AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Technologia Web Service

Systemy Rozproszone

Marek Psiuk

Katedra Informatyki AGH

Kraków, 08.01.2013

Agenda

Presentation available at:
http://home.agh.edu.pl/~mpsiuk/ws/

- Rules of the game
- Web Service theory
- Exercises

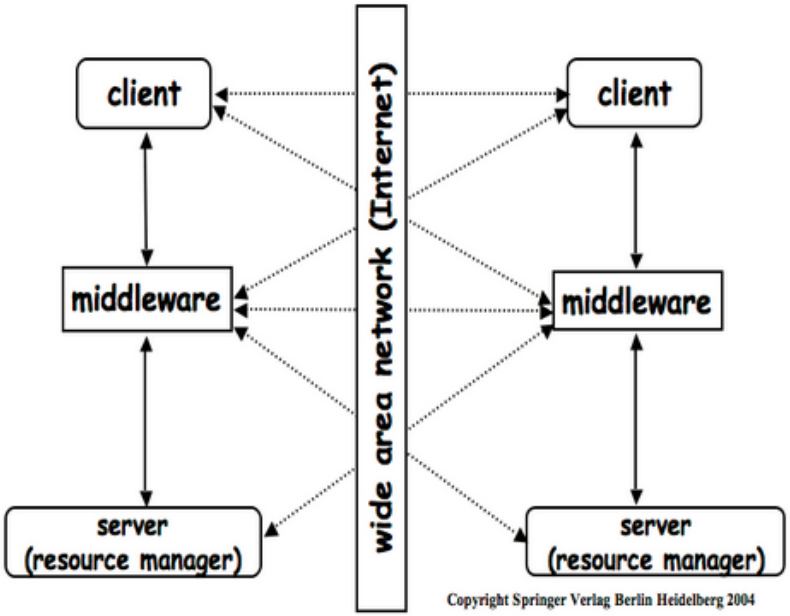
Rules of the game

- In the lab:
  - I provide You with WS theory
  - I explain the **<u>10</u> very simple** exercises
  - You do as many exercises as possible
  - You can ask for help anytime
  - Near the lab's end You present to me completed exercises. Your code and your knowledge are rated.
- At home:
  - You do the exercises which you HAVE NOT presented to me in the lab
  - You can ask for help through mail of ask for face-to-face meeting

At background

- **Launch Spring Tool Suite (f:\springsource)**
  - **Can be downloaded from http://www.springsource.org/downloads/sts**
- **Create and open fresh workspace**

Web Services - origin



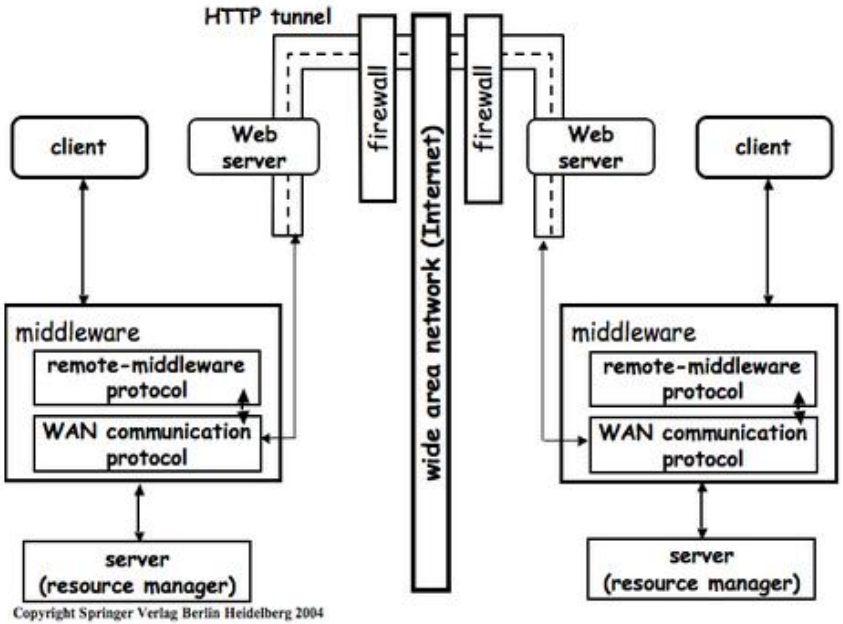
Before the Web, for inter-enterprise integration only two options existed due to lack of standardization:

- Integrate via clients, i.e. clients know about both systems
- Integrate via middleware

