

Web Services Technologies

XML and SOAP
WSDL and UDDI

Version 16

Making Software Work Together™



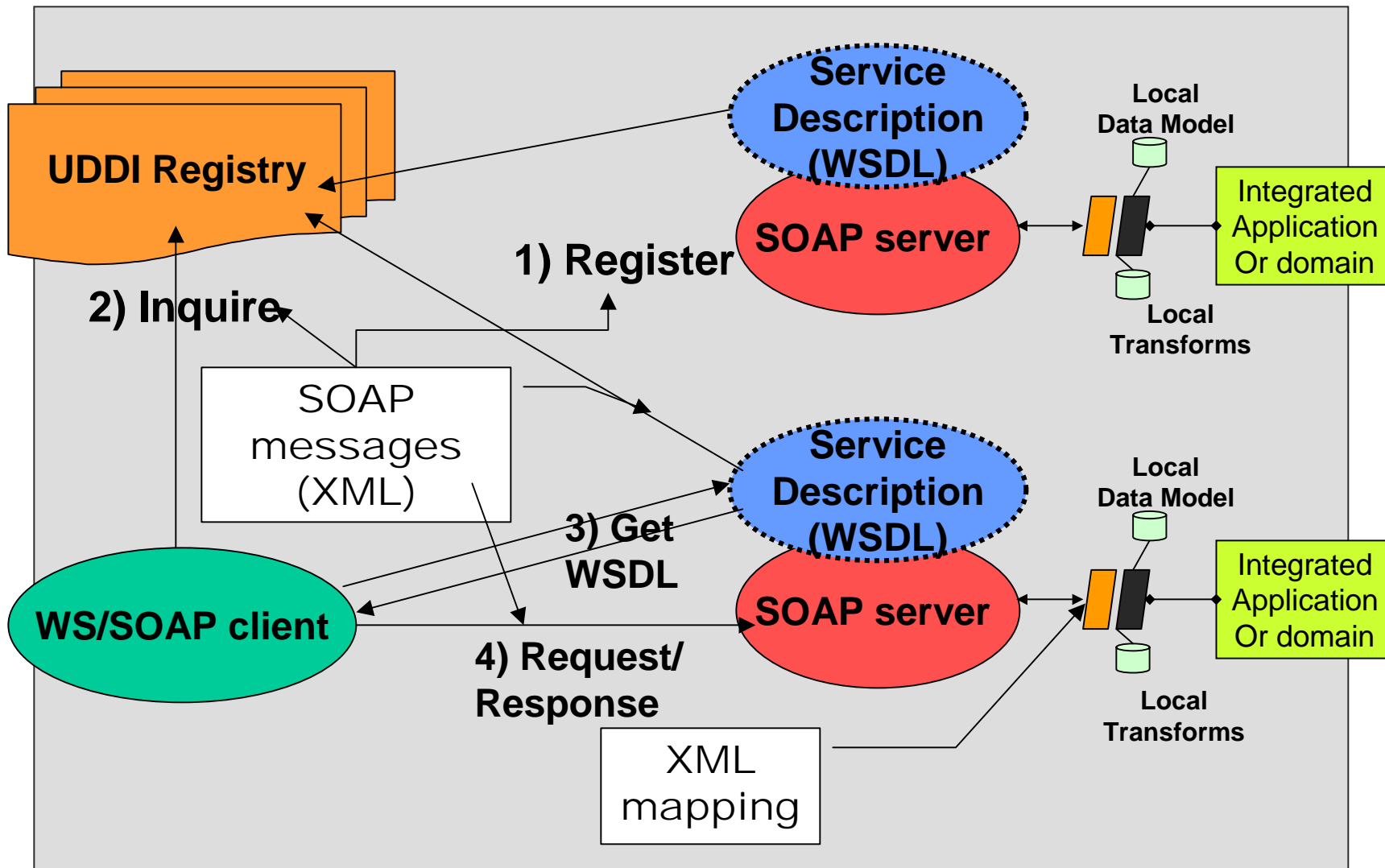
Web Services Technologies

- A collection of XML technology standards that work together to provide Web Services capabilities
 - We will provide an overview of the more prominent WS standards
- Web Services Technologies are primarily an integration or interfacing technology
 - NOT an application development technology
 - Still can develop in existing or new software development environments
- Web Services Technologies make it easier to tie together existing or planned software components
 - Due to the language-, platform-, OS-, hardware-neutral characteristics of the standards
- As we will see a later chapter, Web Services technologies can be used to implement the interfaces and messages for a service-oriented architecture (SOA)

Participants in a Web Services Environment: the roadmap for this chapter



XML / Web services – in action



Web Services Technologies Overview

- XML Technologies

- “Extensible Markup Language”
- Base XML for documents
- XML Schema for describing XML documents

- SOAP

- “Simple Object Access Protocol”
 - A simple way to send documents (some people have called it “email for documents”)
- How to format XML documents for transmission

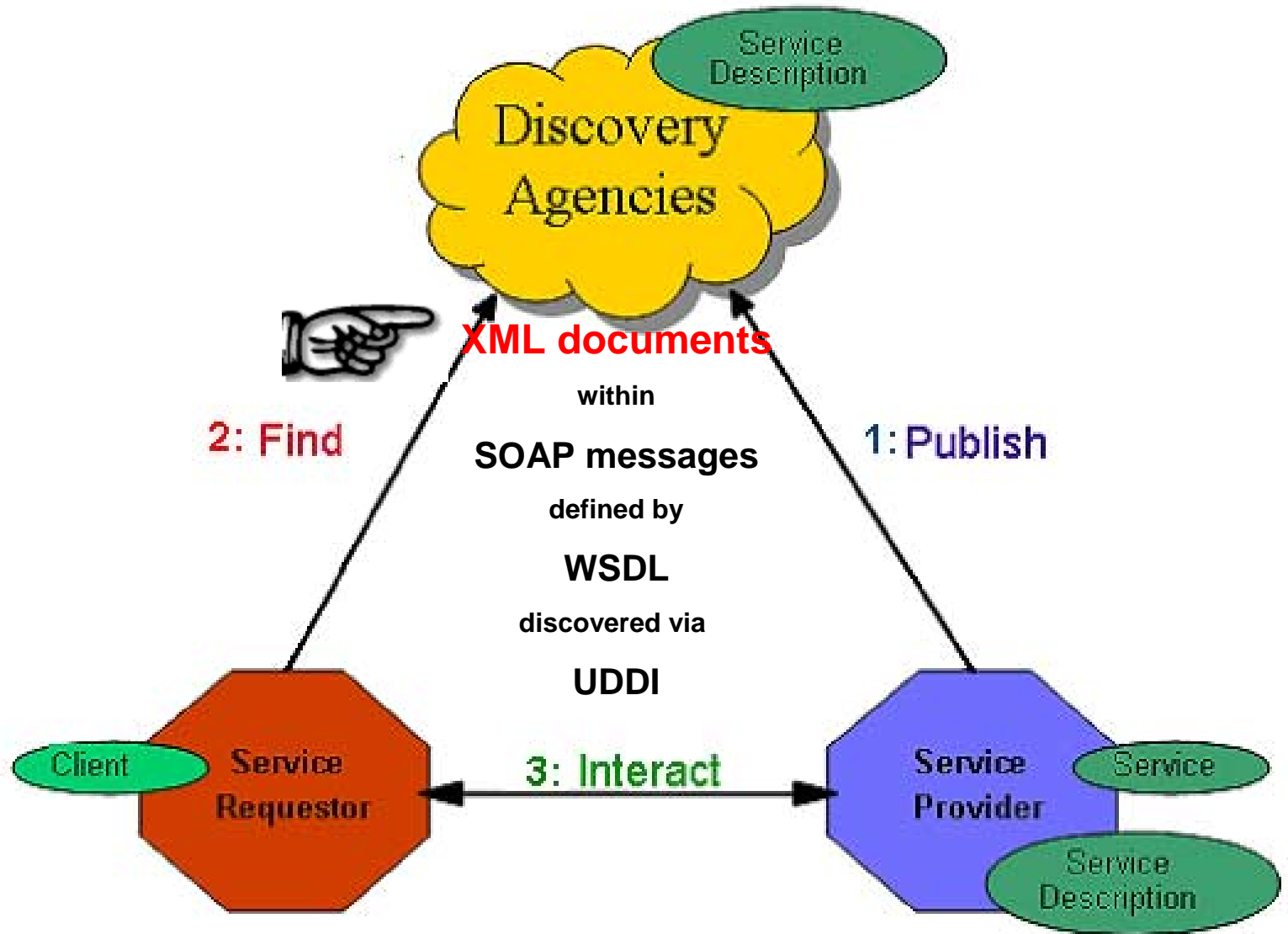
- WSDL

- “Web Services Description Language”
- Defines all details about a service

- UDDI

- “Universal Description, Discovery and Integration”
- One way to advertise and discover services

Participants in a Web Services Environment: Part 1 - XML Technologies



XML Technologies

- Collection of extensible information representation and manipulation technologies
 - XML itself – the base document standard
 - XML Schema – for data typing and document structuring
 - XSLT – for transforming XML to other XML or other formats
 - Parsing strategies for XML document processing
 - DOM – Document Object Model
 - SAX – Simple API for XML
 - Others: XPATH, XQUERY, etc.

XML – eXtensible Markup Language

- Originally a derivative of SGML in the family of markup languages
- Similar to HTML in appearance, but with a very different goal
 - to separate data content from data presentation
- Now, a full and evolving W3C standard
- Parsable, extensible and self-describing text format for exchanging information
 - Platform-, hardware-, programming-language-neutral
 - Highly portable across heterogeneous networks
- Reference: <http://www.w3.org/XML/>

SGML : Structured Generalized Markup Language

W3C : World Wide Web Consortium

XML Technologies

Important characteristics of XML

- Similar in appearance to HTML
 - HTML has a fixed set of tags, XML has a variable set
- Two major structuring mechanisms:
 - *Elements* that may contain text and/or other elements
 - *Attributes* characterize elements with simple strings
- Elements are demarcated by **Tags** at the start and end:
 - E.g., a MyElement element

```
<MyElement> .... </MyElement>
```
- **Nested** elements are used to model complex data
 - We'll see many examples of this
- **Extensible** – can define new elements and attributes easily

- Let's look at very simple XML documents – teach by example

Simple XML Document Exchange

Example of an XML-based request document

```
<getQuote>  
  <symbol>FORD</symbol>  
</getQuote>
```

Four elements:
getQuote
symbol
getQuoteResponse
Return

Example of an XML-based response document

```
<getQuoteResponse>  
  <Result>56.5</Result>  
</getQuoteResponse>
```

*Two contain
other elements.
Two contain
simple text content.*

Each XML document can have only one top-most element.

Simple XML Document Exchange with attributes

Example of an XML-based request

Date attribute

```
<getQuote date="02-01-2004">  
  <symbol>FORD</symbol>  
</getQuote>
```

Example of an XML-based response document

Date attribute

```
<getQuoteResponse date="02-01-2004">  
  <Return>56.5</Return>  
</getQuoteResponse>
```

XML Technologies

Shortcut if there is no text value between the opening/closing tags

- If the `symbol` was changed to be an attribute, then you could write either this:

```
<getQuote      symbol="FORD"  
              date="02-01-2004">  
  
</getQuote>
```

- Or use a shortcut for empty elements:

```
<getQuote      symbol="FORD"  
              date="02-01-2004" />
```

XML Namespaces

- Used to avoid element and attribute *name clashes* when creating complex XML documents
- Attributes of the form *xmlns:prefix=identifier* are used to introduce a namespace and establish the prefix
 - Subsequently, names within the namespace must be written as *prefix:name*
- There is also a way of setting up a default namespace so you do not have to write the prefix: *xmlns=identifier*
- As we'll see, the identifier is typically a URL or URI.
- Let's look at some examples . . .

Namespaces – a simple example

- Example of a simple XML request document with a namespace

```
<WSMsg:getQuote  
  xmlns:WSMsg="http://www.WS.com/msg">  
  <WSMsg:symbol>FORD</WSMsg:symbol>  
</WSMsg:getQuote>
```

- Example of a simple XML response document with a namespace

```
<WSMsg:getQuoteResponse  
  xmlns:WSMsg="http://www.WS.com/msg" >  
  <WSMsg:Result>78.9</WSMsg:Result>  
</WSMsg:greetMeResponse>
```

More Complex Namespace Example

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<ENV:Envelope
```

```
  xmlns:ENV="http://schemas.xmlsoap.org/soap/envelope/"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"  
  xmlns:m1="http://IONA.com/HelloWorld">
```

```
  <ENV:Body>
```

```
    <m1:greetMe  >
```

```
      <m1:stringParam0 xsi:type="xsd:string">
```

```
        Hello From WS Client
```

```
      </m1:stringParam0>
```

```
    </m1:greetMe>
```

```
  </ENV:Body>
```

```
</ENV:Envelope>
```

XML Schema – Data Typing for XML documents

- How do we know what elements and attributes to use?
- XML Schema provides the way to define an XML document type
- So, the structure of a document is expressed in an **XML Schema file**
 - The schema is an XML document itself
 - It supplies the definitions of elements and types
- XML tools can check if a given XML document conforms to a given XML Schema
- Resource: <http://www.w3.org/XML/Schema>
 - the primer document is quite readable
(<http://www.w3.org/TR/xmlschema-0/>)

XML Schema Fundamentals

- Define XML **elements** using the built-in and user-defined **types**:
 - **Simple types**
 - Built-in types (e.g. string, numbers, unsigned bytes, characters)
 - Restricted & enumerated types
 - **Complex types** by composing other types using
 - Sequence (an ordered record)
 - All (an unordered record)
 - Choice (selecting only one of several options)
 - Able to specify limits on repetition (e.g., 0 to N, 5 to unlimited ...) using minOccurs, maxOccurs attributes
 - So you can create lists
- Can access element definitions from other Schema files using the **include** element

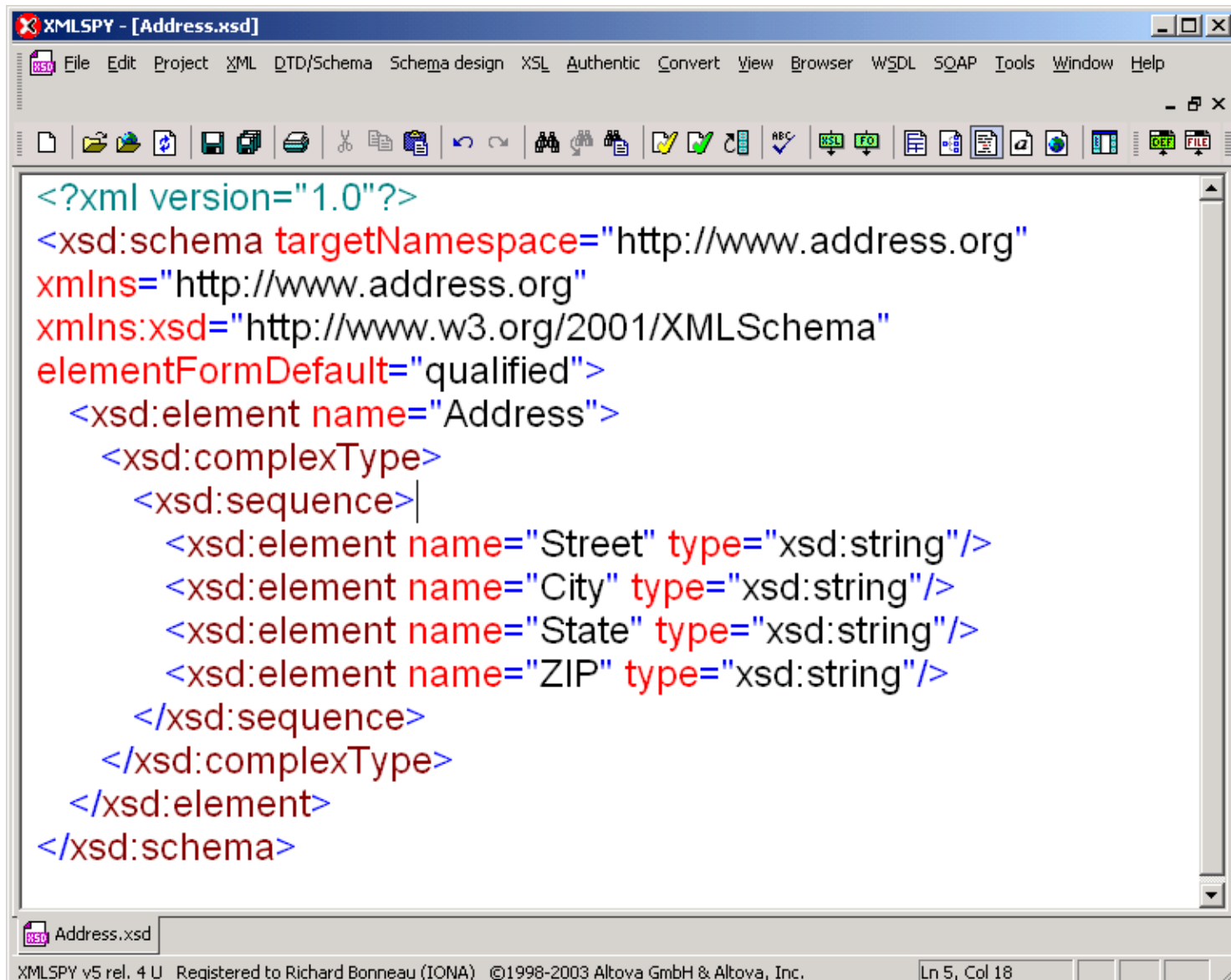
XML Schema - examples

- [Address.xsd](#) – a very simple schema
- [Company.xsd](#) – nested schema – imports two other schemas:
 - [Person.xsd](#) – referenced by Company
 - [Product.xsd](#) – referenced by Company
- [BookStore.xsd](#) – more complex example
 - A nested list of nested structures

Each of these specifies its **target namespace**, using an attribute.
For example:

