ULB UNIVERSITÉ LIBRE DE BRUXELLES

WSDL

INFO-H-511 - Web Services

Mohamed Chajii

Irina Kameniar

Ondrej Svoboda

Content

- Introduction to SDL
- History
- Structure and semantics WSDL
- SOAP encodings
- RPC/Literal, RPC/Encoding
- Structure and semantics WADL
- Overview

Introduction to SDL

- Service Description Language
- Without WSDL, calling syntax must be determined from documentation that must be provided, or from examining wire messages
- With WSDL, the generation of proxies for Web services is automated in a truly language- and platform-independent way

WSDL

- Web Service Description Language
  - In version 1.1 D was for Definition
- XML based interface description language
- Describes functionality offered by a web service
- Machine readable
- Primarily for SOAP

History

- WSDL 1.0 – September 2000
  - By merging NASSL from IBM and SDL from Microsoft
- WSDL 1.1 – March 2001
  - Formalization of WSDL 1.0
- WSDL 1.2 - June 2003
  - Slightly skipped to version 2.0
- WSDL 2.0 – June 2007

WSDL 2.0 changes(to 1.1)

- Further semantics to description language
- Remove message constructs
- Operator overloading not supported
- PortTypes -> Interfaces
- Ports -> endpoints

Structure

<description> - root element

<types> … </types>

<interface>

<operation> … </operation>

</interface>

<binding> … </binding>

<service>

<endpoint> … </endpoint>

</service>

</description>

Abstract

Concrete

Define Target Namespace

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns=http://www.w3.org/2006/01/wsdl
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= http://greath.example.com/2004/wsdl/resSvc
    . . . >
   …
</description>
```

- Description - root tag
- xmlns – namespace for WSDL 2.0 itself
- targetNamespace – not actual XML namespace, analogous to XML Schema target namespace
- xmlns:tns – actual namespace for our service, we can actually use tns: prefix

Define Message Types

- Define the messages that are exchanged
- Usually defined by XML Schema
  - Not restricted only to XML Schema
- Defined inside <type> element
  - Directly
  - Can be imported
- Code example
  - http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106/#example-initial-types

Define Interface

- Abstract interface as a set of abstract operations(more than 1 interface)
- Operation
  - Each operation simple client/service interaction
  - Pattern – in-out,in-only, out-only …
- Code example
  - http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106/#example-initial-interface

Define Binding

- Defines how are messages exchanged
- Specifies concrete message format and transmission protocol
  - For each operation and fault in the interface
- Separate symbol space for bindings, interfaces and services
- Code example
  - http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106/#example-initial-binding

Define Services

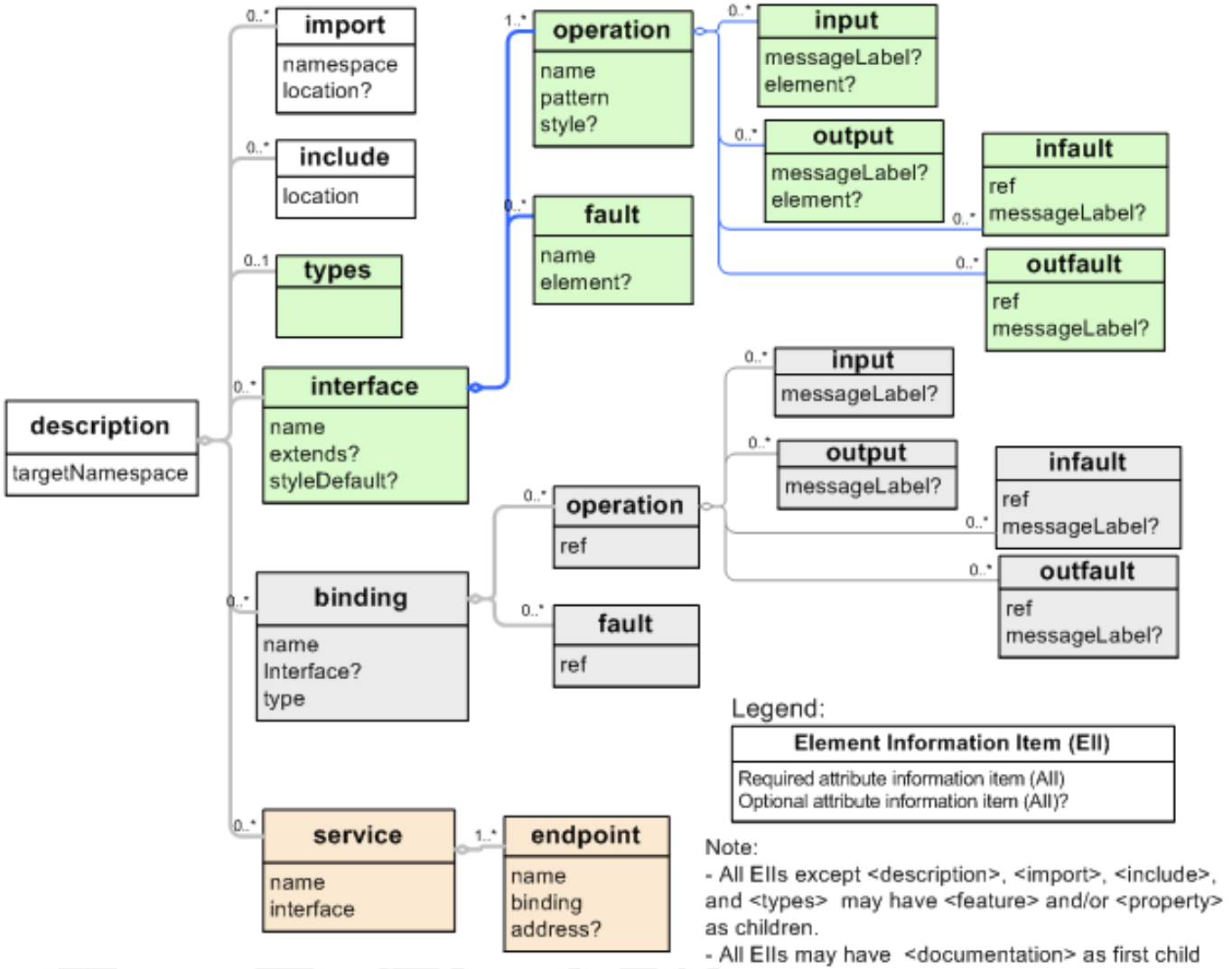
- Defines where the service can be accessed
- Specifies a single interface the service will support – service allowed only 1 interface
  - List of endpoint locations where could be accessed, each must reference binding

- Code example
  - http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106/#example-initial-service

Documentation

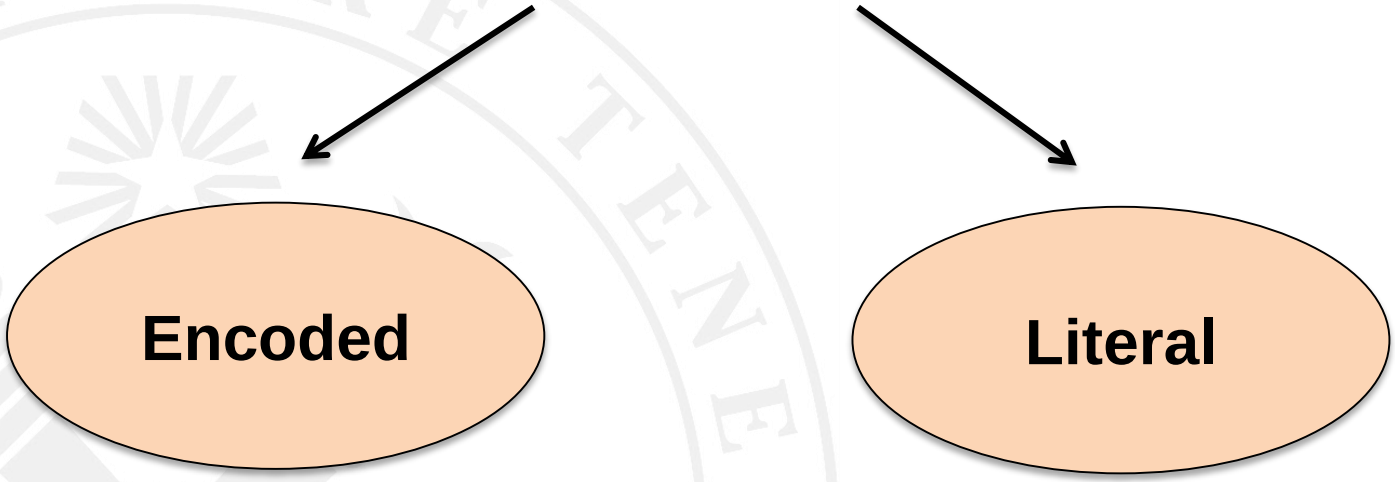
```
<?xml version="1.0" encoding="utf-8" ?>
<description
 . . . >