http://www.cs.colorado.edu/~kena/classes/7818/f06/lectures/06/

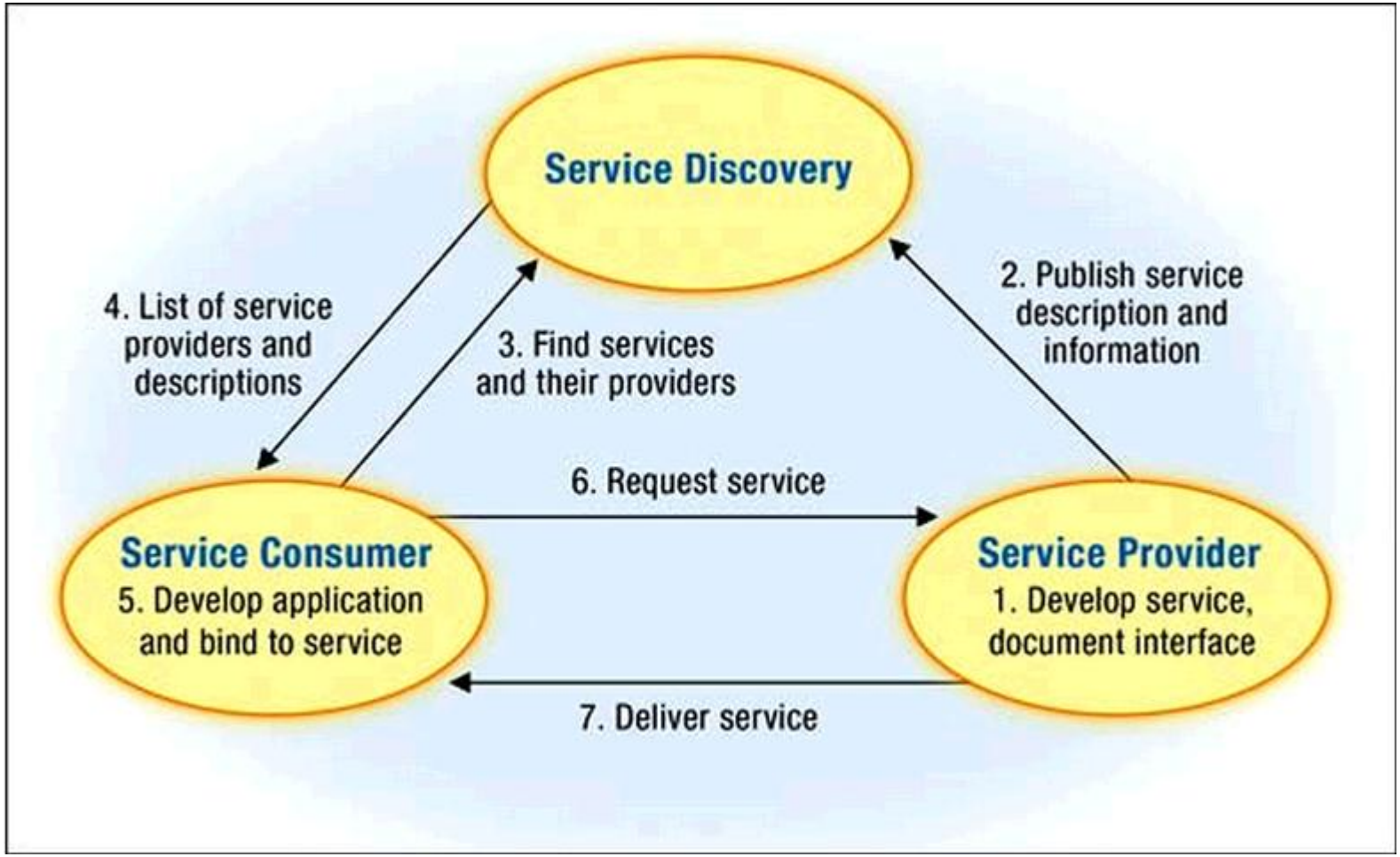
Web Services – origin

Such integration was difficult because third-party systems typically live behind a firewall.
Because Web is so popular the port 80 is rarely blocked.
As a result, companies started tunnelling the messages of other protocols by hiding them within HTML (or XML) documents and sending them to programs via the open HTTP port.

