
        <xElement>5</xElement>
        <yElement>5.0</yElement>
    </soap:body>
</soap:envelope>
```

Document/literal SOAP for myMethod

- **Strengths**
  - There is no type encoding info.
  - You can finally validate this message with any XML validator. Everything within the soap:body is defined in a schema.
  - Document/literal is WS-I compliant, but with restrictions

Document/literal SOAP for myMethod

- **Weaknesses**
  - The WSDL is getting a bit more complicated.
  - This is a very minor weakness, however, since WSDL is not meant to be read by humans.
  - The operation name in the SOAP message is lost. Without the name, dispatching can be difficult, and sometimes impossible.
  - WS-I only allows one child of the soap:body in a SOAP message.

Document/literal SOAP for myMethod

- The **document/literal** style seems to be an rearranged the strengths and weaknesses from the RPC/literal model.
- One can validate the message, but lose the operation name.
- Is there anything we can do to improve upon this?
- Yes. It's called the document/literal wrapped pattern.

Document/literal wrapped WSDL for myMethod

```
<types>
    <schema>
        <strong><element name="myMethod">
            <complexType>
                <sequence>
                    <element name="x" type="xsd:int"/>
                    <element name="y" type="xsd:float"/>
                </sequence>
            </complexType>
        </element>
        <element name="myMethodResponse">
            <complexType/>
        </element></strong>
    </schema>
</types>
<message name="myMethodRequest">
