

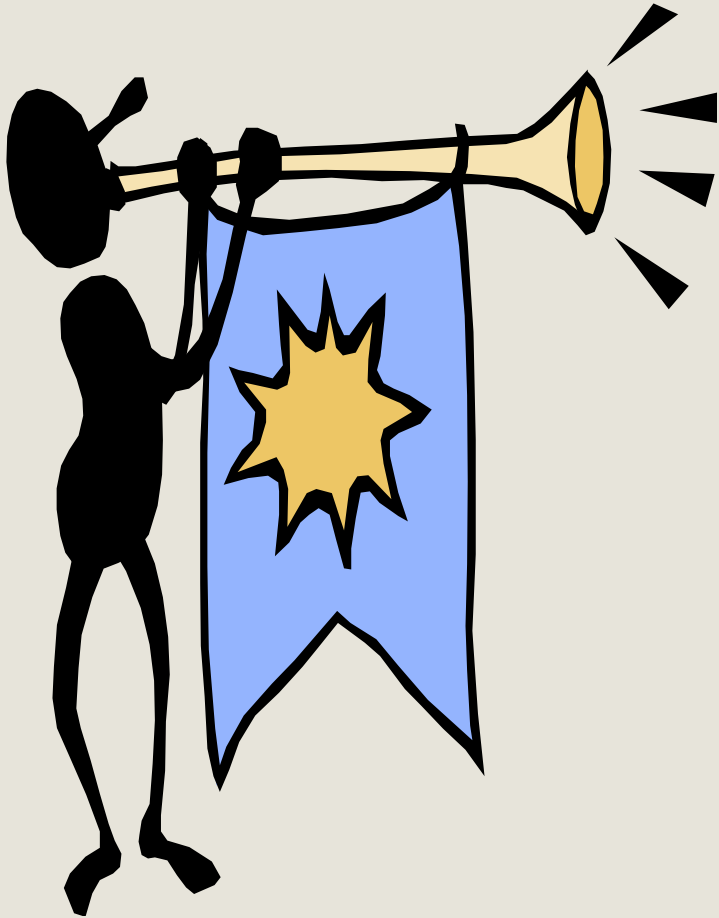
Web Services Based Architectures I

INSE 7110 – Winter 2007

Value Added Services in Next Generation Networks

Week #9

Outline



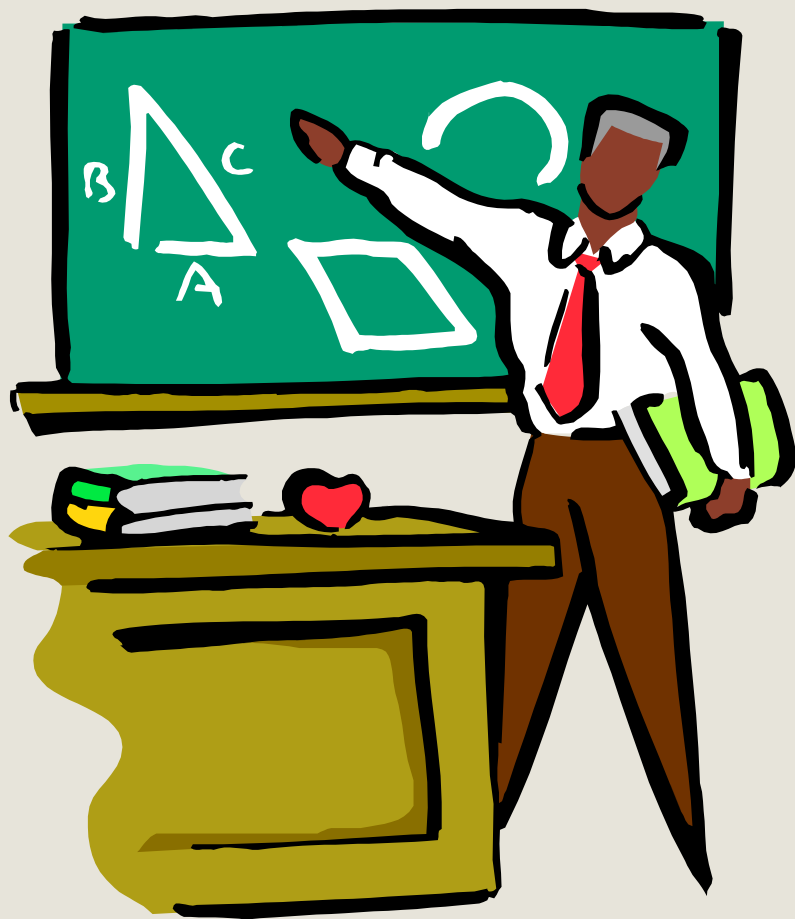
- Basics
- Technologies

Basics



- Fundamental principles
- Business model

Fundamental principles ...



1. Evolution of the Web
2. Definitions and principles
3. Standards

Evolution of the Web

Today

- Publication of documents
- Human interaction
- Proprietary ad-hoc interfaces

XML Technology



Tomorrow

- Publication of **“reusable business logic”**
- Automated Program to program interaction
- Industry standard interfaces

Definitions and principles

“The term Web Services refers to an architecture that allows applications (on the Web) to talk to each other. Period. End of statement”

Adam Bobsworth in ACM Queue, Vol1, No1

The three fundamental principles, still according to Adam Bobsworth:

- 1. Coarse grained approach (I.e. high level interface)**
- 2. Loose coupling (e.g. application A which talks to application B should not necessarily be re-written if application B is modified)**
- 3. Synchronous mode of communication, but also asynchronous mode**

Standards

Some of the involved standards bodies / Consortia

- Architectures and Technologies

- World Wide Web Consortium (W3C)
 - Interoperable technologies for the Web
- Liberty Alliance
 - Open standards for federated network identities (pertinent to Web service security)
 - Network identity refers to the global set of attributes that are contained in an individual's various accounts with different service providers:
 - Personal (name, phone number,
 - Professional (signing key, encryption key, ...)
 - Preferences (music preferences, airline seating preferences, ...)

Standards

Some of the involved standards bodies / Consortia

Application to specific areas

Telecom

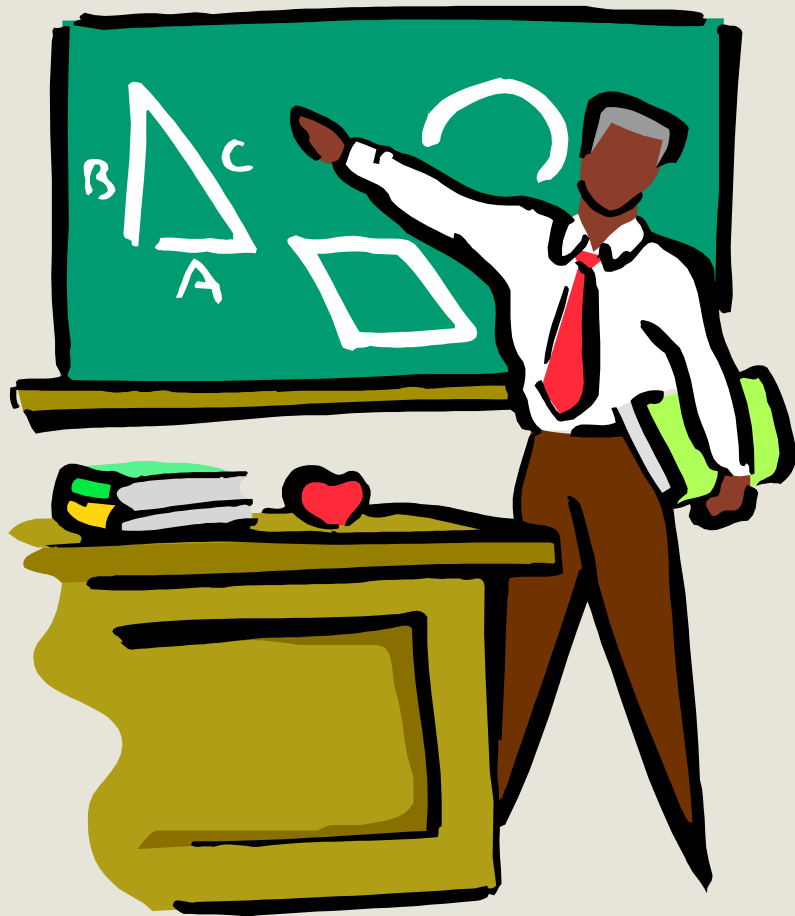
Parlay-X

Open Mobile Alliance (OMA)

Digital images

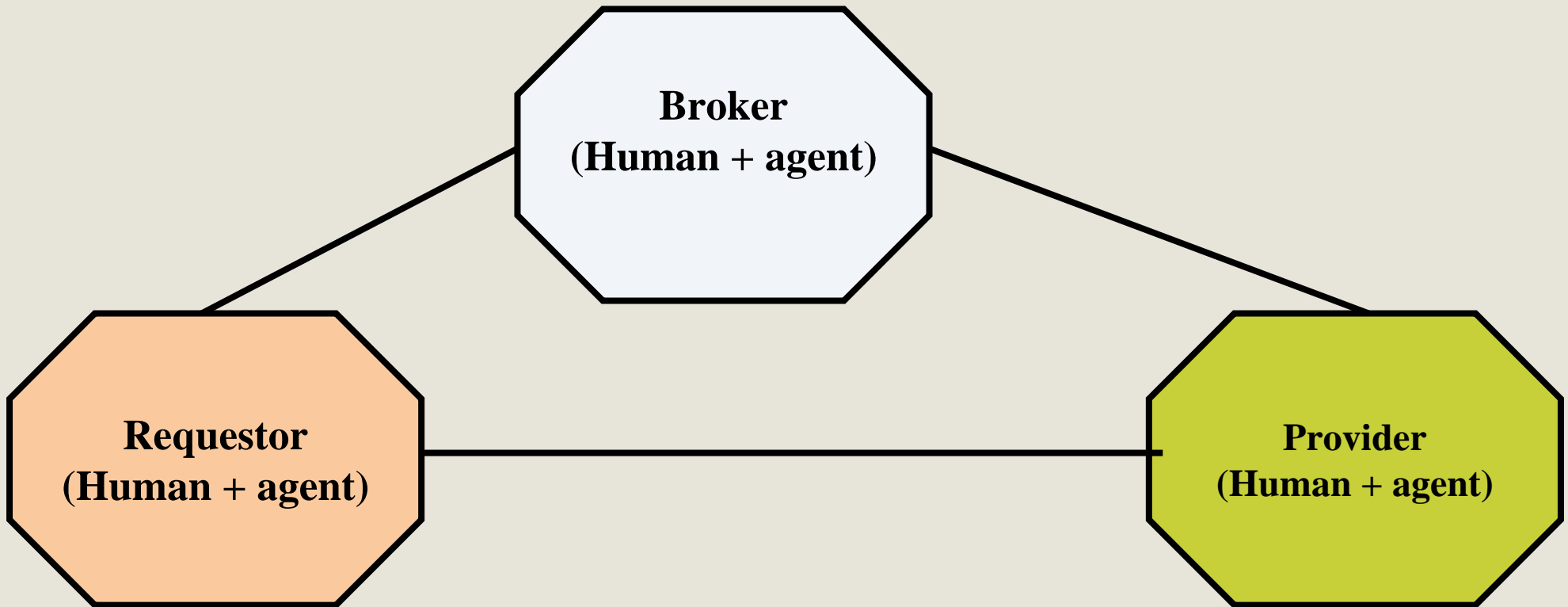
- International Imagery Association

Business model ...



1. Entities
2. Interactions

Entities



Entities

Requestor

- Person or organization that wishes to make use of a Web service.
- Uses an agent (i.e. requestor agent) to exchange messages with both broker agent and provider agent.