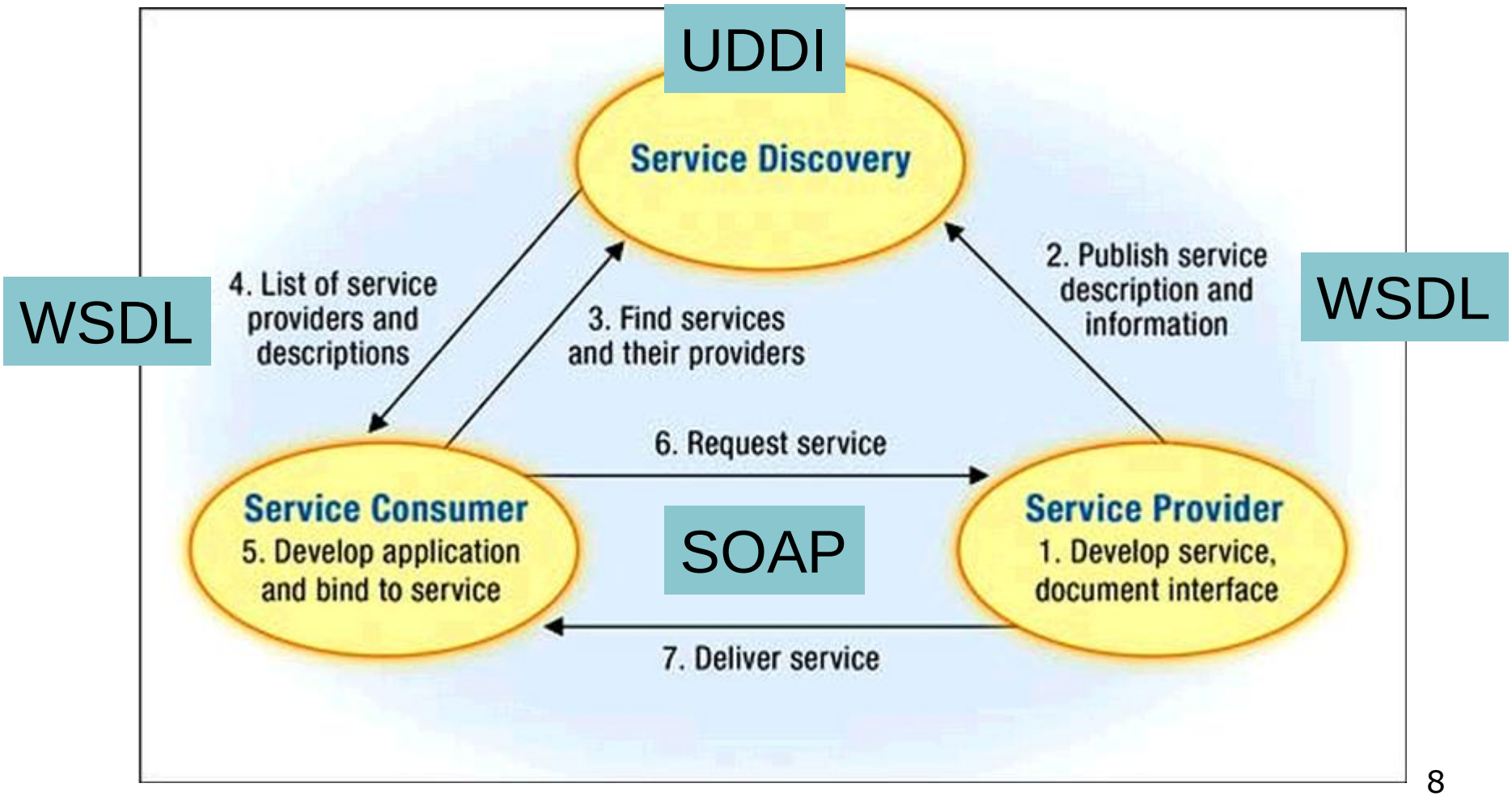


http://www.cs.colorado.edu/~kena/classes/7818/f06/lectures/06/

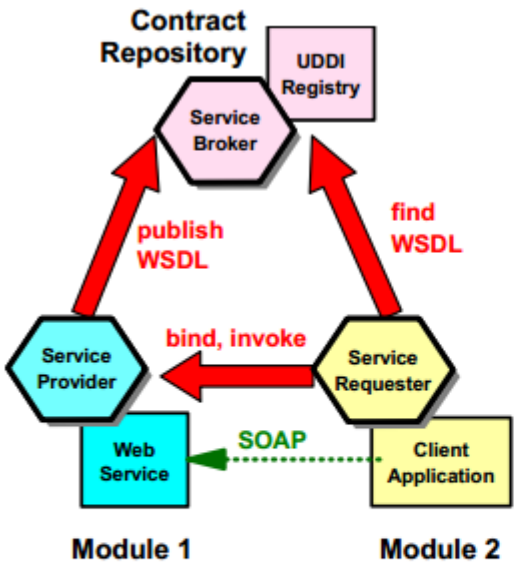
Service Oriented Architecture - Model



SOA implementation – Web Services

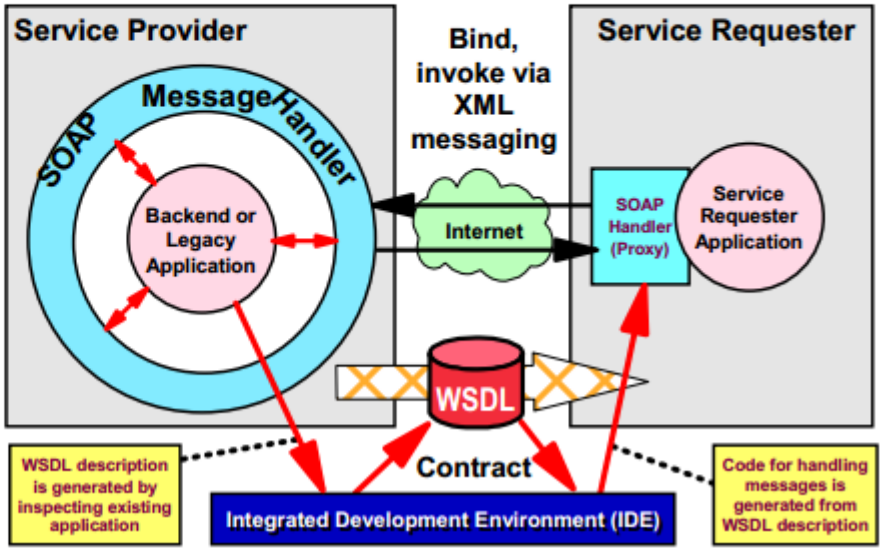


SOA with the use of Web Services: WSDL, SOAP, UDDI



