# Introduction to Web Services

Ioannis G. Baltopoulos

Department of Computer Science
Imperial College London

CERN School of Computing (iCSC), 2005
Geneva, Switzerland

1. Web Services
   - Fundamental Concepts
   - Architectures & eScience example

2. Related Standards
   - XML
   - SOAP
   - WSDL

# Distributed Computing Technologies

## Historic Review (20 years in 5 minutes!)

- **CORBA (OMG)**
  It is standards-based, vendor-neutral, and language-agnostic. Very powerful but limited however by its complicated way of utilizing the power and flexibility of the Internet.
- **DCOM (Microsoft)**
  Distributed Computing platform closely tied to Microsoft component efforts such as OLE, COM and ActiveX.
- **RMI (Sun Microsystems)**
  Java based effort which doesn't play well with other languages. The J2EE platform integrated RMI with IIOP.
- **Web Services (W3C)**
  Web services are more of an evolution than a revolution

# What is a Web Service?

## Definition

A **Web Service** is a standards-based, language-agnostic software entity, that accepts specially formatted requests from other software entities on remote machines via vendor and transport neutral communication protocols, producing application specific responses.

- **Standards based**
- **Language agnostic**
- **Formatted requests**
- **Remote machines**
- **Vendor neutral**
- **Transport neutral**
- **Application specific responses**

# Benefits of Web Services

- **Loosely Coupled**
  Each service exists independently of the other services that make up the application. Individual pieces of the application to be modified without impacting unrelated areas.
- **Ease of Integration**
  Data is isolated between applications creating 'silos'. Web Services act as glue between these and enable easier communications within and across organisations.
- **Service Reuse**
  Takes code reuse a step further. A specific function within the domain is only ever coded once and used over and over again by consuming applications.

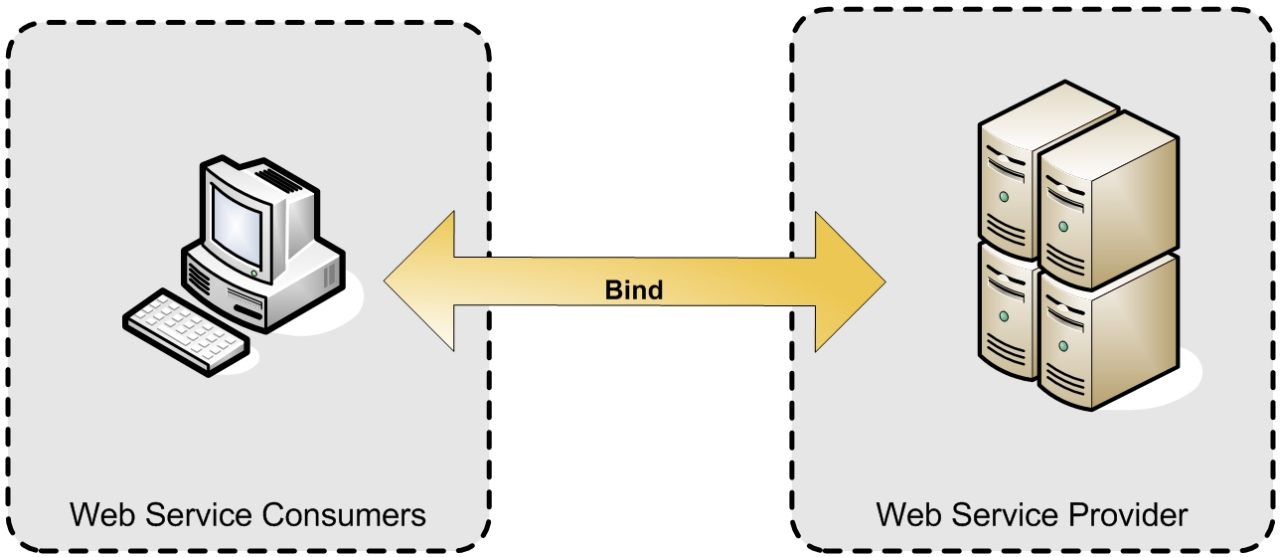
# Web Services Architectures

## The simplest Web Service System

The simplest Web service system has two participants:

- A service **producer** (provider)
- A service **consumer** (requester).