```
<xsd:schema  
  targetNamespace="http://www.address.org"    ...
```

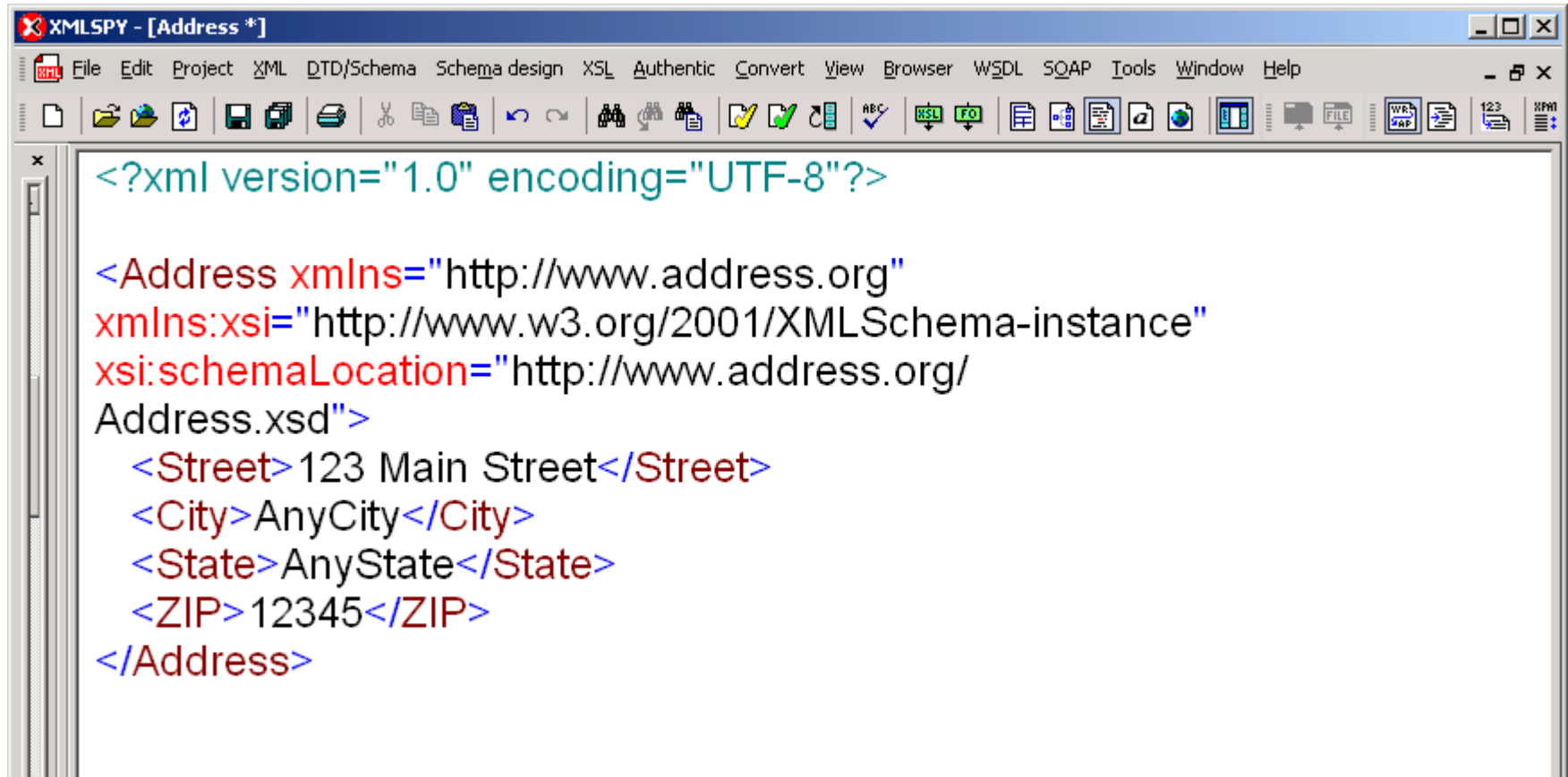
Address.xsd – simple element with several string components



```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://www.address.org"
xmlns="http://www.address.org"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:element name="Address">
    <xsd:complexType>
      <xsd:sequence|
        <xsd:element name="Street" type="xsd:string"/>
        <xsd:element name="City" type="xsd:string"/>
        <xsd:element name="State" type="xsd:string"/>
        <xsd:element name="ZIP" type="xsd:string"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

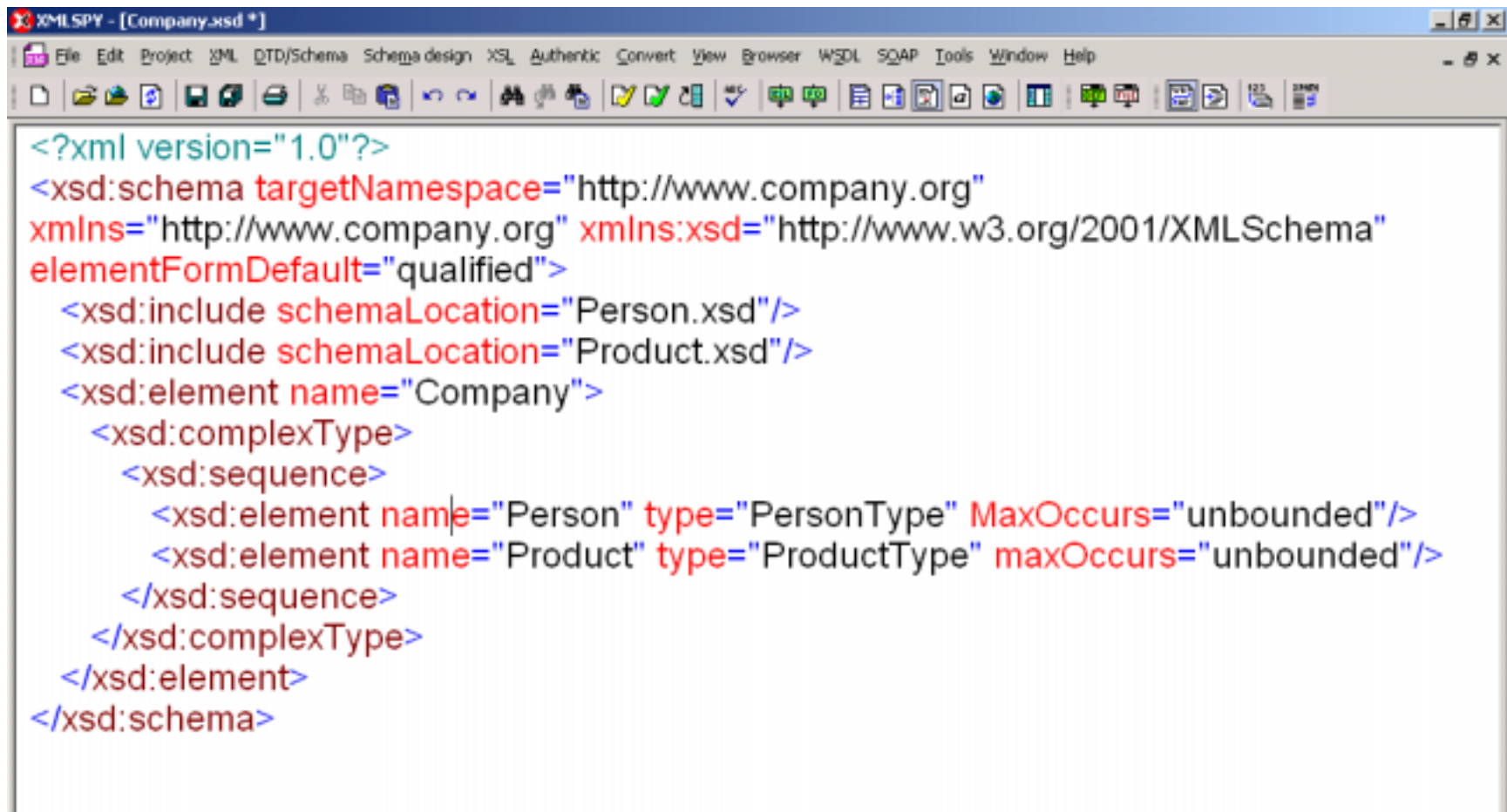
XMLSPY v5 rel. 4 U Registered to Richard Bonneau (IONA) ©1998-2003 Altova GmbH & Altova, Inc. Ln 5, Col 18

Address.xml – consistent with schema



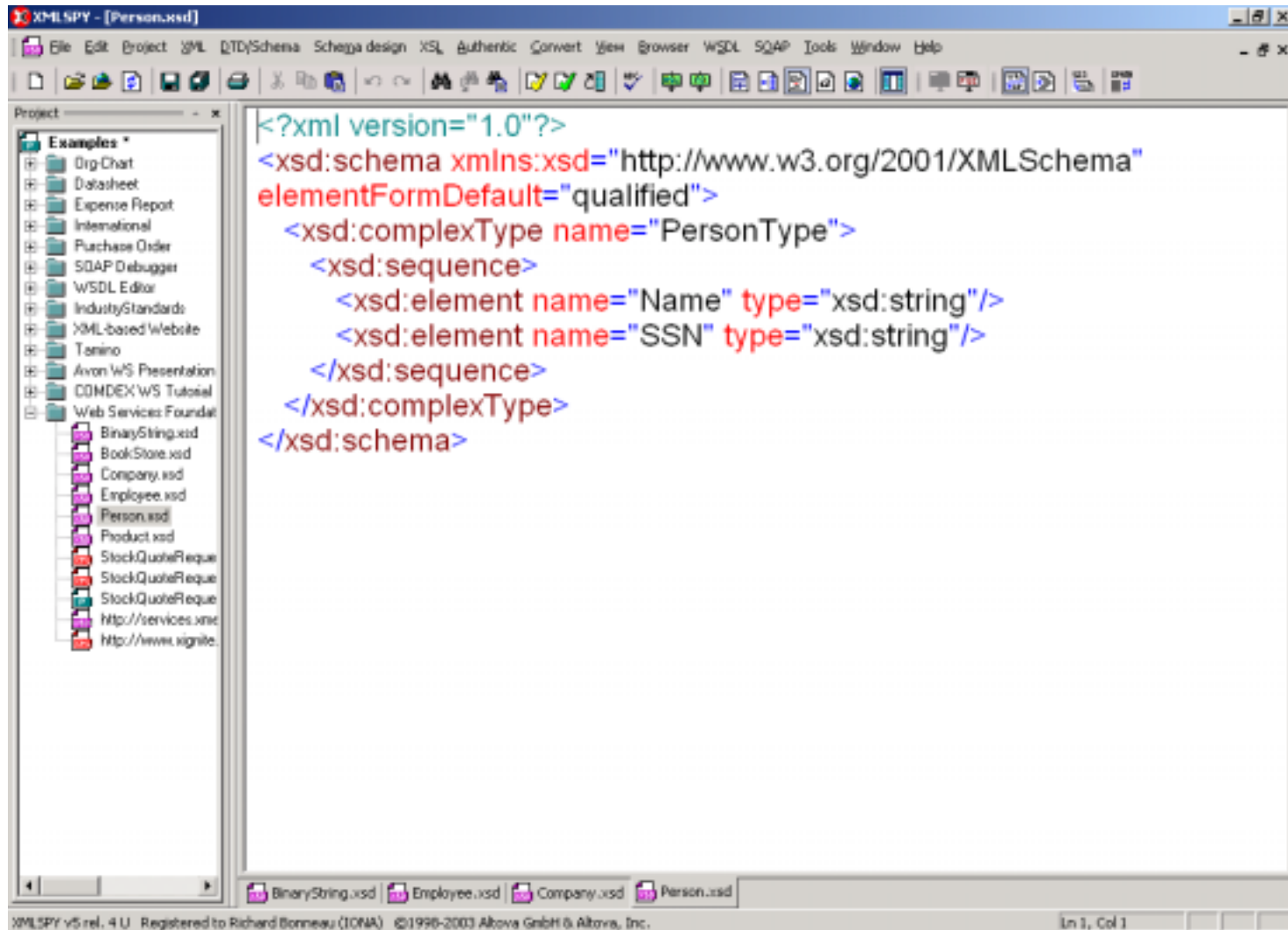
```
<?xml version="1.0" encoding="UTF-8"?>  
  
<Address xmlns="http://www.address.org"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.address.org/  
Address.xsd">  
  <Street>123 Main Street</Street>  
  <City>AnyCity</City>  
  <State>AnyState</State>  
  <ZIP>12345</ZIP>  
</Address>
```