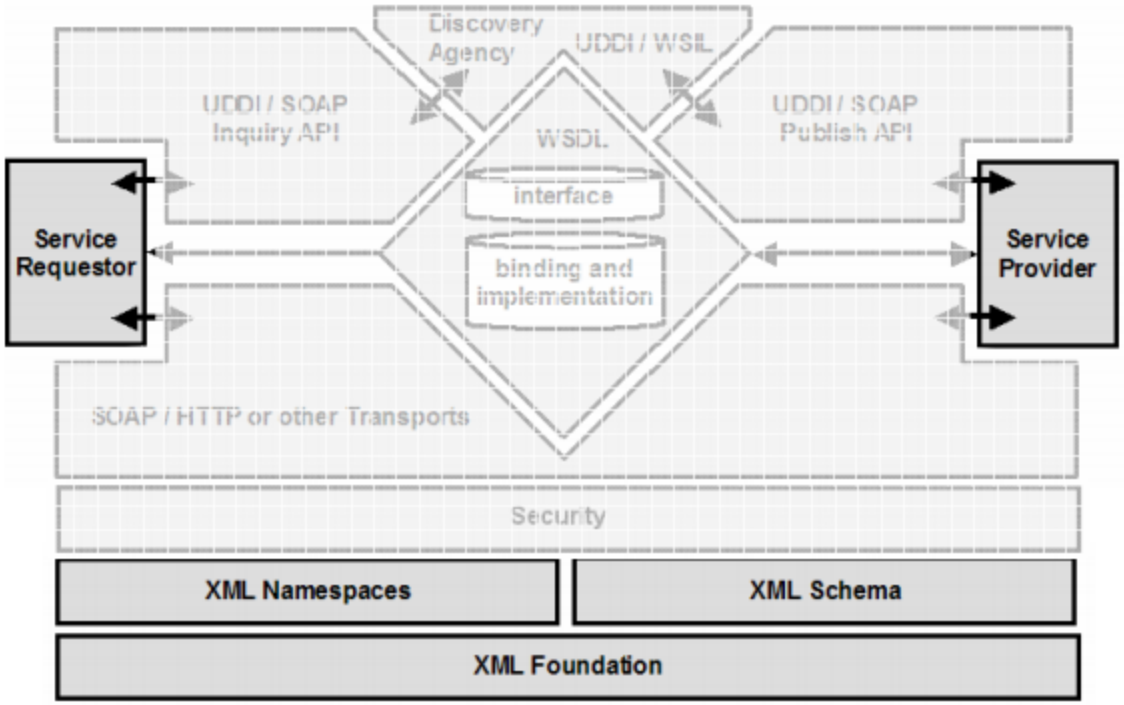
Source: [IBM ITSO]

WSDL: Web Services Description Language
SOAP: (formerly) Simple Object Access Protocol
UDDI: Universal Description, Discovery, and Integration

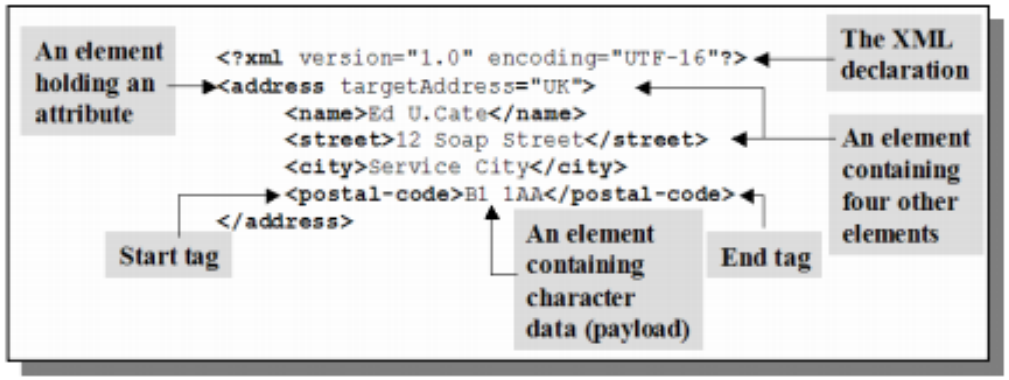


Source: [IBM ITSO]

