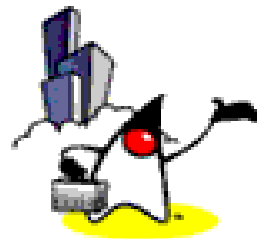




# **WSDL**

## **(Web Services Description Language)**



# Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents.
- Acknowledgments
  - Most slides and speaker notes in this presentation are created from W3C WSDL 1.1 specification

# Revision History

- 10/01/2002: version 1, Created (Sang Shin)
- 01/30/2004: version 2, updated (Sang Shin)
- Things to do
  - speaker notes need to be added

# Agenda

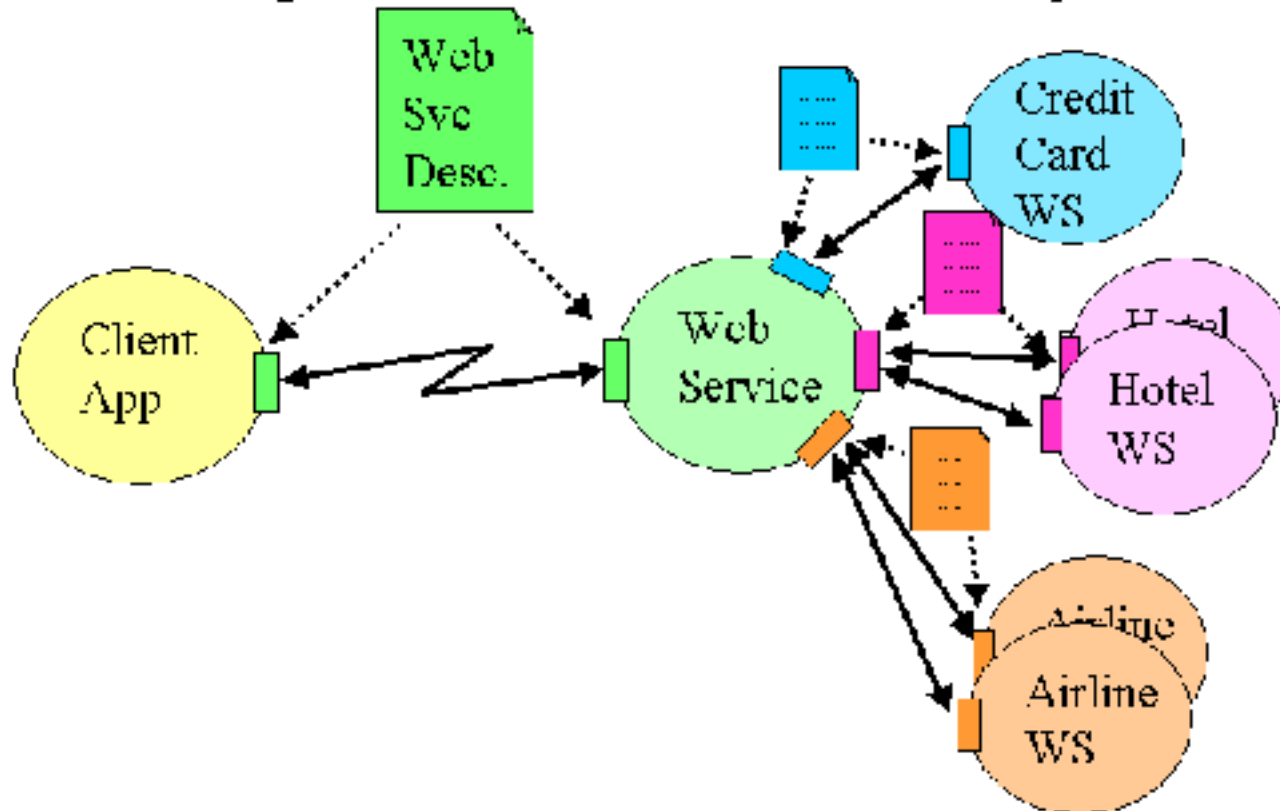
- What and Why WSDL?
- Example WSDL Document
- WSDL Document Elements
  - Binding and extensibility
- Importing & authoring style
- Application design & Tools
- Limitations of WSDL
- WSCI
- WSDL 1.2
- Java APIs for WSDL

# What is WSDL?

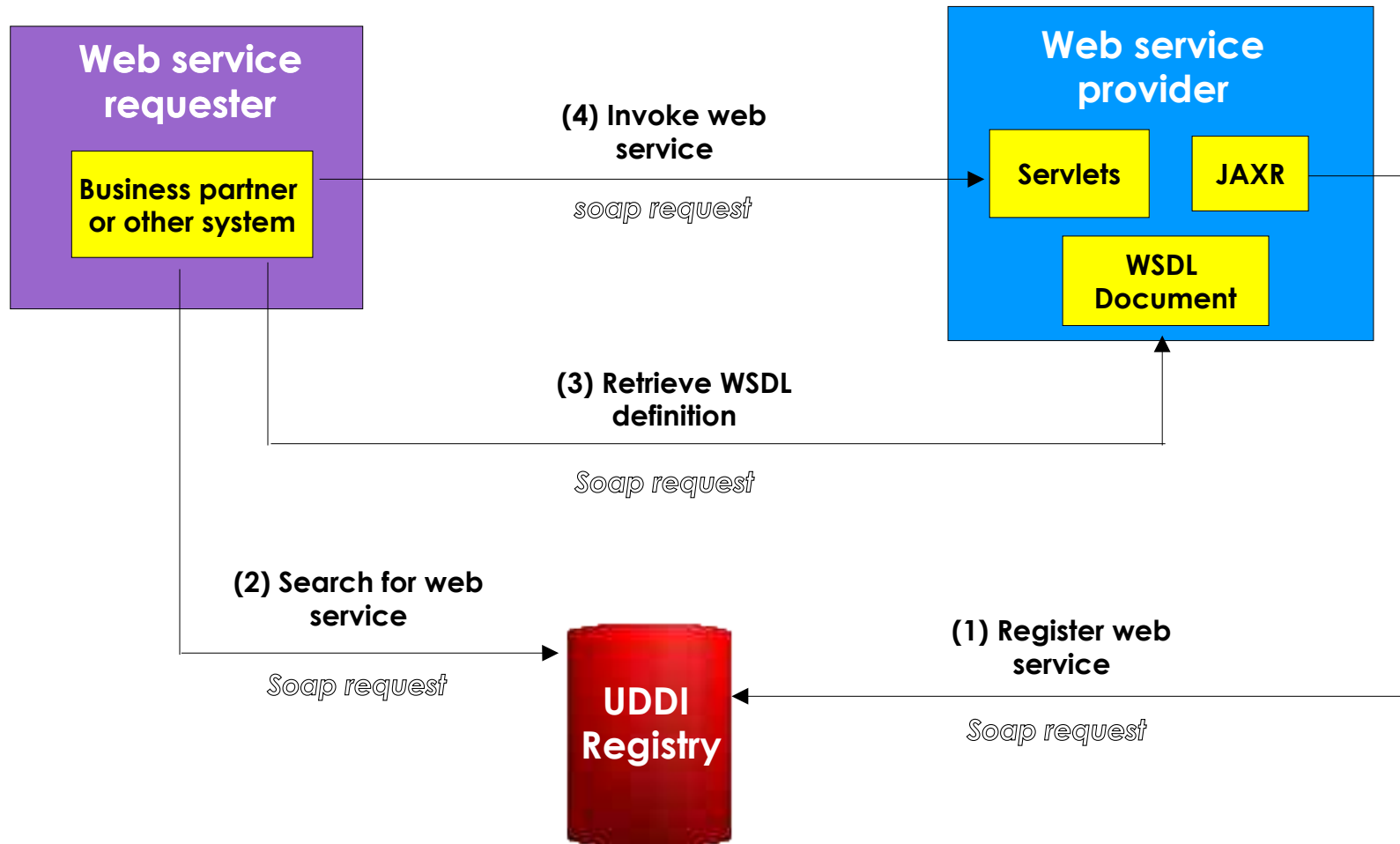
- XML language for **describing** web services
- Web service is described as
  - A set of **communication endpoints (ports)**
- Endpoint is made of two parts
  - **Abstract definitions of operations and messages**
  - **Concrete binding** to networking protocol (and corresponding endpoint address) and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

# What is WSDL?

## Many Web Service Descriptions



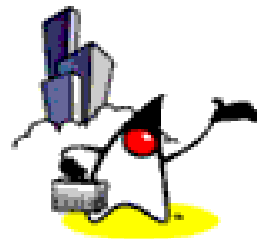
# Where is WSDL Used?