Company.xsd – unlimited number of Persons and Products – types from ‘included’ schema files



```
<?xml version="1.0"?>
<xsd:schema targetNamespace="http://www.company.org"
xmlns="http://www.company.org" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:include schemaLocation="Person.xsd"/>
  <xsd:include schemaLocation="Product.xsd"/>
  <xsd:element name="Company">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Person" type="PersonType" MaxOccurs="unbounded"/>
        <xsd:element name="Product" type="ProductType" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Person.xsd – Persontype elements

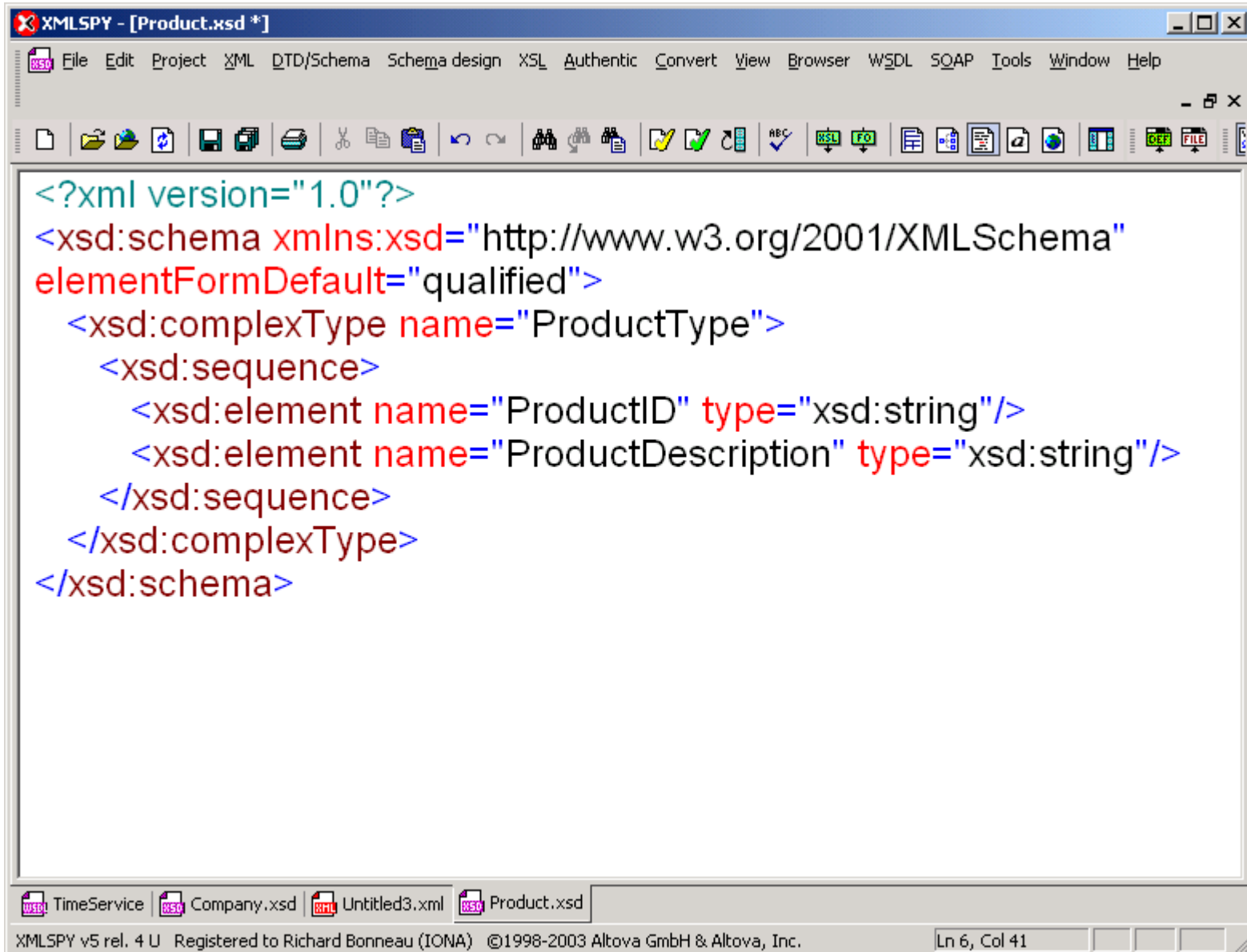


The screenshot displays the XMLSpy v5.0.1.4 interface with the Person.xsd schema file open. The left-hand pane shows a project tree under 'Examples *' with 'Web Services: Foundat' expanded to show 'Person.xsd'. The main editor area contains the following XML Schema Definition (XSD) code:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:complexType name="PersonType">
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string"/>
      <xsd:element name="SSN" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

The status bar at the bottom indicates 'In 1, Col 1' and provides registration information for XMLSpy v5.0.1.4 U. Registered to Richard Borneau (IONA) ©1998-2003 Altova GmbH & Altova, Inc.

Product.xsd – ProductType element

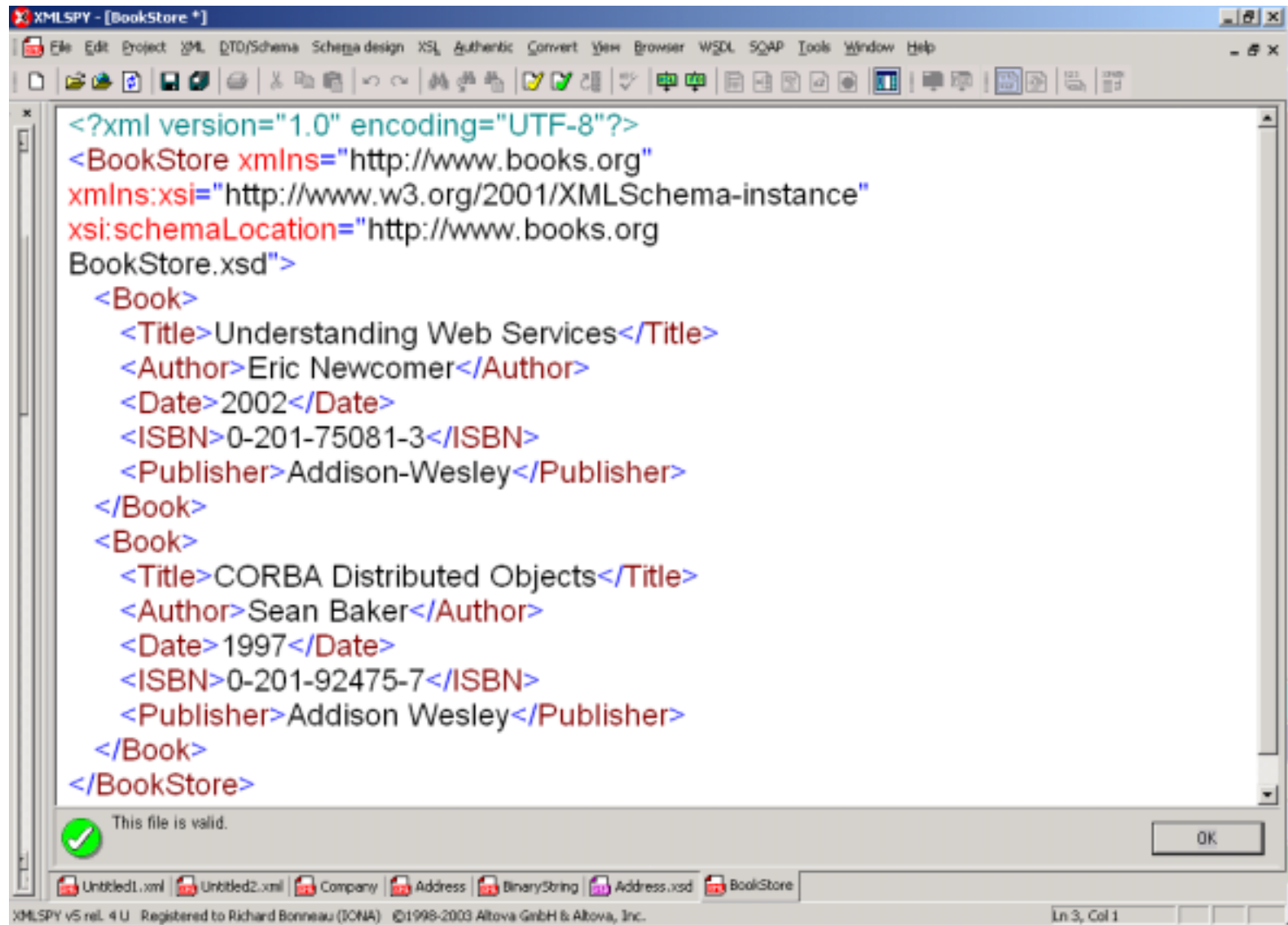


The screenshot shows the XMLSpy application window titled "XMLSPY - [Product.xsd *]". The menu bar includes File, Edit, Project, XML, DTD/Schema, Schema design, XSL, Authentic, Convert, View, Browser, WSDL, SOAP, Tools, Window, and Help. The toolbar contains various icons for file operations and schema management. The main text area displays the following XML Schema Definition (XSD) code:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:complexType name="ProductType">
    <xsd:sequence>
      <xsd:element name="ProductID" type="xsd:string"/>
      <xsd:element name="ProductDescription" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

The taskbar at the bottom shows several open files: TimeService, Company.xsd, Untitled3.xml, and Product.xsd. The status bar at the bottom indicates "XMLSPY v5 rel. 4 U Registered to Richard Bonneau (IONA) ©1998-2003 Altova GmbH & Altova, Inc." and the current cursor position is "Ln 6, Col 41".

Sample Company XML file

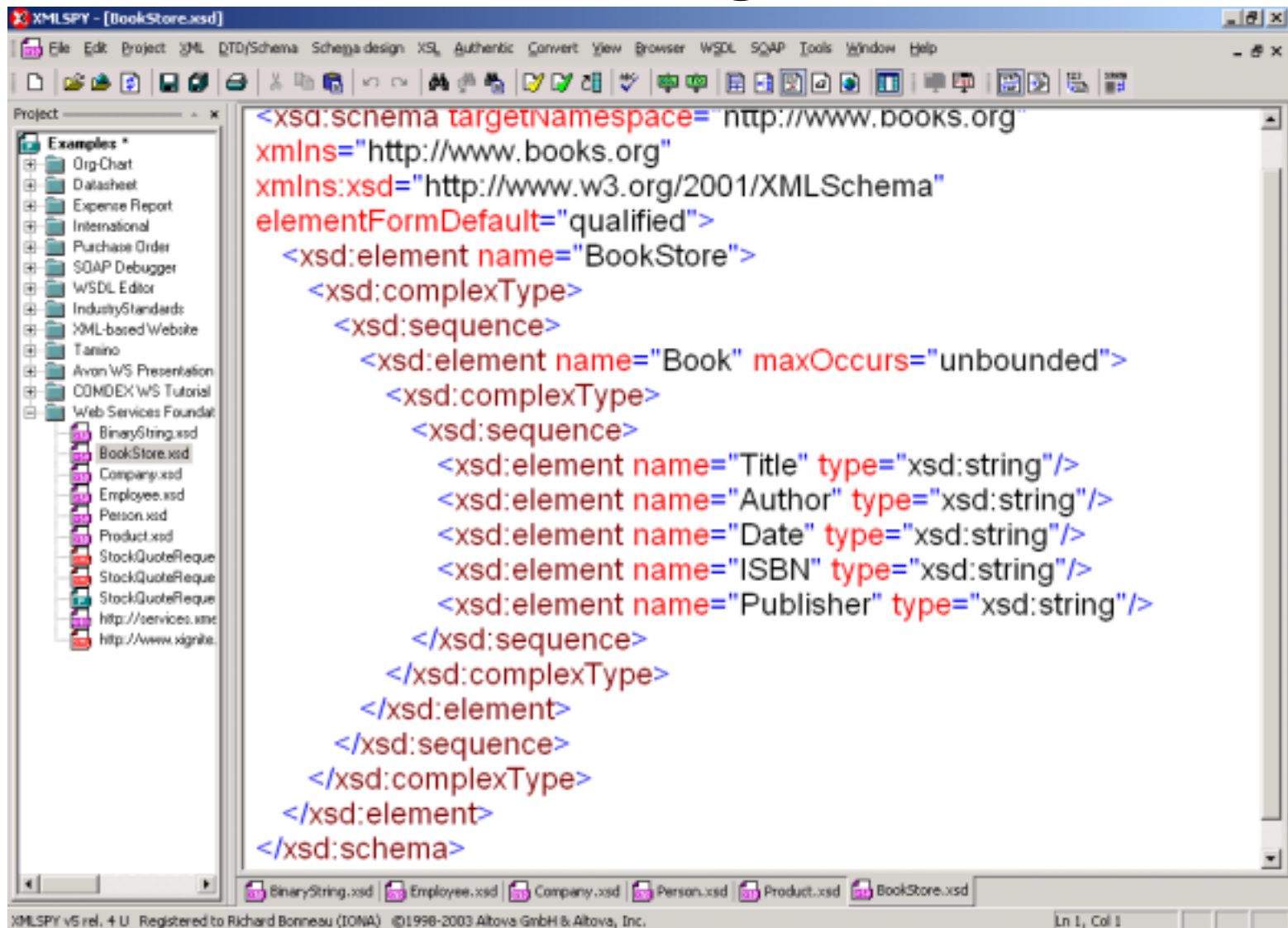


```
<?xml version="1.0" encoding="UTF-8"?>
<BookStore xmlns="http://www.books.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.books.org
BookStore.xsd">
  <Book>
    <Title>Understanding Web Services</Title>
    <Author>Eric Newcomer</Author>
    <Date>2002</Date>
    <ISBN>0-201-75081-3</ISBN>
    <Publisher>Addison-Wesley</Publisher>
  </Book>
  <Book>
    <Title>CORBA Distributed Objects</Title>
    <Author>Sean Baker</Author>
    <Date>1997</Date>
    <ISBN>0-201-92475-7</ISBN>
    <Publisher>Addison Wesley</Publisher>
  </Book>
</BookStore>
```

This file is valid.

XMLSPY v5 rel. 4 U Registered to Richard Bonneau (ICNA) ©1998-2003 Altova GmbH & Altova, Inc. Ln 3, Col 1

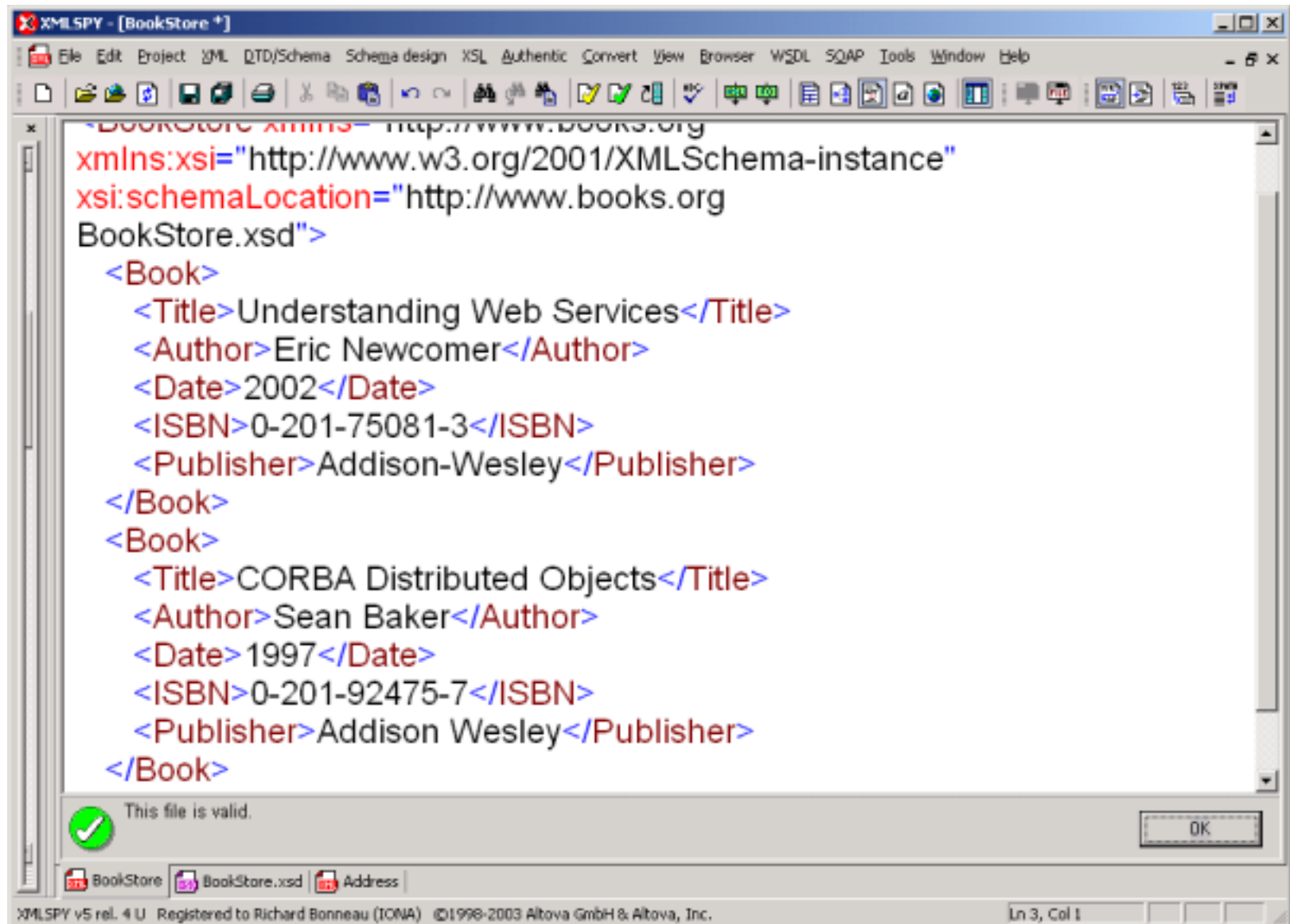
BookStore.xsd – unlimited list of Book structures each containing five fields