Web Services Building Blocks: XML



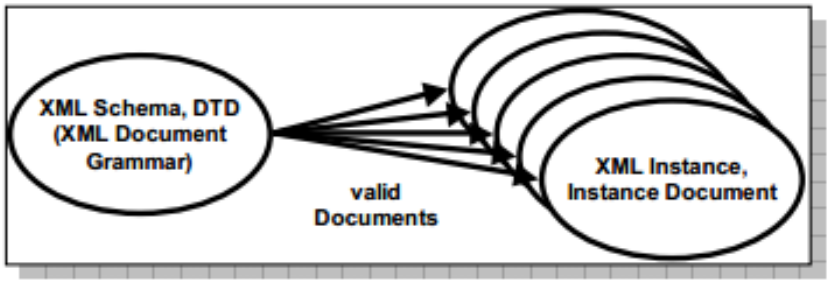
Web Services Building Blocks: XML



XML instance document example

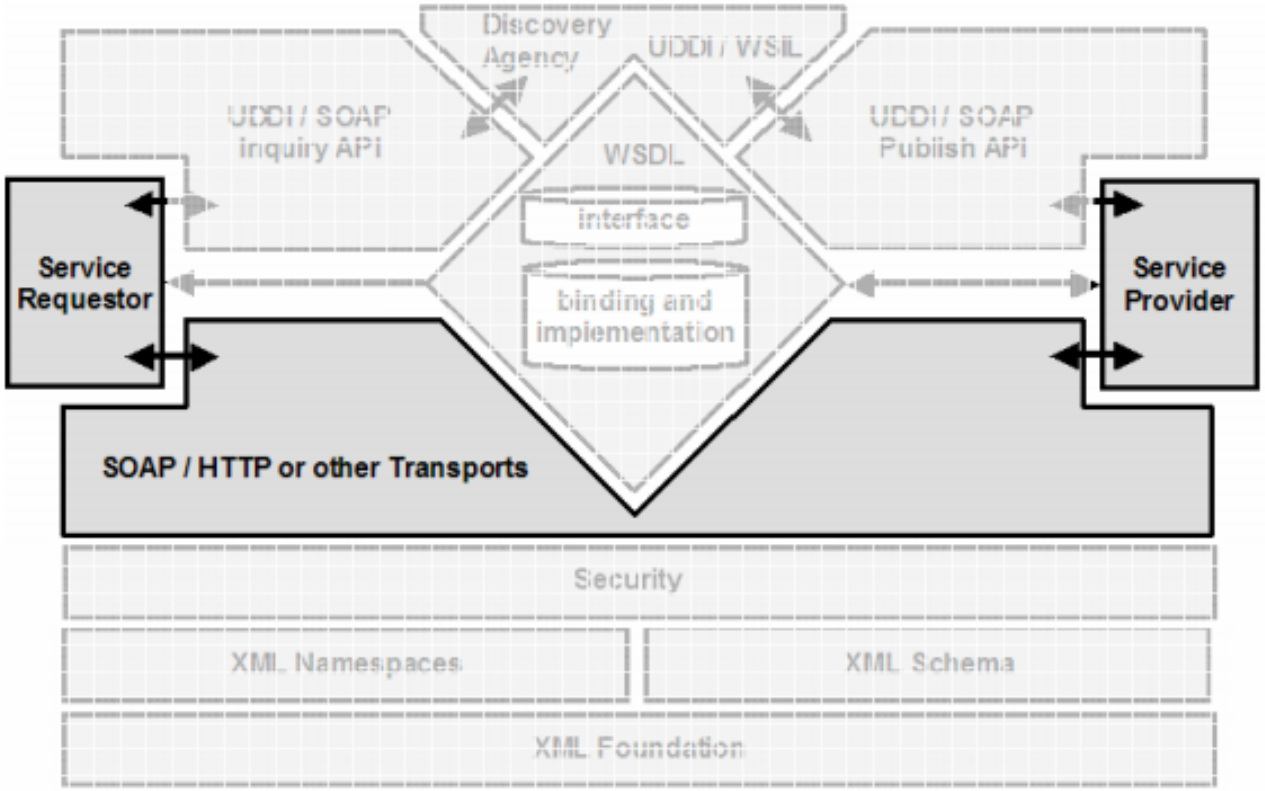
- XML [XML]
  - Markup language composed of tags and data
  - Elements and attributes
  - Read by an XML processor
  - Requires grammar definition
  - Valid and well-formed

- XML Namespaces [XMLNS]
  - Global naming mechanism for XML
  - Qualified names: prefix and local parts
  - Multiple namespaces in same document
- XML Schema [XMLSch]
  - Provides grammar for XML instance docs
  - Built-in types
  - Simple and complex custom data types



XML document grammar and valid XML instances

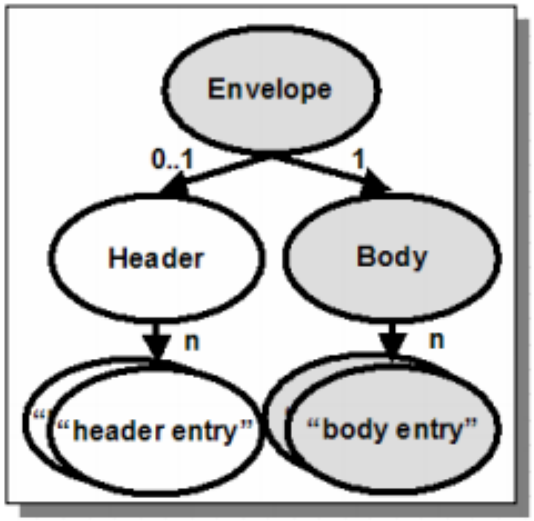
Web Services Building Blocks: SOAP



Web Services Building Blocks: SOAP

- SOAP message elements: Envelope, Headers, Message Body and Faults
- Two communication styles: Document style, RPC style
- Literal or SOAP encoding of message body plus attachments support

Reference: [SOAP]