# Why WSDL?

- Enables **automation** of communication details between communicating partners
  - Machines can read WSDL
  - Machines can invoke a service defined in WSDL
- Discoverable through registry
- Arbitration
  - 3rd party can verify if communication conforms to WSDL

# WSDL Document Structure



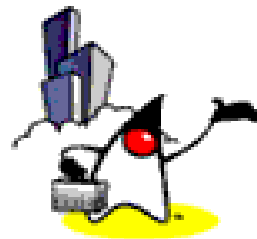
# WSDL Document Structure

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
    targetNamespace="your namespace here"
    xmlns:tns="your namespace here"
    xmlns:soapbind="http://schemas.xmlsoap.org/wsdl/soap">
  <wsdl:types>
    <xs:schema targetNamespace="your namespace here (could be another) "
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      <!-- Define types and possibly elements here -->
    </schema>
  </wsdl:types>
  <wsdl:message name="some operation input">
    <!-- part(s) here -->
  </wsdl:message>
  <wsdl:message name="some operation output">
    <!-- part(s) here -->
  </wsdl:message>
  <wsdl:portType name="your type name">
    <!-- define operations here in terms of their messages -->
  </wsdl:portType>
  <wsdl:binding name="your binding name" type="tns:port type name above">
    <!-- define style and transport in general and use per operation -->
  </wsdl:binding>
  <wsdl:service>
    <!-- define a port using the above binding and a URL -->
  </wsdl:service>
</wsdl:definitions>
```

# WSDL Namespaces

- <http://schemas.xmlsoap.org/wsd>
- <http://schemas.xmlsoap.org/wsd/soap>
- <http://www.w3.org/2001/XMLSchema>

# Example WSDL Document



# 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**

# WSDL Elements

- Types
- Message
- Operation
- Port Type
- Binding
- Port
- Service

# WSDL Elements

- Types
  - Data type definitions
  - Used to describe exchanged messages
  - Uses W3C XML Schema as canonical type system

# WSDL Example: Types

```
<definitions name="StockQuote"
  targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd="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>
```

# WSDL Elements

- Messages
  - **Abstract**, typed definitions of **data** being exchanged
- Operations
  - **Abstract** description of an **action**
  - Refers to an **input** and/or **output messages**
- Port type
  - **Collection** of operations
  - **Abstract definition** of a service

# Example:

## 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>
```

# WSDL Elements

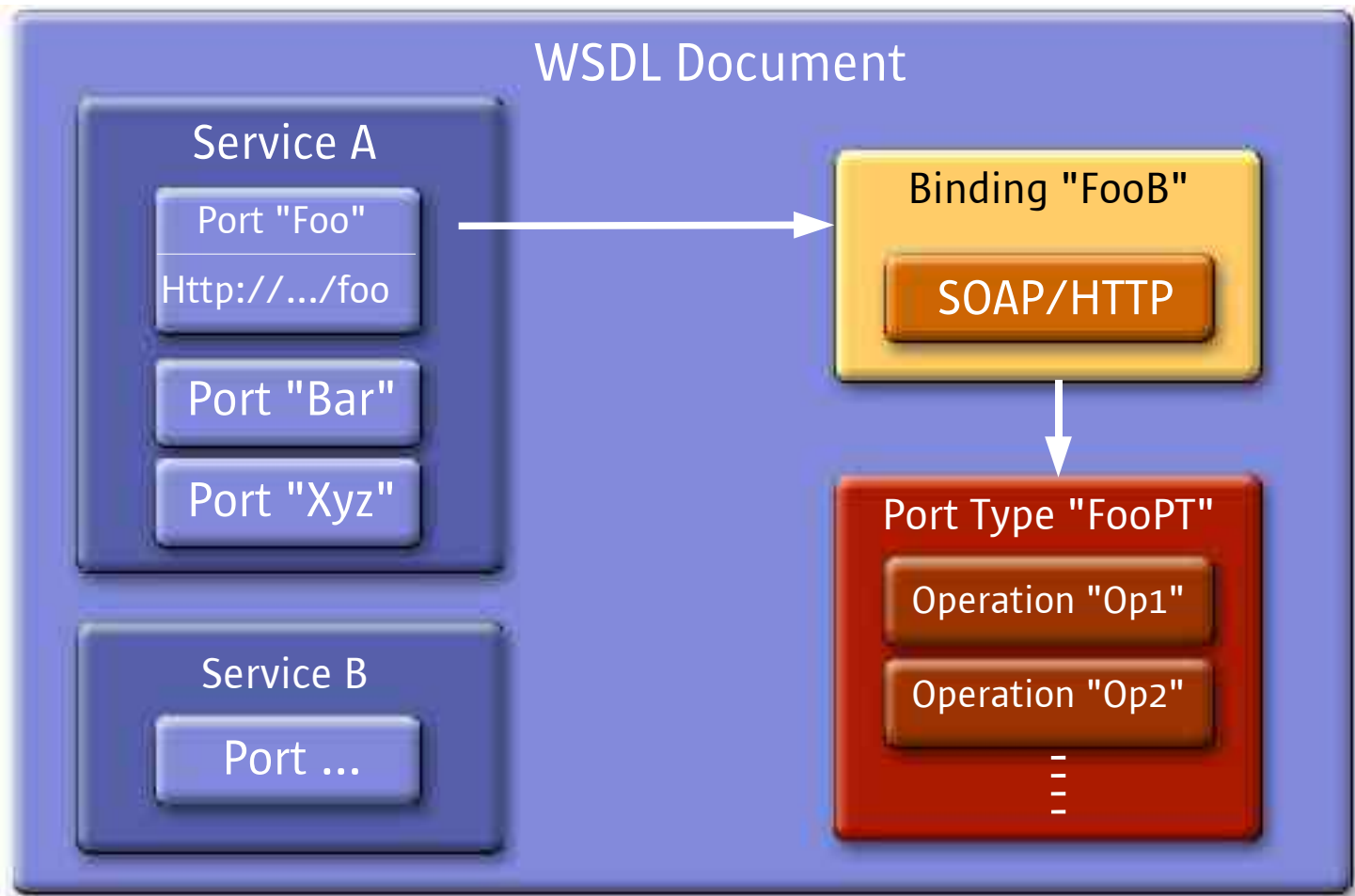
- Binding
  - **Concrete protocol and data format** (encoding) for a particular Port type
    - Protocol examples: SOAP 1.1 over HTTP or SOAP 1.1 over SMTP
    - Encoding examples: SOAP encoding, RDF encoding
- Port
  - Defines a single communication endpoint
  - **Endpoint address** for binding
  - URL for HTTP, email address for SMTP
- Service
  - **Aggregate set of related ports**