    <documentation>
        This document describes the GreatH Web service.  Additional
        application-level requirements for use of this service –
        beyond what WSDL 2.0 is able to describe -- are available
        at http://greath.example.com/2004/reservation-documentation.html
    </documentation>
. . .
</description>
```

XML Infoset


description
targetNamespace
import
namespace
location?
include
location
types
interface
name
extends?
styleDefault?
operation
name
pattern
style?
input
messageLabel?
element?
output
messageLabel?
element?
infault
ref
messageLabel?
outfault
ref
messageLabel?
fault
name
element?
binding
name
Interface?
type
operation
ref
input
messageLabel?
output
messageLabel?
infault
ref
messageLabel?
outfault
ref
messageLabel?
fault
ref
service
name
interface
endpoint
name
binding
address?
Legend:
Element Information Item (EII)
Required attribute information item (AII)
Optional attribute information item (AII)?
Note:
- All EIIs except <description>, <import>, <include>, and <types> may have <feature> and/or <property> as children.
- All EIIs may have <documentation> as first child

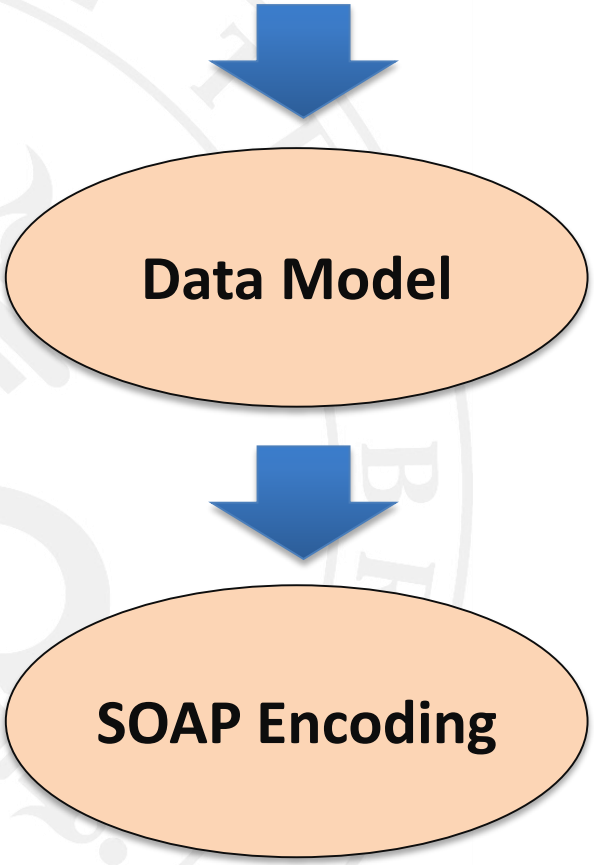
WSDL SOAP binding encoded

2 different type of encoding



Data Model and SOAP Encoding

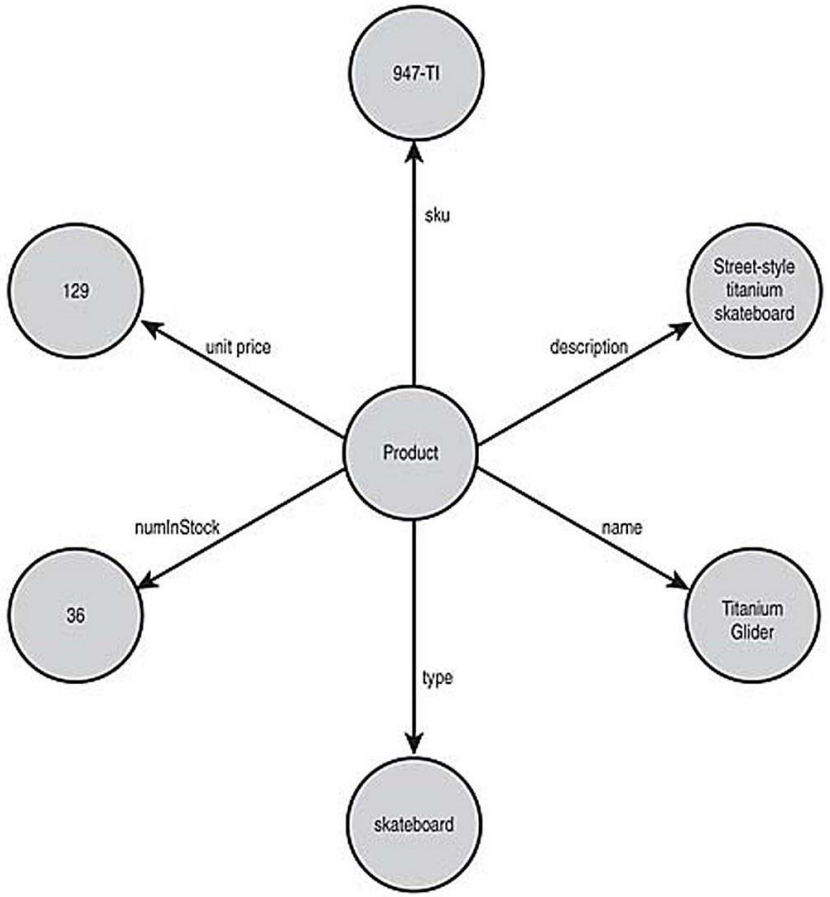
how to map common object-oriented programming constructs to XML?



Data Model

- graphs of *nodes.*
- each *node* may be connected via directional edges to other nodes.
- Node can be:
  1. *Compound* values → have outgoing edges.
  2. *Simple* values → only incoming edges.
- In Java, the object representing this structure might look like this:

```
class Product {
  String description;
  String sku;
  double unitPrice;
  String name;
  String type;
  int numInStock;}
```



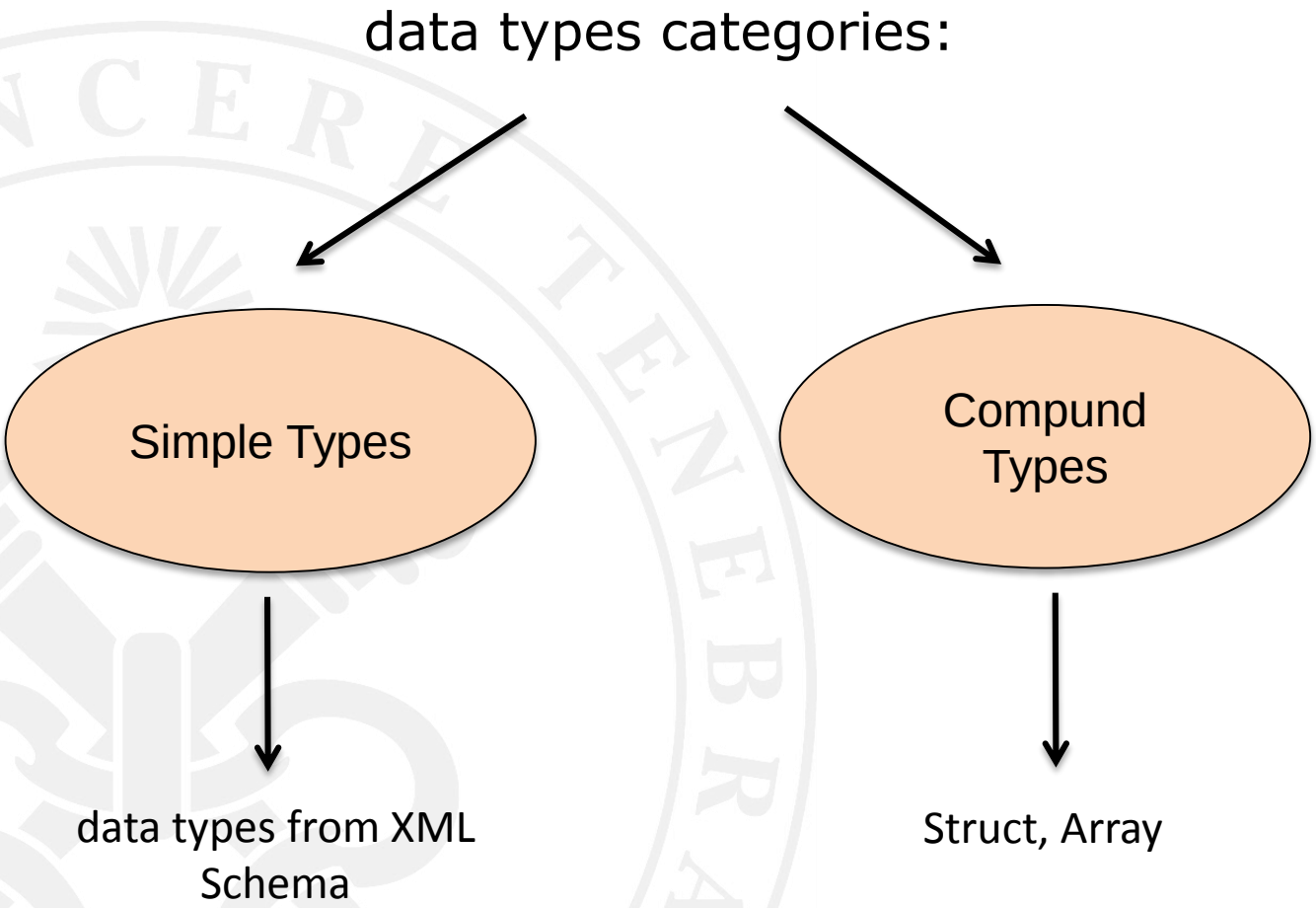
Example data model for a *Product* object.

SOAP Encoding

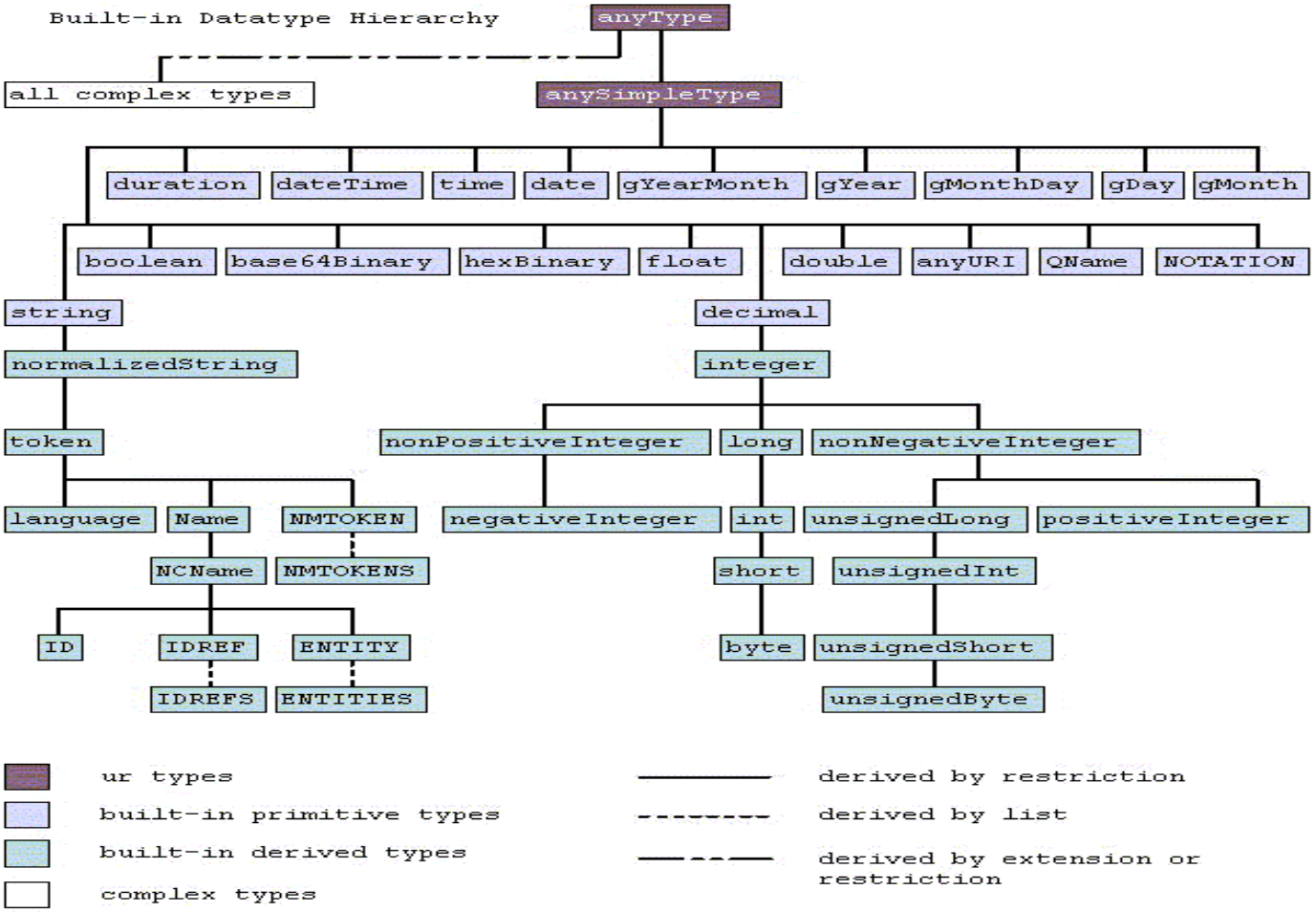
- Section 5 of SOAP Spec.
- Set of serialization rules.
- XML representation of Data Model.
- Encoding is specified by *EncodingStyle* attribute.