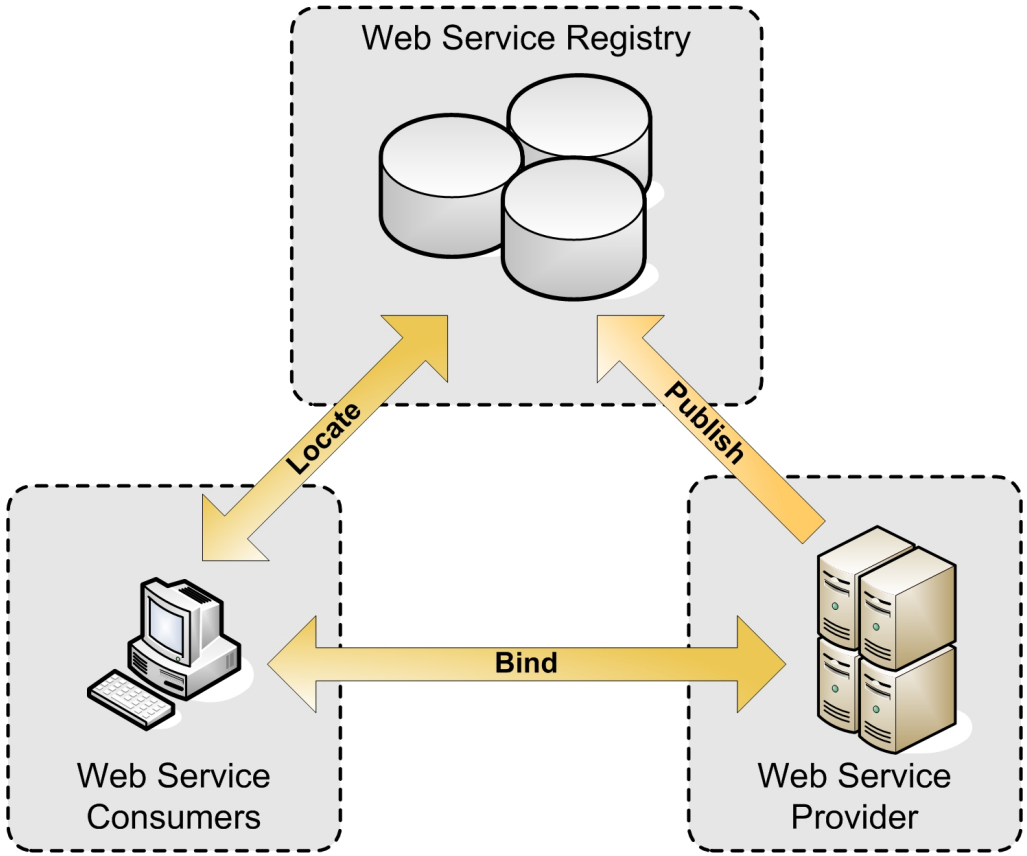
The provider presents the interface and implementation of the service, and the requester uses the Web service.



# Web Services Architectures
## A Service Oriented Architecture (SOA)

A more sophisticated system:

- A **registry**, acts as a broker for Web services.
- A **provider**, can publish services to the registry
- A **consumer**, can then discover services in the registry