# Example: Binding, Port, Service

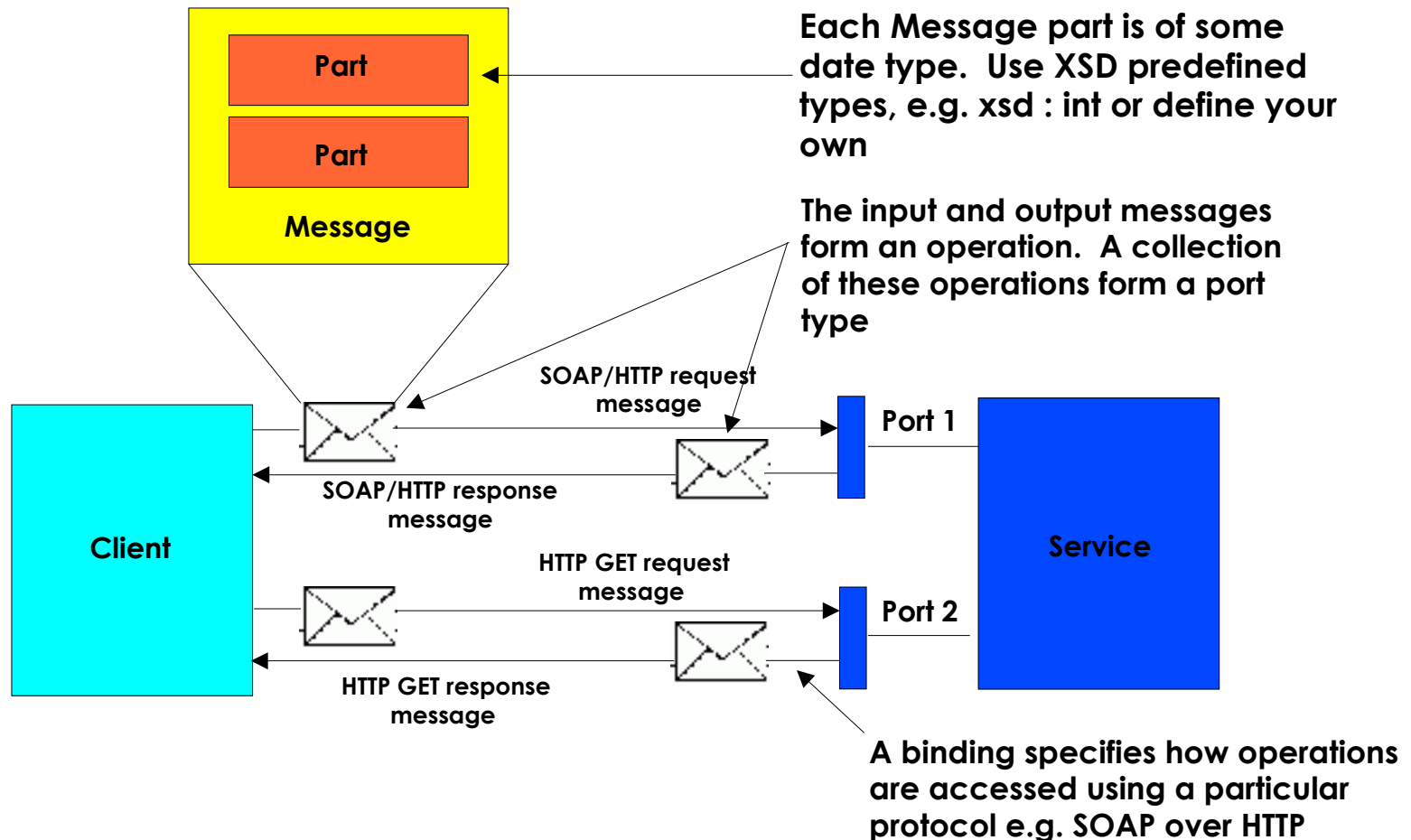
```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
    <soap:operation
      soapAction="http://example.com/GetLastTradePrice"/>
    <input> <soap:body 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>
```

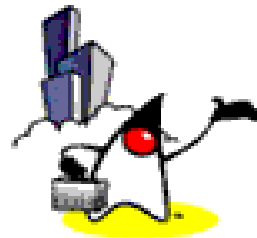
# WSDL View of a Web Service



# Web Service Invocation



# More Detailed Description on WSDL Elements



# Message Element

- Consist of one or more logical **parts**
- Syntax

```
<definitions .... >
```

```
  <message name="nmtoken"> *
```

```
    <part name="nmtoken" element="qname"? type="qname"?/> *
```

```
  </message>
```

```
</definitions>
```

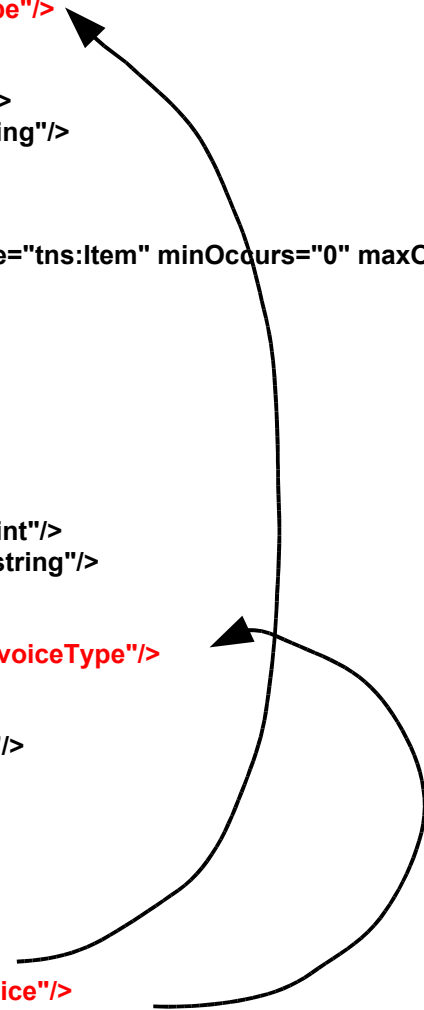
- *element* attribute refers to an XSD element using a QName
- *type* attribute refers to an XSD *simpleType* or *complexType* using a QName

# Example: Message Element

```
<definitions .... >
  <types>
    <schema .... >
      <element name="PO" type="tns:POType"/>
      <complexType name="POType">
        <all>
          <element name="id" type="string"/>
          <element name="name" type="string"/>
          <element name="items">
            <complexType>
              <all>
                <element name="item" type="tns:Item" minOccurs="0" maxOccurs="unbounded"/>
              </all>
            </complexType>
          </element>
        </all>
      </complexType>

      <complexType name="Item">
        <all>
          <element name="quantity" type="int"/>
          <element name="product" type="string"/>
        </all>
      </complexType>
      <element name="Invoice" type="tns:InvoiceType"/>
      <complexType name="InvoiceType">
        <all>
          <element name="id" type="string"/>
        </all>
      </complexType>
    </schema>
  </types>

  <message name="PO">
    <part name="po" element="tns:PO"/>
    <part name="invoice" element="tns:Invoice"/>
  </message>
</definitions>
```



The diagram consists of two curved arrows. The first arrow starts at the message definition `<part name="po" element="tns:PO"/>` and points to the `<element name="PO" type="tns:POType"/>` definition. The second arrow starts at the message definition `<part name="invoice" element="tns:Invoice"/>` and points to the `<complexType name="InvoiceType">` definition.

# Types of Operations

- One-way
  - The endpoint receives a message
- Request/response
  - The endpoint receives a message, and sends a correlated message
- Notification
  - The endpoint sends a message
- Solicit/response
  - The endpoint sends a message, and receives a correlated message

# One-way Operation

```
<operation name="submitPurchase">  
  <input message="purchase"/>  
</operation>
```

# Request/Response Operation

```
<operation name="submitPurchase">  
  <input message="purchase"/>  
  <output message="confirmation"/>  
</operation>
```

```
<operation name="submitPurchase">  
  <input message="purchase"/>  
  <output message="confirmation"/>  
  <fault message="faultMessage"/>  
</operation>
```

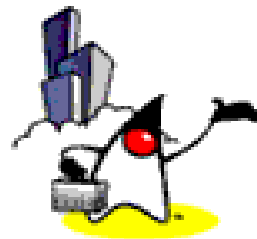
# Notification Operation

```
<operation name="deliveryStatus">  
  <output message="trackingInformation"/>  
</operation>
```

# Solicit/Response Operation

```
<operation name="clientQuery">  
  <output message="bandwidthRequest"/>  
  <input message="bandwidthInfo"/>  
  <fault message="faultMessage"/>  
</operation>
```

# More Detailed Description on WSDL Elements on Binding element and Binding extensions



# Binding Element

- Defines **protocol details** and **message format** for *operations* and *messages* defined by a particular *portType*
- Specify one protocol out of
  - SOAP (SOAP over HTTP, SOAP over SMTP)
  - HTTP GET/POST
- Provides **extensibility** mechanism
  - Can includes **binding extensibility elements**
  - Binding extensibility elements are used to specify the concrete grammar

# Binding Element Syntax

```
<wsdl:definitions .... >
```

```
...
```

```
<wsdl:binding name="nmtoken" type="qname"> *  
  <!-- extensibility element per binding --> *
```

```
<wsdl:operation name="nmtoken"> *  
  <!-- extensibility element per operation --> *
```

```
<wsdl:input name="nmtoken"? > ?  
  <!-- extensibility element per input -->
```

```
</wsdl:input>
```

```
<wsdl:output name="nmtoken"? > ?  
  <!-- extensibility element per output --> *
```

```
</wsdl:output>
```

```
<wsdl:fault name="nmtoken"> *  
  <!-- extensibility element per fault --> *
```

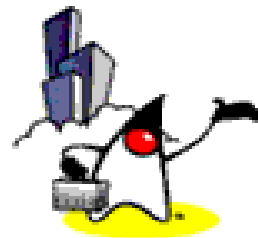
```
</wsdl:fault>
```

```
</wsdl:operation>
```

```
</wsdl:binding>
```

```
</wsdl:definitions>
```