```
<xsd:schema targetNamespace="http://www.books.org"
xmlns="http://www.books.org"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:element name="BookStore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Book" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="Title" type="xsd:string"/>
              <xsd:element name="Author" type="xsd:string"/>
              <xsd:element name="Date" type="xsd:string"/>
              <xsd:element name="ISBN" type="xsd:string"/>
              <xsd:element name="Publisher" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XMLSpy v5 rel. 4 U Registered to Richard Bonneu (IONA) ©1998-2003 Altova GmbH & Altova, Inc. Ln 1, Col 1

BookStore.XML



```
XMLSPY - [BookStore +]  
File Edit Project XML QTD/Schema Schema design XSL Authentic Convert View Browser WSDL SOAP Tools Window Help  
BookStore.xml - http://www.books.org  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.books.org  
BookStore.xsd">  
  <Book>  
    <Title>Understanding Web Services</Title>  
    <Author>Eric Newcomer</Author>  
    <Date>2002</Date>  
    <ISBN>0-201-75081-3</ISBN>  
    <Publisher>Addison-Wesley</Publisher>  
  </Book>  
  <Book>  
    <Title>CORBA Distributed Objects</Title>  
    <Author>Sean Baker</Author>  
    <Date>1997</Date>  
    <ISBN>0-201-92475-7</ISBN>  
    <Publisher>Addison Wesley</Publisher>  
  </Book>  
This file is valid. OK  
BookStore BookStore.xsd Address  
XMLSPY v5 rel. 4 U Registered to Richard Bonneau (IONA) ©1998-2003 Altova GmbH & Altova, Inc. Ln 3, Col 1
```

XML Technologies

XSLT for Transforming Data

- eXtensible Stylesheet Language - Transform
- Need ways to take XML data and **transform** it to other forms for various purposes
 - XML isn't always the format that you want to work with
- Examples
 - Transform XML information into HTML format for web page display
 - Transform XML data to conform to another XML Schema
 - Transform between international representations (dates, currency)
 - Transform data into several different presentation formats (browser screen, handheld, pure text ...)
- Actually *three* specs: XSLT (Transform), XML Path Language (XPath) and XSL Formatting Objects (XSL-FO)
- Resource: <http://www.w3.org/Style/XSL/>

XSLT Fundamentals

- Applies transformation rules to elements found in the XML
 - You tell it what elements to find, and what to output when each is found
- It leverages the XPath (XML Path Language) specification
 - A powerful way to identify/find elements within an XML document
- Uses a nested programming-language-like syntax for matching, repetition, etc. to support complex document parsing
- Let's look at an example

XML document containing company info

The screenshot shows the XMLSpy application window titled "XMLSPY - [OrgChart]". The interface includes a menu bar (File, Edit, Project, XML, DTD/Schemas, Schema design, XSL, Authentic, Convert, View, Browser, WSDL, SOAP, Tools, Window, Help) and a toolbar. On the left, a "Project" pane displays a tree view of files, with "OrgChart.xml" selected. The main window displays the XML content, which is a mix of plain text and XML tags. The text describes a company's history and provides contact details. The XML tags are highlighted in red, and the plain text is in black. The status bar at the bottom indicates "Ln 1, Col 1" and provides copyright information for XMLSpy v5 rel. 4 U.

```

involved in developing nanoelectronic software technologies
since 1996 and released the first version of its products in
February 1999.</para>
  <para>
    <strong>Due to the fact </strong>that nanoelectronic
software components are so small that <strong>nobody can see
</strong>it the company is not well known to the public.</para>
  </Desc>
  <Address>
    <ipo:street> 119 Oakstreet, Suite 4876</ipo:street>
    <ipo:city>Vereno</ipo:city>
    <ipo:state>DC</ipo:state>
    <ipo:zip>29213</ipo:zip>
  </Address>
  <Phone>+1 (321) 555 5155</Phone>
  <Fax>+1 (321) 555 5155 - 9</Fax>
  <EMail>office@nanonull.com</EMail>
  <WebStore>>true</WebStore>
  <CustomerSupport>>true</CustomerSupport>
  <Department>
    <Name>Administration</Name>
  <Person>

```

XMLSPY v5 rel. 4 U Registered to Richard Bonneau (DONA) ©1998-2003 Altova GmbH & Altova, Inc. Ln 1, Col 1

(Part of) XSLT for transforming data

The screenshot shows the XMLSpy interface with the following XSLT code:

```
<!--XSL Stylesheet for generating simple Orgchart-->
<xsl:template match="a:OrgChart">
  <html>
    <head>
      <title>
        <xsl:value-of select="a:Name"/>
      </title>
    </head>
    <body>
      <table width="100%">
        <tr>
          <td>
            <h1>
              <xsl:value-of select="a:Name"/>
            </h1>
          </td>
          <td align="right">
            <img alt="Logo">
              <xsl:attribute name="src">
                <xsl:value-of select="a:CompanyLogo/@href" />
              </xsl:attribute>
            </td>
          </tr>
        </table>
      </body>
    </html>
  </template>
</xsl:stylesheet>
```

Three yellow arrows point to the following elements in the code:

- The `match="a:OrgChart"` attribute in the `<xsl:template>` tag.
- The `select="a:Name"` attribute in the `<xsl:value-of>` tag inside the `<title>` element.
- The `name="src"` attribute in the `<xsl:attribute>` tag inside the `` element.

At the bottom of the window, the status bar reads: XMLSPY v5 rel. 4 U Registered to Richard Bonneau (IONA) ©1998-2003 Altova GmbH & Altova, Inc. Ln 8, Col 28

Transformed Data in HTML format ...

The screenshot shows the XMLSpy application window titled "XMLSPY - [OrgChart]". The main content area displays the following information:

Organization Chart

Nanonull, Inc.

119 Oakstreet, Suite 4876
Vereno DC 29213

Phone: +1 (321) 555 5155
Fax: +1 (321) 555 5155 - 9
E-Mail: office@nanonull.com

The company was established in Vereno, in 1995 and is privately held. Nanonull has been actively involved in developing nanoelectronic software technologies since 1996 and released the first version of its products in February 1999.

Due to the fact that nanoelectronic software components are so small that nobody can see it the company is not well known to the public.

Administration

First	Last	Title	Ext	E-Mail
Vernon	Callaby	Office Manager	582	v.callaby@nanonull.com
Frank	Further	Accounts Receivable	471	f.further@nanonull.com
Loby	Matise	Accounting Manager	963	l.matise@nanonull.com

Marketing

First	Last	Title	Ext	E-Mail
-------	------	-------	-----	--------

The project tree on the left shows a list of files including XSD and XML files, with "OrgChart.xml" selected.

Parsing XML documents

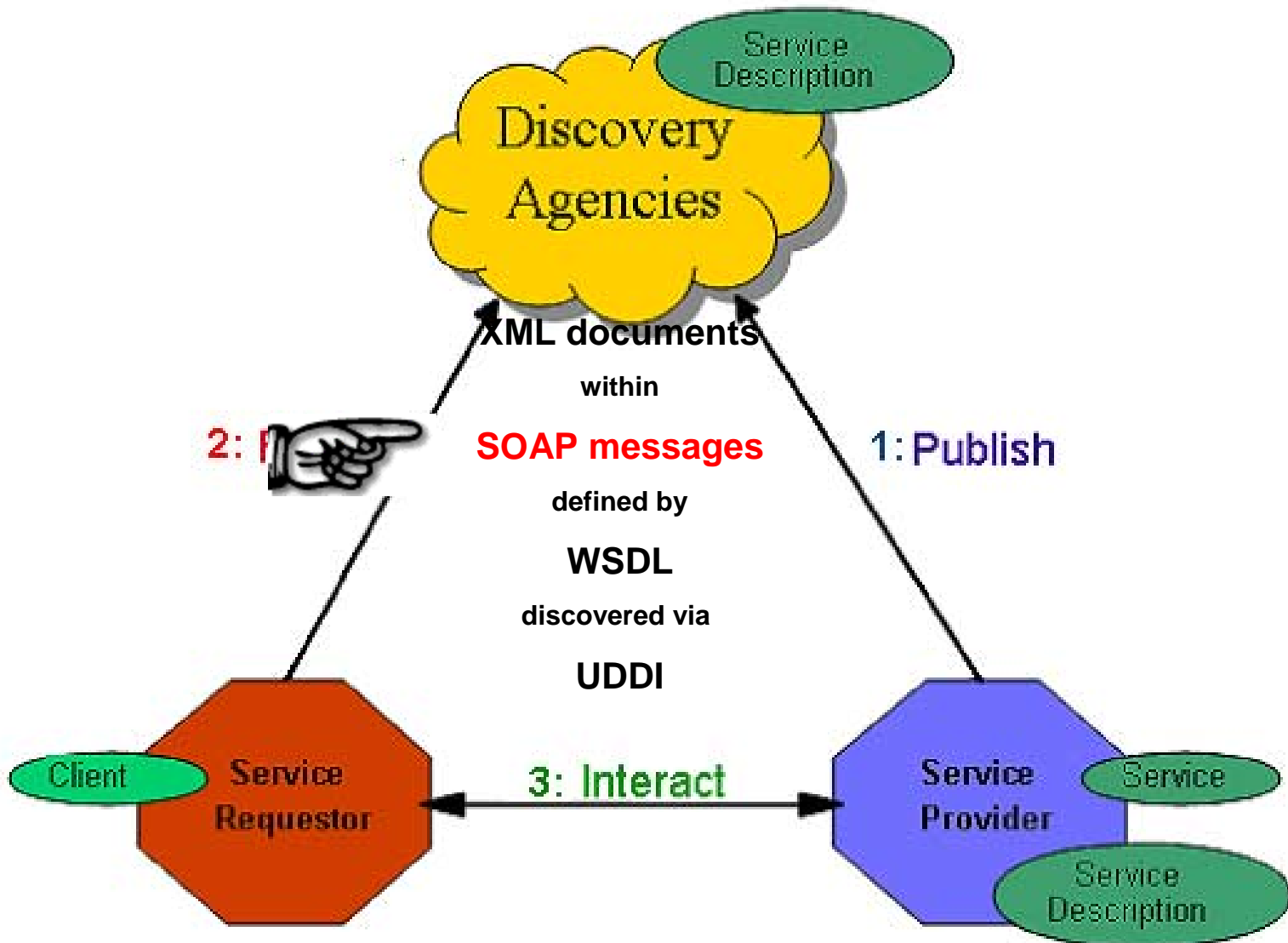
- Processing XML documents within programs typically requires *XML parsing* capabilities
- Two most popular parsing approaches:
 - **DOM – Document Object Model**
 - Read and represent entire document in memory as a tree of nodes allowing for easy traversal throughout the document
 - You can modify the tree, and make a call to DOM to tell it to output the new tree as a document
 - **SAX – Simple API for XML**
 - Event driven parsing model that invokes callbacks for major XML elements as they are read from the document
 - Can be more efficient than DOM for large document processing
 - Used more for extracting sub-document information and specific processing

Other XML Specifications

- **XPATH**
 - XML-based syntax to provide sophisticated access to parts of an XML document – used heavily with XSLT for transforming XML
- **XQUERY**
 - An SQL-like dialect for querying parts of varying XML data sources
 - Extension to XPATH
- **XML-Encryption** – (to be seen later in more detail)
 - Process for encrypting/decrypting parts of XML docs
- **XML-Signature** – (to be seen later in more detail)
 - To ensure origin and integrity of XML docs

Participants in a Web Services Environment.

Part 2 - SOAP Technology



SOAP Technology

- Originally, the acronym stood for

Simple Object Access Protocol

– but not really a valid acronym any more.

- Much more like

“How to exchange XML documents programmatically over the Internet using messages”

- Also known as “e-mail for XML documents”
- Several versions of standard available or in progress
 - Latest version: SOAP 1.2. Most widely used version: SOAP 1.1.
- Resource: <http://www.w3.org/TR/SOAP/>

SOAP Overview

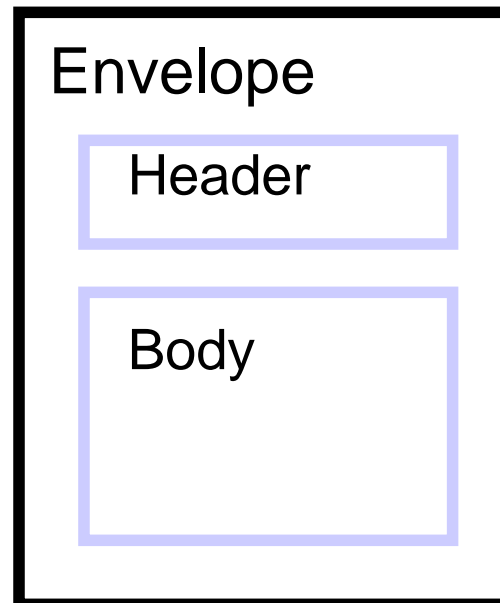
What Is SOAP now?

- SOAP is
 - an XML-based messaging framework specifically designed for exchanging formatted data across the internet
- Latest version: SOAP 1.2
- Examples of usage
 - (RPC Model) Using request and reply messages
 - (Document Model) Sending entire documents between correspondents.
- Resource: <http://www.w3.org/TR/SOAP/>

SOAP Overview

Structure of a SOAP Message

- A SOAP message consists of:
 - **Envelope** - identifies the message boundary and includes:
 - **Header** - contains meta-data/auxiliary (optional)
 - **Body** - contains the request and response XML documents



- Fault information may also be supplied – not shown here
- Often the header contains system level data (see later chapters)

SOAP Overview

Skeleton SOAP Message in XML

```
<?xml version="1.0"?>  
<soap:Envelope xmlns:soap=" ... ">  
  <soap:Header>  
    ... ..  
  </soap:Header>  
  <soap:Body>  
    ... ..  
    < ... message payload goes here ... >  
  </soap:Body>  
</soap:Envelope>
```

SOAP Overview

Our Simple Example – in SOAP

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<ENV:Envelope
```

```
  xmlns:ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:m1="http://IONA.com/HelloWorld">
```

```
  <ENV:Body>
```

```
    <m1:greetMe >
```

```
      <stringParam0 xsi:type="xsd:string">
```

```
        Hello From WS Client
```

```
      </stringParam0>
```

```
    </m1:greetMe>
```

```
  </ENV:Body>
```

```
</ENV:Envelope>
```

SOAP Overview

Simple Example Response – in SOAP

```
<?xml version='1.0' encoding='utf-8'?>
```

```
<ENV:Envelope
```

```
  xmlns:ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```
  xmlns:ENC="http://schemas.xmlsoap.org/soap/encoding/"
```

```
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:m1="http://IONA.com/HelloWorld">
```