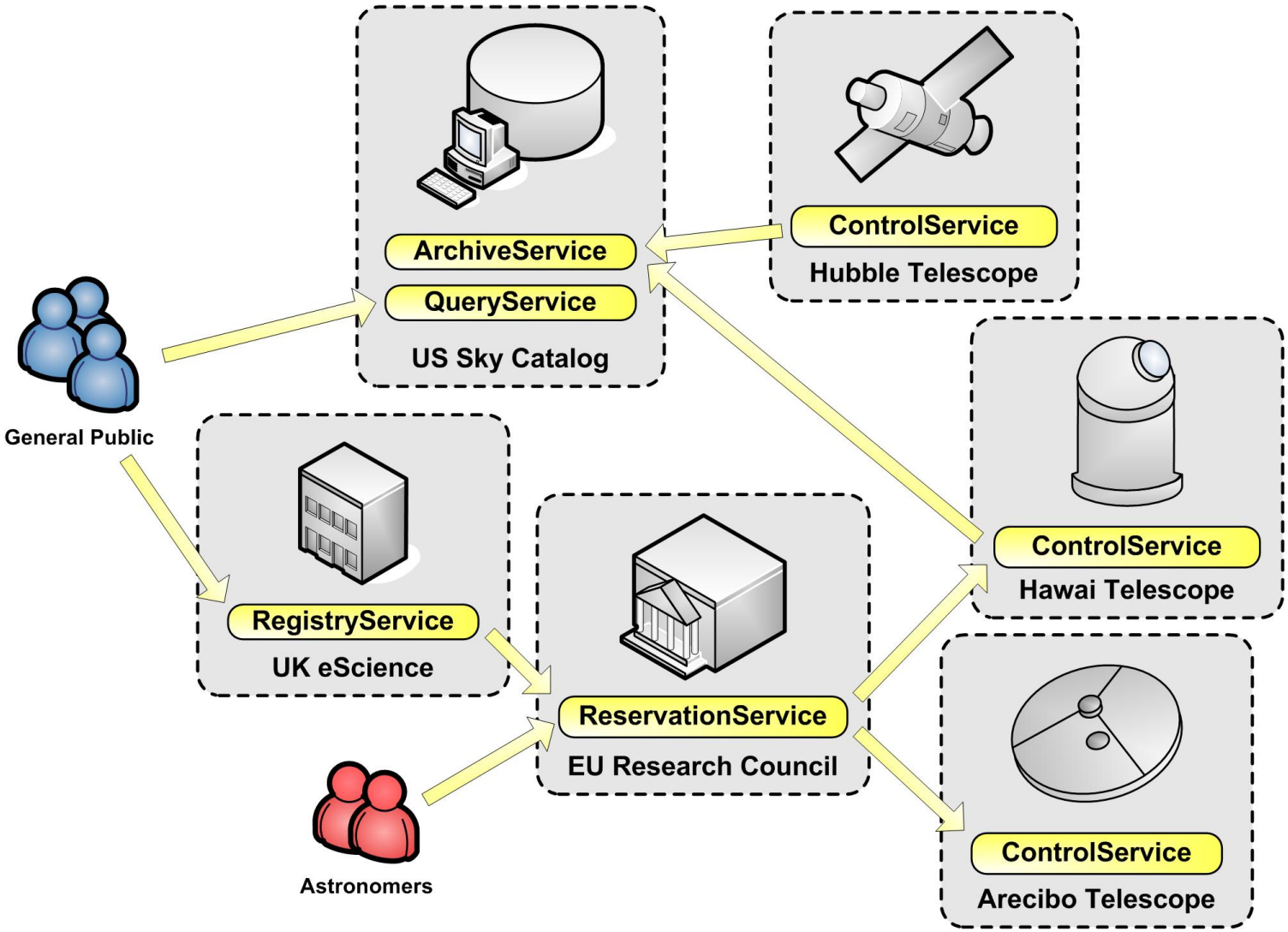
# e-Science example

## Web Enabled Telescope Access Requirements

In the context of eScience and observatories, there are several requirements from a distributed astronomical system.
For example,

- different people need access to subsets of the same data,
- data needs to be archieved for future use,
- same functionality implemented using different technologies,
- certain authorities authorize the use of resources,
- others are responsible for cataloging available resources.

# e-Science example
## Web Enabled Telescope Access



# eXtensible Markup Language (XML)

## Definition

The eXtensible Markup Language (XML) is a W3C recommendation for creating special-purpose markup languages that enable the structuring, description and interchange of data.

- A simplified subset of SGML capable of describing many different kinds of data for any imaginable application domain.
- It facilitates the sharing of structured text and information in databases and across the Internet.
- Languages based on XML are themselves described in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form.
- Separate syntax from semantics.
- Inherently supports internationalization (Unicode) and platform independence.

# XML Building Blocks

- **Elements**
  The pairing of a start tag and an end tag.
- **Attributes**
  A name-value pair that is part of a starting tag of an Element.
- **Processing Instructions**
  Special directives to the application that will process the XML document.
- **Comments**
  Messages helping a human reader understand the source code.
- **Character Data**
  - Characters (in a specific encoding)
  - Entities
  - Whitespace

# XML Elements

## Formal Definition & Rules

### Definition

The term **element** is a technical name for the pairing of a start tag and an end tag in an XML Document.

### Production Rule

$$
\begin{aligned}
\langle element \rangle &::= \langle EmptyElement \rangle \\
&\quad | \quad \langle STag \rangle \ \langle content \rangle \ \langle ETag \rangle \\
\langle STag \rangle &::= \text{`<'} \ \langle Name \rangle \ \langle Attribute \rangle^{*} \ \text{`>'} \\
\langle ETag \rangle &::= \text{`</'} \ \text{Name} \ \text{`>'} \\
\langle EmptyElement \rangle &::= \text{`<'} \ \text{Name} \ \langle Attribute \rangle^{*} \ \text{`/>'}
\end{aligned}
$$

- XML Elements must be strictly nested!
- Element names can include letters, the underscore, hyphen and colon; they **must** begin with a letter.
- Element names are case sensitive!