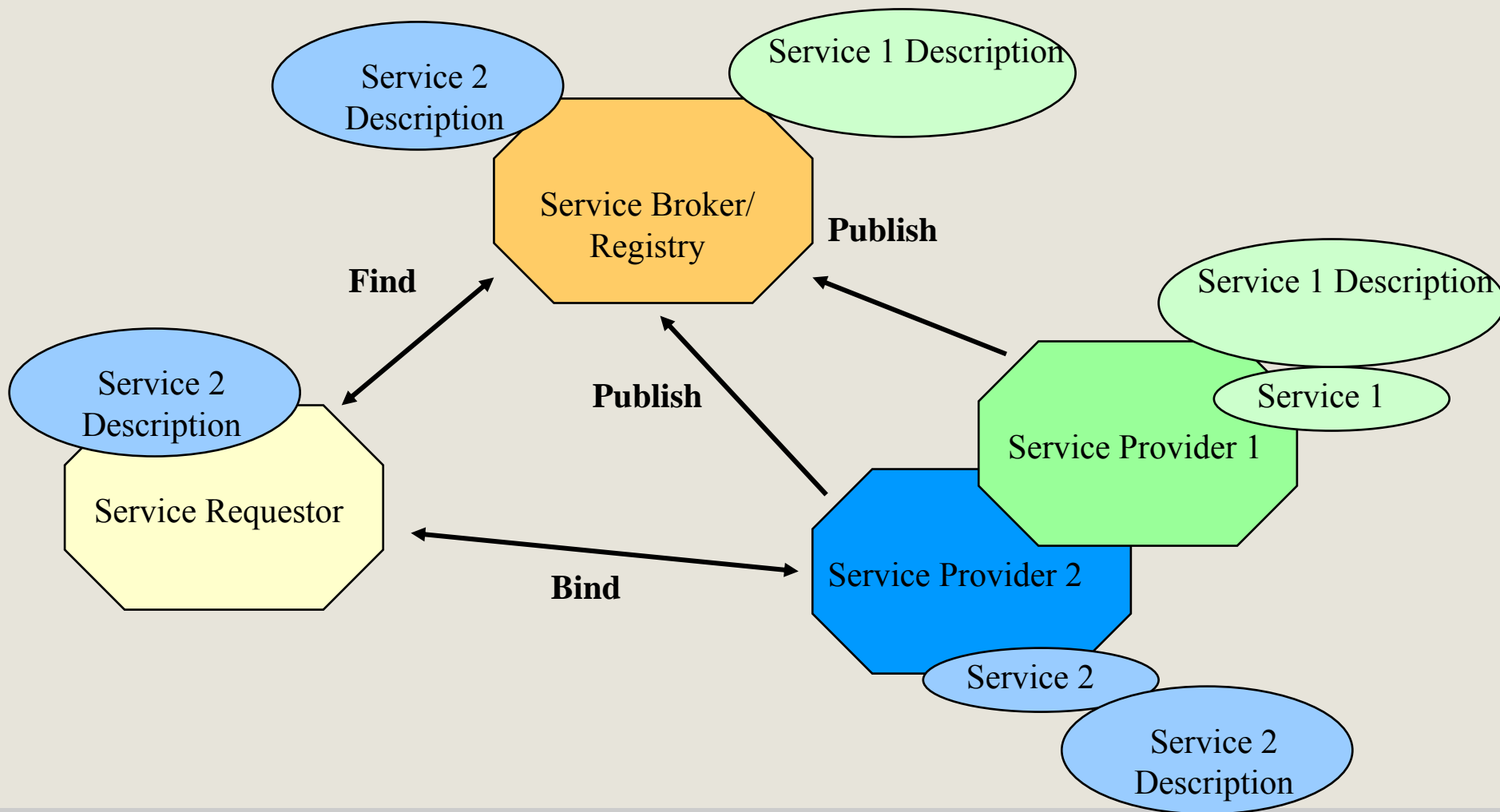
Provider

- Person or organization that owns a Web service it wants to make available for usage
- Use an agent (i.e. provider agent) to exchange messages with broker agent and requestor agent.
- The provider agent is also the software piece which implements the Web service (e.g. mapping towards legacy)

Broker

- Person or organization that puts requestors and providers in contact
 - Providers use brokers to publish Web services
 - Requestors use brokers to discover Web services
- Use an agent (i.e. broker agent) to exchange messages with requestor agent and provider agent

Interactions

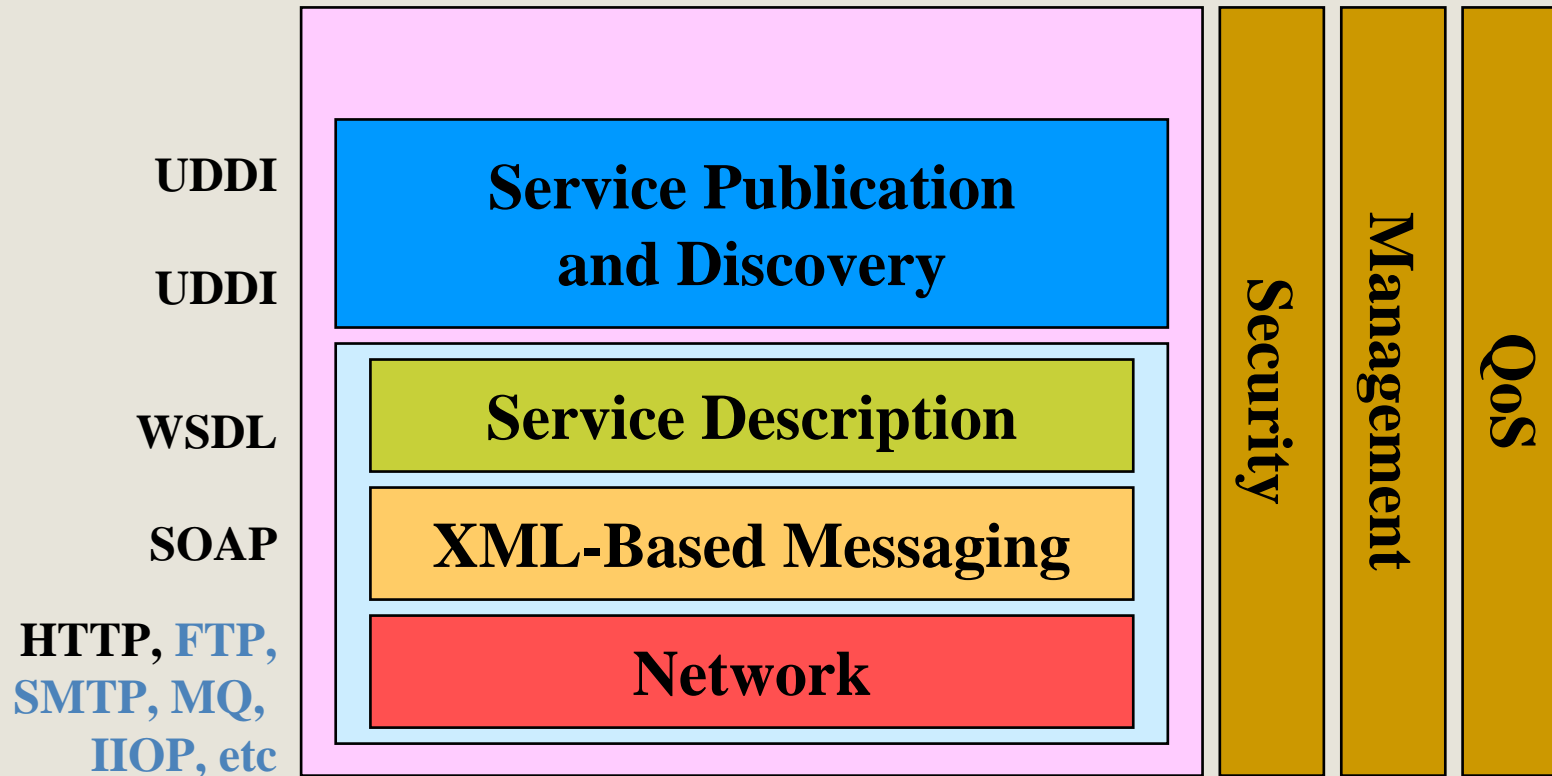


Technologies

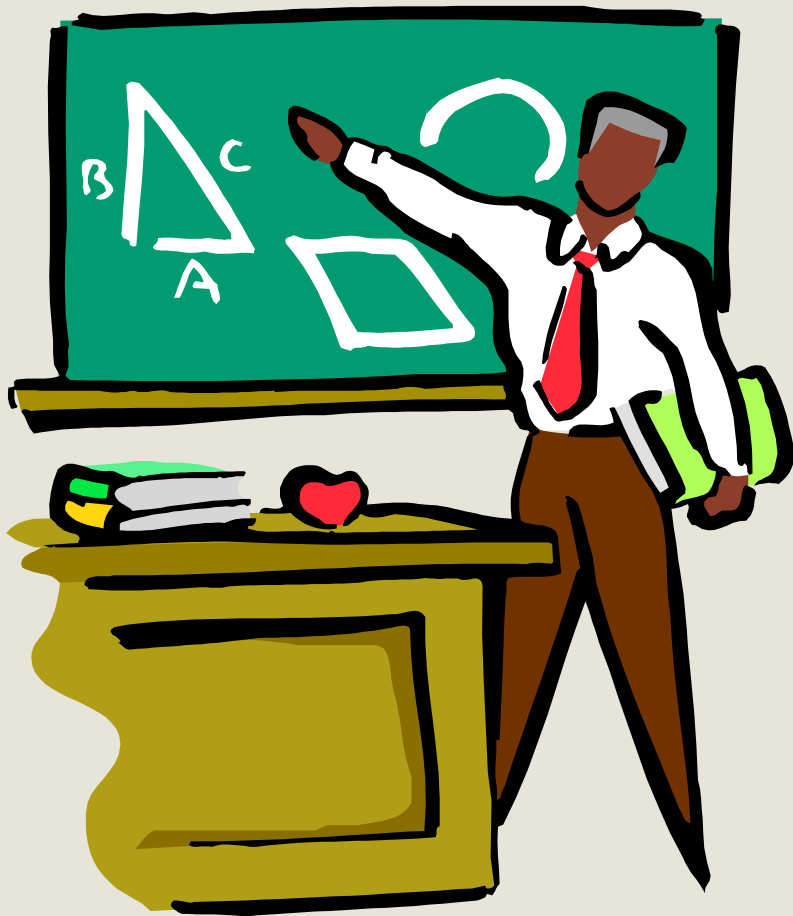


- Extensible Mark Up Language (XML)
- Web Service Description Language (WSDL)
- Simple Object Access Protocol (SOAP)
- Universal Description Discovery and Integration (UDDI)
- Putting it together

Technologies ...



XML ...



- Introduction
- Objectives
- Fundamental concepts
- Examples

Introduction

XML is a markup language for documents containing structured information

XML makes use of tags just like HTML.

- In HTML, both tag semantics (<p> means paragraph) and tag set are fixed)

XML was designed to overcome the limitations of HTML

- Better support for dynamic content creation and management
- Interactions between programs going further than browser / Web page

W3C recommendation

Introduction

Main differences between HTTP and XML

- XML was designed to carry data
- XML is not a replacement for HTML
- XML and HTML were designed with different goals
 - XML was designed to describe data and to focus on what data is.
 - HTML was designed to display data and to focus on how data looks.
 - HTML is about displaying information, while XML is about describing information.
- XML is free and extensible (xml tags are not predefined)

Introduction

XML advantages

- Structure information
- Separate actual data from data representation
- Store data in plain text format
- Share data in software-and-hardware independent way
- Exchange data between incompatible systems
- Create new languages (WAP, WML).
- Platform independent

Objectives (As per the W3C recommendation)

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs which process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

Fundamental concepts

XML documents

Data objects made of elements

- `<element> content </element>`

Well-formed Documents

- If it obeys to the XML syntax
 - Exp: - All XML elements must have a closing tag
 - The name in an element's end-tag **MUST** match the element type in the start-tag.
 - All XML elements must be properly nested

Valid document

- A well-formed document is valid if it obeys to the structural rules of the associated DTD or schema document

Fundamental concepts

XML documents

Schema and DTD (Document Type Definition)

- Provide a grammar for a class of documents
- Define the legal elements of an XML document

Namespace

- An XML namespace is a collection of names, identified by a URI reference
- Provides a simple method for qualifying element and attribute names used in XML documents.