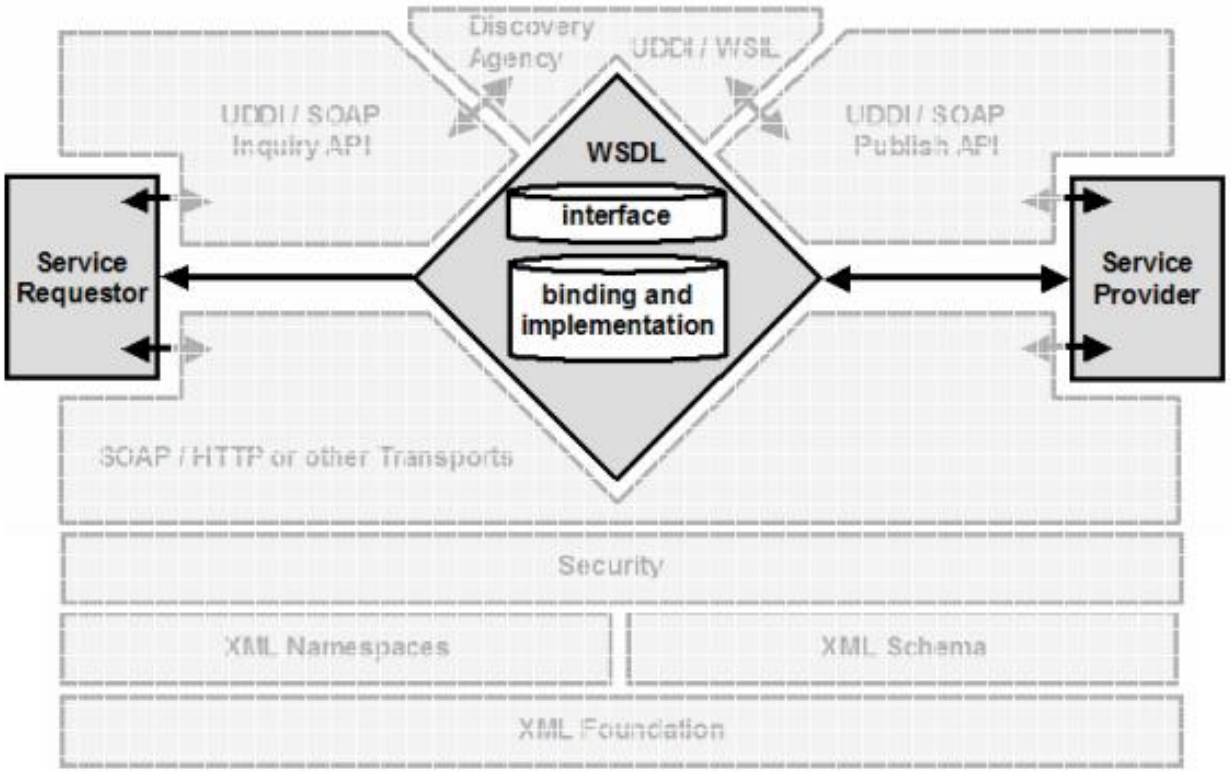
SOAP message containment structure

```
<SOAP-ENV:Envelope
                xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
                xmlns:xsd="http://www.w3.org/2001/Schema"
                xmlns:xsi="http://www.w3.org/2001/Schema-instance">
    <SOAP-ENV:Header>
                <pQP:priority
                xmlns:pQP="http://premierquotes.com/ns/priority"
                SOAP-ENV:mustUnderstand="1">
                high
                </pQP:priority>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
                <pQ:getPhoneNumber
                xmlns:pQ="http://premierquotes.com/ns/employees"
                SOAP-ENV:encodingStyle=
                "http://schemas.xmlsoap.org/soap/encoding/">
                <first-name xsi:type="xsd:string">Ed U.</first-name>
                <last-name xsi:type="xsd:string">Cate</last-name>
                </pQ:getPhoneNumber>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

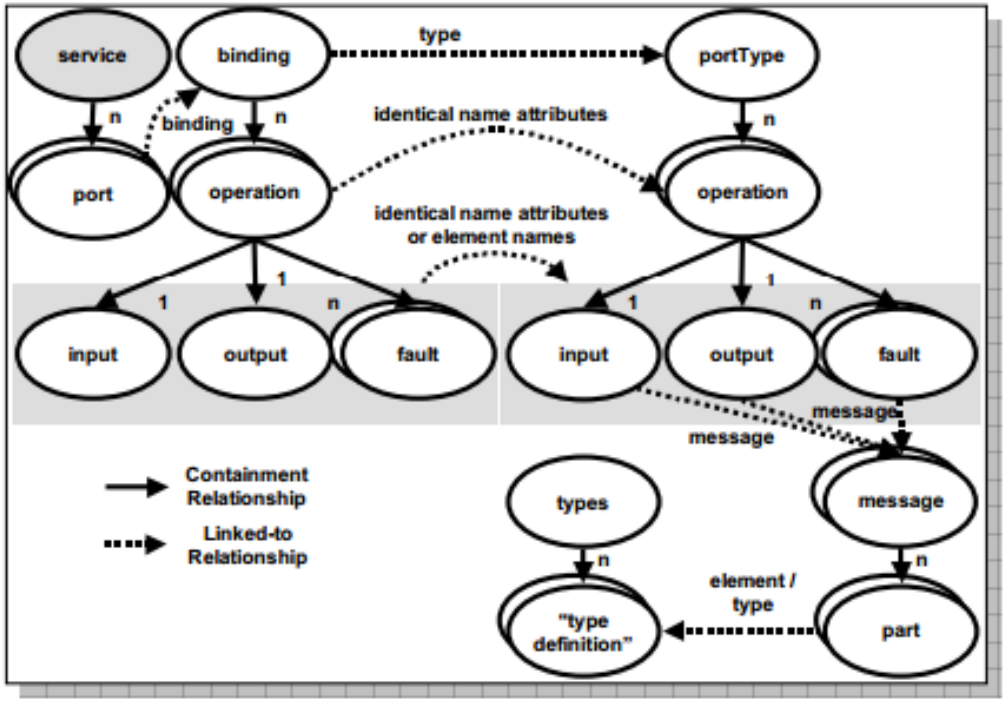
Header entry
Header must be processed
Method invocation
Body entry
Use SOAP encoding
Method parameters
Method parameter types
Method parameter values

SOAP message example

Web Services Building Blocks: WSDL



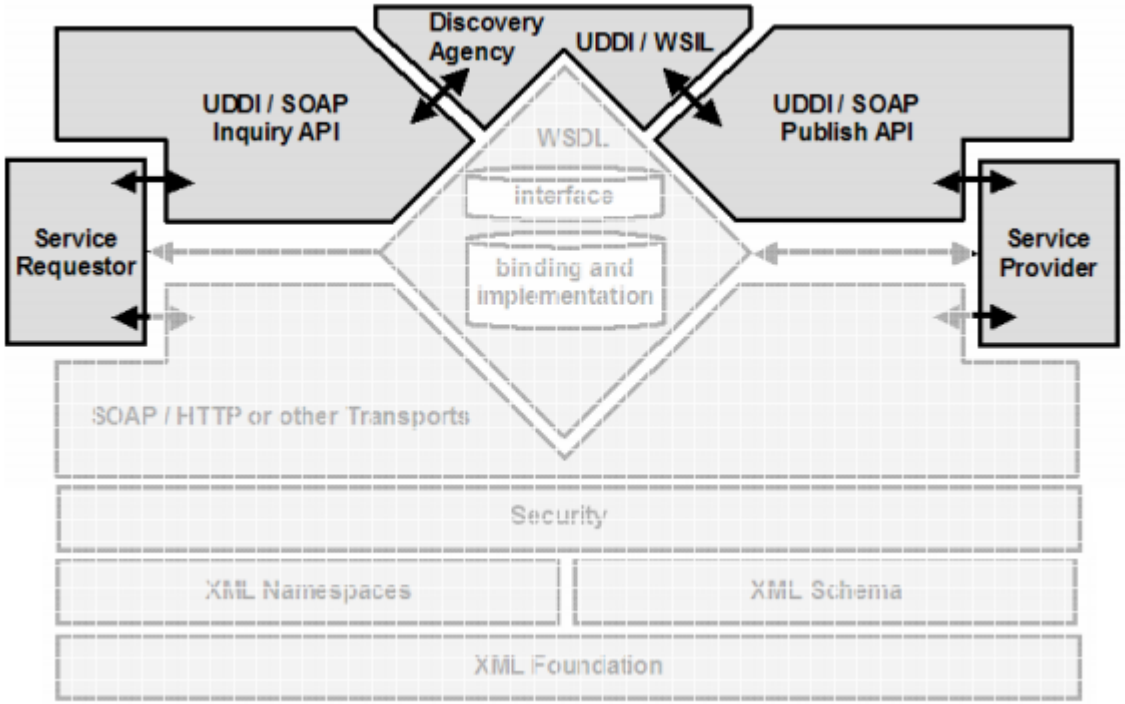
Web Services Building Blocks: WSDL



Logical relationships between WSDL elements

- WSDL document elements
  - Type definitions and imports
  - Interface description (Port Type, Operations, Messages)
  - Extensible binding section
  - Implementation description (Ports)
- WSDL SOAP binding
  - Defines header and fault support
  - Extensibility element for addressing
- HTTP binding also defined

Web Services Building Blocks: UDDI … and others



- Was not extensively adopted. Complementary approaches:
  - WS-Inspection (http://www.ibm.com/developerworks/webservices/library/ws-wsiluddi/index.html)
  - WS-Discovery (http://en.wikipedia.org/wiki/WS-Discovery)
  - JAXR (http://docs.oracle.com/javaee/1.3/tutorial/doc/JAXR2.html)

Web Service design approaches

There can be distinguished two main way od creating WS