# XML Elements

## Some right & wrong examples

### Example

```
<!-- Example 1: Element with two tags -->
<message> Welcome! </message>

<!-- Example 2: Empty Element (Single tag) -->
<message/>
```

### Wrong Examples

```
<!-- Example 1: Incorrect Nesting -->
<ATag><BTag> Nesting Problem </ATag></BTag>

<!-- Example 2: Invalid Element name -->
<.wrong.element> some text </.wrong.element>
```

# XML Attributes

## Formal Definition & Rules

### Definition

The term **attribute**(s) refers to a theoretically arbitrary number of name-value pairs that can be included in the starting tag of an XML element.

### Production Rule

$\langle STag \rangle$ ::= '<' $\langle TagName \rangle$ $\langle Attribute \rangle^*$ '>'

$\langle Attribute \rangle$ ::= AttrName '=' Value

- The value part of the attribute has to be **quoted**.
- Attribute names starting with `xml:` are **reserved** by the XML specification.

### Example

```
<!-- Single attribute -->
<yacht length="60f"/>
```

# Processing Instructions
## Definition, Rule & Example

**Definition**

A special directive to the applications processing the XML documents.

**Production Rule**

$\langle PI \rangle$ ::= '<?' PITarget ... '?>'

**Example**

```
<!-- Example: A popular one! -->
<?xml version="1.0" encoding="UTF-8"?>
```

- The PI Target keyword is meaningful to the processing application and hence could be different between applications.
- Everything between the PI Target and the closing question mark is considered the contents of the processing instruction.

# Comments & Character Data

## Definition, Rule & Example

**Comment** A message that helps the human reader understand the program and the processing that takes place at a particular point of the source code.

**Production Rule**

$\langle Comment \rangle ::=$ '<!--' Char* '-->'

**Character Data**

- **Encoding:** All characters in an XML document must comply with the document's encoding; those outside the encoding must be escaped and are called **character references**.
- **Whitespace:** Whitespace can be treated as either significant or insignificant. Most XML applications care little about whitespace.
- **Entities:** Like character references, they are predefined escape sequences that map to specific characters.

# An XML Document

## Putting it all together!

```
<?xml version="1.0" encoding="UTF-8"?>
<message from="yiannis" to="family">
    <text>Hey, I'm at the iCSC!
    </text>
    <!-- Attachment is optional -->
    <attachment>
        <desc>Photo from Geneva</desc>
        <item>
            <?BinaryDataStart ?>
            01001000010100010010100 10
            <?BinaryDataEnd ?>
        </item>
    </attachment>
</message>
```

An XML Document consists of:

- Optional prolog
- A root element
- Comments
- Processing Instructions

But...

# Some Problems

## And how we solved them!

The problems in the previous example relate with the:

- **Physical Structure** of the document
  Well formedness (Parsers)
- **Logical Structure** of the document
  Validity (Schemas). Semantics of the elements?
- **Element Name clashes** between Documents
  Namespaces

# XML Namespaces

## Motivating the Problem

Solve the problem of recognition and collision of elements in an XML Document.

- **Recognition**
  How does an XML processing application distinguish between the XML elements that describe the message and the XML elements that are part of a Purchase Order?
- **Collision**
  Does the element description refer to attachment descriptions in messages or order item descriptions? Does the item element refer to an item of attachment or an order item?

# XML Namespaces

## Detailing the Solution

XML Namespaces uses Uniform Resource Identifiers for uniquely qualifying local names. As URIs can be long and typically contain characters that arent allowed in XML element names, the process of including namespaces in XML document involved two steps:

- A namespace identifier is associated with a prefix, a name that contains only legal XML element name characters with the exception of the colon (;)
- Qualified names are obtained as a combination of the prefix, the colon character, and the local element name

```
Qualified Name = Namespace Identifier + Local Name
```

# A Namespaces XML Document

```
<msg:message from="yiannis" to="family"
  xmlns:msg="http://www.w2c.com/ns/email"
  xmlns:po="http://www.w2c.com/ns/purchase">
   <msg:text>
      <msg:desc>A Purchase Order</msg:desc>
      <msg:item>
         <po:order>
            <po:item>
               <po:desc>Laptop Computer</po:desc>
               <po:price>1300 GBP</po:price>
            </po:item>
         </po:order>
      </msg:item>
   </msg:text>
</msg:message>
```

# XML Namespaces

## A couple more last things

- **Default Namespaces**
  Adding a prefix to every element in the document decreases readability and increases document size. Therefore, XML Namespaces allow us to use a default namespace in a document. Elements belonging to the default namespace don't require prefixes.
- **Namespace prefixed attributes**
  Attributes can also have namespaces associated with them. The desire to extend the information provided by an XML element without having to make changes directly to its document type.