```
    <ENV:Body>
```

```
      <m1:greetMeResponse>
```

```
        <return xsi:type="xsd:string">
```

```
          Echo: Hello From WS Client
```

```
        </return>
```

```
      </m1:greetMeResponse>
```

```
    </ENV:Body>
```

```
</ENV:Envelope>
```

SOAP Overview

SOAP Processing

- Within the SOAP environment, the software responsible for the generation, transmission, reception and analysis of these messages is known as a **SOAP Processor**
- These processors can be standalone listeners on TCP ports
 - Accepting incoming SOAP messages and passing them up in the stack
- Also, most Web Servers include a SOAP processor
 - but if your Web Server doesn't include a SOAP processor, then most Web application development environments allow you to add a "SOAP plug-in".

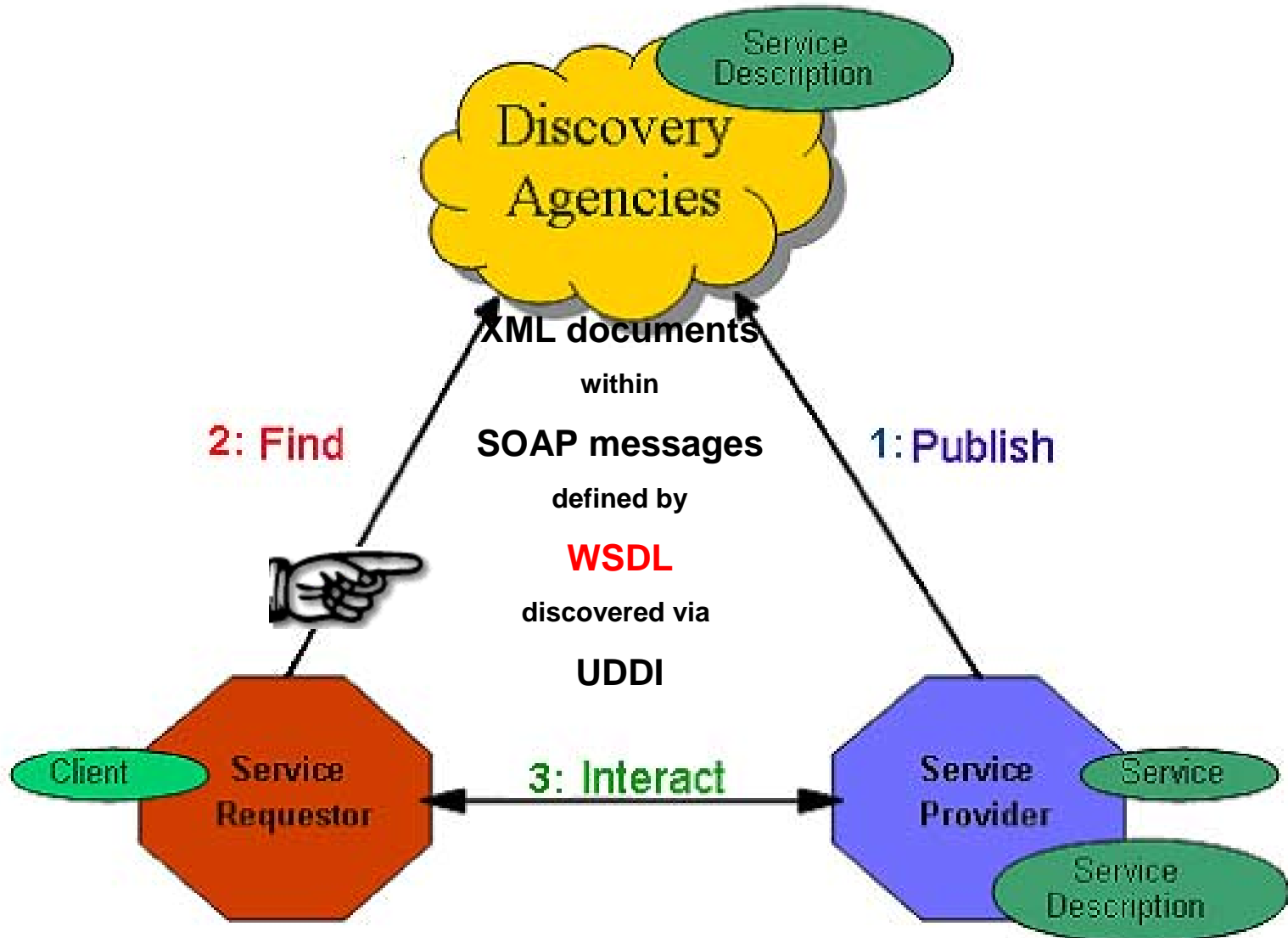
SOAP Overview

SOAP 1.2

- SOAP 1.1 is the most used version of SOAP specification
 - Recommended by WS-I in its Basic Profile 1.0
- The latest release is SOAP 1.2
 - Incorporates SOAP with attachments (controversial)
 - Allows for additional information payload as an attached file – that may contain binary as well as text data
 - Now being incorporated into existing SOAP implementations
 - Other web service standards and interop efforts also starting to build on top of SOAP to allow for support of attachments
 - Other definitions being put forth for consideration

Participants in a Web Services Environment:

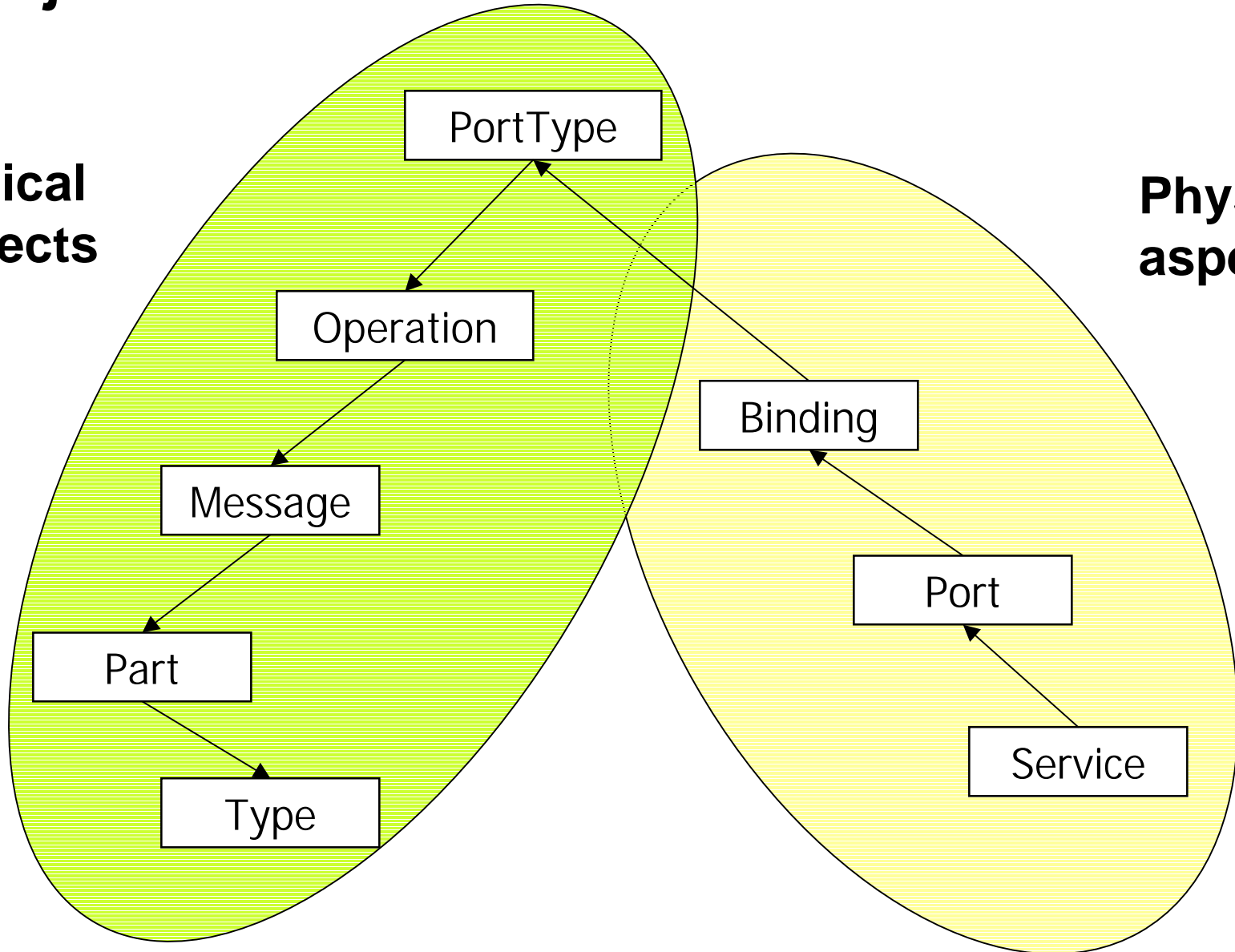
Part 3 - WSDL Technology



WSDL Technology

- **Web Services Description Language**
- WSDL is the KEY to using Web Services
 - WSDL provides an OS-, platform- language-neutral definition of a Web Service
- THE **contract** between Web Service provider and consumer
- Completely describes a Web Service:
 - Messages and types of data used in messages
 - Operations with associated in and out messages
 - Bindings of operations to transports
 - Physical location of service (endpoint) specifications
- Large and complex standard, continually undergoing development and proposed extensions
- Reference: <http://www.w3.org/TR/wsdl>

WSDL Overview

Major WSDL Elements**Logical aspects****Physical aspects**

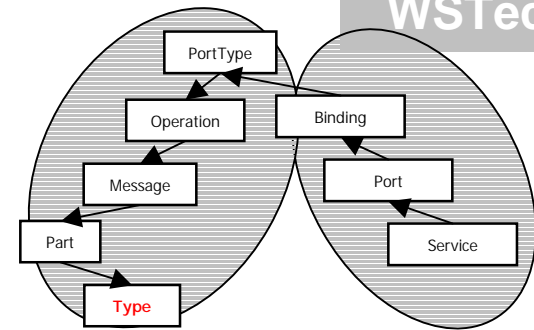
Major WSDL Elements

- The top level XML element is the **definitions** element
`<definitions> . . . </definitions>`
- Services are defined using eight major elements:
 - **types**: data type definitions used to describe the messages exchanged.
 - **message**: abstract definition of the data being transmitted in a message. A message consists of logical **parts**, each of which has one of the defined types.
 - **operation**: Each operation can have an input message and an output message.
 - **porttype**: a set of operations.

 - **binding**: specifies concrete protocol and data formats for the operations and messages defined by a particular portType.
 - **port**: specifies an address for a binding, thus defining a single communication endpoint.
 - **service**: used to aggregate a set of related ports.

WSDL Overview

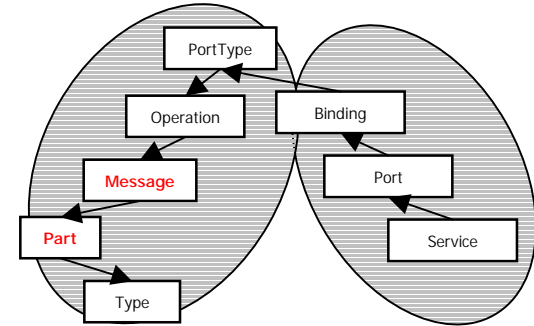
The `<types>` element



- Types are specialized building blocks for data content exchanged via Web Services
- Normally the `<types>` element uses XML Schema to define types and elements used in the WSDL document
 - But another type system could be used (this isn't very well defined now)
- So normally specified in terms of XML Schema constructs
 - Simple types
 - Complex types – including sequences and nested structures
 - E.g. a purchase order with multiple items
- We'll see examples of these later.

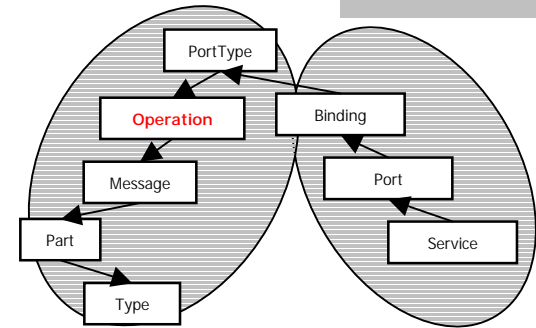
WSDL Overview

The `<message>` element



- Messages are the packaging of data that are transferred (in or out) via web service operations
- A message can contain zero or more *parts*
 - And each part can contain one element or one type

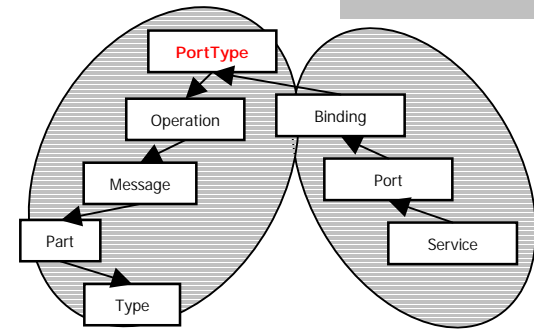
WSDL Overview

The *<operation>* element

- An operation can have an input message and an output message

WSDL Overview

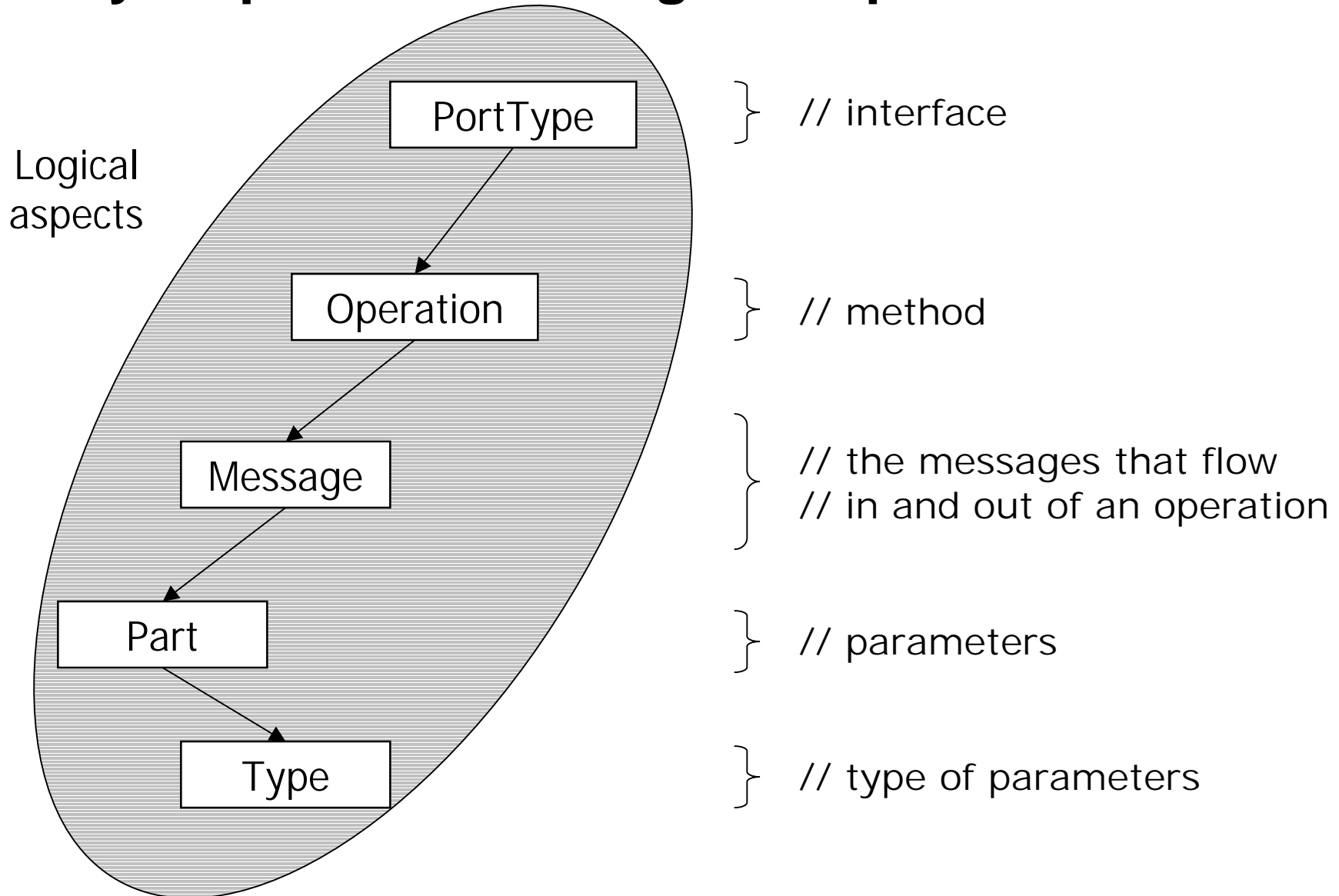
The `<porttype>` element



- A `<porttype>` is a collection of named operations
 - each with an input and/or an output message
- This is the key to the logical definition of a web service
 - What operations are available for access
 - What input message is required for each operation
 - What output message is generated by each operation

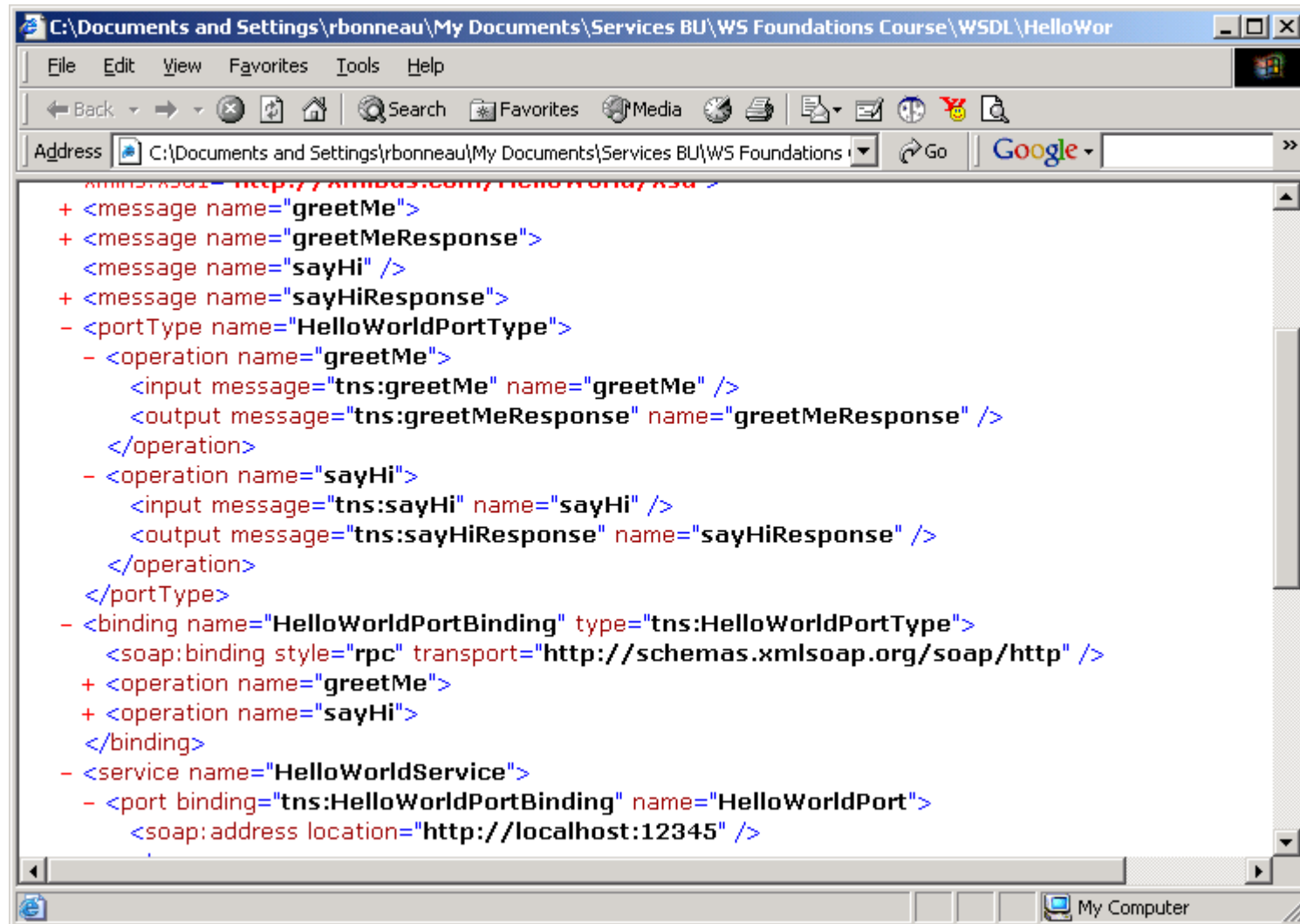
WSDL Overview

It may help to view the logical aspects as follows:



WSDL Overview

Let's look at our simple example,
within a WSDL file ... [HelloWorld.wsdl](#)

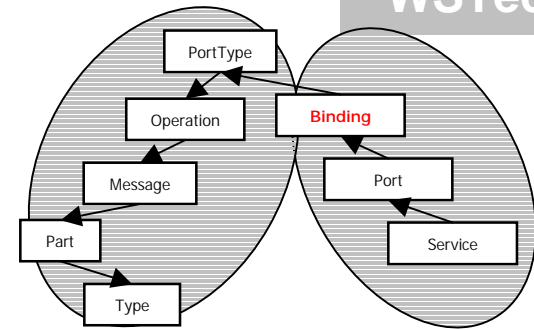