# SOAP Binding



# SOAP Binding Extension

- WSDL includes binding for SOAP 1.1 endpoints and supports:
  - Indication of binding to SOAP as a protocol
  - Address for SOAP endpoint
  - The URI for SOAPAction HTTP header (applies only for HTTP binding of SOAP)
  - List of definitions for Headers for SOAP envelope
- “soap” namespace
  - `xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"`

# SOAP Binding Extensions Syntax

```
<definitions .... >
  <binding .... >
    <soap:binding style="rpc|document" transport="uri">
      <operation .... >
        <soap:operation soapAction="uri"? style="rpc|document"?>?
          <input>
            <soap:body parts="nmtokens"? use="literal|encoded" encodingStyle="uri-list"? namespace="uri"?>
            <soap:header message="qname" part="nmtoken" use="literal|encoded"
              encodingStyle="uri-list"? namespace="uri"?>*
            <soap:headerfault message="qname" part="nmtoken" use="literal|encoded"
              encodingStyle="uri-list"? namespace="uri"?/>*
            <soap:header>
          </input>
          <output>
            <soap:body parts="nmtokens"? use="literal|encoded" encodingStyle="uri-list"? namespace="uri"?>
            <soap:header message="qname" part="nmtoken" use="literal|encoded"
              encodingStyle="uri-list"? namespace="uri"?>*
            <soap:headerfault message="qname" part="nmtoken" use="literal|encoded"
              encodingStyle="uri-list"? namespace="uri"?/>*
            <soap:header>
          </output>
          <fault>*
            <soap:fault name="nmtoken" use="literal|encoded" encodingStyle="uri-list"? namespace="uri"?>
          </fault>
        </operation>
      </binding>

    <port .... >
      <soap:address location="uri"/>
    </port>
  </definitions>
```

# soap:binding

```
<definitions .... >  
  <binding .... >  
    <soap:binding transport="uri"? style="rpc|document"?>  
  </binding>  
</definitions>
```

- Must be present when using SOAP binding
- *style* attribute applies to each contained operation (default: *document*) unless it is overridden by operation specific *style* attribute
- *transport* attribute indicates which transport to use
  - <http://schemas.xmlsoap.org/soap/http> (for HTTP)
  - <http://schemas.xmlsoap.org/soap/smtp> (for SMTP)

# soap:operation

```
<definitions .... >  
  <binding .... >  
    <operation .... >  
      <soap:operation soapAction="uri"? style="rpc|document"?>  
    </operation>  
  </binding>  
</definitions>
```

- *style* attribute indicates whether the operation is **RPC-oriented** (messages containing parameters and return values) or **document-oriented** (message containing document(s))
  - Affects the way in which the Body of the SOAP message is constructed on the wire
- *soapAction* attribute specifies the value of the *SOAPAction* header for this operation

# soap:body

```
<definitions .... >
  <binding .... >
    <operation .... >
      <input>
        <soap:body parts="nmtokens"? use="literal|encoded"?
          encodingStyle="uri-list"? namespace="uri"?>
      </input>
      <output>
        <soap:body parts="nmtokens"? use="literal|encoded"?
          encodingStyle="uri-list"? namespace="uri"?>
      </output>
    </operation>
  </binding>
</definitions>
```

# soap:body

- Specifies how the message parts appear inside the SOAP Body element
  - Provides information on how to assemble the different message parts inside the Body element
- Used in both RPC-oriented and document-oriented messages
  - Which one to use is determined via *style* attribute of `soap:binding` or `soap:operation` elements

# soap:body for RPC style

- WSDL document
  - The **operation** name of WSDL document is used to name the wrapper element (immediate child element under <soap:Body> element)
    - Name of the wrapper element is a method
  - Each part is a parameter or a return value and appears **inside a wrapper element** within the <soap:Body>
- SOAP message:
  - Contents of the Body are formatted as a struct
  - Parts are arranged in the same order as the parameters of the call

# MyHelloServiceRpcLiteral.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:Foo"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="MyHelloService"
  targetNamespace="urn:Foo">
  <types/>
  <message name="HelloIF_sayHello">
    <part name="String_1" type="xsd:string"/>
    <part name="Integer_2" type="xsd:int"/></message>
  <message name="HelloIF_sayHelloResponse">
    <part name="result" type="xsd:string"/></message>
  <portType name="HelloIF">
    <operation name="sayHello" parameterOrder="String_1 Integer_2">
      <input message="tns:HelloIF_sayHello"/>
      <output message="tns:HelloIF_sayHelloResponse"/></operation></portType>
  <binding name="HelloIFBinding" type="tns:HelloIF">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="sayHello">
      <input>
        <soap:body use="literal" namespace="urn:Foo"/></input>
      <output>
        <soap:body use="literal" namespace="urn:Foo"/></output>
      <soap:operation soapAction=""/></operation>
    </binding>
  <service name="MyHelloService">
    <port name="HelloIFPort" binding="tns:HelloIFBinding">
      <soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        location="http://localhost:8080/hello-jaxrpc/hello"/></port></service></definitions>
```

operation name

## RPC, Literal

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:n="urn:Foo"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <n:sayHello>
      <String_1>MyRpcLiteralMessage</String_1>
      <Integer_2>79</Integer_2>
    </n:sayHello>
  </soap:Body>
</soap:Envelope>
```

<part name="String\_1" type="xsd:string"/>  
<part name="Integer\_2" type="xsd:int"/>

<operation name="sayHello" parameterOrder="String\_1 Integer\_2">

# soap:body for Document style

- WSDL document:
  - Each `<message>` has single `<part>` element
  - The `element` attribute of `<part>` refers to schema definition of XML document fragment, which is defined inside `<types>`
- SOAP message:
  - SOAP Body element contains an XML document fragment (document)
    - Ex) Purchase order XML document fragment
  - There are no wrappers

# MyHelloServiceDocLiteral.wsdl(1)

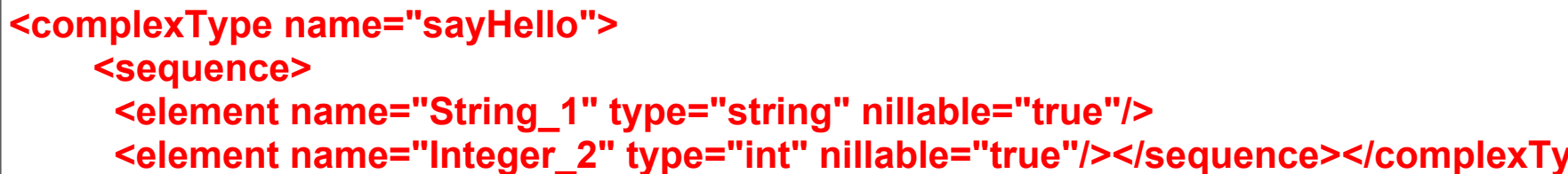
```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:Foo"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="MyHelloService"
  targetNamespace="urn:Foo">
<types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:soap11-
    enc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:Foo">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <complexType name="sayHello">
      <sequence>
        <element name="String_1" type="string" nillable="true"/>
        <element name="Integer_2" type="int" nillable="true"/></sequence></complexType>
    <complexType name="sayHelloResponse">
      <sequence>
        <element name="result" type="string" nillable="true"/></sequence></complexType>
    <element name="sayHello" type="tns:sayHello"/>
    <element name="sayHelloResponse" type="tns:sayHelloResponse"/></schema></types>
<message name="HelloIF_sayHello">
  <part name="parameters" element="tns:sayHello"/></message>
<message name="HelloIF_sayHelloResponse">
  <part name="result" element="tns:sayHelloResponse"/></message>
```

XML schema definition  
of XML document frag.

# SOAP Request Message:

## Doc, Literal

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:tns="urn:Foo"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <tns:sayHello>
      <String_1>MyDocLiteralMessage</String_1>
      <Integer_2>78</Integer_2>
    </tns:sayHello>
  </soap:Body>
</soap:Envelope>
```



The diagram shows a callout box containing the XSD complex type definition for 'sayHello'. An arrow points from the top-left corner of this box to the corresponding XML element in the SOAP message above.