# XML Schema

An XML Schema enables the following:

- Identification of the elements that can be in a document
- Identification of the order and relation between elements
- Identification of the attributes of every element and whether they're optional or required or have some other special properties
- Identification of the datatype of attribute content

Think of it as an elaborate UML Class diagram where classes only have field and no methods.

# Simple Object Access Protocol (SOAP)

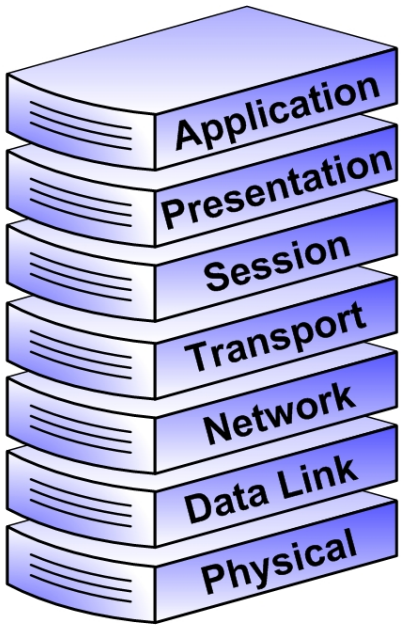
## What's the big deal?

**Definition**

SOAP is an industry accepted W3C specification for a ubiquitous XML distributed computing infrastructure.

- A mechanism for defining the unit of communication.
- A mechanism for error handling.
- An extensibility mechanism
- Lives above the transport layer of OSI

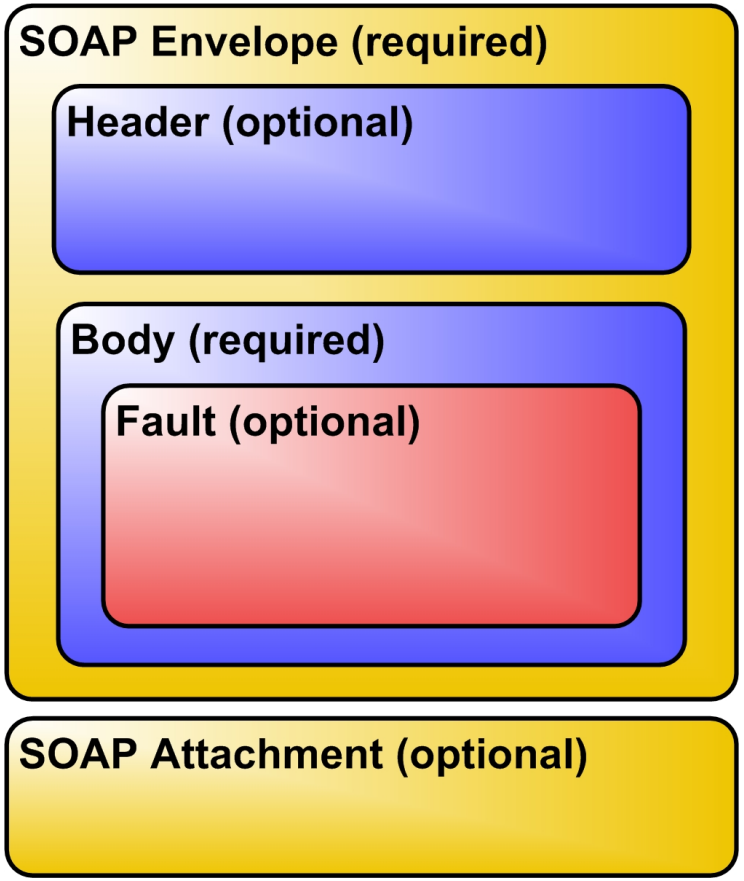
Simply put its a mechanism that allows the transmission of XML documents, regardless of transport layer protocol.




**OSI Reference Model**

# SOAP Messages
## Logical & Physical Structure

- The root element of a SOAP message is the `Envelope` element.
- It contains an optional `Header` element and the required `Body`
- Elements called `Faults` can be used to describe exceptional situations.
- It can contain optional Attachments in MIME encoding for exchanging binary data.