```

+ <message name="greetMe">
+ <message name="greetMeResponse">
  <message name="sayHi" />
+ <message name="sayHiResponse">
- <portType name="HelloWorldPortType">
  - <operation name="greetMe">
    <input message="tns:greetMe" name="greetMe" />
    <output message="tns:greetMeResponse" name="greetMeResponse" />
  </operation>
  - <operation name="sayHi">
    <input message="tns:sayHi" name="sayHi" />
    <output message="tns:sayHiResponse" name="sayHiResponse" />
  </operation>
</portType>
- <binding name="HelloWorldPortBinding" type="tns:HelloWorldPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  + <operation name="greetMe">
  + <operation name="sayHi">
</binding>
- <service name="HelloWorldService">
  - <port binding="tns:HelloWorldPortBinding" name="HelloWorldPort">
    <soap:address location="http://localhost:12345" />

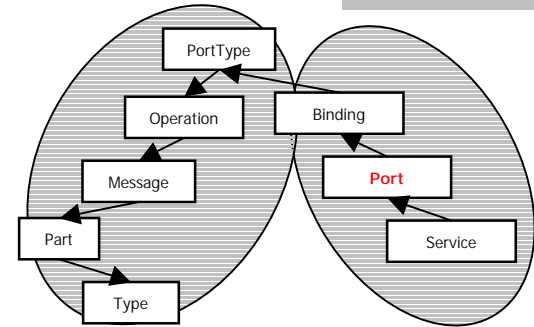
```

WSDL Overview

The *<binding>* element

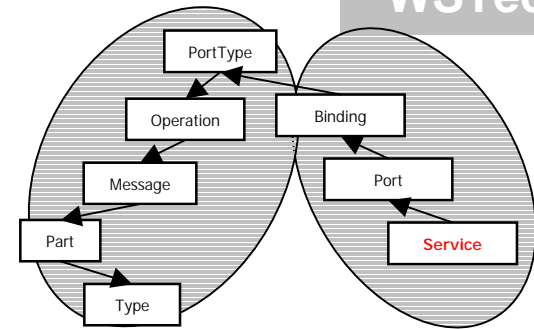
- The `<binding>` element specifies how the logical content (messages, operations) is mapped onto transports for exchange between requestor and provider
- HTTP + SOAP are the predominant binding options, but many more are becoming available to support integration throughout and among enterprises
- Note that multiple bindings for the same PortType are possible!

WSDL Overview

The `<port>` element

- The `<port>` element specifies the physical location/destination for sending the service requests
- Typically supplied as a URL

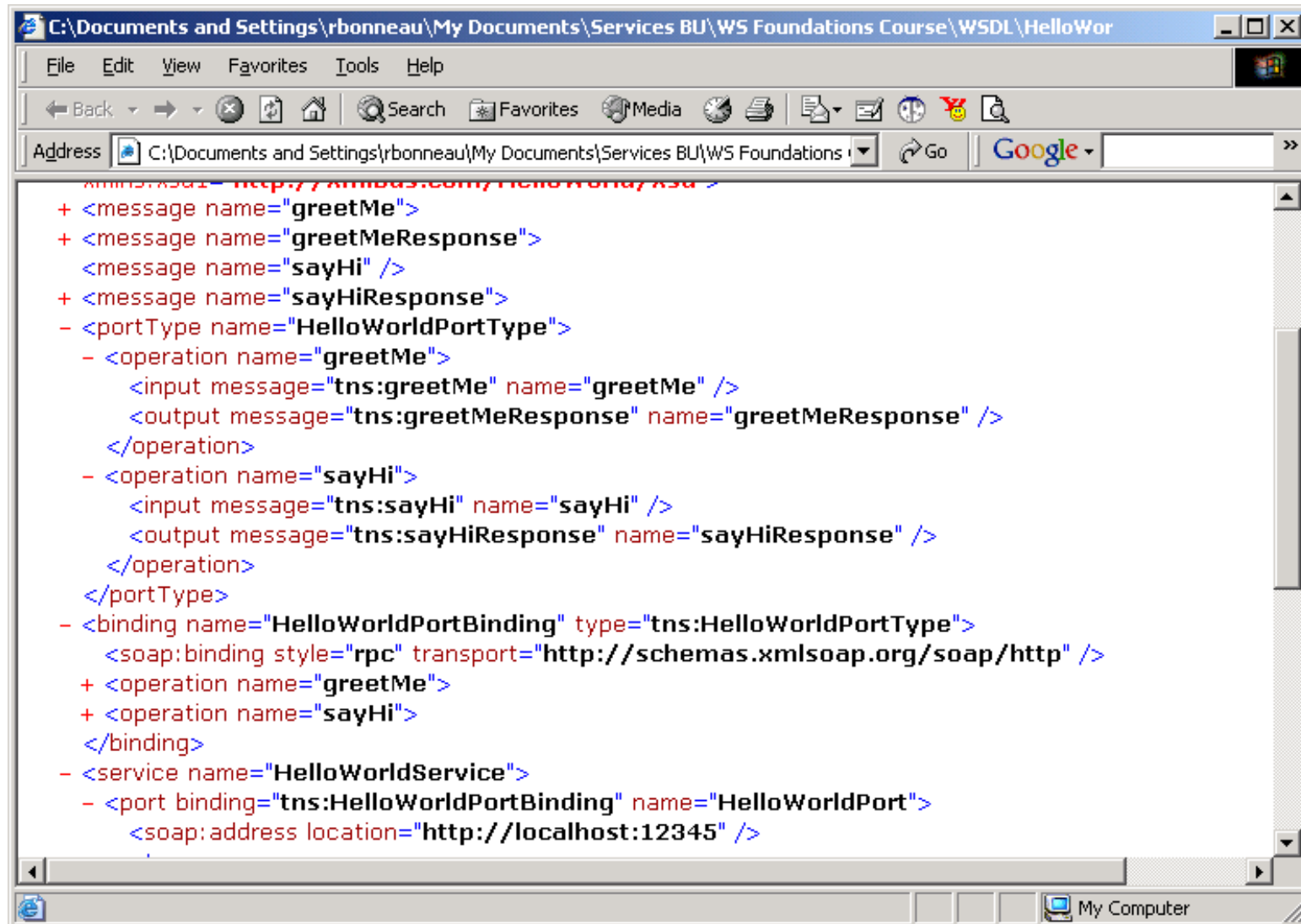
WSDL Overview

The <service> element

- The <service> element specifies a collection of ports – I.e. locations where the operations of the service are accessible to potential service requestors
- Typically, the ports of a service refer to the different locations where the different possible bindings specified elsewhere in the WSDL file are implemented and available
- Example: one port may supply the location where HTTP is the transport binding, another port may supply the location where it is FTP
- Let's look at the rest of our simple example within a WSDL file ... [HelloWorld.wsdl](#)

WSDL Overview

HelloWorld.WSDL



```
+ <message name="greetMe">
+ <message name="greetMeResponse">
  <message name="sayHi" />
+ <message name="sayHiResponse">
- <portType name="HelloWorldPortType">
  - <operation name="greetMe">
    <input message="tns:greetMe" name="greetMe" />
    <output message="tns:greetMeResponse" name="greetMeResponse" />
  </operation>
  - <operation name="sayHi">
    <input message="tns:sayHi" name="sayHi" />
    <output message="tns:sayHiResponse" name="sayHiResponse" />
  </operation>
</portType>
- <binding name="HelloWorldPortBinding" type="tns:HelloWorldPortType">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  + <operation name="greetMe">
  + <operation name="sayHi">
</binding>
- <service name="HelloWorldService">
  - <port binding="tns:HelloWorldPortBinding" name="HelloWorldPort">
    <soap:address location="http://localhost:12345" />
```

WSDL Overview

A Few Concrete WSDL Examples

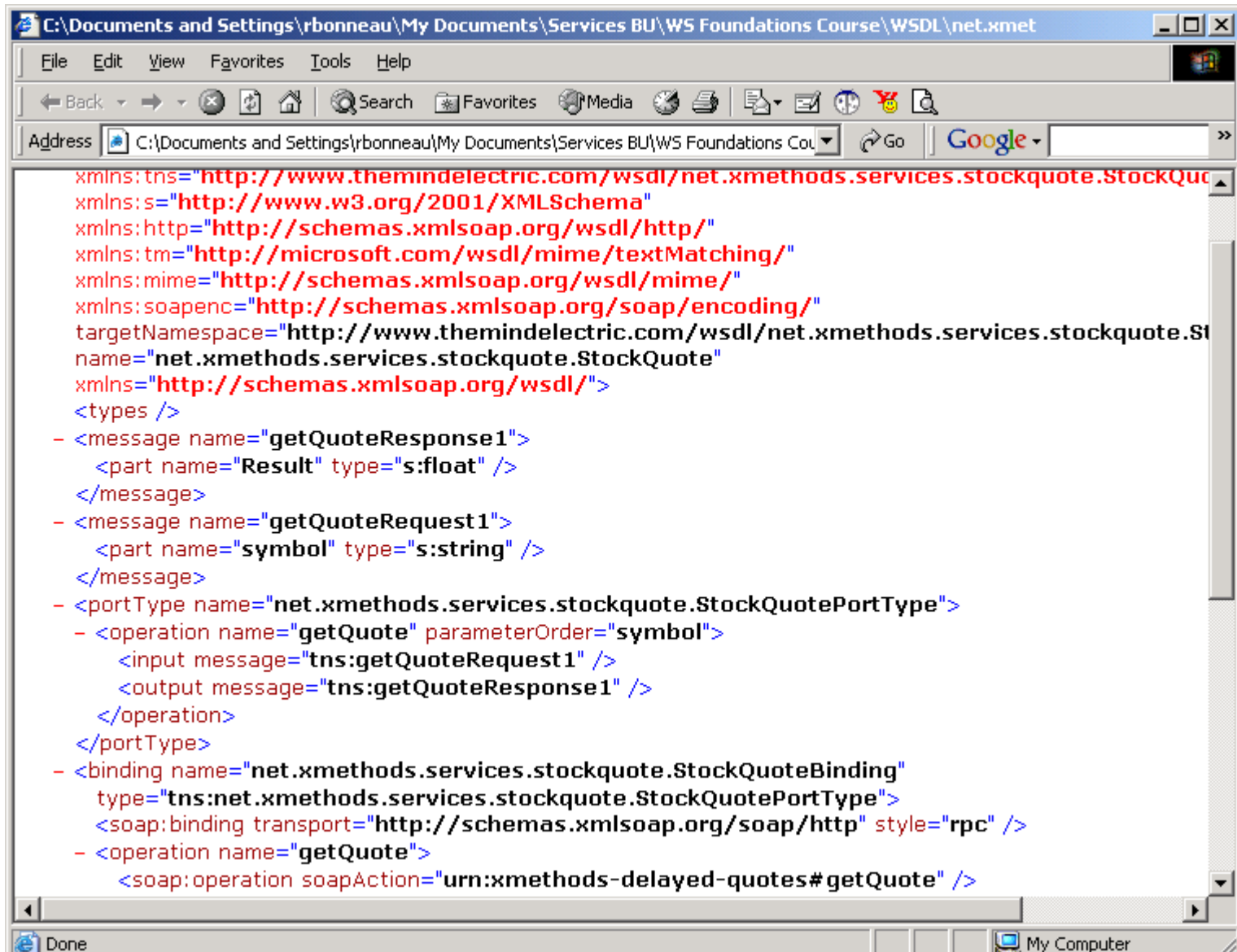
- [Stock Quote WSDL](#) – remotely stored WSDL file at Xmethods.net
 - Simple data model and operations list used for our early demo
- WS-I Manufacturer WSDL file ([Warehouse.wsdl](#))
 - Imported schemas, shared among different WSDL files
 - Sample request and response SOAP messages documented in the file
 - Documentation discipline – commentary, headers, etc.

More Concrete WSDL Examples

- [Stock Quote WSDL](#) – remotely stored WSDL file at Xmethods.net
 - Simple data model used for our early demo
- A [SearchService WSDL](#) file
 - More complex data (types section)
 - More operations
 - Optional transports/bindings/services/ports (some in comments)
- One of the Web Services Interoperability Demo WSDL files
 - Another remotely available WSDL file
 - [Isupplier](#) – complex types and numerous operations
- WS-I Manufacturer WSDL file ([Warehouse.wsdl](#))
 - Imported schemas, shared among different WSDL files
 - Sample request and response SOAP messages documented in the file
 - Good documentation discipline – commentary, headers, etc.

WSDL Overview

Stock Quote WSDL File

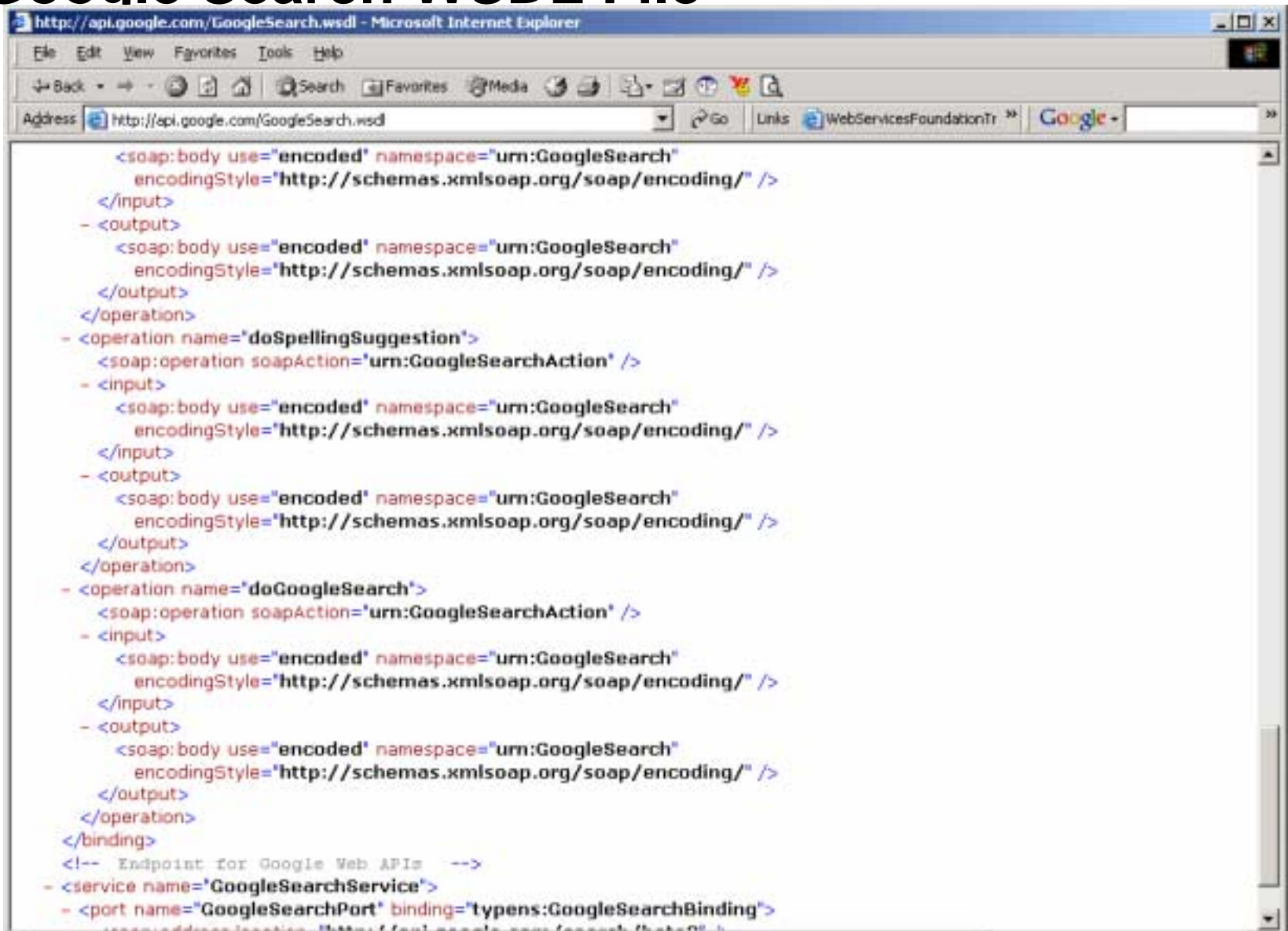


The screenshot shows a web browser window with the address bar containing the file path: C:\Documents and Settings\rbonneau\My Documents\Services BU\WS Foundations Course\WSDL\net.xmet. The browser displays the XML content of the WSDL file, which defines a service for getting stock quotes. The XML includes namespace declarations, a port type, a binding, and a message structure.