```
<complexType name="sayHello">
  <sequence>
    <element name="String_1" type="string" nillable="true"/>
    <element name="Integer_2" type="int" nillable="true"/>
  </sequence></complexType>
```

# use attribute of soap:body

- *use="literal|encoded"*
- *literal*
  - parts define the concrete schema of the message
  - XML document fragment can be validated against its XML schema
- *encoded*
  - Indicates whether the message parts are encoded using some encoding rules

# use attribute of soap:body

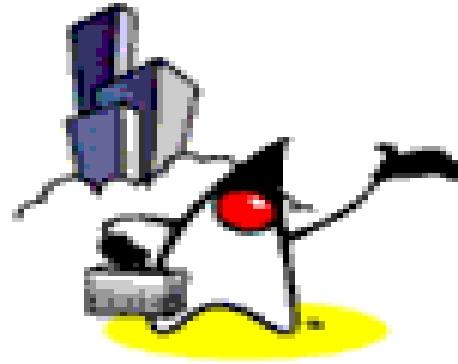
- use="literal"
  - each part references a **concrete schema definition** using either the *element* or *type* attribute (WS-I profile says use *element*)
  - *element* attribute
    - Document style: the element referenced by the part will appear directly under the Body element
    - RPC style: the element referenced by the part will appear under an accessor element named after the message part
  - *type* attribute
    - the type referenced by the part becomes the schema type of the enclosing element

# *use* attribute of *soap:body*

- use="encoded"
  - each message part references an abstract type using the *type* attribute
  - abstract types are used to produce a concrete message by applying an encoding specified by the *encodingStyle* attribute
  - part names, types and value of the namespace attribute are all inputs to the encoding

# Possible *Style/Use* Combinations

- style="rpc" and use="encoded"
- style="rpc" and use="literal"
- style="document" and use="encoded"
- style="document" and use="literal"



SOAP Binding  
**style="rpc"**  
**use="literal"**

# MyHelloServiceRpcLiteral.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:Foo"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="MyHelloService"
  targetNamespace="urn:Foo">
  <types/>
  <message name="HelloIF_sayHello">
    <part name="String_1" type="xsd:string"/>
    <part name="Integer_2" type="xsd:int"/></message>
  <message name="HelloIF_sayHelloResponse">
    <part name="result" type="xsd:string"/></message>
  <portType name="HelloIF">
    <operation name="sayHello" parameterOrder="String_1 Integer_2">
      <input message="tns:HelloIF_sayHello"/>
      <output message="tns:HelloIF_sayHelloResponse"/></operation></portType>
  <binding name="HelloIFBinding" type="tns:HelloIF">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="sayHello">
      <input>
        <soap:body use="literal" namespace="urn:Foo"/></input>
      <output>
        <soap:body use="literal" namespace="urn:Foo"/></output>
      <soap:operation soapAction=""/></operation>
    </binding>
  <service name="MyHelloService">
    <port name="HelloIFPort" binding="tns:HelloIFBinding">
      <soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
        location="http://localhost:8080/hello-jaxrpc/hello"/></port></service></definitions>
```

operation name

# SOAP Request Message: RPC, Literal

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:n="urn:Foo"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body>
    <n:sayHello>
      <String_1>MyRpcLiteralMessage</String_1>
      <Integer_2>79</Integer_2>
    </n:sayHello>
  </soap:Body>
</soap:Envelope>
```

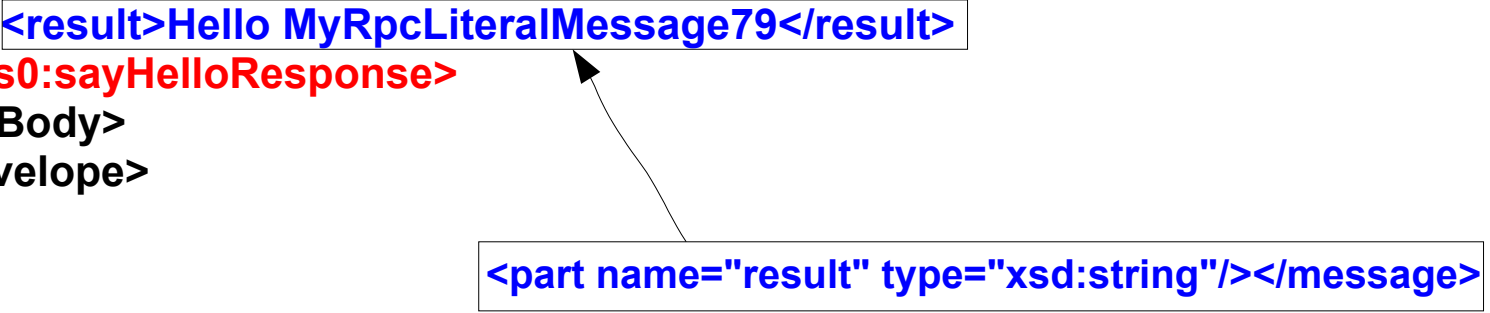
**<operation name="sayHello" parameterOrder="String\_1 Integer\_2">**

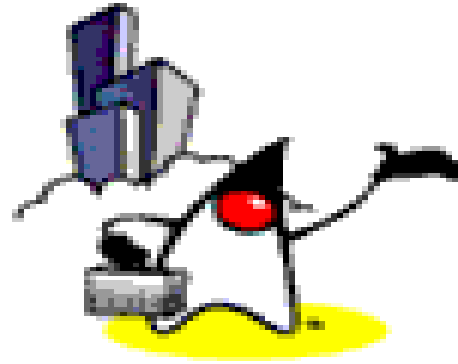
**<part name="String\_1" type="xsd:string"/>**  
**<part name="Integer\_2" type="xsd:int"/>**

# SOAP Response Message: Rpc, Literal

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns0="urn:Foo">
  <env:Body>
    <ns0:sayHelloResponse>
      <result>Hello MyRpcLiteralMessage79</result>
    </ns0:sayHelloResponse>
  </env:Body>
</env:Envelope>
```

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

A diagram consisting of two rectangular boxes with black borders. The top box contains the XML element `<result>Hello MyRpcLiteralMessage79</result>` in blue text. The bottom box contains the SOAP message part `<part name="result" type="xsd:string"/></message>` in blue text. A black arrow points from the top box down to the bottom box, indicating that the content of the top box is the value of the 'result' part in the message.



SOAP Binding  
**style="document"**  
**use="literal"**

# document/literal

- SOAP Body element contains an XML document

# MyHelloServiceDocLiteral.wsdl(1)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:Foo"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="MyHelloService"
  targetNamespace="urn:Foo">
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:soap11-
      enc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" targetNamespace="urn:Foo">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
      <complexType name="sayHello">
        <sequence>
          <element name="String_1" type="string" nillable="true"/>
          <element name="Integer_2" type="int" nillable="true"/></sequence></complexType>
      <complexType name="sayHelloResponse">
        <sequence>
          <element name="result" type="string" nillable="true"/></sequence></complexType>
      <element name="sayHello" type="tns:sayHello"/>
      <element name="sayHelloResponse" type="tns:sayHelloResponse"/></schema></types>
    <message name="HelloIF_sayHello">
      <part name="parameters" element="tns:sayHello"/></message>
    <message name="HelloIF_sayHelloResponse">
      <part name="result" element="tns:sayHelloResponse"/></message>
```

XML schema definition of XML document frag.


# MyHelloServiceDocLiteral.wsdl(2)

```
<portType name="HelloIF">
  <operation name="sayHello">
    <input message="tns:HelloIF_sayHello"/>
    <output message="tns:HelloIF_sayHelloResponse"/></operation></portType>
<binding name="HelloIFBinding" type="tns:HelloIF">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <operation name="sayHello">
    <input>
      <soap:body use="literal"/></input>
    <output>
      <soap:body use="literal"/></output>
    <soap:operation soapAction=""/></operation>
  </binding>
<service name="MyHelloService">
  <port name="HelloIFPort" binding="tns:HelloIFBinding">
    <soap:address xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
      location="http://localhost:8080/hello-jaxrpc/hello"/></port></service></definitions>
```

# SOAP Request Message:

## Doc, Literal