# SOAP Example

## Structure of a real XML SOAP Message

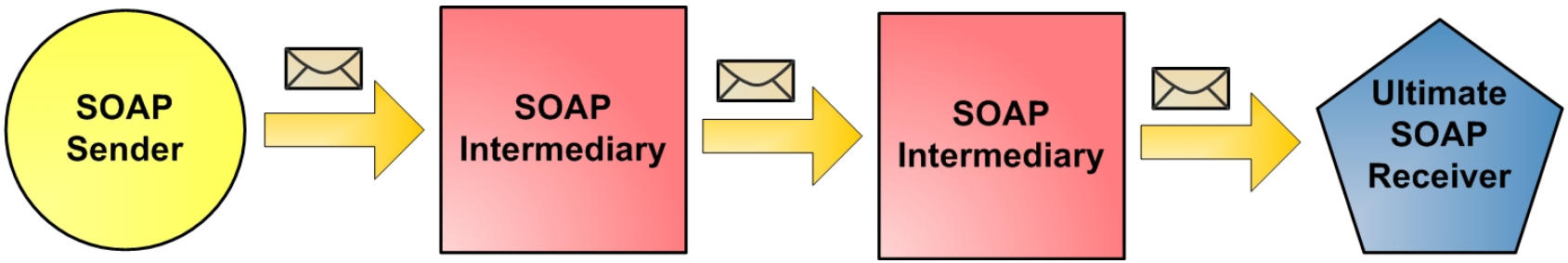
```
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
  soap:encodingStyle="http://soap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst">
  <soap:Header>
    <!-- Transactions, priorites, etc. -->
  </soap:Header>
  <soap:Body>
    <!-- Some content -->
  </soap:Body>
</soap:Envelope>
```

# SOAP Message Transmission

## Message delivery path using Intermediaries

The SOAP Message Transmission involves three main roles:

- The **SOAP Sender** creates and sends a SOAP Message to an ultimate SOAP Receiver.
- One or more optional **SOAP Intermediaries** can be positioned to intercept messages between the the sender and the receiver. They can perform filtering, logging, catching etc.
- The SOAP sender's intended destination is called the **Ultimate SOAP Receiver**.



# Web Services Description Language (WSDL)

Web Services Description Language (WSDL) is an XML format for describing all the information needed to invoke and communicate with a Web Service. It gives the answers to the questions Who? What? Where? Why? How? A service description has two major components:

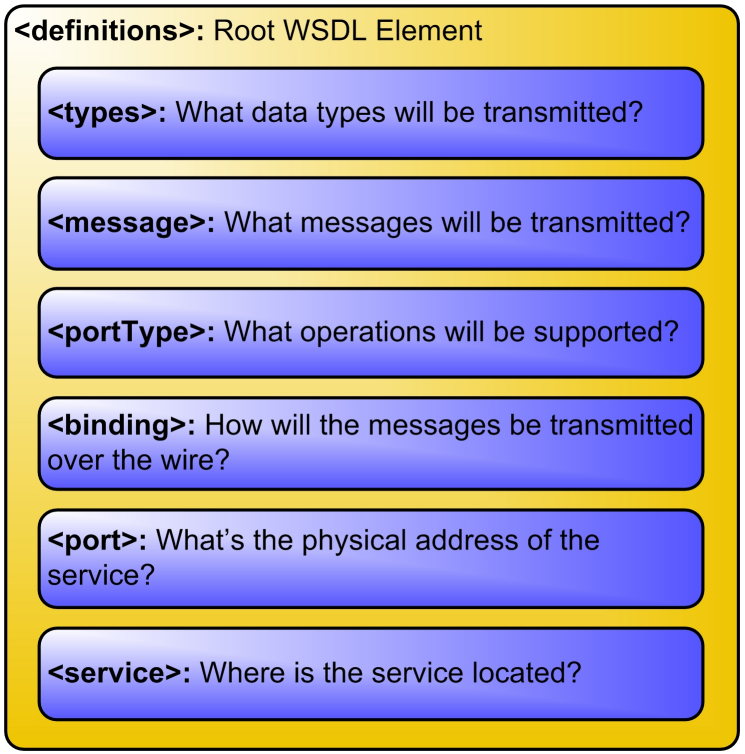
- **Functional Description**
  Defines details of how the Web Service is invoked, where it's invoked. Focuses on the details of the syntax of the message and how to configure the network protocols to deliver the message.
- **Nonfunctional Description**
  Provides other details tha are secondary to the message (such as security policy) but instruct the requestor's runtime environment to include additional SOAP headers.