```
<product soapenv:encodingStyle="http://www.w3.org/2003/05/soap-
encoding">
 <sku>947-TI</sku>
 <name>Titanium Glider</name>
 <type>skateboard</type>
 <desc>Street-style titanium skateboard.</desc>
 <price>129.00</price>
 <inStock>36</inStock>
</product>
```

SOAP Encoding



Simple Type Encoding



Simple Type Encoding Examples

Java examples

int a = 3

float pi = 3.14

String s = "Hello"

Soap Encoding

<a
xsi:type="xsd:int> 10
"

<pi
xsi:type="xsd:float"> 3.14
</pi>

<s
xsi:type="xsd:string"> Hello
</s>

Enumeration and References encoding

- **Enumeration**

```
<element name="EyeColor" type="tns:EyeColor"/>
<simpleType name="EyeColor" base="xsd:string">
  <enumeration value="Green"/>
  <enumeration value="Blue"/>
  <enumeration value="Brown"/>
</simpleType>
```

Boolean simple-type is not supported.

- **References**

```
<manager>
    <fname>Geoffrey</>
    <lname>Fox</>
</manager>
<employee>
    <fname>Marlon</>
    <lname>Pierce</>
    <manager>
        <fname>Geoffrey</>
        <lname>Fox</>
    </manager>
</employee>
```

AFTER

```
<manager>
<manager id="GCF">
        <fname>Geoffrey</>
        <lname>Fox</>
</manager>
<employee>
        <fname>Marlon</>
        <lname>Pierce</>
        <manager href="#gcf">
</employee>
```

Compound Type Encoding - 1

- *Array*

```
<element name="myFavoriteNumbers"
         type="SOAP-ENC:Array"/>
```

←

Arrays represented as element values with type *SOAP-ENC:Array*

```
<myFavoriteNumbers
   SOAP-ENC:arrayType="xsd:int[2]">
   <number>3</number>
   <number>4</number>
</myFavoriteNumbers>
```

←

contains several members each of which is a value of type *SOAP-ENC:int*.

Compound Type Encoding – 2

- *Struct*

```
<e:Book>
   <author>Henry Ford</author>
         <preface>Prefatory text</preface>
         <intro>This is a book.</intro>
</e:Book>
```

Struct of type *Book*.