```
xmlns:tns="http://www.theminelectric.com/wsd/net.xmethods.services.stockquote.StockQuote"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:http="http://schemas.xmlsoap.org/wsd/http/"
xmlns:tm="http://microsoft.com/wsd/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsd/mime/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
targetNamespace="http://www.theminelectric.com/wsd/net.xmethods.services.stockquote.St
name="net.xmethods.services.stockquote.StockQuote"
xmlns="http://schemas.xmlsoap.org/wsd/">
<types />
- <message name="getQuoteResponse1">
  <part name="Result" type="s:float" />
</message>
- <message name="getQuoteRequest1">
  <part name="symbol" type="s:string" />
</message>
- <portType name="net.xmethods.services.stockquote.StockQuotePortType">
  - <operation name="getQuote" parameterOrder="symbol">
    <input message="tns:getQuoteRequest1" />
    <output message="tns:getQuoteResponse1" />
  </operation>
</portType>
- <binding name="net.xmethods.services.stockquote.StockQuoteBinding"
  type="tns:net.xmethods.services.stockquote.StockQuotePortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
  - <operation name="getQuote">
    <soap:operation soapAction="urn:xmethods-delayed-quotes#getQuote" />
```

WSDL Overview

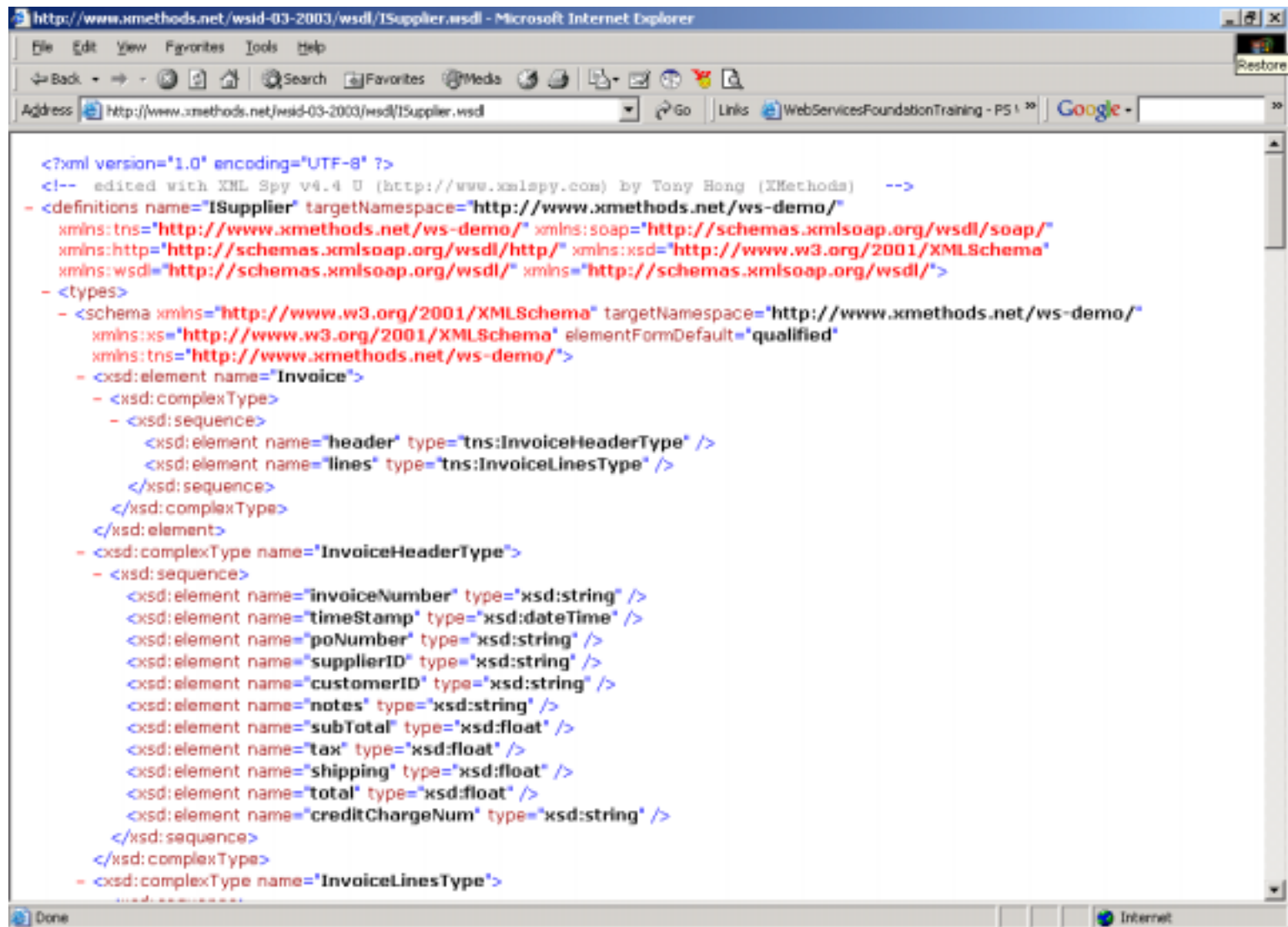
Google Search WSDL File



```
http://api.google.com/GoogleSearch.wsdl - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Search Favorites Media
Address http://api.google.com/GoogleSearch.wsdl
<soap:body use="encoded" namespace="urn:GoogleSearch"
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</input>
- <output>
  <soap:body use="encoded" namespace="urn:GoogleSearch"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
  </output>
</operation>
- <operation name="doSpellingSuggestion">
  <soap:operation soapAction="urn:GoogleSearchAction" />
  - <input>
    <soap:body use="encoded" namespace="urn:GoogleSearch"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
    <soap:body use="encoded" namespace="urn:GoogleSearch"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
- <operation name="doGoogleSearch">
  <soap:operation soapAction="urn:GoogleSearchAction" />
  - <input>
    <soap:body use="encoded" namespace="urn:GoogleSearch"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
  - <output>
    <soap:body use="encoded" namespace="urn:GoogleSearch"
      encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
<!-- Endpoint for Google Web APIs -->
- <service name="GoogleSearchService">
  - <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
```

WSDL Overview

WSID Supplier WSDL



The screenshot shows a Microsoft Internet Explorer browser window with the address bar displaying `http://www.xmethods.net/wsdl-03-2003/wsdl/ISupplier.wsdl`. The main content area displays the XML WSDL document for the 'ISupplier' service. The XML is color-coded and includes the following structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- edited with XML Spy v4.4 U (http://www.xmlspy.com) by Tony Hong (XMethods) -->
- <definitions name="ISupplier" targetNamespace="http://www.xmethods.net/ws-demo/"
  xmlns:tns="http://www.xmethods.net/ws-demo/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsd="http://schemas.xmlsoap.org/wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/">
- <types>
- <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.xmethods.net/ws-demo/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  xmlns:tns="http://www.xmethods.net/ws-demo/">
- <xsd:element name="Invoice">
- <xsd:complexType>
- <xsd:sequence>
  <xsd:element name="header" type="tns:InvoiceHeaderType" />
  <xsd:element name="lines" type="tns:InvoiceLinesType" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
- <xsd:complexType name="InvoiceHeaderType">
- <xsd:sequence>
  <xsd:element name="invoiceNumber" type="xsd:string" />
  <xsd:element name="timeStamp" type="xsd:dateTime" />
  <xsd:element name="poNumber" type="xsd:string" />
  <xsd:element name="supplierID" type="xsd:string" />
  <xsd:element name="customerID" type="xsd:string" />
  <xsd:element name="notes" type="xsd:string" />
  <xsd:element name="subTotal" type="xsd:float" />
  <xsd:element name="tax" type="xsd:float" />
  <xsd:element name="shipping" type="xsd:float" />
  <xsd:element name="total" type="xsd:float" />
  <xsd:element name="creditChargeNum" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
- <xsd:complexType name="InvoiceLinesType">
```

WSDL Overview

WS-I Warehouse WSDL

```

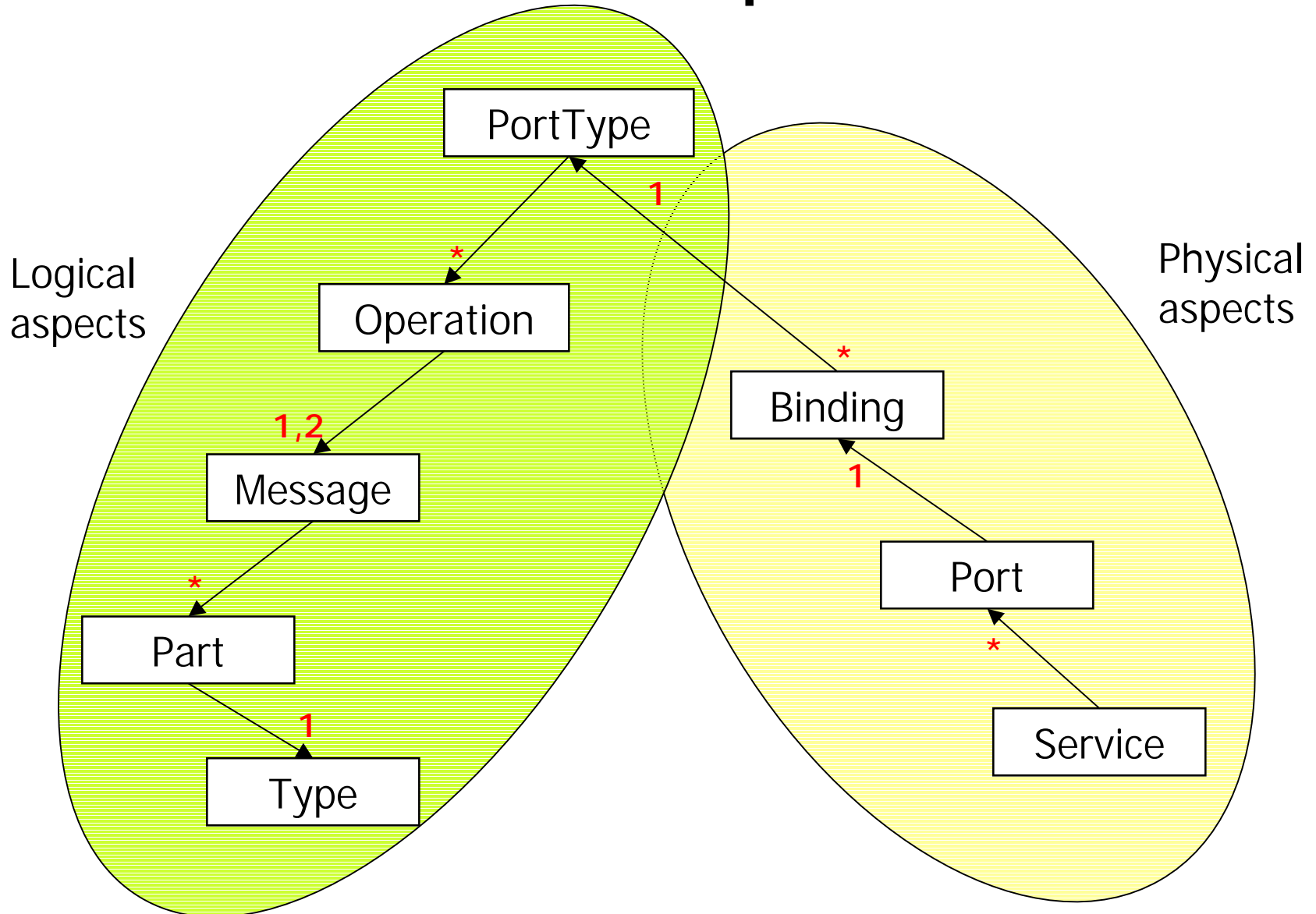
</soap:header>
</wsdl:input>
- <wsdl:output>
  <soap:body use="literal" namespace="http://www.ws-
    i.org/SampleApplications/SupplyChainManagement/2002-
    08/Warehouse.wsdl" />
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
</wsdl:definitions>
- <!--
  The following is an example of a SOAP request message compliant with the above
  WSDL:

  <s:Envelope xmlns:s="http://schemas.xmlsoap.org/envelope/">
    <s:Header>
      <h:Configuration
        xmlns:h="http://www.ws-i.org/SampleApplications/SupplyChainManagemen
        <h:UserId>8bf7ec9a-f3b2-4f39-9807-c55c860a5983</h:UserId>
        <h:ServiceUrl Role="WarehouseA">http://www.ws-i.org/BasicSampleApp/WarehouseA<
        <h:ServiceUrl Role="LoggingFacility">http://www.ws-i.org/BasicSampleApp/Loggin
        <h:ServiceUrl Role="WarehouseC">http://www.ws-i.org/BasicSampleApp/WarehouseC<
        <h:ServiceUrl Role="Retailer">http://www.ws-i.org/BasicSampleApp/Retailer</h:S