# WSDL Document Structure

## The 6 basic building blocks

A WSDL Document is a set of definitions with a single root element. Services can be defined using the following XML elements:

- **Types**, think Data Type
- **Message**, think Methods
- **PortType**, think Interfaces
- **Binding**, think Encoding Scheme
- **Port**, think URL
- **Service**, many URLs

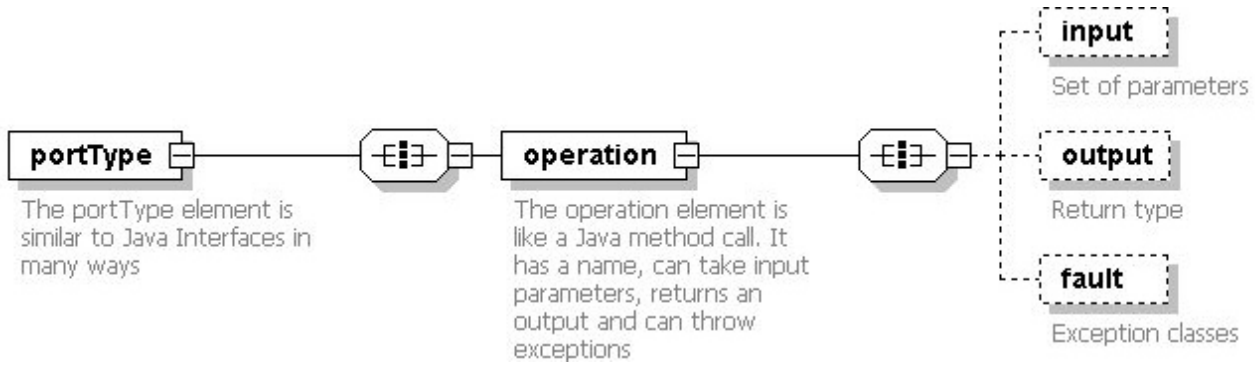


# PortType Element
## Definition and Usage

### Definition

The `portType` element describes the interface to a Web Service

- A WSDL Document can contain zero or more `portType`
- A `portType` element contains a single `name` attribute. Naming convention *`nameOfWebService`*`PortType`
- A `portType` contains one or more operation elements, with a `name` attribute can contain input, output and fault elements



# PortType Element
## Example

### Example

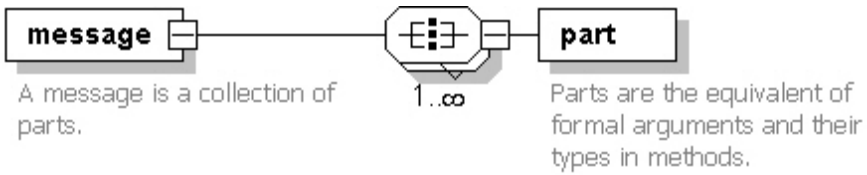
```
<!-- Port Type Definition Example -->
<portType name="weatherCheckPortType">
  <operation name="checkTemperature">
    <input message="checkTemperatureRequest"/>
    <output message="checkTemperatureResponse"/>
  </operation>
  <operation name="checkHumidity">
    <input message="checkHumidityRequest"/>
    <output message="checkHumidityResponse"/>
  </operation>
</portType>
```

# Message Element
## Definition and Usage

### Definition

A `message` is a collection of parts; intuitively a `part` is a named argument with its type. A `message` is a collection of these parts.

- A WSDL document can contain zero or more `message` elements.
- Each `message` element can be used as an input, output or fault message within an `operation` .
- The `type` attribute of `part` can be any standard data type from the XSD Schema or a user defined one.



# Message Element
## Example

### Example

```
<!-- Message Definitions -->
<message name="checkTemperatureRequest">
    <part name="location" type="xsd:string">
</message>
<message name="checkTemperatureResponse">
    <part name="result" type="xsd:double">
</message>
<message name="checkHumidityRequest">
    <part name="location" type="xsd:string">
</message>
<message name="checkHumidityResponse">
    <part name="result" type="ns:HummidityType"
</message>
```