```
<element name="Book">
   <complexType>
    <element name="author" type="xsd:string"/>
    <element name="preface" type="xsd:string"/>
   <element name="intro" type="xsd:string"/>
  </complexType>
</e:Book>
```

schema fragment describing the above structure.

SOAP encoding problem

Data Model

XML Schema

graph of untyped structures

tree of typed elements

How can an encoding scheme for SOAP data model be applied to abstract XML Schema definitions ?

No answers from the SOAP specification

SOAP encoding problem - example

Input – WSDL:

```
<wsdl:portType name="Geometry">
  <wsdl:operation name="Distance">
    <wsdl:input message="tns:DistanceInput" />
    <wsdl:output message="tns:DistanceOutput" />
  </wsdl:operation>
 </wsdl:portType>

 <!-- RPC style message definitions -->
 <wsdl:message name="DistanceInput">
  <wsdl:part name="p1" type="tns:Point" />
  <wsdl:part name="p2" type="tns:Point" />
 </wsdl:message>

<xsd:complexType name="Point">
      <xsd:sequence>
        <xsd:element name="x" type="xsd:int" />
        <xsd:element name="y" type="xsd:int" />
      </xsd:sequence>
</xsd:complexType>
```

portType called *Geometry* that contains the *Distance* operation

SOAP encoding problem - example

The client's serialized request message:

```
<soap:Envelope xmlns:soap=
      "http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body soap:encodingStyle=
      "http://schemas.xmlsoap.org/soap/encoding/">
  <ns:Distance xmlns:ns=
      "http://www.gotdotnet.com/team/tewald/samples">
   <p1 href="#id1" />
   <p2 href="#id1" />
  </ns:Distance>
  <ns:Point id="id1"
   xmlns:ns="http://www.gotdotnet.com/team/tewald/samples">
    <x>10</x>
    <y>20</y>
  </ns:Point>
 </soap:Body>
</soap:Envelope>
```

do not match the schema definition for the *Point* type.

Literal

Data is serialized according to a schema.

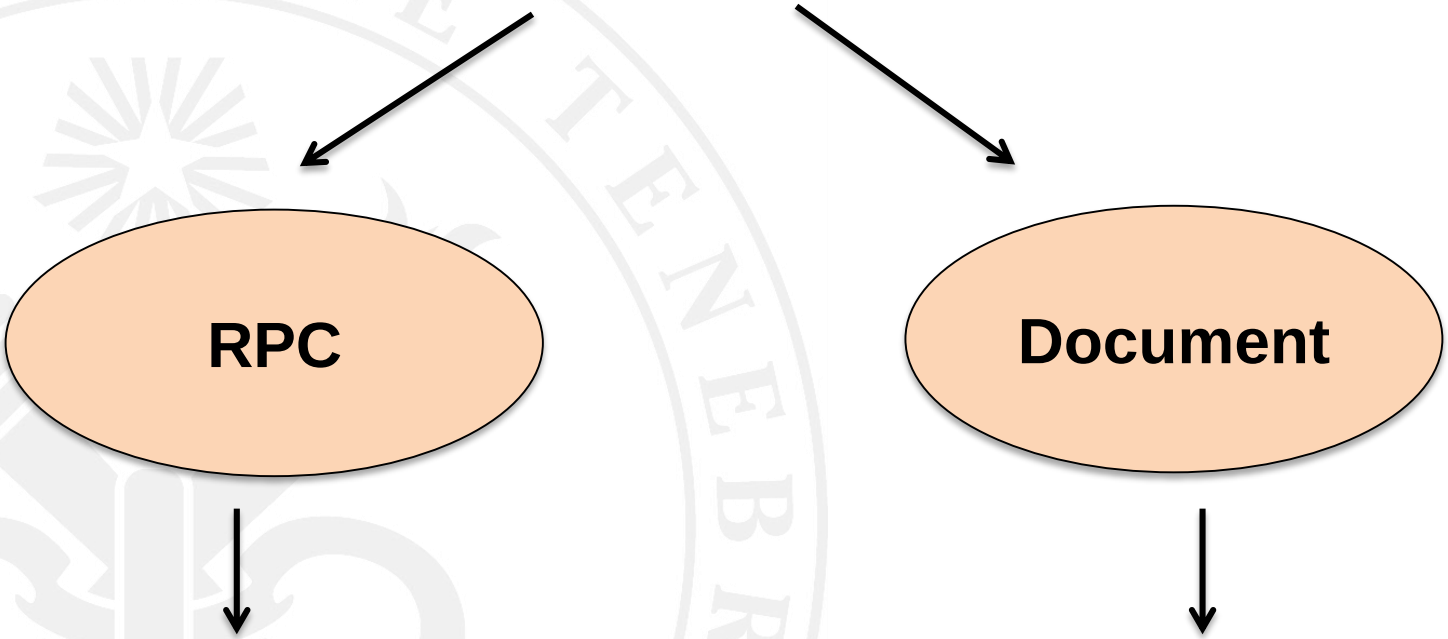
Soap body follows a W3C **XML Schema**.

XML Schema included in the **WSDL** Document.

WS-I Standard.

WSDL SOAP binding style

Define message style to use:

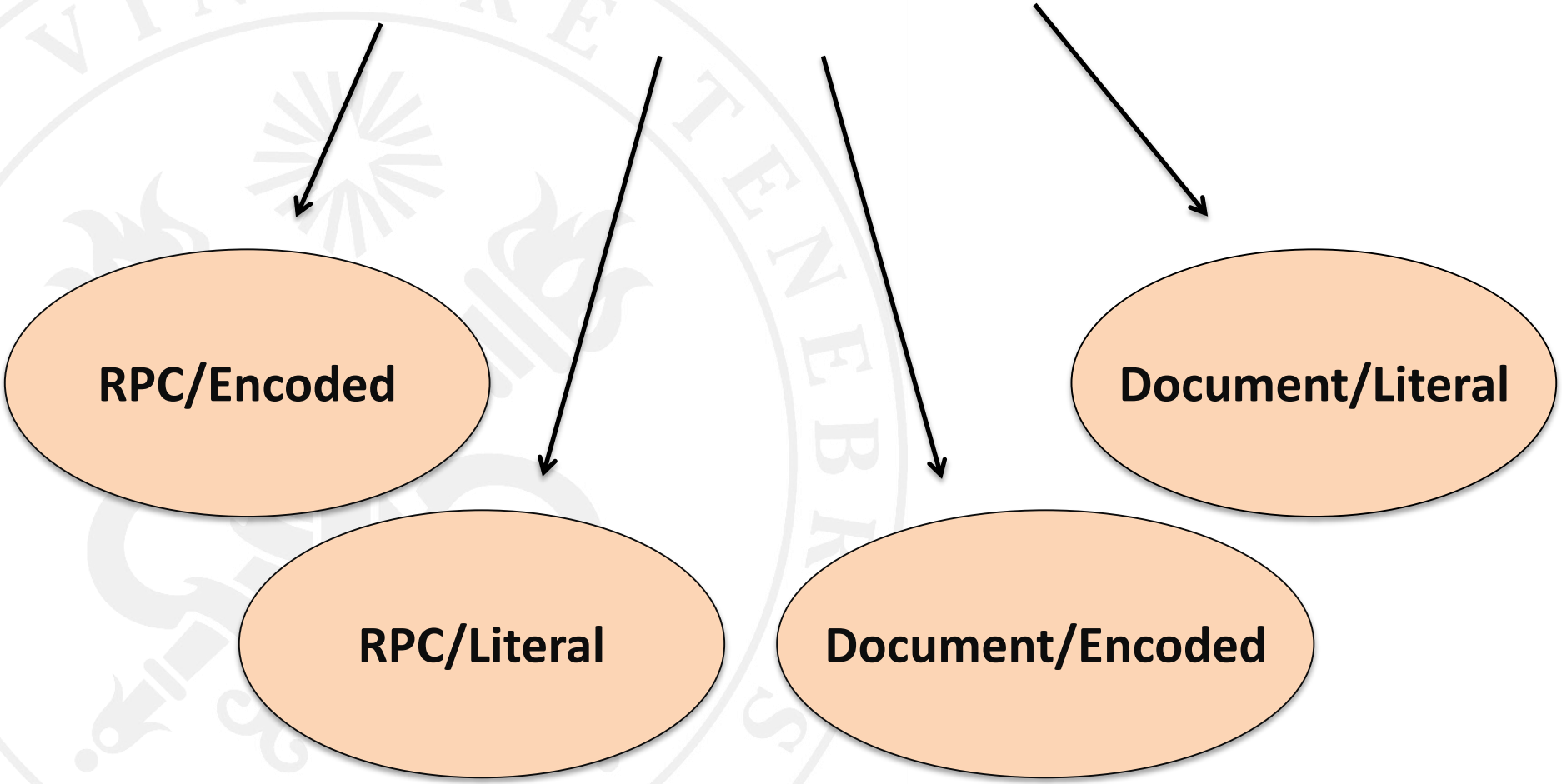


- XML element with sub elements as **content** of the SOAP Body.
- allows **input** and **output** parameters to be exchanged.

- specifically structured XML document as **content** of the SOAP Body.
- allows **any** type of messages to be exchanged.

Types of WSDL SOAP binding

4 different **Style/Encoded** combinations



RPC/Encoded - WSDL (1.1)

- Let's start considering the method