```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:tns="urn:Foo"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <tns:sayHello>
      <String_1>MyDocLiteralMessage</String_1>
      <Integer_2>78</Integer_2>
    </tns:sayHello>
  </soap:Body>
</soap:Envelope>
```



The diagram illustrates the relationship between the XML payload and its schema definition. A callout box highlights the `<tns:sayHello>` element within the SOAP body. An arrow points from this callout box to a larger callout box at the bottom of the slide, which contains the XSD `<complexType>` definition for `sayHello`. This definition specifies a sequence of two elements: `String_1` (type `string`) and `Integer_2` (type `int`), both with `nillable="true"`.

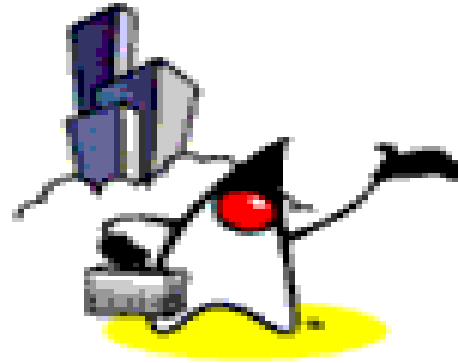
```
<complexType name="sayHello">
  <sequence>
    <element name="String_1" type="string" nillable="true"/>
    <element name="Integer_2" type="int" nillable="true"/>
  </sequence>
</complexType>
```

# SOAP Response Message:

## Doc, Literal

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:ns0="urn:Foo">
  <env:Body>
    <ns0:sayHelloResponse>
      <result>Hello MyDocLiteralMessage78</result>
    </ns0:sayHelloResponse>
  </env:Body>
</env:Envelope>
```

```
<complexType name="sayHelloResponse">
  <sequence>
    <element name="result" type="string" nillable="true"/></sequence></complexType>
```



SOAP Binding  
**style="rpc"**  
**use="encoded"**

# MyHelloServiceRpcEncoded.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="urn:Foo"
  xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" name="MyHelloService"
  targetNamespace="urn:Foo">
  <types/>
  <message name="HelloIF_sayHello">
    <part name="String_1" type="xsd:string"/>
    <part name="Integer_2" type="ns2:int"/></message>
  <message name="HelloIF_sayHelloResponse">
    <part name="result" type="xsd:string"/></message>
  <portType name="HelloIF">
    <operation name="sayHello" parameterOrder="String_1 Integer_2">
      <input message="tns:HelloIF_sayHello"/>
      <output message="tns:HelloIF_sayHelloResponse"/></operation></portType>
  <binding name="HelloIFBinding" type="tns:HelloIF">
    <operation name="sayHello">
      <input>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
          namespace="urn:Foo"/></input>
      <output>
        <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" use="encoded"
          namespace="urn:Foo"/></output>
      <soap:operation soapAction=""/></operation>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/></binding>
  <service name="MyHelloService">
    <port name="HelloIFPort" binding="tns:HelloIFBinding">
      <soap:address xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" location="http://localhost:8080/hello-
        jaxrpc/hello"/></port></service></definitions>
```

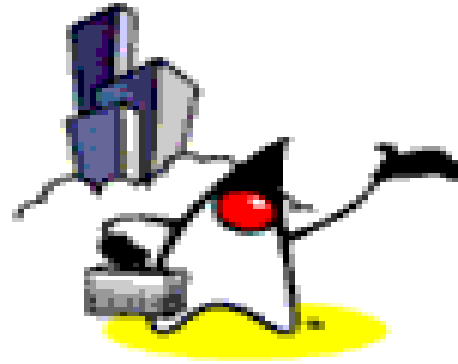
# SOAP Request Message: rpc, encoded

```
<<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  xmlns:n="urn:Foo"
  xmlns:ns2="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <n:sayHello>
      <String_1 xsi:type="xs:string">MyRpcEncodingMessage</String_1>
      <Integer_2 xsi:type="ns2:int">77</Integer_2>
    </n:sayHello>
  </soap:Body>
</soap:Envelope>
```

# SOAP Response Message:

## rpc, encoded

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns0="urn:Foo"
  env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <env:Body>
    <ns0:sayHelloResponse>
      <result xsi:type="xsd:string">Hello MyRpcEncodingMessage77</result>
    </ns0:sayHelloResponse>
  </env:Body>
</env:Envelope>
```



SOAP Binding  
**style="document"**  
**use="encoded"**

# SOAP Message Example2: “document” & “encoded”

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns="http://tempuri.org/" xmlns:types="http://tempuri.org/encodedTypes"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">

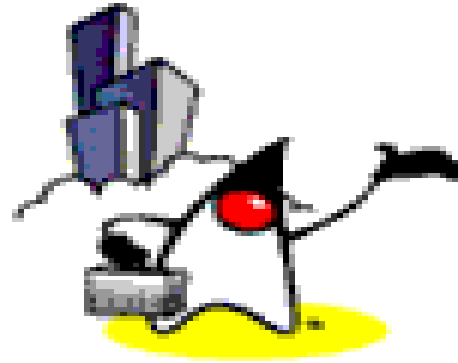
  <soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">

    <types:HelloEncodedWorldResponse
      xsi:type="types:HelloEncodedWorldResponse">
      <HelloEncodedWorldResult href="#id1" />
    </types:HelloEncodedWorldResponse>

    <types:MethodDuration id="id1" xsi:type="types:MethodDuration">
      <start xsi:type="xsd:dateTime">dateTime</start>
      <end xsi:type="xsd:dateTime">dateTime</end>
      <rVal xsi:type="xsd:string">string</rVal>
    </types:MethodDuration>

  </soap:Body>

</soap:Envelope>
```



# SOAP Binding

## When to use What?

# RPC vs. Document-style

## RPC

- Procedure call
- Method signature
- Marshaling
- Tightly-coupled
- Point to point
- Synchronous
- Typically within Intranet

## Document-style

- Business documents
- Schema
- Parsing & Validating
- Loosely coupled
- End to end
- Asynchronous
- Typically over internet

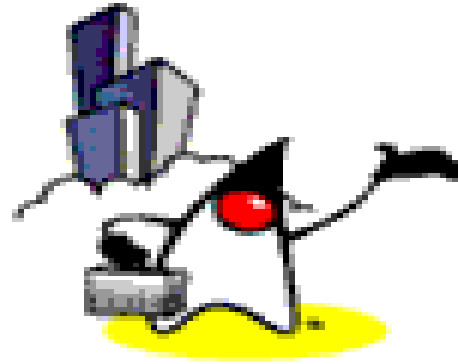
# When to use Which model?

## RPC

- Within Enterprise
- Simple, point-to-point
- Short running business process
- Reliable and high bandwidth
- Trusted environment

## Document-style

- Between enterprise and enterprise
- Complex, end to end with intermediaries
- Long running business process
- Unpredictable bandwidth
- Blind trust



# SOAP Binding

## More Examples

# Example3: SOAP binding of One-way operation over SMTP using a SOAP Header

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

  <message name="SubscribeToQuotes">
    <part name="body" element="xsd1:SubscribeToQuotes"/>
    <part name="subscribeheader" element="xsd1:SubscriptionHeader"/>
  </message>

  <portType name="StockQuotePortType">
    <operation name="SubscribeToQuotes">
      <input message="tns:SubscribeToQuotes"/>
    </operation>
  </portType>

  <binding name="StockQuoteSoap" type="tns:StockQuotePortType">
    <soap:binding style="document" transport="http://example.com/smtp"/>
    <operation name="SubscribeToQuotes">
      <input message="tns:SubscribeToQuotes">
        <soap:body parts="body" use="literal"/>
        <soap:header message="tns:SubscribeToQuotes" part="subscribeheader" use="literal"/>
      </input>
    </operation>
  </binding>
```

# Continued

```
<service name="StockQuoteService">
  <port name="StockQuotePort" binding="tns:StockQuoteSoap">
    <soap:address location="mailto:subscribe@example.com"/>
  </port>
</service>

<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="SubscribeToQuotes">
      <complexType>
        <all>
          <element name="tickerSymbol" type="string"/>
        </all>
      </complexType>
    </element>
    <element name="SubscriptionHeader" type="uriReference"/>
  </schema>
</types>
</definitions>
```

# Example 4: SOAP binding of request-response RPC operation over HTTP

```
<?xml version="1.0"?>
<definitions name="StockQuote"
  targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:xsd1="http://example.com/stockquote.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

```
<message name="GetTradePriceInput">
  <part name="tickerSymbol" element="xsd:string"/>
  <part name="time" element="xsd:dateTime"/>
</message>
```

```
<message name="GetTradePriceOutput">
  <part name="result" type="xsd:float"/>
</message>
```

```
<portType name="StockQuotePortType">
  <operation name="GetTradePrice">
    <input message="tns:GetTradePriceInput"/>
    <output message="tns:GetTradePriceOutput"/>
  </operation>
</portType>
```

# Continued

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetTradePrice">
    <soap:operation soapAction="http://example.com/GetTradePrice"/>
    <input>
      <soap:body use="encoded" namespace="http://example.com/stockquote"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <soap:body use="encoded" namespace="http://example.com/stockquote"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>>
</binding>

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

# Example 5: SOAP binding of request-response RPC operation over HTTP

```
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote.wsdl"
  xmlns:tns="http://example.com/stockquote.wsdl"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
  xmlns:xsd1="http://example.com/stockquote/schema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