Fundamental concepts

XML documents Example

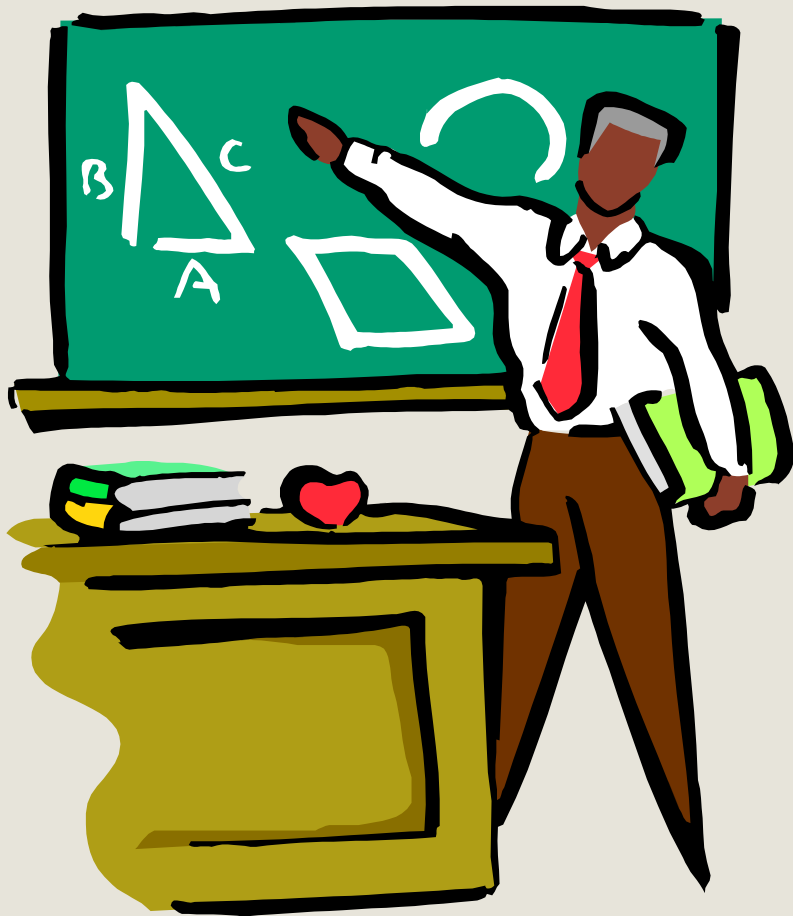
```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<book>  
  <title>Understanding Web Services</title>  
  <author>Eric Newcomer</author>  
  <price>39.99</price>  
</book>
```

Fundamental concepts

XML processor

- Read XML documents
 - Provide access to the content and the structure
 - Behaviour described in the XML specifications
 - Navigate XML document structure and add, modify, or delete its elements.
-
- Most popular programming APIs
 - Document Object Model (DOM) from W3C
 - Simple API for XML (SAX) – From XML-DEV mailing list

SOAP ...



- Introduction
- Message structure
- Bindings

Introduction

SOAP is

- A simple XML based communication protocol between applications
- Platform and language independent

Purpose: Get the XML data from one point to another point over the network

- Provider / UDDI
- Requestor / UDDI
- Provider / Requestor

W3C recommendation

- Effort initiated by IBM and IONA

Introduction

Purpose: Get the data from one point to another point over the network

- One way XML messaging protocol that can be used to build models such as
 - Request / reply
 - Asynchronous messaging
 - Event notification
- Entities
 - Sender
 - Receiver
 - Intermediary

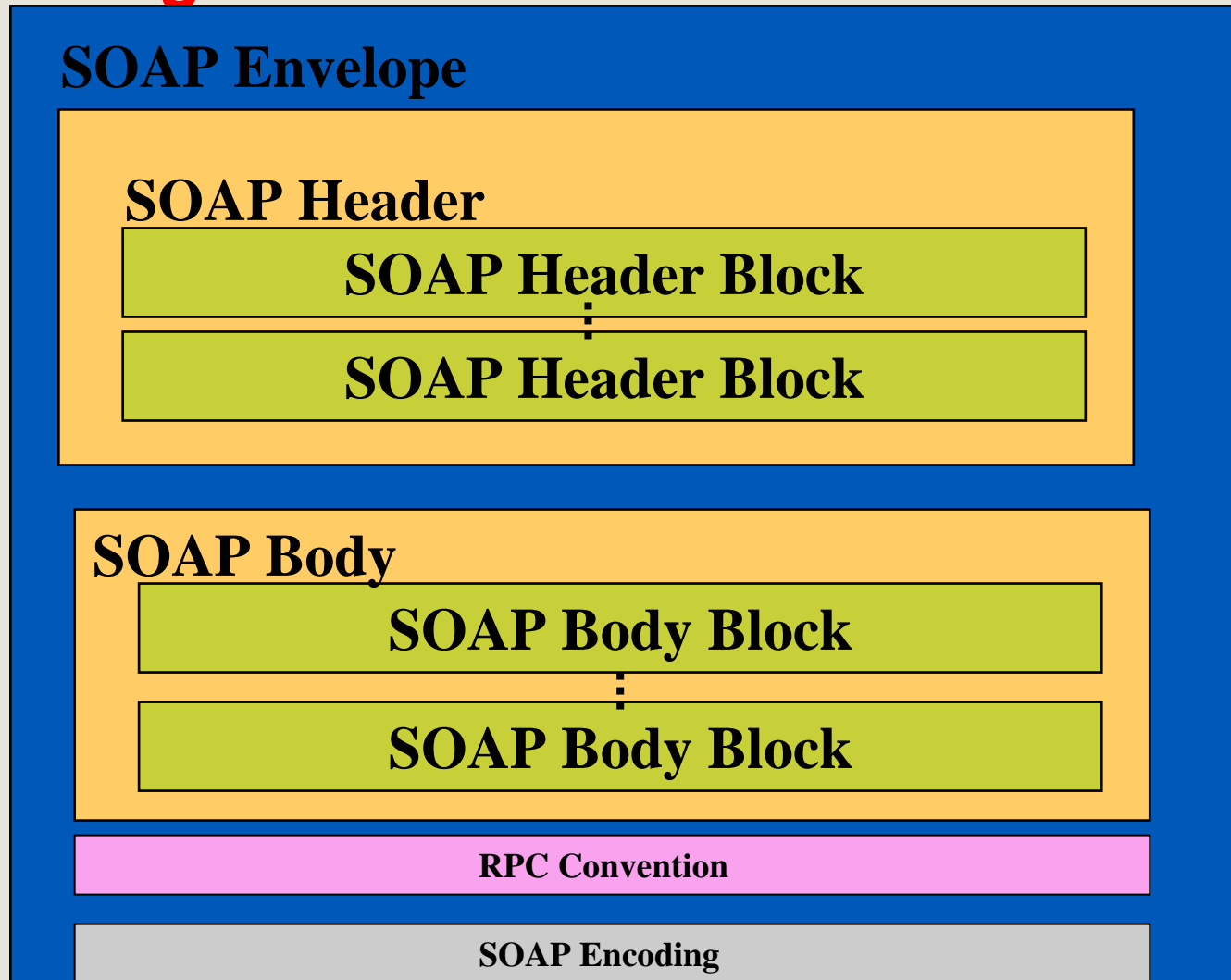
Message structure

Several parts

- **Envelope** (mandatory): Start and end of message
- **Header** (optional): Optional attributes used in the processing
 - May be negotiated
 - Examples: transactions, priority, QoS, security

- **Body** (mandatory): Message being sent
 - Actual message
 - Fault codes
- **Attachment** (optional) : Self-explanatory
- **RPC convention** (optional) : Requirements for RPC mapping
 - Target URI for the SOAP node, procedure name/signature
- **SOAP Encoding** (optional) : How to represent data being transmitted in the message
 - Encoding scheme

Message structure



Bindings

Purpose: Specification of how SOAP messages may be passed from one node to another node using a concrete lower layer protocol

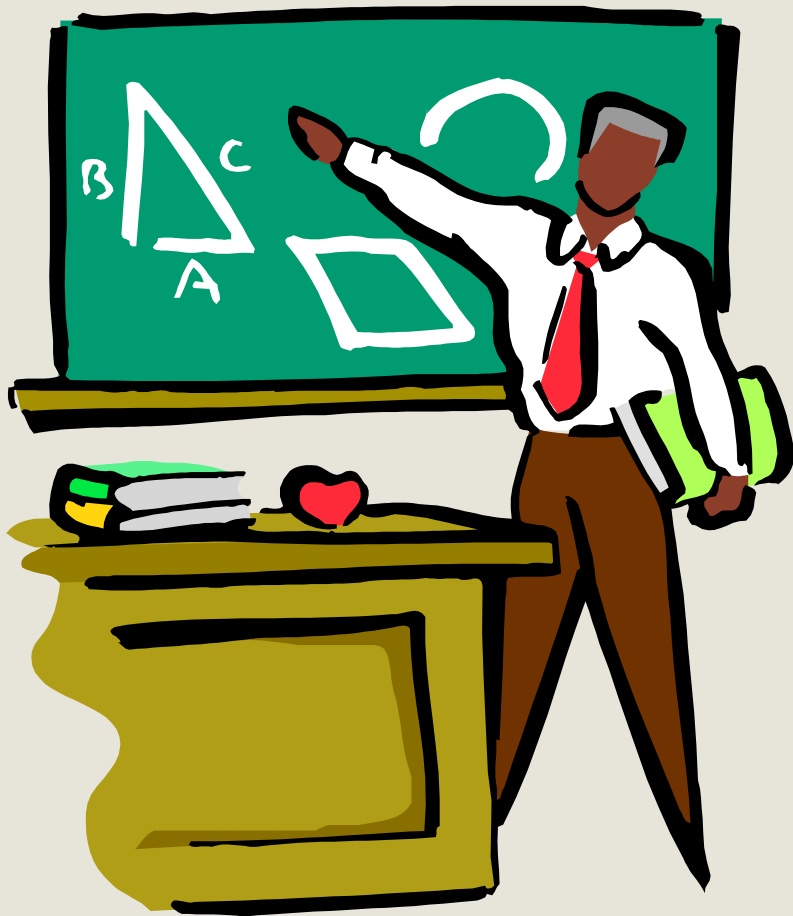
Existing bindings

- HTTP
- SOAP over email

HTTP binding

- HTTP Request URI used to identify SOAP node
- Commonly used HTTP request for carrying SOAP messages: HTTP Post

Additional information on SOAP ...



Reminder: SOAP message sender and receiver ...

Concepts

- **Sender**
 - Initial sender
 - Intermediary sender
- **Receiver**
 - Intermediary receiver
 - Ultimate receiver

Nodes

- **Sender**
- **Intermediary (Intermediary sender + intermediary receiver)**
- **Ultimate receiver**

Why SOAP? Why not just send XML documents in HTTP or via Email ...

Give application level control:

- Priority
- Security
- And other

Via header processing by intermediaries:

- Take actions according to headers
- Replace headers
- And others ..