```
Public void MyMethod(int x, float y)
```

RPC encoded WSDL for *MyMethod:*

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

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

<soap:binding
      transport="http://schemas.xmlsoap.org/soap/http"
      style="rpc" />
                ...
                <output>
     <soap:body use="encoded" />
                </output>
<binding .../>
```

In the SOAP binding, the style is *RPC* and the use is *encoded*.

RPC/Encoded SOAP

RPC/encoded SOAP message for *myMethod*

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

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

    </myMethod>
  </soap:Body>
</soap:Envelope>
```

SOAP encoding has a *URI* to identify it

message will be exchange as typed(encoded)

SW Analisys of RPC/Encoded

STRENGHTS
Operation name appears in the message.
The WSDL is about as straightforward as it's possible for WSDL to be.

WEAKNESSES
Type encoding information overhead (such as xsi:type="xsd:int”).
Legal WSDL but RPC/encoded is not WS-I compliant.
SOAP message cannot be validated except against WSDL.

RPC/Literal

Is there a way to keep the strengths and remove the weaknesses of RPC/Encoded?

Let's look at the RPC/Literal style.

RPC/Literal WSDL (1.1)

- RPC/literal WSDL message 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>

<soap:binding
       transport="http://schemas.xmlsoap.org/soap/http"
         style="rpc" />
                ...
                <output>
      <soap:body use="literal" />
                </output>
<binding .../>
```

In the SOAP binding, the style is *RPC* and the use is *literal*.

RPC/Literal SOAP

- RPC/encoded SOAP message for *myMethod*

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

Contents of *soap:Body* will validate against the schema.

The *type* encodings have been removed.

WS Analisys of RPC/Literal

STRENGHTS
The WSDL is still about as straightforward as it is possible for WSDL to be.
The operation name still appears in the message.
The type encoding info is eliminated.
RPC/literal is WS-I compliant.

WEAKNESSES
Nearly all the definitions in WSDL so not independently validatable.

Document/Literal

Is there a way to overcome the weakness regarding RPC/Literal?

Let's look at the Document/Literal style.

Document/Literal WSDL (1.1)

Document/literal WSDL for *myMethod*:

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

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

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

<binding .../>
```

Describe the content of the message body.

Define *part* element that points to a schema element.

Document/Literal SOAP

Document/literal SOAP message for *myMethod:*

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

soap:Body formatted according to the schema included in WSDL

WS Analisys of Document/Literal

STRENGHTS
There is no type encoding info.
Everything within the **soap:body** is defined in a schema.
Document/literal is WS-I compliant.

WEAKNESSES
The WSDL is getting a bit more complicated.
The operation name in the SOAP message is lost.
WS-I only allows one child of the **soap:body** in a SOAP message.

Document/Encoding

Nobody follow this style!

It is not WS-I compliant and not useful.

Describe REST Web services with WSDL

WSDL 1.1
WSDL 2.0

A WSDL description contains all the details of a Web service, including:

- The service's URL.
- The communication mechanisms it understands.
- What operations it can perform.
- The structure of its messages.

Example:

Bookstore, URL: http://www.bookstore.com

- The **book list service** retrieves the list of the books that you sell in your store

http://www.bookstore.com/books/

Parameters: *Author, Language, Publisher, Subject, Title*

- The **book details service** retrieves the details about a specific book

http://www.bookstore.com/books/ISBN_NUMBER

http://www.bookstore.com/books/?subject=computers/eclipse

```
<booklist:bookList xmlns:booklist="http://www.bookstore.org/booklist/xsd"
   xmlns:book="http://www.bookstore.org/book/xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bookstore.org/booklist/xsd booklist.xsd
                       http://www.bookstore.org/book/xsd book.xsd">