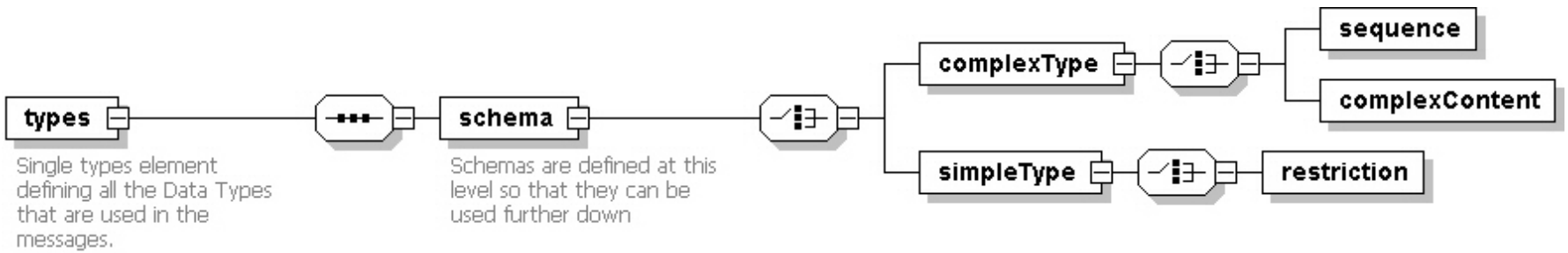
# Types Element
## Definition and Usage

### Definition

Custom user data types defined in an abstract way.

- The default type system in WSDL is the XML Schema (XSD)
- A WSDL document can have at most one `types` element.
- The types element can contain simpleType or complexType.
- At the lowest level elements intuitively named (again!) element are defined with a name and a type attribute.

NOTE! The diagram bellow is incomplete! This is considered an advanced topic and for more information you should look at data modelling using the XML Schema.



# Types Element
## Example

### Example

```
<!-- Type Definitions -->
<types>
  <xsd:schema targetNamespace="http://weather.com/ns"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="HumidityType">
      <xsd:sequence>
        <xsd:element name="loc" type="xsd:string">
        <xsd:element name="humd" type="xsd:double">
        <xsd:element name="temp" type="xsd:double">
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

# Binding Element
## Definition and Usage

**Definition**

The binding element specifies to the service requester how to format the message in a protocol-specific manner.

- Each portType can have one or more binding elements associated with it.
- For a given portType the binding element has to specify an messaging and transport pair. (SOAP/HTTP, SOAP/SMTP, etc).

# Binding Element

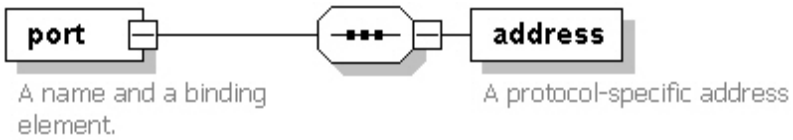
```
<binding name="WeatherBinding" type="weatherCheckPortType">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="checkTemperature">
    <soap:operation soapAction="" />
      <input>
        <soap:body use="encoded" namespace="checkTemperature"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="checkTemperature"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
  </operation>
</binding>
```

# Port Element
## Definition, Usage & Example

### Definition

The `port` element specifies the network address of the endpoint hosting the Web Service.

- It associates a single protocol-specific address to an individual binding element.
- Ports are named and must be unique within the document.




### Example

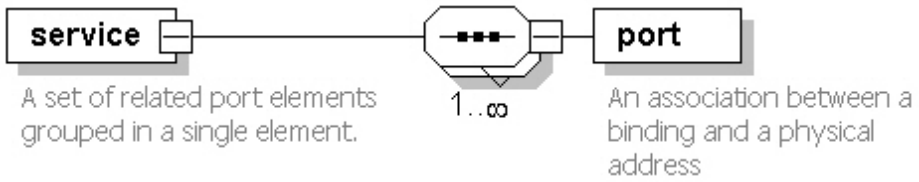
```
<port name="WeatherCheck"
      binding="wc:WeatherCheckSOAPBinding">
  <soap:address location="http://host/WeatherCheck"/>
</port>
```

# Service Element
## Definition and Usage

### Definition

The service element is a collection of related port elements identified by a single service name.

- A WSDL Document is allowed to contain multiple service elements, but conventionally contains a single one.
- Each service must be uniquely named.
- The naming convention is GeneralInfoService



# Service Element
## Example

### Example

```
<!-- Service definition -->
<service name="WeatherCheckService">
  <port name="WeatherCheckSOAP"
      binding="wc:WeatherCheckSOAPBinding">
  <soap:address location="http://host/WeatherCheck"/>
  </port>
  <port name="WeatherCheckSMTP"
      binding="wc:WeatherCheckSMTPBinding">
  <soap:address location="http://host/WeatherCheck"/>
</port>
</service>
```

# Concluding Remarks

In this first lecture we saw

- the position of Web Services within the Distributed Computing Environment.
- the XML primitives and touched upon Namespaces and Schemas.
- how SOAP is used for transferring platform and language independent messages between software entities on different hosts.
- how to describe Web Services using WSDL.

...now...GO FOR COFFEE!