- contract-first (top-down) - first formal description of services expressed in therms fo WSDL document is created, than the implementation is provided/generated
- code-first (bottom-up) - first implementation is provided, then portable artefacts (WSDL,XSD) are generated

Pros and cons of both approaches?

Java Web Service Runtime environments

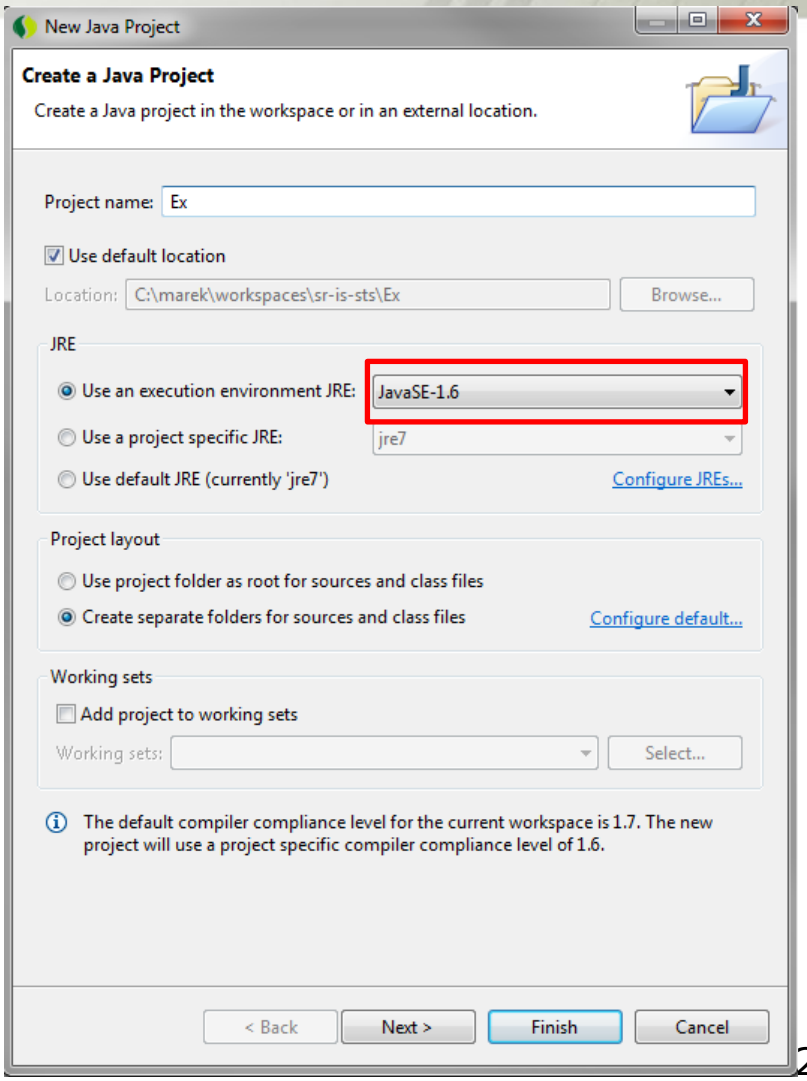
- Apache Axis 2 - The Axis2 web application itself is a container for Web Services. Web Services are packed into an own file format with the file extension aar. Implementations of Axis2 are available in Java and C. Services are configured using the service.xml configuration file
- Apache CXF - CXF was developed with the intention to simply integrate it into other, existsing systems. This is reflected in the CXF API and the use of the Spring framework. For instance CXF respectively its predecessor XFire was integrated into numerous open and closed source projects like ServiceMix or Mule. Custom API as well as the standardized JAX-WS interface are available for the development and use of Web Services.
- Sun Metro - reference implementation of JAX-WS 2.0

JAX-WS Type Mapping

JAX-WS delegates the mapping of Java programming language types to and from XML definitions to JAXB. Application developers don't need to know the details of these mappings, but they should be aware that not every class in the Java language can be used as a method parameter or return type in JAX-WS.

Exercises Hint

- Always Java 6 and NOT 7



Exercise 1

- The simplest hello world - method with 3 arguments
- Looking into online WSDL (available at: http://localhost:8181/SimplestHelloService?wsdl)
- Use Eclipse Web Service Explorer to test the server side (http://www.eclipse.org/webtools/jst/components/ws/1.0/tutorials/WebServiceExplorer/WebServiceExplorer.html)
- TCP/IP Monitor (http://www.mkyong.com/webservices/jax-ws/how-to-trace-soap-message-in-eclipse-ide/)
  - When using TCP/IP Monitor remember to change the port called by the client. Can be done as follows:

Client.java