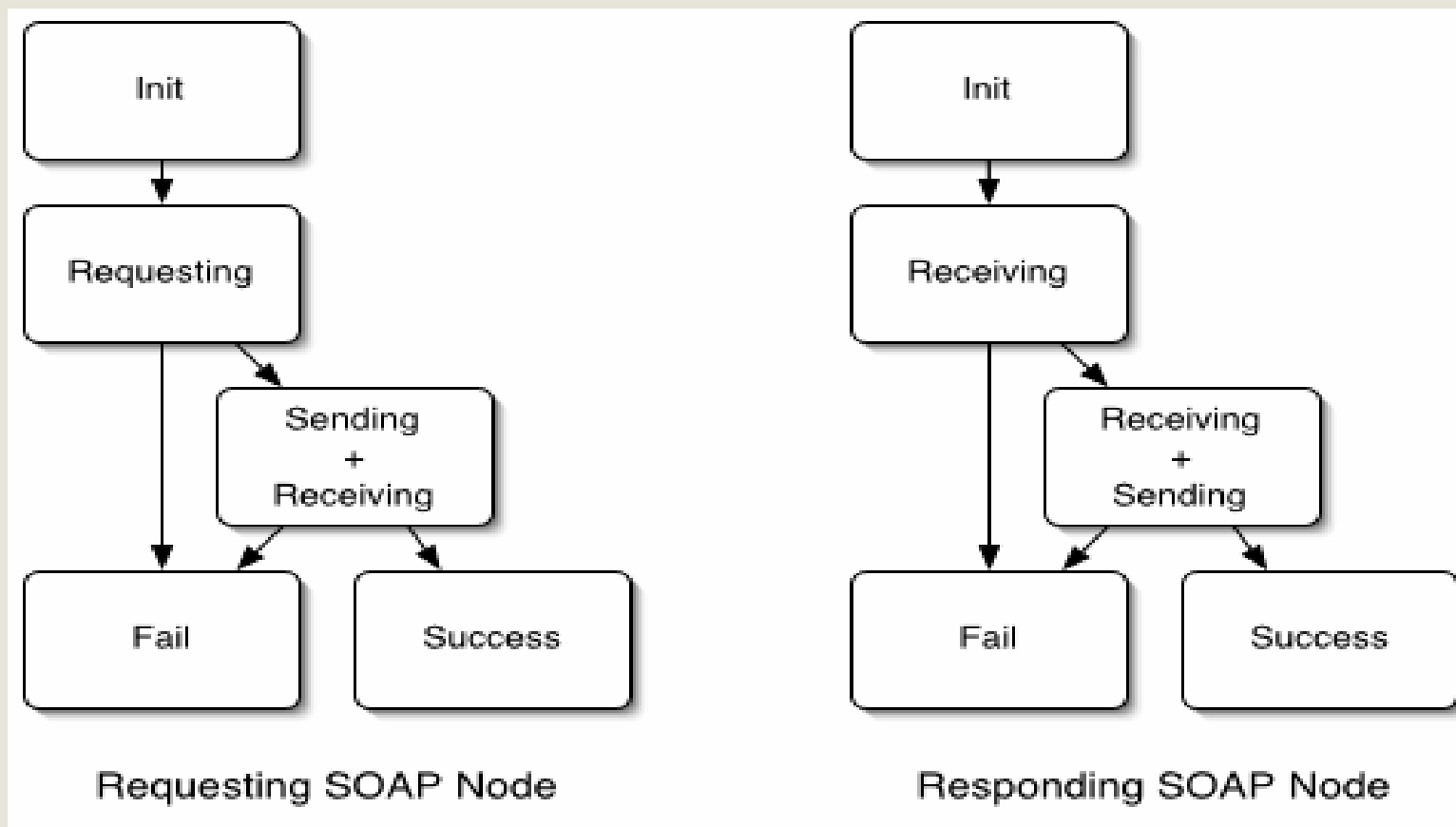
And independently of the binding (e.g. email, HTTP):

Why SOAP? Why not just send XML documents in HTTP or via Email ...

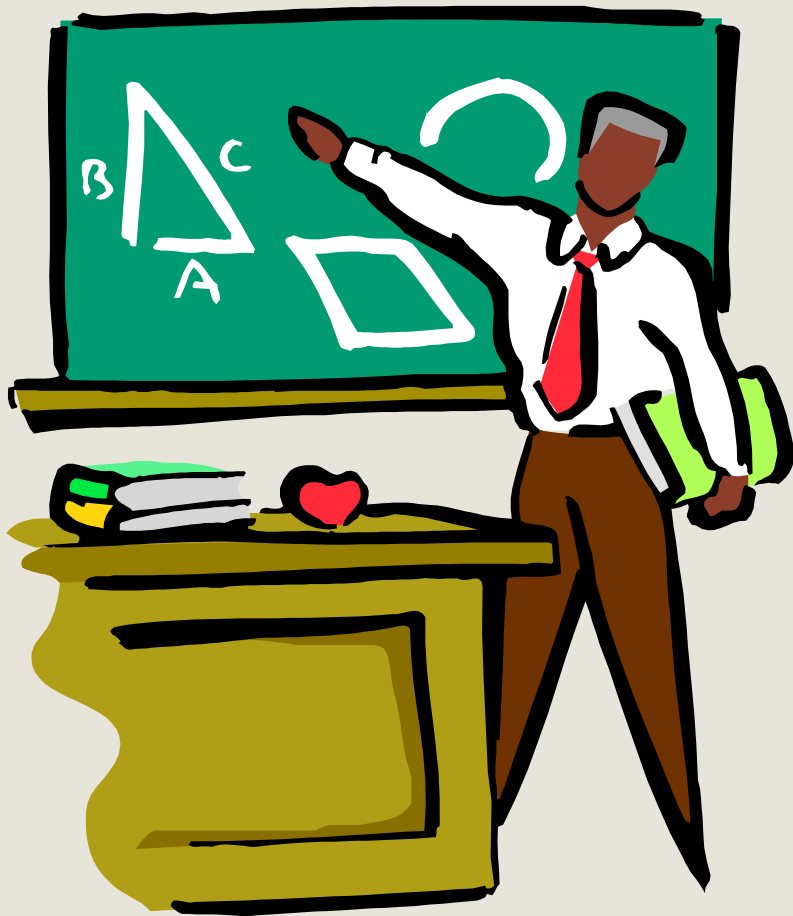
Example 1: SOAP message containing a SOAP header block and a SOAP body

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">  
  <env:Header>  
    <n:alertcontrol xmlns:n="http://example.org/alertcontrol">  
      <n:priority>1</n:priority>  
      <n:expires>2001-06-22T14:00:00-05:00</n:expires>  
    </n:alertcontrol>  
  </env:Header>  
  <env:Body>  
    <m:alert xmlns:m="http://example.org/alert">  
      <m:msg>Pick up Mary at school at 2pm</m:msg>  
    </m:alert>  
  </env:Body>  
</env:Envelope>
```

A message exchange pattern



WSDL ...



- Introduction
- Elements
- Grammar

Introduction

WSDL is an XML-based language for describing Web services and how to access them

Purpose: XML grammar for describing a Web service

- Formats and protocols
 - Input data to the Web service
 - Operations to be performed on the data
 - Binding to a transport protocol

Initially developed by a handful of companies (e.g. IBM, Microsoft)

Now a W3C recommendation

Elements

Types

–Definition of data types used to describe the exchanged messages

Messages

–Describes the abstract format of a particular message that a Web service sends or receives, using the defined types

- One way
- Request / reply
- Solicit response
- Notification

Operation

–Abstract definition of an action supported by the service

Interface

-Abstract set of operations supported by one or more endpoints

Elements

Binding

- Concrete protocol and data format specification for a particular EndPoint

EndPoint

- Defined as a combination of a binding and a network address

Service

- Collection of related endpoints

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="TicketAgent"
  targetNamespace="http://airline.wsd/ticketagent/"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://airline.wsd/ticketagent/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsd1="http://airline/">
  <import location="TicketAgent.xsd" namespace="http://airline/">
  <message name="listFlightsRequest">
    <part name="depart" type="xs:dateTime"/>
    <part name="origin" type="xs:string"/>
    <part name="destination" type="xs:string"/>
  </message>
  <message name="listFlightsResponse">
    <part name="result" type="xsd1:ArrayOfString"/>
  </message>
  <interface name="TicketAgent">
    <operation name="listFlights" parameterOrder="depart origin destination">
      <input message="tns:listFlightsRequest" name="listFlightsRequest"/>
      <output message="tns:listFlightsResponse" name="listFlightsResponse"/>
    </operation>
  </interface>
</definitions>
```

Grammar

Conventions used in the specifications

? (0 or 1)

* (0 or more)

+ (1 or more)

Grammar

Examples

One way messaging vs. two way messaging

```
<wsdl:definitions .... > <wsdl:portType .... > *  
  <wsdl:operation name="nmtoken">  
    <wsdl:input name="nmtoken"? message="qname"/>  
  </wsdl:operation>  
</wsdl:portType >  
</wsdl:definitions>
```

```
<wsdl:definitions .... >  
  <wsdl:portType .... > *  
    <wsdl:operation name="nmtoken" parameterOrder="nmtokens">  
      <wsdl:input name="nmtoken"? message="qname"/>  
      <wsdl:output name="nmtoken"? message="qname"/>  
      <wsdl:fault name="nmtoken" message="qname"/>*  
    </wsdl:operation>  
  </wsdl:portType >  
</wsdl:definitions>
```

Example from WSDL specification

Example 1 SOAP 1.1 Request/Response via HTTP

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

Example from WSDL specification

Example 1 SOAP 1.1 Request/Response via HTTP - Continued

```
<types>
  <schema targetNamespace="http://example.com/stockquote.xsd"
    xmlns="http://www.w3.org/2000/10/XMLSchema">
    <element name="TradePriceRequest">
      <complexType>
        <all>
          <element name="tickerSymbol" type="string"/>
        </all>
      </complexType>
    </element>
    <element name="TradePrice">
      <complexType>
        <all>
          <element name="price" type="float"/>
        </all>
      </complexType>
    </element>
  </schema>
</types>
```

Example from WSDL specification

Example 1 SOAP 1.1 Request/Response via HTTP - Continued

```
<message name="GetLastTradePriceInput">  
  <part name="body" element="xsd1:TradePriceRequest"/>  
</message>
```

```
<message name="GetLastTradePriceOutput">  
  <part name="body" element="xsd1:TradePrice"/>  
</message>
```

```
<portType name="StockQuotePortType">  
  <operation name="GetLastTradePrice">  
    <input message="tns:GetLastTradePriceInput"/>  
    <output message="tns:GetLastTradePriceOutput"/>  
  </operation>  
</portType>
```

Example from WSDL specification

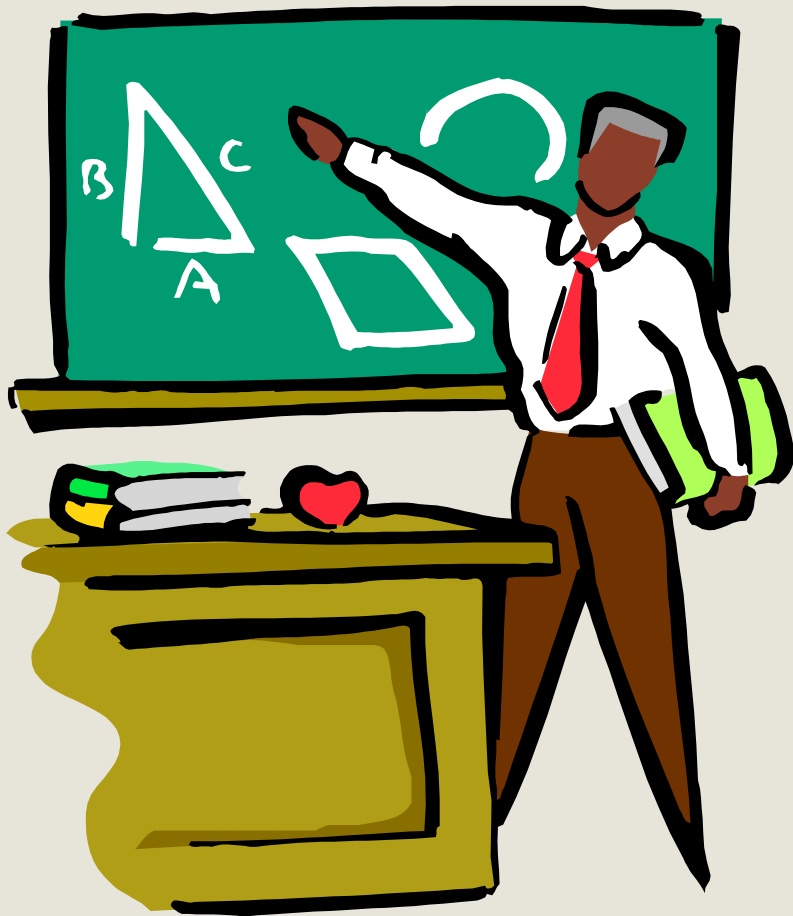
Example 1 SOAP 1.1 Request/Response via HTTP - Continued