    <part name="<strong>parameters</strong>" element="<strong>myMethod</strong>"/>
</message>
<strong><message name="empty">
    <part name="parameters" element="myMethodResponse"/>
</message></strong>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
```

Document/literal wrapped SOAP message for myMethod

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

- It looks exactly like the RPC/literal SOAP message, but there's a subtle difference.
- In the RPC/literal SOAP message, the <myMethod> child of <soap:body> was the name of the operation.
- In the document/literal wrapped SOAP message, the <myMethod> clause is the name of the wrapper element which the single input message's part refers to.

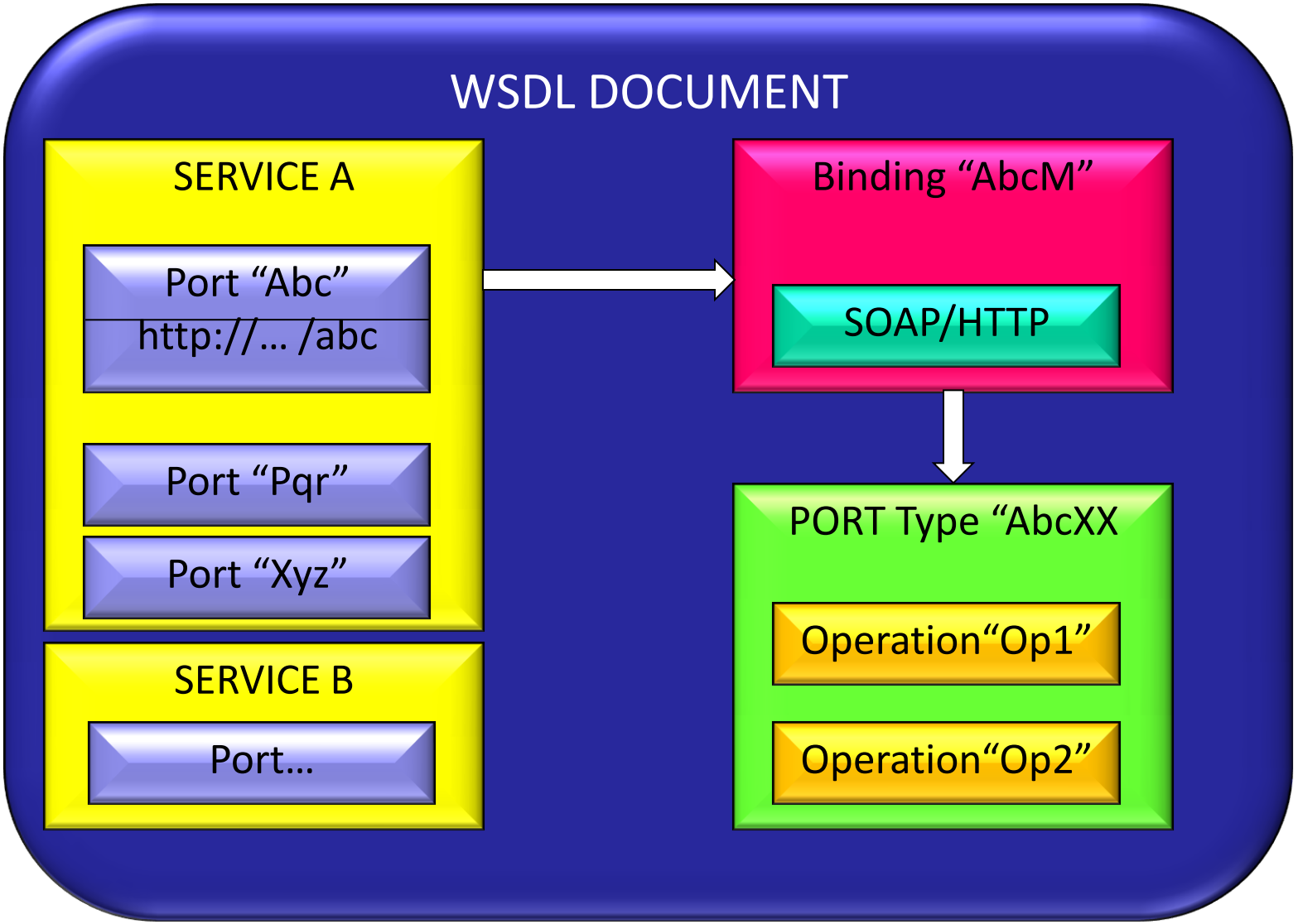
Document/literal wrapped SOAP message for myMethod

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

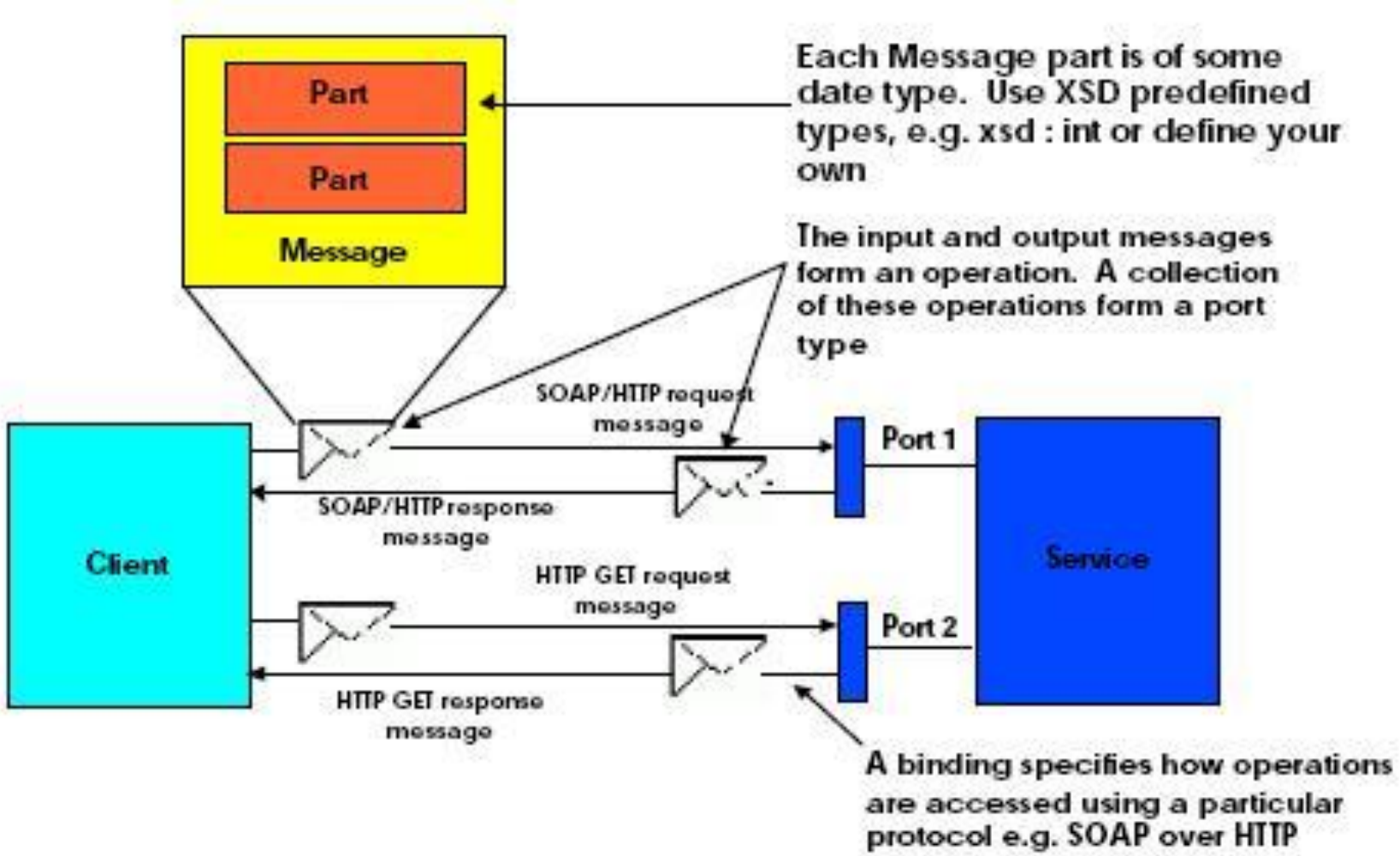
```
<soap:envelope>
    <soap:body>
        <myMethod>
          <x>5</x>
          <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

- It looks exactly like the RPC/literal SOAP message, but there's a subtle difference.
- In the RPC/literal SOAP message, the <myMethod> child of <soap:body> was the name of the operation.
- In the document/literal wrapped SOAP message, the <myMethod> clause is the name of the wrapper element which the single input message's part refers to.

WSDL view of web service



Web Service Invocation



Message definitions

Click here to see the Google Search wsdl file

For more information

- SOAP

http://www.w3c.org/TR/soap

- WSDL

http://www.w3c.org/TR/wsdl