```

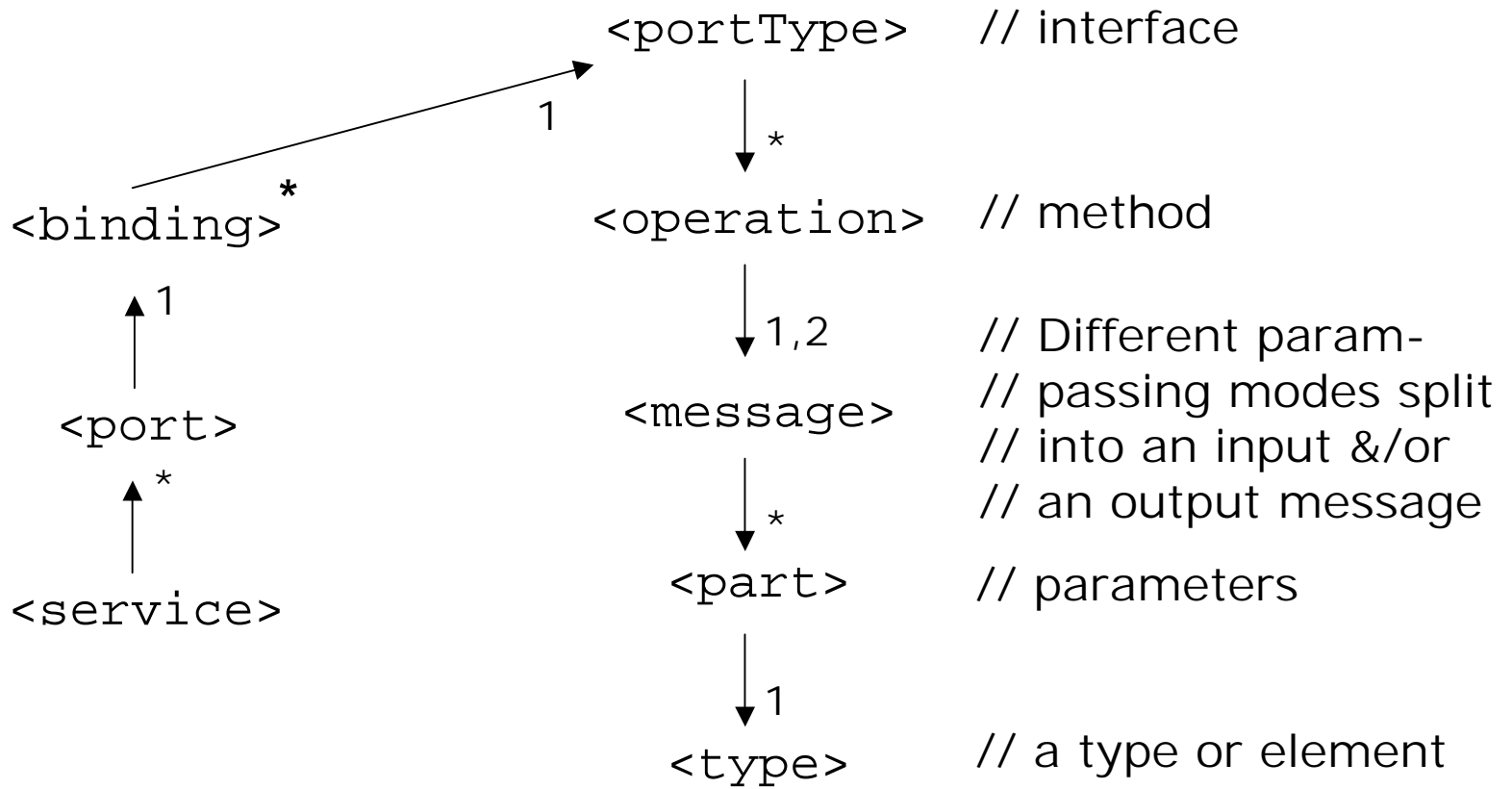
WSDL Overview

WSDL Element Relationships



WSDL Overview

WSDL Element Relationships

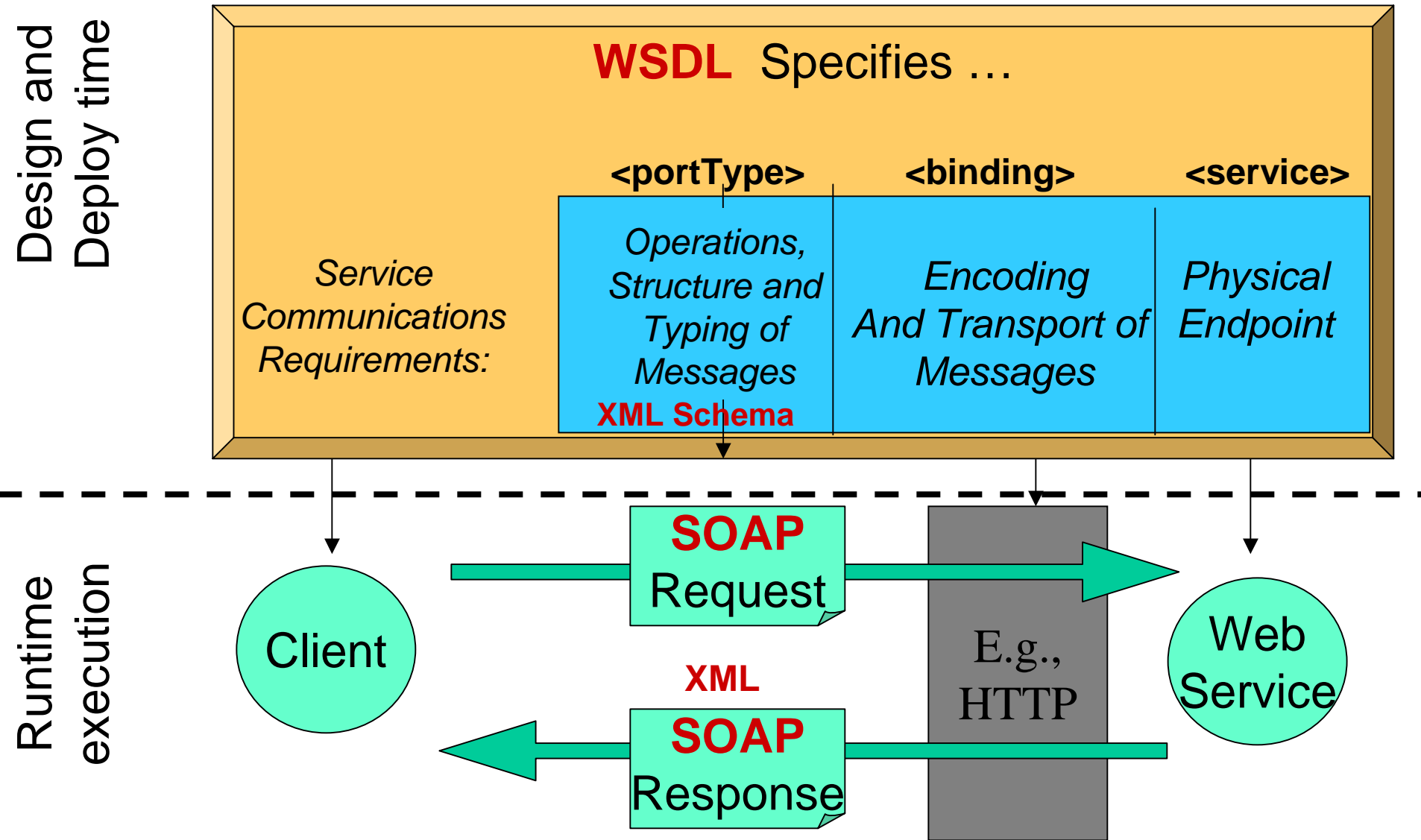


Physical Layer

Logical Layer

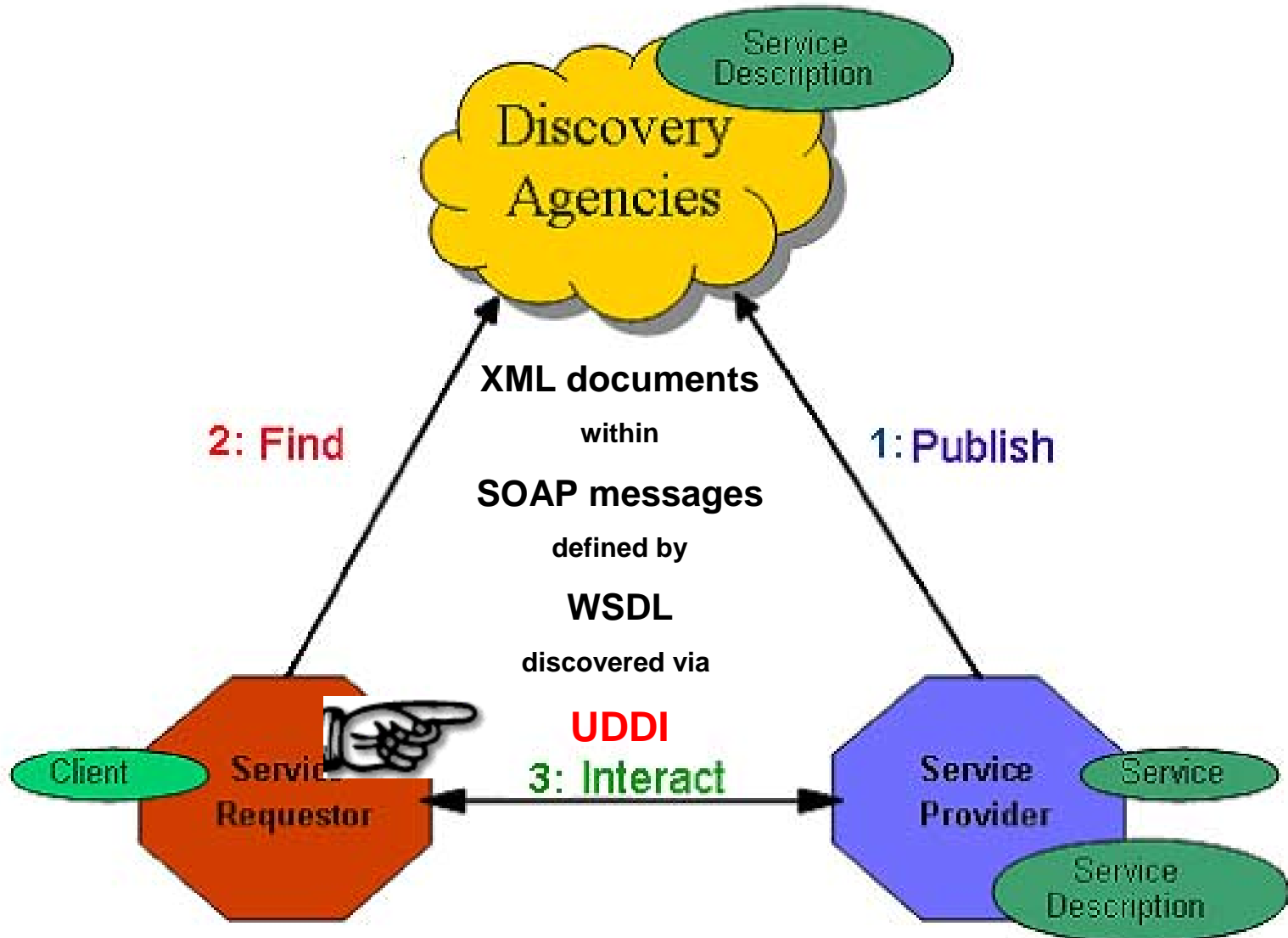
Web Service Technologies Relationships

Run time vs. Design Time



Participants in a Web Services Environment: WSTech-66

Part 4 - UDDI Technology



UDDI Technology

- Universal Description, Discovery and Integration of Web Services
- Sponsored by the OASIS organization
- Supports the ability to register and find services on the internet
 - Service Providers register/publish services including WSDL file along with searchable attributes
 - Potential clients search UDDI registries to retrieve WSDL suiting their service needs
- Resource: <http://www.uddi.org>

UDDI is the least accepted of the SOAP, WSDL, UDDI triangle. It is overly complex, and over burdened by a lot of business level aspects (e.g., descriptions of a business).

UDDI Overview

UDDI API's

- Defined by – what else by – WSDL files
 - UDDI services are Web Services !
- **Register/publish** web services API
 - [Register WSDL](#) for publishing
 - Major operations: save_service, save_binding, ...
- **Inquire** for web services API
 - [Inquire WSDL](#) for inquiring
 - Major inquire operations: find_business, find_service, find_binding, get_binding_detail, get_service_detail, ...

UDDI Overview

UDDI Registries online

- Public UDDI Business Registry (UBR) Nodes available from
 - IBM
 - Inquiry API
 - <http://uddi.ibm.com/testregistry/inquiryapi>
 - Publish API
 - <https://uddi.ibm.com/testregistry/publishapi>
 - Microsoft ...
 - SAP ...
- Test UBRs sponsored by these and other organizations

UDDI Overview

Sample UDDI Registry Listing

Orbix E2A Business Registry Manager for JAXR/UDDIv2 Version 5.4 Build #20021010-1851

File Tasks Help

Current Registry: Orbix E2A UDDI Business Registry Service

Select Registry Registry Listings

Search Registry by Organization Name (% is wildcard)

%

Search

Search Results:

- TransformService
- BrokerService
- FinanceService
- ElectricityService
- KnowledgeBaseService
- ChainService
- LiveEjbService
- InteropTest1999Service
- AttachmentAppService**
- FarmsAndRegalService
- DeliveryConfirmationService
- UDDIRegistryService
- DomesticCalculatorService
- CarShopFlowService
- IONAWarehouseService
- IONAWarehouseService
- IONASupplierService
- IONASupplierService
- IONACreditBureauService

View Service Edit Service Publish Deployed Web Service

Service AttachmentAppService

Name AttachmentAppService

Description This example demonstrates how to write a Web service that handles a SOAP message with attachments.

Service Bindings

Description SOAP binding for AttachmentAppService

AccessURL <http://localhost:8080/xmlbus/AttachmentApp/AttachmentAppService/AttachmentAppPort/>

Correlating the WS Standards

Web Services Interoperability Organization (WS-I)

- Web Services depend heavily on agreed-upon standards (XML, SOAP, WSDL, etc.) that are relatively independent of each other
- Keeping up with the changes and nuances that exist among these is difficult and time-consuming
- WS-I has the mission of doing this for the IT industry
 - advising on how to use these WS standards appropriately and interoperably
- Primary deliverables:
 - **Basic Profile 1.0**: this supplies advice about which combinations of the standards should be used together, and which features should be avoided or emphasized from each standard
 - **Test Tools** for verifying compliance with the Basic Profile
 - **Sample Applications** for assisting others in building compliant systems
- Visit <http://ws-i.org/> for more details ...

Conclusions

- Web Service technologies allow the exchange of XML documents
 - XML, XML Schema define data to be exchanged
 - SOAP is the protocol used for the exchange of the data
 - WSDL specifies the details of services, messages, and endpoints
 - UDDI supports the publishing and locating of services
- The platform-, OS-, programming language-neutral aspects of Web Services allow heterogeneous environments to interoperate easily
- Defining the information/data structures/messages is central to the overall process
 - Utilize the XML Schema and WSDL types facilities to represent your data appropriately
 - Let development tools provide automatic mappings between programming language structures and Web Services structures

Participants in a Web Services Environment: Review