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

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

</definitions>
```

UDDI ...



- Introduction
- Content
- Publishing APIs
- Inquiry APIs

Introduction

Purpose: Enable the publication, the discovery and the usage of Web services

- Integral part of the Web services infrastructure
 - Public
 - Semi-public (e.g. circle of trust)
 - Private (e.g. enterprise)
- Data bases accessible via SOAP APIs
 - Publishing API
 - Inquiry APIs

Introduction

UDDI.ORG

- Initiated by a a handful of companies (e.g. IBM, Microsoft)
- Now open to all companies
- Produce specifications for UDDI

Initial public UDDI repository

- Operated by founders of UDDI.ORG, later joined by HP and SAP
- Synchronized data bases called operator sites (one at each site)
- Test UDDI
 - Allow requestors and providers to test their UDDI clients
- Production UDDI
 - Allow providers to actually publish Web Services and requestors to actually inquire about Web services
 - Need to register with one of the operators for publishing services (authorization)

The content ...

White pages

Business address

Contact person / number

Yellow pages

More info about the business

- Type of business
- Industry type
- Products / services

Green pages

Technical information about the services

- Service features/functionality
- Pointer to the WSDL file

The content ...

UDDI data model

Business entity

- Top level structure
- Description of the entity for which information is being registered
- Include the list of Web services provided by the entity

Business services

- Name and description of services being published
- Include binding templates

Binding templates

- Information about the services
- Include entry point for accessing the services

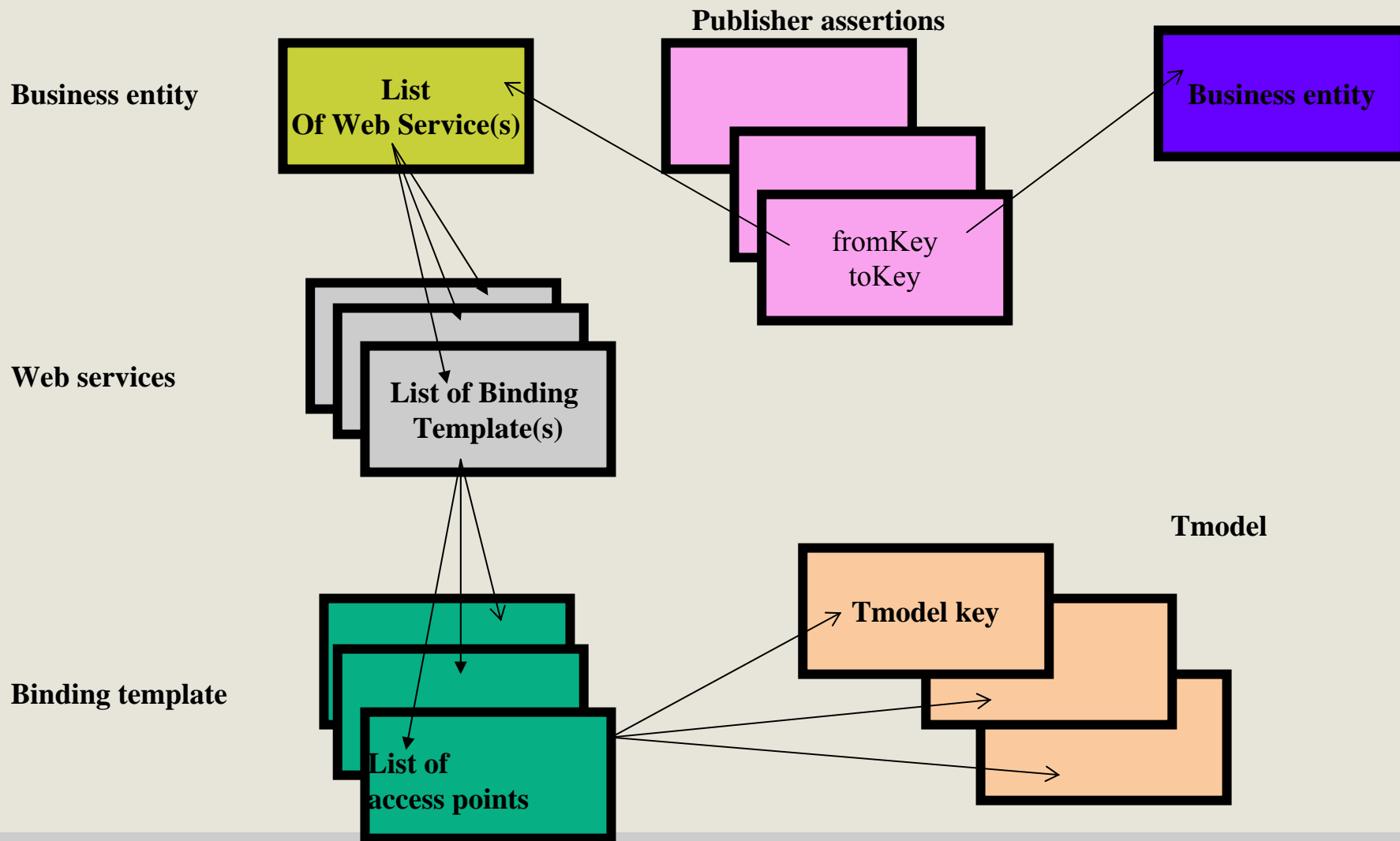
tModel

- Fingerprint, collection of information that uniquely identify the service

Publisher assertion

- Business relationship between business entities (e.g. subsidiary of ..)

Data model ...



Publishing APIs

Some examples

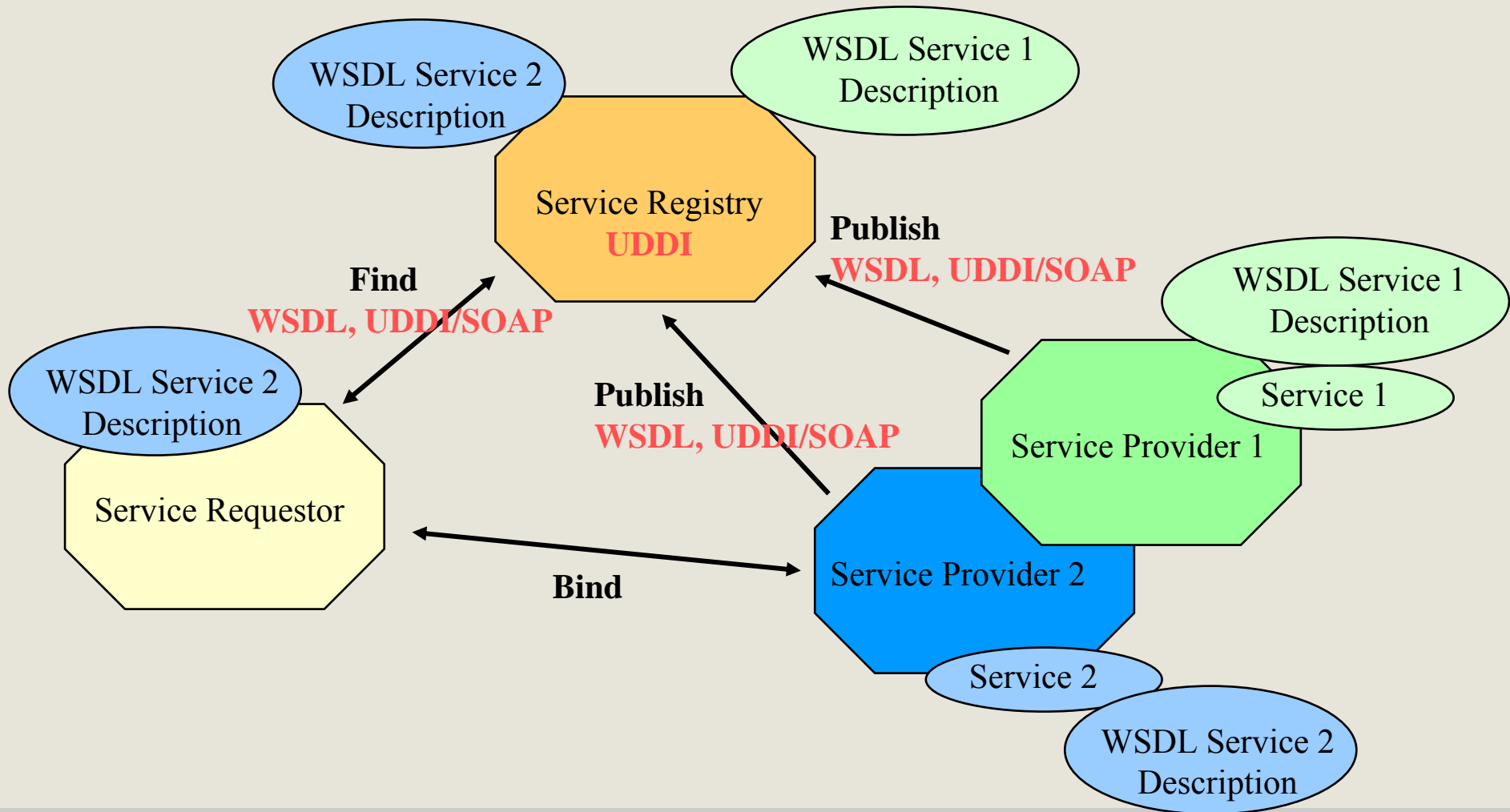
- Add Publisher Assertions
- Save/Delete binding
- Save/Delete Business
- Save/Delete Service
- Save/Delete tModel
- Set/Get Publisher assertions
- Delete_Publisher_Assertion
- Get Registered assertions
- Get Assertions status report (used by UDDI operators)

Inquiry APIs

Some examples

- Find binding
- Find business
- Find related business
- Find service
- Find tModel
- Get binding details
- Get business details
- Get tModel details

Putting it together ...



Web Service Discovery

- The Registry Approach
- The Index Approach
- Peer-to-Peer (P2P) Discovery

Web Service Discovery

- The Registry Approach

- A registry is an authoritative, centrally controlled store of information
- Publishing is active: Service description must be explicitly placed in the registry by the service provider before being available
- The registry owner decides what information is placed in the registry
- Access control needed

➤ UDDI is often seen as an example of the registry approach, but it can also be used as an index.

Web Service Discovery

The Index Approach

- Is a compilation or guide to information that exists elsewhere
 - Publishing is passive
 - Anyone can create their own index
 - The information contained in an index could be out of date
- Google is often cited as an example of the index approach.

Web Service Discovery

- Peer-to-Peer (P2P) Discovery
 - Centralized registry not needed
 - Allows dynamic discovery of web services
 - Each web service is a node in a network of peers
 - Each node may contain its own indexing of the existing Web services.
 - The discovered information is known to be current.

Web Service Discovery

- Functional Descriptions and Discovery
 - Web services discovery requires the ability to search for appropriate Web services based on functional descriptions or other criteria
 - Functional descriptions need to be:
 - machine processable
 - Unambiguous
 - Capable of expressing any existing or future functionality
 - Capable of expressing existing and new vocabularies and relationships between functionalities
- Need for web services semantics description
 - ❖ Share the same concepts understanding between the different roles

Web Services and Semantics (WS2) project

- Is a specific support action (financed by European Commission's IST Programme) running for 2 years, from July 2004 to June 2006, in order to promote Web services and work on integration of semantics.
- WS2 Goal:
 - Support the W3C in
 - Administering its Web services related standard activities
 - Their wide adoption in the European industry
 - Their evolution toward achieving efficient and interoperable communications within distributed software, as well as rich and complex interactions, **exploring use of Semantic Web technologies**

Web Services and Semantics (WS2) project

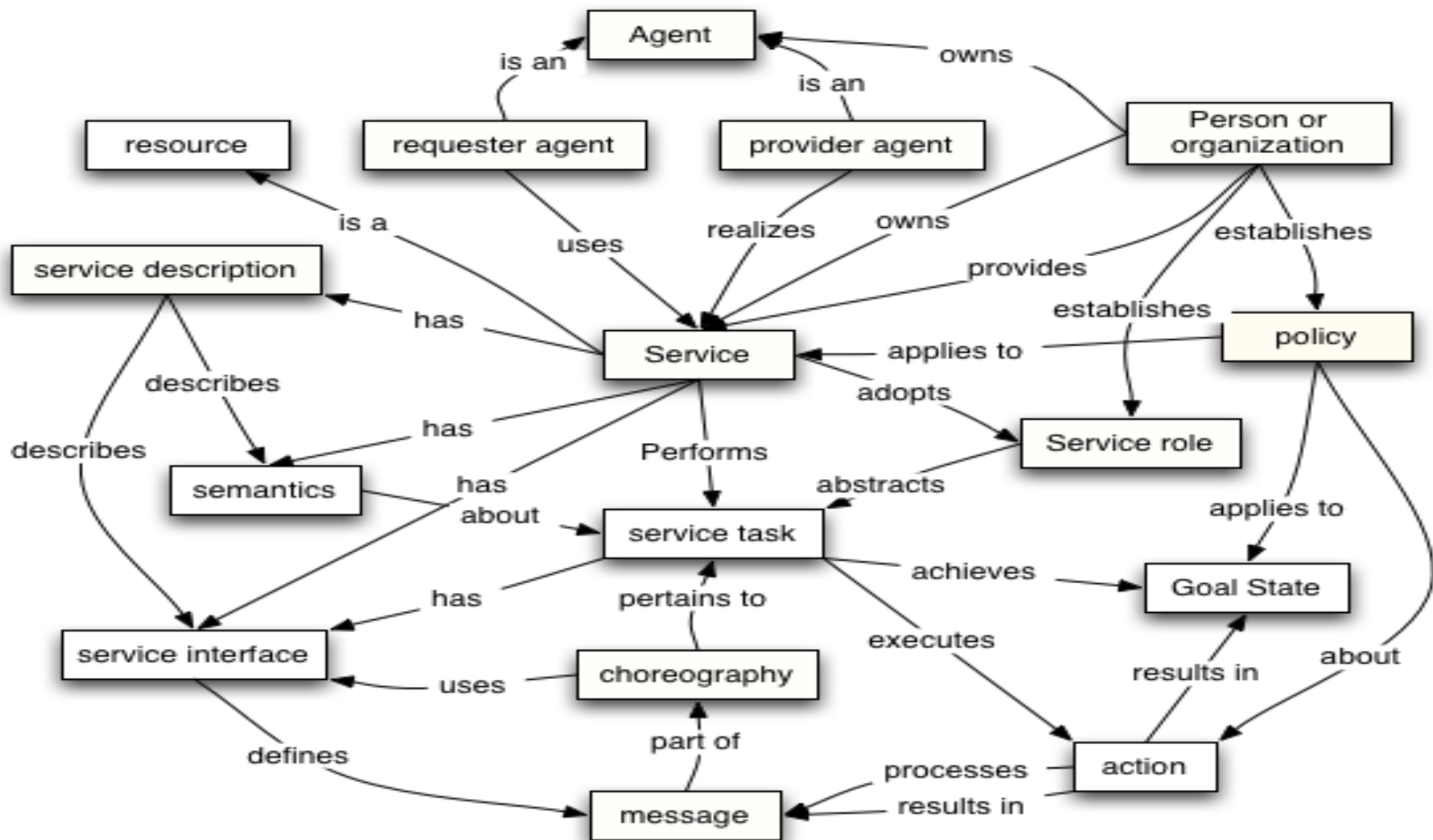
Service Model Concepts

- **Web service:** is an abstract notion that must be implemented by a concrete agent.
- **Agent:** is the concrete piece of software or hardware that sends and receives messages, while the service is the resource characterized by the abstract set of functionality that is provided.
- **Provider entity:** is the person or organization that provides an appropriate agent to implement a particular service
- **Requester entity:** is a person or organization that wishes to make use of a provider entity's Web service

Web Services and Semantics (WS2) project

Service Model Concepts

- **Provider agent:** is a web service software agent that realizes one or more services
- **requester agent:** is an agent that uses a web service
- **Service task:** is an abstraction that encapsulates some intended effect of invoking a service.
- **Service description:** is a machine-processable description of a Service and service's interface



Example from OWL description

- Service model: Agent