<types>
  <schema targetNamespace="http://example.com/stockquote/schema"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
    <complexType name="TimePeriod">
      <all>
        <element name="startTime" type="xsd:timeInstant"/>
        <element name="endTime" type="xsd:timeInstant"/>
      </all>
    </complexType>
    <complexType name="ArrayOfFloat">
      <complexContent>
        <restriction base="soapenc:Array">
          <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:float[]"/>
        </restriction>
      </complexContent>
    </complexType>
  </schema>
</types>
```

# Continued

```
<message name="GetTradePricesInput">  
  <part name="tickerSymbol" element="xsd:string"/>  
  <part name="timePeriod" element="xsd1:TimePeriod"/>  
</message>
```

```
<message name="GetTradePricesOutput">  
  <part name="result" type="xsd1:ArrayOfFloat"/>  
  <part name="frequency" type="xsd:float"/>  
</message>
```

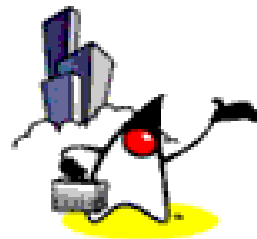
```
<portType name="StockQuotePortType">  
  <operation name="GetLastTradePrice" parameterOrder="tickerSymbol timePeriod  
frequency">  
    <input message="tns:GetTradePricesInput"/>  
    <output message="tns:GetTradePricesOutput"/>  
  </operation>  
</portType>
```

# Continued

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetTradePrices">
    <soap:operation soapAction="http://example.com/GetTradePrices"/>
    <input>
      <soap:body use="encoded" namespace="http://example.com/stockquote"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </input>
    <output>
      <soap:body use="encoded" namespace="http://example.com/stockquote"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
    </output>
  </operation>>
</binding>

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

# HTTP GET/POST Binding



# HTTP Get & Post Binding

- Allows applications to interact Web sites
- Can specify
  - An indication that a binding uses HTTP GET or POST
  - An address for the port
  - A relative address for each operation (relative to the base address defined by the port)
- It is rarely used

# Example 4A: GET and FORM POST returning GIF or JPG

```
<definitions .... >
  <message name="m1">
    <part name="part1" type="xsd:string"/>
    <part name="part2" type="xsd:int"/>
    <part name="part3" type="xsd:string"/>
  </message>

  <message name="m2">
    <part name="image" type="xsd:binary"/>
  </message>

  <portType name="pt1">
    <operation name="o1">
      <input message="tns:m1"/>
      <output message="tns:m2"/>
    </operation>
  </portType>
```

# Example 4B: GET and FORM POST returning GIF or JPG

```
<service name="service1">
  <port name="port1" binding="tns:b1">
    <http:address location="http://example.com/" />
  </port>
  <port name="port2" binding="tns:b2">
    <http:address location="http://example.com/" />
  </port>
  <port name="port3" binding="tns:b3">
    <http:address location="http://example.com/" />
  </port>
</service>
```

```
<binding name="b1" type="pt1">
  <http:binding verb="GET" />
  <operation name="o1">
    <http:operation location="o1/A(part1)B(part2)/(part3)" />
    <input>
      <http:urlReplacement />
    </input>
    <output>
      <mime:content type="image/gif" />
      <mime:content type="image/jpeg" />
    </output>
  </operation>
</binding>
```

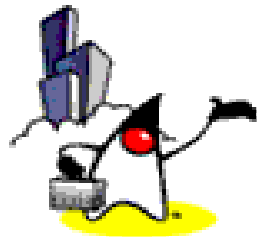
# Example 4C: GET and FORM POST returning GIF or JPG

```
<binding name="b2" type="pt1">  
  <http:binding verb="GET"/>  
  <operation name="o1">  
    <http:operation location="o1"/>  
    <input>  
      <http:urlEncoded/>  
    </input>  
    <output>  
      <mime:content type="image/gif"/>  
      <mime:content type="image/jpeg"/>  
    </output>  
  </operation>  
</binding>
```

```
<binding name="b3" type="pt1">  
  <http:binding verb="POST"/>  
  <operation name="o1">  
    <http:operation location="o1"/>  
    <input>  
      <mime:content type="application/x-www-form-urlencoded"/>  
    </input>  
    <output>  
      <mime:content type="image/gif"/>  
      <mime:content type="image/jpeg"/>  
    </output>  
  </operation>  
</binding>  
</definitions>
```



# MIME Binding



# MIME Binding

- Binding abstract types to concrete messages
- The following MIME types are defined:
  - multipart/related
  - text/xml
  - application/x-www-form-urlencoded
  - Others (by specifying the MIME type string)

# Example 6A: Using MIME multipart/related with SOAP

```
<definitions .... >
```

```
<types>
```

```
<schema .... >
```

```
<element name="GetCompanyInfo">
```

```
<complexType>
```

```
<all>
```

```
<element name="tickerSymbol " type="string"/>
```

```
</all>
```

```
</complexType>
```

```
</element>
```

```
<element name="GetCompanyInfoResult">
```

```
<complexType>
```

```
<all>
```

```
<element name="result" type="float"/>
```

```
</all>
```

```
</complexType>
```

```
</element>
```

```
<complexType name="ArrayOfBinary">
```

```
<complexContent>
```

```
<restriction base="soapenc:Array">
```

```
<attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:binary[]"/>
```

```
</restriction>
```

```
<complexContent>
```

```
</complexType>
```

```
</schema>
```

```
</types>
```

# Example 6B: Using multipart/related with SOAP

```
<message name="m1">  
  <part name="body" element="tns:GetCompanyInfo"/>  
</message>
```

```
<message name="m2">  
  <part name="body" element="tns:GetCompanyInfoResult"/>  
  <part name="docs" type="xsd:string"/>  
  <part name="logo" type="tns:ArrayOfBinary"/>  
</message>
```

```
<portType name="pt1">  
  <operation name="GetCompanyInfo">  
    <input message="m1"/>  
    <output message="m2"/>  
  </operation>  
</portType>
```

# Example 6C: Using multipart/related with SOAP

```
<binding name="b1" type="tns:pt1">
  <operation name="GetCompanyInfo">
    <soap:operation soapAction="http://example.com/GetCompanyInfo"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <mime:multipartRelated>
        <mime:part>
          <soap:body parts="body" use="literal"/>
        </mime:part>
        <mime:part>
          <mime:content part="docs" type="text/html"/>
        </mime:part>
        <mime:part>
          <mime:content part="logo" type="image/gif"/>
          <mime:content part="logo" type="image/jpeg"/>
        </mime:part>
      </mime:multipartRelated>
    </output>
  </operation>
</binding>

<service name="CompanyInfoService">
  <port name="CompanyInfoPort" binding="tns:b1">
    <soap:address location="http://example.com/companyinfo"/>
  </port>
</service>
</definitions>
```



# Importing & Authoring Style

# Authoring Style Recommendation

- Reusability and maintainability
- Maintain WSDL document in 3 separate parts
  - Data type definitions
  - Abstract definitions
  - Specific service bindings
- Use “import” element to import necessary part of WSDL document

# Example7A:

<http://example.com/stockquote/stockquote.xsd>

```
<?xml version="1.0"?>
<schema targetNamespace="http://example.com/stockquote/schemas"
  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>
```

# Example 7B:

<http://example.com/stockquote/stockquote.wsdl>

```
<?xml version="1.0"?>
<definitions name="StockQuote"

targetNamespace="http://example.com/stockquote/definitions"
  xmlns:tns="http://example.com/stockquote/definitions"
  xmlns:xsd1="http://example.com/stockquote/schemas"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <import namespace="http://example.com/stockquote/schemas"
    location="http://example.com/stockquote/stockquote.xsd"/>

  <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>
  </portType>
</definitions>
```

# Example7C:

<http://example.com/stockquote/stockquoteservice.wsdl>

```
<?xml version="1.0"?>
<definitions name="StockQuote"
targetNamespace="http://example.com/stockquote/service"
  xmlns:tns="http://example.com/stockquote/service"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:defs="http://example.com/stockquote/definitions"
  xmlns="http://schemas.xmlsoap.org/wsdl/">

  <import namespace="http://example.com/stockquote/definitions"
    location="http://example.com/stockquote/stockquote.wsdl"/>