  <booklist:book url="http://www.bookstore.com/books/0321442598"
     title="BIRT: A Field Guide to Reporting"/>
  <booklist:book url="http://www.bookstore.com/books/0321205758"
     title="Contributing to Eclipse: Principles, Patterns, and Plug-Ins"/>
  <booklist:book url="http://www.bookstore.com/books/0321245873"
     title="Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the..."/>
  <booklist:book url="http://www.bookstore.com/books/0321288157"
     title="Eclipse Distilled"/>
  <booklist:book url="http://www.bookstore.com/books/0131425420"
     title="Eclipse Modeling Framework"/>
  <booklist:book url="http://www.bookstore.com/books/0321334612"
     title="Eclipse Rich Client Platform: Designing, Coding, and Packaging Java..."/>
  <booklist:book url="http://www.bookstore.com/books/0321396855"
     title="Eclipse Web Tools Platform: Developing Java Web Applications"/>
  <booklist:book url="http://www.bookstore.com/books/032142672X"
     title="Eclipse: Building Commercial-Quality Plug-Ins (2nd Edition)"/>
  <booklist:book url="http://www.bookstore.com/books/0321443853"
     title="Integrating and Extending BIRT"/>
  <booklist:book url="http://www.bookstore.com/books/0321268385"
     title="Official Eclipse 3.0 FAQs"/>
  <booklist:book url="http://www.bookstore.com/books/0321256638"
     title="Official Eclipse 3.0 FAQs"/>
</booklist:bookList>
```

http://www.bookstore.com/books/0321396855

```
<book:book xmlns:book="http://www.bookstore.org/book/xsd"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.bookstore.org/book/xsd book.xsd">
  <book:title>Eclipse Web Tools Platform: Developing Java Web Applications</book:title>
  <book:author>
    <book:firstName>Naci</book:firstName>
    <book:lastName>Dai</book:lastName>
  </book:author>
  <book:author>
    <book:firstName>Lawrence</book:firstName>
    <book:lastName>Mandel</book:lastName>
  </book:author>
  <book:author>
    <book:firstName>Arthur</book:firstName>
    <book:lastName>Ryman</book:lastName>
  </book:author>
  <book:overview>
</book:overview>
  <book:pages>752</book:pages>
  <book:publisher>Addison-Wesley Professional</book:publisher>
  <book:language>English</book:language>
  <book:isbn-10>0321396855</book:isbn-10>
  <book:isbn-13>978-0321396853</book:isbn-13>
  <book:price>54.99</book:price>
</book:book>
```

Address the book list service:

http://www.bookstore.com/books/

http://www.bookstore.org/booklist/wsdl

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
   targetNamespace="http://www.bookstore.org/booklist/wsdl"
   xmlns:tns="http://www.bookstore.org/booklist/wsdl">

  <wsdl:service name="BookList" interface="">
    <wsdl:documentation>
      The bookstore's book list service.
    </wsdl:documentation>
    <wsdl:endpoint name="BookListHTTPEndpoint"
      binding=""
      address="http://www.bookstore.com/books/">
    </wsdl:endpoint>
  </wsdl:service>
</wsdl:description>
```

Talking HTTP with the book list service

The book list binding definition:

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
  targetNamespace="http://www.bookstore.org/booklist/wsdl"
  xmlns:tns="http://www.bookstore.org/booklist/wsdl"
  xmlns:whttp="http://www.w3.org/ns/wsdl/http">

  <wsdl:binding name="BookListHTTPBinding"
    type="http://www.w3.org/ns/wsdl/http"
    interface="">
    <wsdl:documentation>
      The RESTful HTTP binding for the book list service.
    </wsdl:documentation>
    <wsdl:operation ref="" whttp:method="GET"/>
  </wsdl:binding>

  <wsdl:service name="BookList" interface="">
    <wsdl:documentation>
      The bookstore's book list service.
    </wsdl:documentation>
    <wsdl:endpoint name="BookListHTTPEndpoint"
      binding="tns:BookListHTTPBinding"
      address="http://www.bookstore.com/books/">
    </wsdl:endpoint>
  </wsdl:service>
</wsdl:description>
```

Define the book list service operation

```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
  targetNamespace="http://www.bookstore.org/booklist/wsdl"
  xmlns:tns="http://www.bookstore.org/booklist/wsdl"
  xmlns:whttp="http://www.w3.org/ns/wsdl/http"
  xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions">

 <wsdl:interface name="BookListInterface">
  <wsdl:operation name="getBookList"
    pattern="http://www.w3.org/ns/wsdl/in-out"
    style="http://www.w3.org/ns/wsdl/style/iri"
    wsdlx:safe="true">
   <wsdl:documentation>
     This operation returns a list of books.
   </wsdl:documentation>
   <wsdl:input element=""/>
   <wsdl:output element=""/>
  </wsdl:operation>
 </wsdl:interface>

 <wsdl:binding name="BookListHTTPBinding"
   type="http://www.w3.org/ns/wsdl/http"
   interface="tns:BookListInterface">
  <wsdl:documentation>
    The RESTful HTTP binding for the book list service.
  </wsdl:documentation>
  <wsdl:operation ref="tns:getBookList" whttp:method="GET"/>
 </wsdl:binding>

 <wsdl:service name="BookList" interface="tns:BookListInterface">
  <wsdl:documentation>
    The bookstore's book list service.
  </wsdl:documentation>
  <wsdl:endpoint name="BookListHTTPEndpoint"
    binding="tns:BookListHTTPBinding"
    address="http://www.bookstore.com/books/">
  </wsdl:endpoint>
 </wsdl:service>
</wsdl:description>
```

Define the book list service operation messages

Input message: *getBookList.*

Contains a sequence of elements, including each of the query parameters you allow on the service, namely author, title, publisher, subject, and language

Output message: *bookList*

Contains a sequence of book elements. Each book element contains **title** and **url** attributes. The **title** attribute should be self explanatory. The **url** attribute is a link to a book details REST Web service, which returns the details for the specific book.

WSDL 2.0 description of the book list REST Web service


```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
  targetNamespace="http://www.bookstore.org/booklist/wsdl"
  xmlns:tns="http://www.bookstore.org/booklist/wsdl"
  xmlns:whttp="http://www.w3.org/ns/wsdl/http"
  xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:msg="http://www.bookstore.org/booklist/xsd">
 <wsdl:documentation>
    This is a WSDL 2.0 description of a sample bookstore service
    listing for obtaining book information.
 </wsdl:documentation>