```
<!-- 2.3.2.2 Agent -->
_ <owl:Class rdf:ID="Agent">
  <rdfs:label>Agent</rdfs:label>
  <rdfs:comment> And agent is a program acting on behalf of another person, entity, or process</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2004/02/wsa/ResourceModel.owl#Resource" />
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2004/02/wsa/Extensions.owl#Computational_Resource" />
</owl:Class>
```

```
<owl:ObjectProperty rdf:ID="has_owner">
  <rdfs:label>has_owner</rdfs:label>
  <rdfs:comment> An agent has an owner that is a person or organization</rdfs:comment>
  <owl:inverseOf rdf:resource="http://www.w3.org/2004/02/wsa/PolicyModel.owl#owner" />
</owl:ObjectProperty>
```


Example from OWL description

- Service model: Agent

```
<owl:ObjectProperty rdf:ID="realize_service">  
  <rdfs:label>realize_service</rdfs:label>  
  <rdfs:comment>An agent may realize zeror more services</rdfs:comment>  
  <rdfs:domain rdf:resource="#Agent" />  
  <rdfs:range rdf:resource="#Service" />  
</owl:ObjectProperty>
```

Examples of tool kits

- Examples of tool kits
 - Apache / Axis
 - BEA Weblogic
 - SunOne
 - .Net
 - Systinet
 - Get tModel details
- Usage simplicity depends on:
 - Friendly user interface
 - Detail level required

BEA WebLogic

• WebLogic Workshop

- provides a framework for building web services
- **Free Edition** (non-expiring development license)
- Is a Java development environment for BEA
- Simplifies the process of creating web services by insulating developers from the low-level implementation details
- Support web services standards
 - SOAP 1.1 and 1.2
 - WSDL 1.1
 - OASIS Standard 1.0 Web Services Security specifications
 - UDDI 2.0

BEA WebLogic

- **Web Service Development Cycle**
 - Designing a Web Service
 - Implementing Web Service Code
 - Testing a Web Service
 - Deploying a Web Service

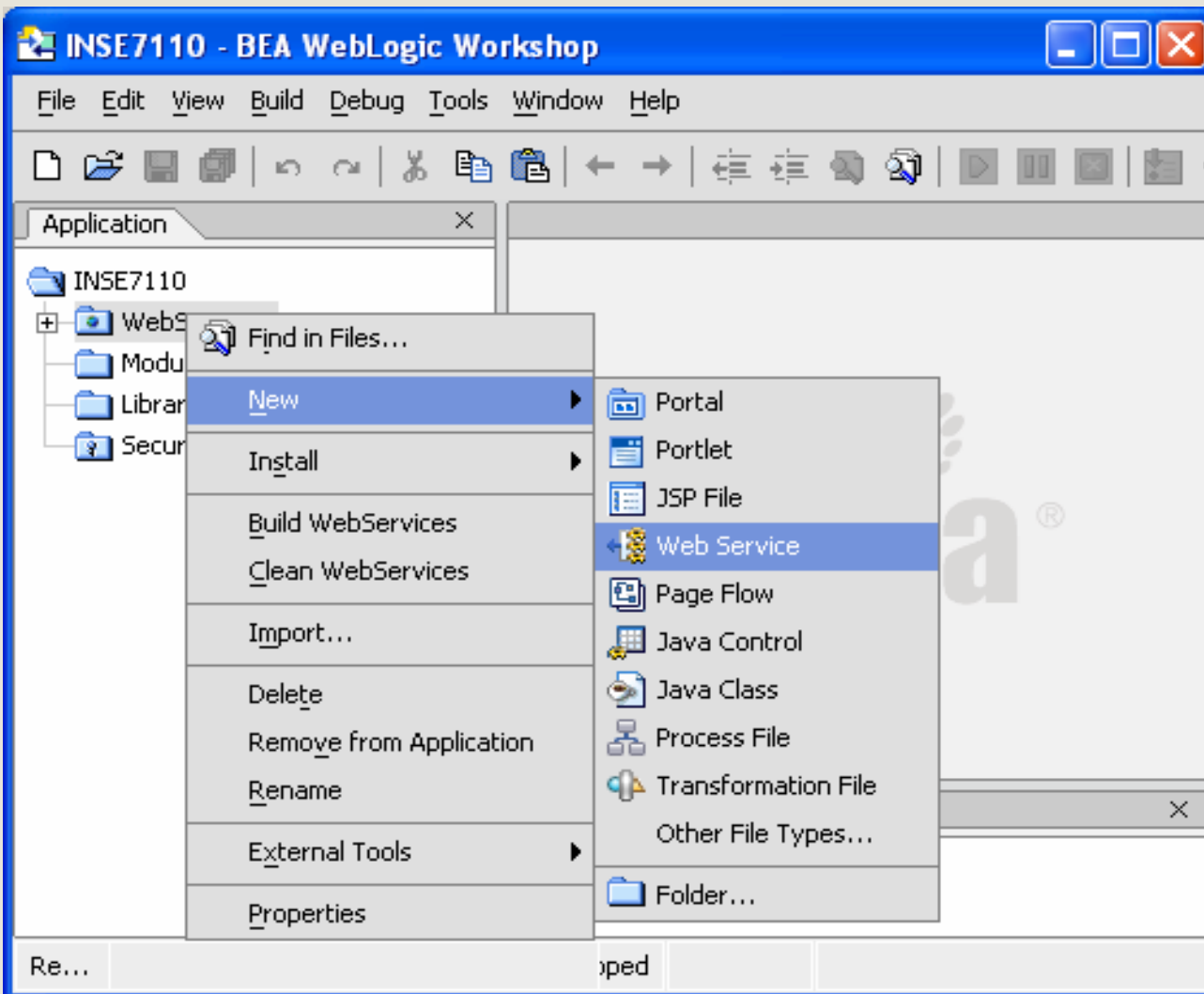
Web services development using BEA WorkShop

• Designing a Web Service

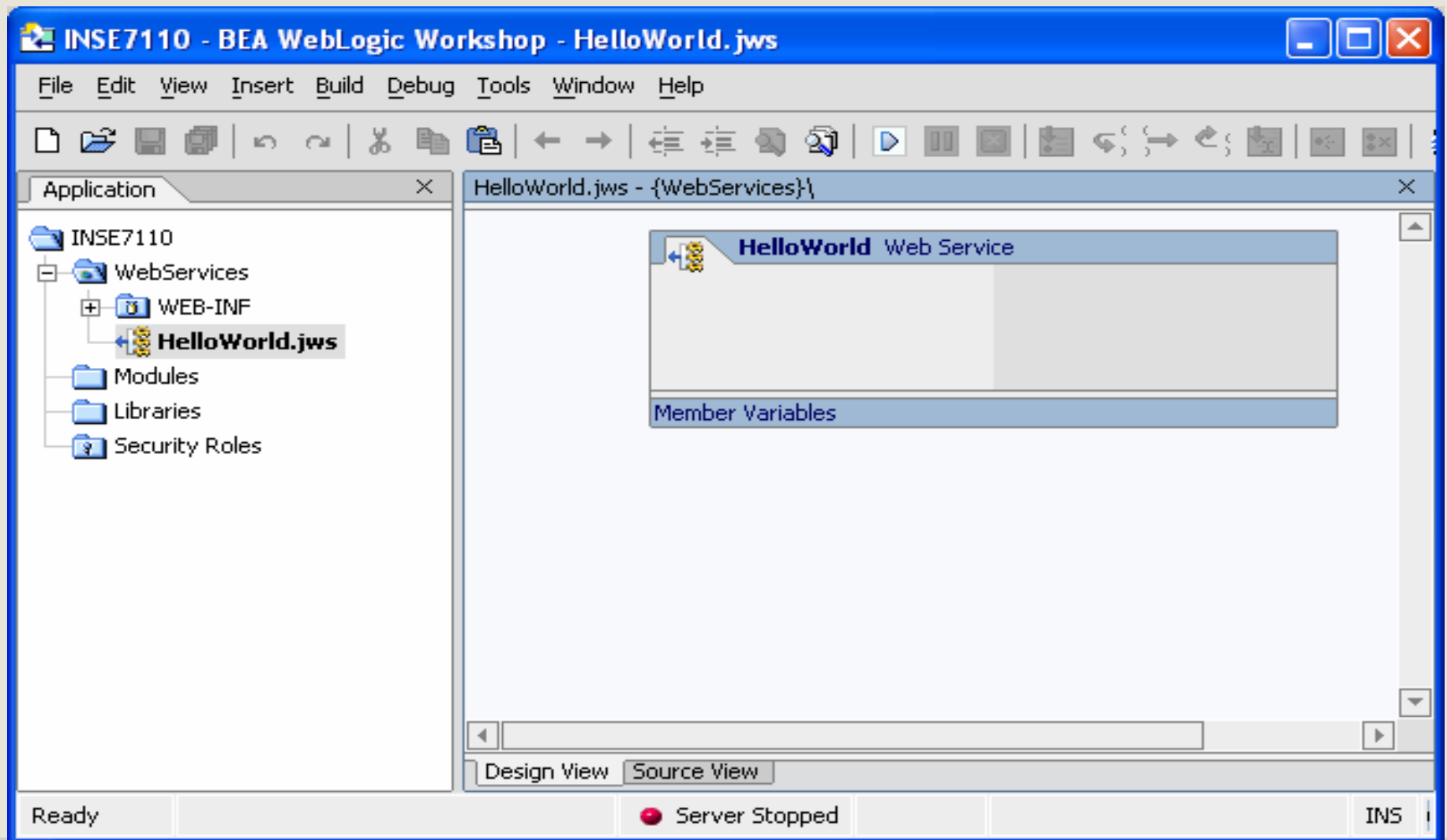
- Specify the names and parameters of all of the service's exposed operations

• Steps

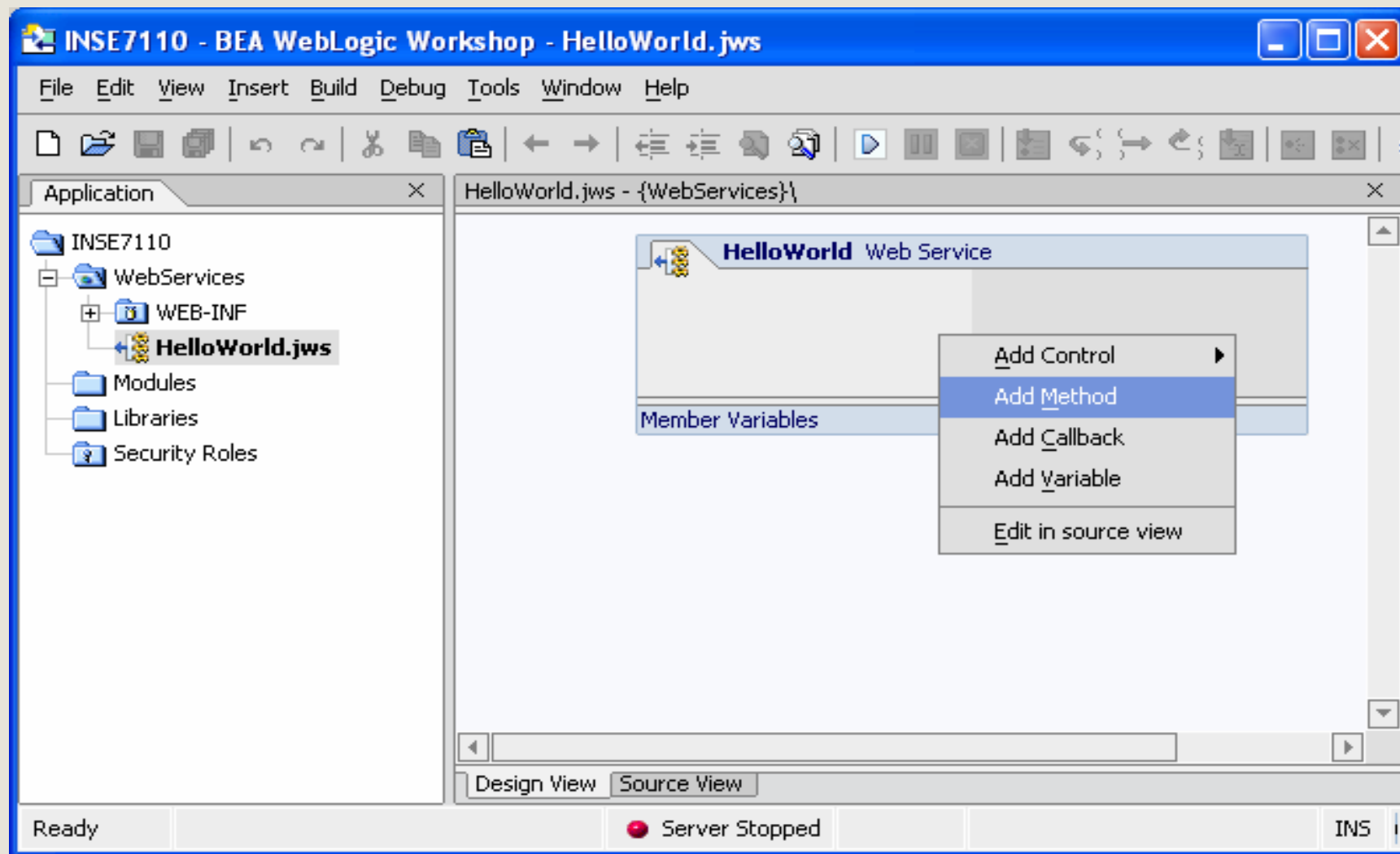
- Create a JWS file
- Add the methods
- Configure each method's parameters
- Add any callbacks and configure each callback's parameters.



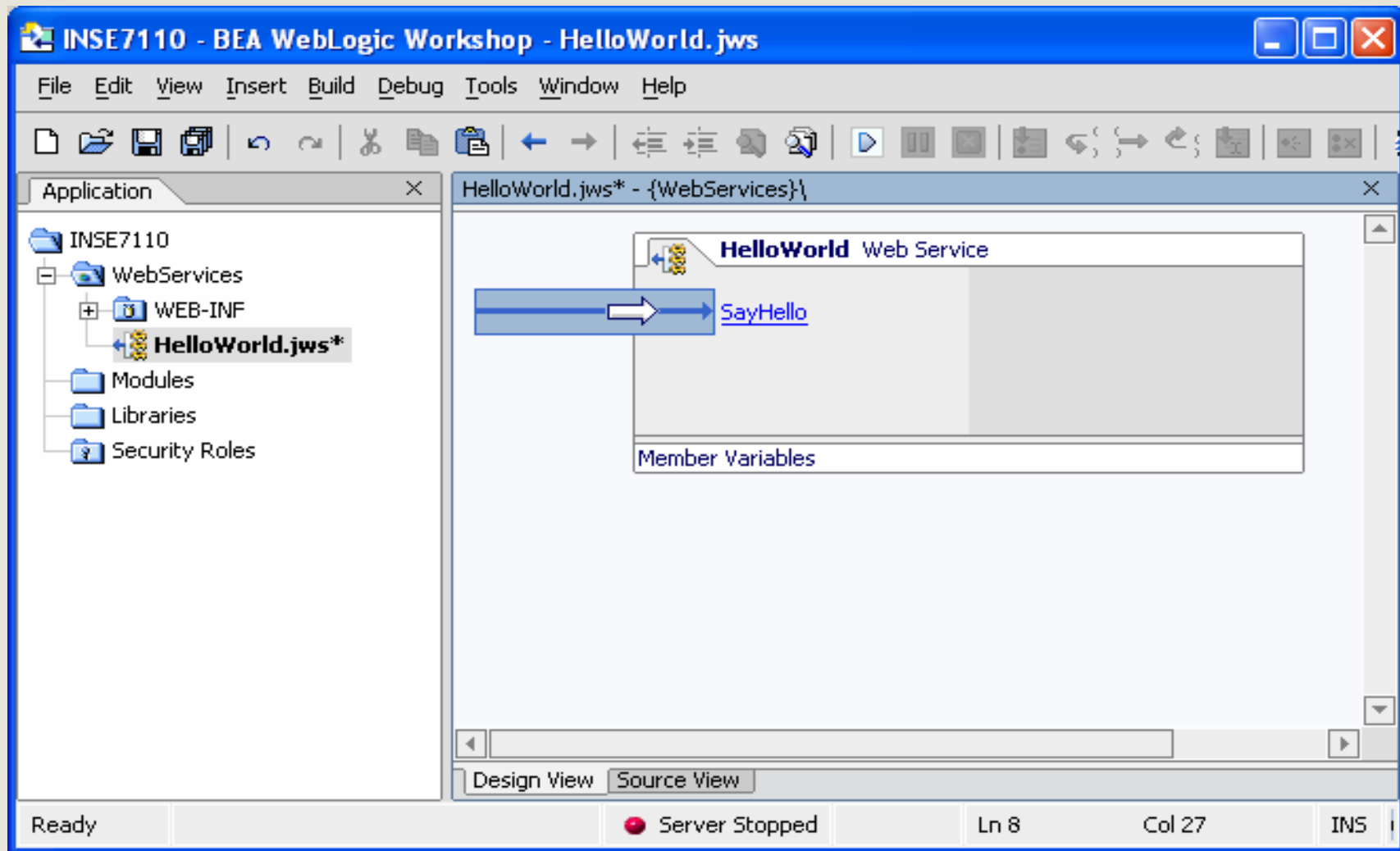
Web services development using BEA WorkShop



Web services development using BEA WorkShop

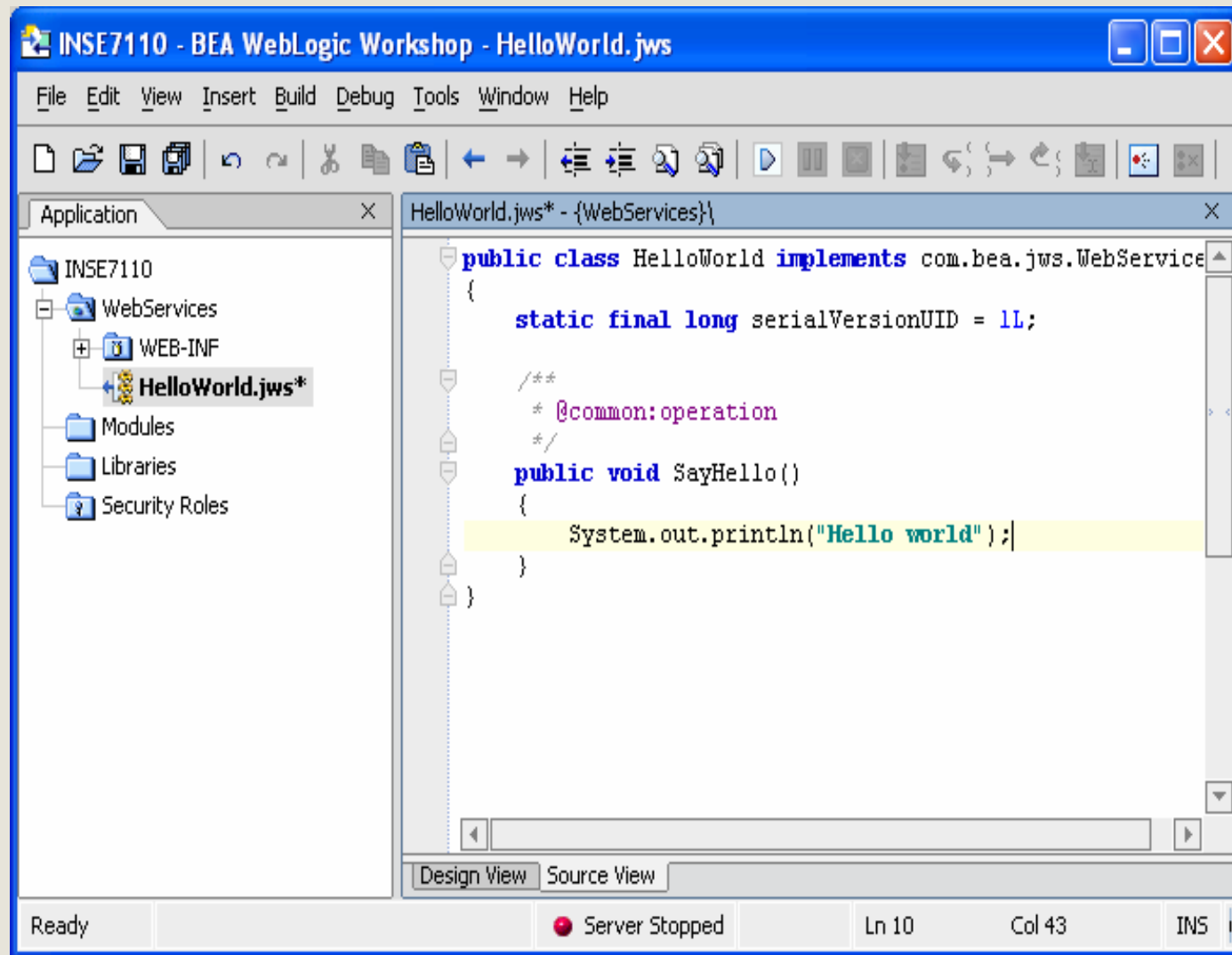


Web services development using BEA WorkShop




Web services development using BEA WorkShop

- Implementing Web Service Code
 - implement the business logic of your web service
- Using both the Design and Source Views
- Programming language
 - Java



Web services development using BEA WorkShop

- Testing a Web Service
 - Using Test View