  <binding name="StockQuoteSoapBinding" type="defs: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:StockQuoteBinding">
      <soap:address location="http://example.com/stockquote"/>
    </port>
  </service>
</definitions>
```



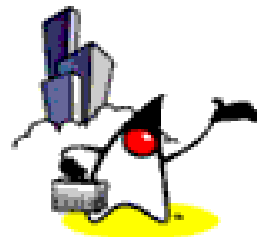
# Limitations of WSDL

# Limitations of WSDL

- Does not support business collaboration
  - ebXML's BPSS does support it
- Does not support partner profile concept
  - ebXML partner profile supports it
- It is **static**
  - WSCI (Web Services Choreography Interface) complements WSDL
- No “native” asynchronous support
- Defines only syntactical aspects
  - No semantic description
  - ebXML Core components and UBL define them

# WSCI

(Web Services  
Choreography Interface)



# WSCI and WSDL

- WSCI “adds the behavior” to the WSDL artifacts
  - It does not extend nor change WSDL
  - It does not compete with WSDL but requires it.
- WSCI makes WSDL usable !!! (finally!!!)
  - Only stateless and trivial w/s can be fully described by WSDL (web-toys...)
- We could see WSDL as a still-camera and WSCI as a video-camera
  - Both of them “describe” the reality...
  - Videos are organized series of snapshots...

# Simplified Ticket Reservation Process (WSDL view)

## Traveler Web Service

```
<portType name = "TravelerToTA">

  <operation name = "OrderTrip">
    <output message = "tripOrderRequest"/>
    <input message =
      "tripOrderAcknowledgment"/>
  </operation>

  <operation name = "BookTickets">
    <output message = "bookingRequest"/>
    <input message = "bookingConfirmation"/>
  </operation>

  <operation name = "ReceiveStatement">
    <input message = "statement"/>
  </operation>

  ...

</portType>
```

## Travel Agent Web Service

```
<portType name = "TravelerToTA">

  <operation name = "OrderTrip">
    <input message =
      "tripOrderRequest"/>
    <output message =
      "tripOrderAcknowledgment"/>
  </operation>

  <operation name = "BookTickets">
    <input message = "bookingRequest"/>
    <output message =
      "bookingConfirmation"/>
  </operation>

  <operation name = "SendStatement">
    <output message = "statement"/>
  </operation>

  ...

</portType>
```

# Simplified Ticket Reservation Process (WSCI view)

## Traveler Web Service

```
interface „Traveler“

process „PlanAndBookTrip“
sequence

    action „OrderTrip“
        operation „OrderTrip“
    ...
    action „BookTickets“
        operation „BookTickets“

    action „ReceiveStatement“
        operation „ReceiveStatement“
```

## Travel Agent Web Service

```
interface „TravelAgent“

process „PlanAndBookTrip“
sequence

    action „ReceiveTripOrder“
        operation „OrderTrip“
    ...
    action „ReceiveBooking“
        operation „BookTickets“
        call „BookSeats“

    action „SendStatement“
        operation „SendStatement“

process „BookSeats“ // to Airline
    action „BookSeats“
        operation „BookSeats“
```



# WSDL 1.2

# WSDL 1.2

- Described in XML Information set
- Aligned with SOAP 1.2
  - Changes are mostly about SOAP binding
- MEP (Message Exchange Pattern)
- Clarifications
  - Solicit-response and notification model
- Still “work in progress” state



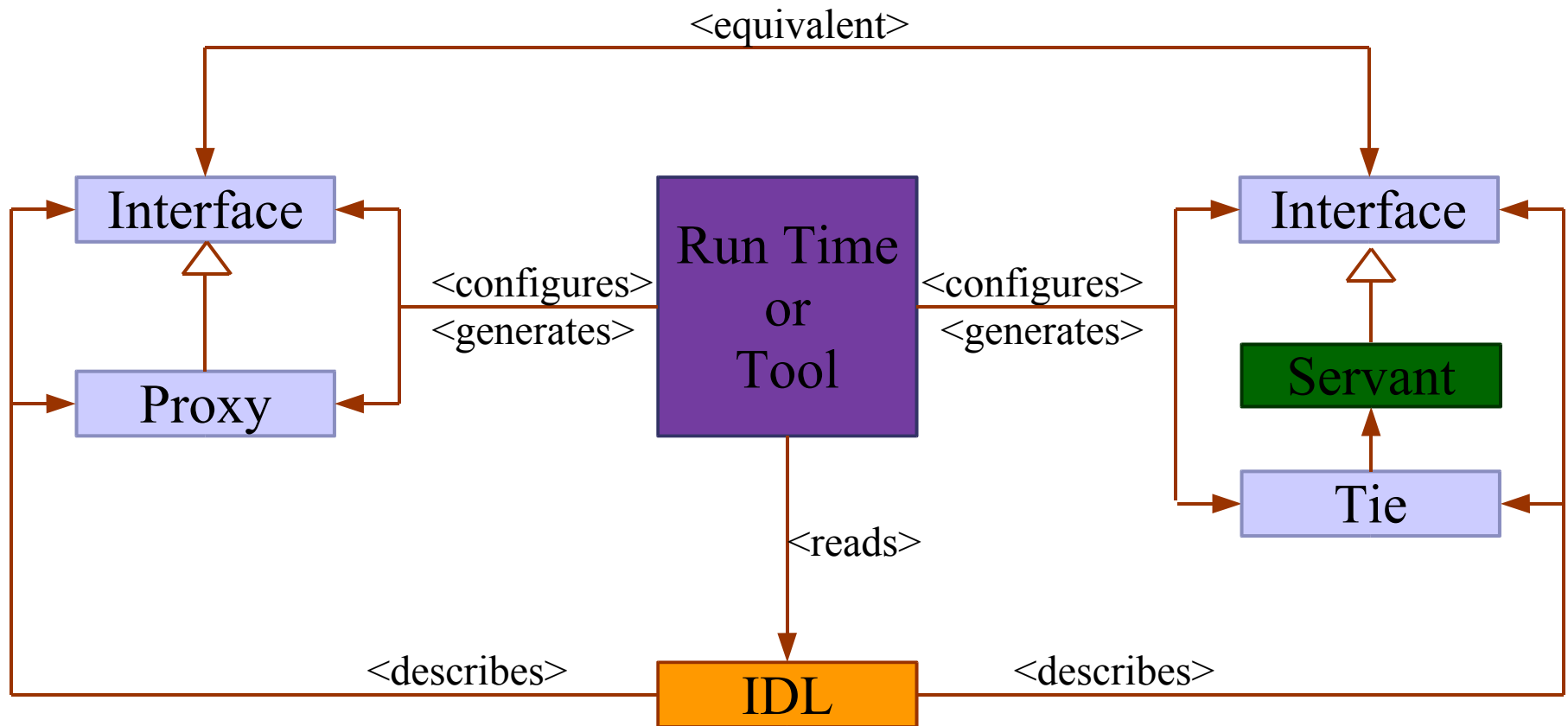
# Application Design & Tools

**(We will talk a lot more on  
this in JAX-RPC Session)**

# IDL

- Interface Description Language
- Describes programming interfaces in a language neutral way
- Used by tools to statically generate or dynamically configure interfaces, proxies, and ties in a specific environment

# The role of IDL



# XML Based RPC

- Uses Standards based on XML
  - SOAP is the “protocol”
  - WSDL is the IDL
- Any text based protocol can be used as a transport (e.g. HTTP, SMTP, FTP etc...)

# Application Design

- Web service is defined in **WSDL**
- Top-down
  - **WSDL is created (or found) first** before its implementation
- Bottom-up
  - WSDL gets generated from **existing J2EE components**
- Middle-ground

# Tools

- **Generate WSDL document** from existing Java classes or EJB components
- **Generate SOAP messages** from WSDL document (via client stub and server skeleton)
- JAX-RPC xrpcc
- Forte for Java with SOAP-RPC module
- Systnet(Idoox), lopsys, CapeClear

# Java Programming APIs for WSDL

- JSR-110
  - Provides API support to:
    - Create WSDL document programmatically
    - Parse WSDL document (from XML)
    - Query WSDL document
- **JSR-101 (JAX-RPC)**
  - Java to/from WSDL mapping



# Resources

# Resources

- W3C WSDL Home
  - [www.w3.org/TR/wsd1](http://www.w3.org/TR/wsd1)
- W3C WSDL Primer
  - [dev.w3.org/cvsweb/~checkout~/2002/ws/desc/wsd12/wsd12-primer.html](http://dev.w3.org/cvsweb/~checkout~/2002/ws/desc/wsd12/wsd12-primer.html)
- WSDL information site
  - [xml.coverpages.org/wsd1.html](http://xml.coverpages.org/wsd1.html)