 <wsdl:types>
   <xs:import namespace="http://www.bookstore.org/booklist/xsd"
      schemaLocation="booklist.xsd"/>
 </wsdl:types>

 <wsdl:interface name="BookListInterface">
   <wsdl:operation name="getBookList"
      pattern="http://www.w3.org/ns/wsdl/in-out"
      style="http://www.w3.org/ns/wsdl/style/iri"
      wsdlx:safe="true">
     <wsdl:documentation>
        This operation returns a list of books.
     </wsdl:documentation>
     <wsdl:input element="msg:getBookList"/>
     <wsdl:output element="msg:bookList"/>
   </wsdl:operation>
 </wsdl:interface>

 <wsdl:binding name="BookListHTTPBinding"
    type="http://www.w3.org/ns/wsdl/http"
    interface="tns:BookListInterface">
   <wsdl:documentation>
      The RESTful HTTP binding for the book list service.
   </wsdl:documentation>
   <wsdl:operation ref="tns:getBookList" whttp:method="GET"/>
 </wsdl:binding>

 <wsdl:service name="BookList" interface="tns:BookListInterface">
   <wsdl:documentation>
      The bookstore's book list service.
   </wsdl:documentation>
   <wsdl:endpoint name="BookListHTTPEndpoint"
      binding="tns:BookListHTTPBinding"
      address="http://www.bookstore.com/books/">
   </wsdl:endpoint>
 </wsdl:service>
</wsdl:description>
```

Web Application Description Language

•WADL is designed to provide a machine process-able description of HTTP-based Web applications, which

- Are based on existing Web architecture and infrastructure
- Are platform and programming language independent
- Promote re-use of the application beyond the browser
- Enable composition with other Web or desktop applications
- Require semantic clarity in content (representations) exchanged during their use

So we can describe:

- Set of resources

(Analogous to a site map showing the resources on offer)

- Relationships between resources

(Describing the links between resources, both referential and causal)

- Methods that can be applied to each resource

(The HTTP methods that can be applied to each resource, the expected inputs and outputs and their supported formats)

- Resource representation formats

(The supported MIME types and data schemas in use)

```
1 <?xml version="1.0"?>
2 <application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3  xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
4  xmlns:tns="urn:yahoo:yn"
5  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6  xmlns:yn="urn:yahoo:yn"
7  xmlns:ya="urn:yahoo:api"
8  xmlns="http://wadl.dev.java.net/2009/02">
9   <grammars>
10    <include
11     href="NewsSearchResponse.xsd"/>
12    <include
13     href="Error.xsd"/>
14  </grammars>
15
16  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
17   <resource path="newsSearch">
18    <method name="GET" id="search">
19     <request>
20      <param name="appid" type="xsd:string"
21       style="query" required="true"/>
22      <param name="query" type="xsd:string"
23       style="query" required="true"/>
24      <param name="type" style="query" default="all">
25       <option value="all"/>
26       <option value="any"/>
27       <option value="phrase"/>
28      </param>
```

```
29          <param name="results" style="query" type="xsd:int" default="10"/>
30          <param name="start" style="query" type="xsd:int" default="1"/>
31          <param name="sort" style="query" default="rank">
32            <option value="rank"/>
33            <option value="date"/>
34          </param>
35          <param name="language" style="query" type="xsd:string"/>
36        </request>
37        <response status="200">
38          <representation mediaType="application/xml"
39            element="yn:ResultSet"/>
40        </response>
41        <response status="400">
42          <representation mediaType="application/xml"
43            element="ya:Error"/>
44        </response>
45      </method>
46    </resource>
47  </resources>
48
49 </application>
```

Description Components

http://wadl.dev.java.net/2009/02

- Cross Referencing (intra-, inter-document)
- Application
- Documentation (*xml:lang, title*)
- Grammars (*include*)
- Resources (*base*)
- Resource (*id, path, type, queryType*)
- Resource Type (*id*)
- Method (Method Reference or Method Defenition)

Description Components

http://wadl.dev.java.net/2009/02

- Request (query parameters example)
- Response
- Representation (Reference or Definition)
- Parameter (Reference or Definition (i*d, name, style, type, default, path, required, repeating, fixed*), Option (value, mediaType), Link (resource_type, rel, rev))
- Extensibility

Query example:

```
 1 <resources base="http://example.com/widgets">
 2   <resource path="{widgetId}">
 3     <param name="customerId" style="query"/>
 4     <method name="GET">
 5       <request>
 6         <param name="verbose" style="query"
type="xsd:boolean"/>
 7       </request>
 8       <response>
 9         ...
10       </response>
11     </method>
12   </resource>
13 </resources>
```

Option parameter example:

```
1 <method name="GET">
 2   <request>
 3     <param name="format" style="query">
 4       <option value="xml"
mediaType="application/xml"/>
 5       <option value="json"
mediaType="application/json"/>
 6     </param>
 7     ...
 8   </request>
 9   <response>
10     <representation mediaType="application/xml"/>
11     <representation mediaType="application/json"/>
12   </response>
13 </method>
```

Link parameter example:

```
<!-- XML-based representation of a widget -->
<w:widget xmlns:w="http://example.com/widgets">
  <w:loc>http://example.com/widgets/110113</w:loc>
  <w:name>A Widget</w:name>
  <w:description>A very useful gizmo.</w:description>
  <w:price currency="USD">19.99</w:price>
  <w:list>http://example.com/widgets</w:list>
</w:widget>

<!-- WADL fragment describing the widget representation
     without parameters-->
<representation mediaType="application/xml"
  element="w:widget"/>

<!-- WADL fragment describing the widget representation
     with parameters -->
<representation mediaType="application/xml"
  element="w:widget">
  <param name="location" style="plain"
    type="xsd:anyURI" path="/w:widget/w:loc">
    <link resource_type="#widget" rel="self"/>
  </param>
  <param name="index" style="plain"
    type="xsd:anyURI" path="/w:widget/w:list">
    <link resource_type="#widgets" rel="index" rev="child"/>
  </param>
</representation>
```

References

- http://www.w3.org/TR/2006/CR-wsdl20-primer-20060106/
- http://msdn.microsoft.com/en-us/library/ms996486.aspx
- http://msdn.microsoft.com/en-us/library/ms995710.aspx
- http://msdn.microsoft.com/en-us/library/ms996466.aspx
- http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/

References

- http://www.w3.org/Submission/wadl/