- Test View
 - Invoke a web service's methods from a browser
 - View the XML messages that are exchanged



The screenshot displays the 'Workshop Test Browser' window. The address bar shows the URL: `http://localhost:7001/WebServices/HelloWorld.jws?.EXPLORE=.TEST`. The main content area is titled 'HelloWorld.jws Web Service' and includes the text 'Created by BEA WebLogic Workshop' and 'User: <anonymous>'. Below the title, there are tabs for 'Overview', 'Console', 'Test Form', and 'Test XML'. The 'Test Form' tab is active, showing a 'SayHello' button. To the left of the button, there is a 'Message Log' section with a 'Refresh' button and the text 'Log is empty'. The 'SayHello' button is highlighted, indicating it has been clicked.

Web services development using BEA WorkShop

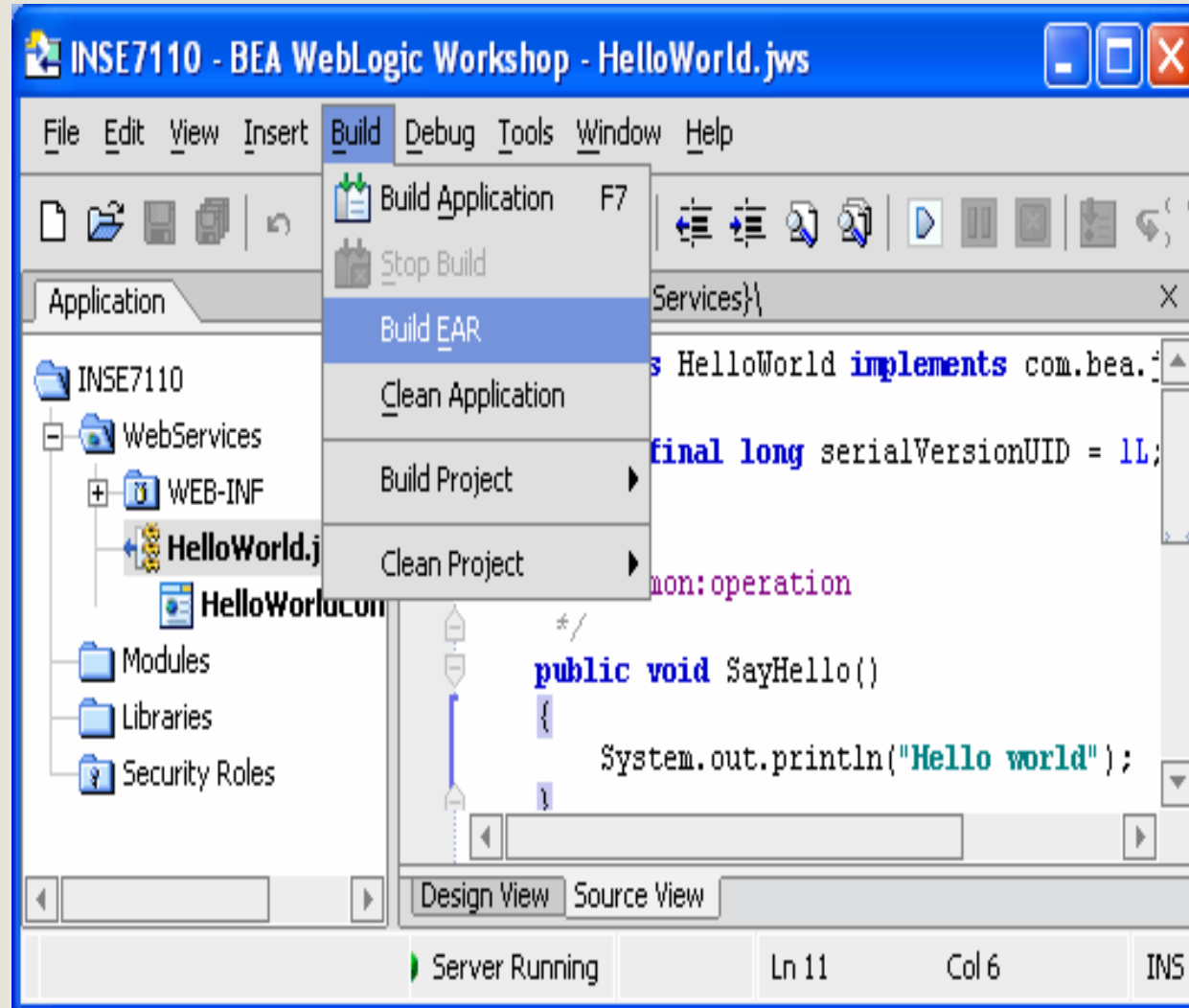


The screenshot shows the BEA Workshop Test Browser interface. The browser window title is "Workshop Test Browser". The address bar shows the URL: `http://localhost:7001/WebServices/HelloWorld.jws?.EXPLORE=.TEST&.LOGENTRY=0`. The page title is "HelloWorld.jws Web Service". The page is created by BEA WebLogic Workshop, and the user is anonymous. The page has four tabs: "Overview", "Console", "Test Form", and "Test XML". The "Test Form" tab is selected. The URL `http://localhost:7001/WebServices/HelloWorld.jws` is displayed. There is a "Test operations" link. On the left, there is a "Message Log" section with a "Refresh" button and a "Clear Log" button. The "SayHello" message is selected. The main content area shows the following log entries:

- Service Request SayHello**
Submitted at Monday, March 7, 2005 3:10:52 PM EST
- Operation SayHello**
Submitted at Monday, March 7, 2005 3:10:54 PM EST
Method: HelloWorld.SayHello
Arguments:
CallStack:
SayHello()
- Returned from SayHello**
Submitted at Monday, March 7, 2005 3:10:54 PM EST
- Service Response**
Submitted at Monday, March 7, 2005 3:10:54 PM EST
<Void xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true">
</Void>

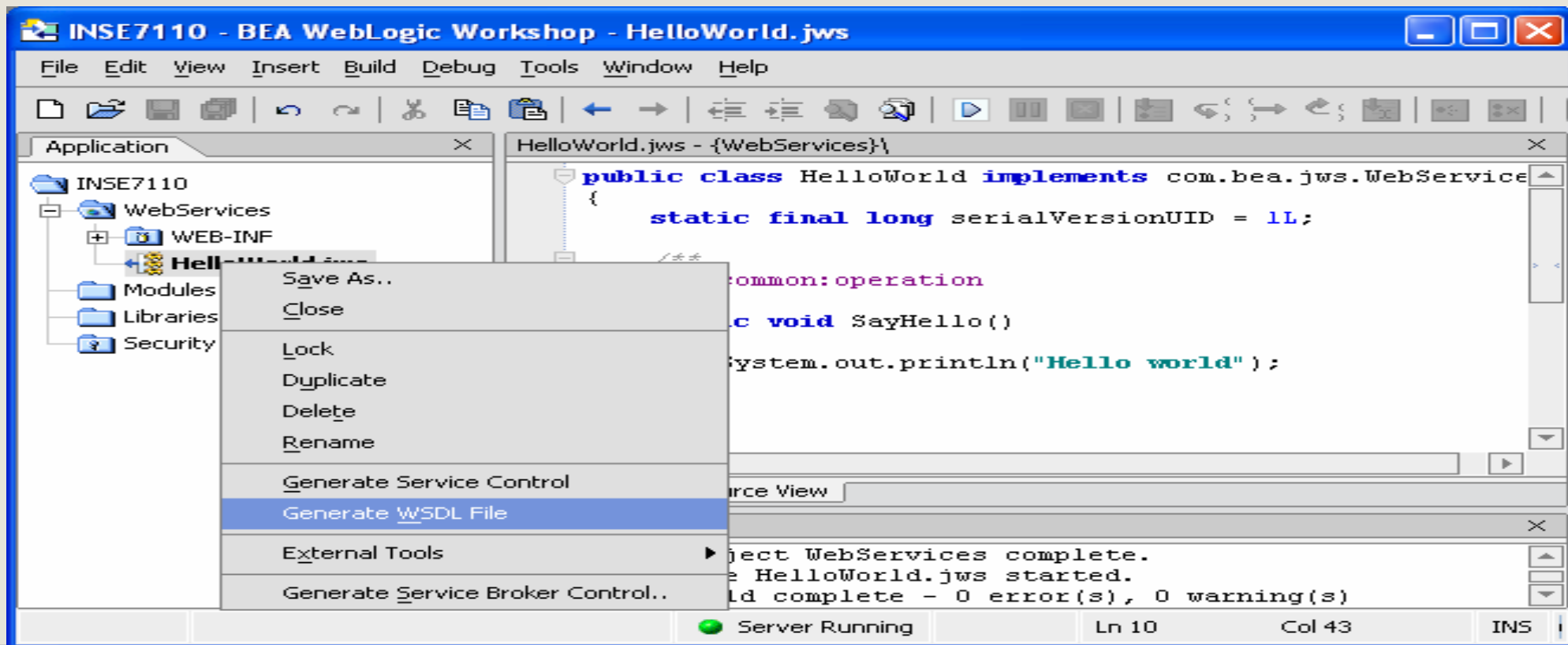
Web services development using BEA WorkShop

- Deploying a Web Service
 - Make that web service available to potential clients
- Steps
 - Package the application containing the web services as an EAR file
 - Copy the EAR file to a production server
 - From the menu bar
 - Using the **wlwBuild.cmd** from the command line tool



Web services development using BEA WorkShop

- Others
 - Create WSDL file
 - Create proxy



INSE7110 - BEA WebLogic Workshop - HelloWorldContract.wsdl

File Edit View Build Debug Tools Window Help

Application

INSE7110

- WebServices
 - WEB-INF
 - HelloWorld.jws
 - HelloWorldContract.wsdl
 - Modules
 - Libraries
 - Security Roles

HelloWorldContract.wsdl - {WebServices}

```
<?xml version="1.0" encoding="utf-8"?>
<!-- @editor-info:link autogen="true" source="HelloWorld.jws" -->
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:conv="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://schemas.xmlsoap.org/wsdl/soap/">
      <s:element name="SayHello">
        <s:complexType>
          <s:sequence/>
        </s:complexType>
      </s:element>
      <s:element name="SayHelloResponse">
        <s:complexType>
          <s:sequence/>
        </s:complexType>
      </s:element>
    </s:schema>
  </types>
  <message name="SayHelloSoapIn">
    <part name="parameters" element="s0:SayHello"/>
  </message>
  <message name="SayHelloSoapOut">
    <part name="parameters" element="s0:SayHelloResponse"/>
  </message>
  <message name="SayHelloHttpGetIn"/>
  <message name="SayHelloHttpGetOut"/>
  <message name="SayHelloHttpPostIn"/>
  <message name="SayHelloHttpPostOut"/>
  <portType name="HelloWorldSoap">
    <operation name="SayHello">
      <input message="s0:SayHelloSoapIn"/>
      <output message="s0:SayHelloSoapOut"/>
    </operation>
  </portType>
  <portType name="HelloWorldHttpGet">
    <operation name="SayHello">
      <input message="s0:SayHelloHttpGetIn"/>
      <output message="s0:SayHelloHttpGetOut"/>
    </operation>
  </portType>

```

To probe further ...

- F. Curbera et al., **Unraveling the Web services Web: An Introduction to SOAP, WSDL and UDDI**, IEEE Internet Computing, Vol. 6, No2, March-April 2002, pp. 86-93
- E. Newcomer, **Understanding Web Services: XML, WSDL, and UDDI**, Addison Wesley, 2002
- W3C specifications
- OASIS specifications (UDDI)
- <http://www.projectliberty.org/>
- <http://www.bea.com/>