```
package agh.sr.ws.simplest;

import java.net.MalformedURLException;

public class Client {
    public static void main(String[] args) throws MalformedURLException {
        Service srv = Service.create(new URL(
                "http://localhost:8181/SimplestHelloService"), new QName(
                "http://simplest.ws.sr.agh/", "SimplestHelloImplService"));
        SimplestHello simplestClient = srv.getPort(SimplestHello.class);
        ((BindingProvider) simplestClient).getRequestContext().put(
                BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                "http://localhost:9191/SimplestHelloService");
        System.out.println(simplestClient.sayHello("The Client", 23,
                (float) 100.0));
    }
}
```

Exercise 1

Package Explorer

- Ex1
  - src
    - agh.sr.ws.simplest
      - Client.java
      - Server.java
      - SimplestHello.java
      - SimplestHelloImpl.java
  - JRE System Library [JavaSE-1.7]

SimplestHello.java

```java
package agh.sr.ws.simplest;

import javax.jws.WebService;

@WebService
@SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL) // this is the defualt binding
public interface SimplestHello {

    public String sayHello(String sender, int age, float salary);

}
```

SimplestHelloImpl.java

```java
package agh.sr.ws.simplest;

import javax.jws.WebService;

@WebService(endpointInterface = "agh.sr.ws.simplest.SimplestHello")
public class SimplestHelloImpl implements SimplestHello {

    @Override
    public String sayHello(String sender, int age, float salary) {
        String ret = "Saying hello to: " + sender + " at age: " + age
                + "with salary: " + salary;
        System.out.println("Processing request with the following result:" + ret);
        return ret;
    }
}
```

Exercise 1

Package Explorer

- Ex1
  - src
    - agh.sr.ws.simplest
      - Client.java
      - Server.java
      - SimplestHello.java
      - SimplestHelloImpl.java
  - JRE System Library [JavaSE-1.7]

Server.java

```java
package agh.sr.ws.simplest;

import javax.xml.ws.Endpoint;

public class Server {
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:8181/SimplestHelloService",
                new SimplestHelloImpl());
    }
}
```

Client.java

```java
package agh.sr.ws.simplest;

import java.net.MalformedURLException;

public class Client {
    public static void main(String[] args) throws MalformedURLException {
        Service srv = Service.create(new URL(
                "http://localhost:8181/SimplestHelloService"), new QName(
                "http://simplest.ws.sr.agh/", "SimplestHelloImplService"));
        SimplestHello simplestClient = srv.getPort(SimplestHello.class);
        System.out.println(simplestClient.sayHello("The Client", 23,
                (float) 100.0));
    }
}
```

Exercise 1

When analyzing WSDL in document literal style do not forget to analyze imported schema

localhost:8181/SimplestHe

localhost:8181/SimplestHelloService?wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<!--
    Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
  -->
▼<!--
    Generated by JAX-WS RI at http://jax-ws.dev.java.net. RI's version is JAX-WS RI 2.2.4-b01.
  -->
▼<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http:/
  xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns
  xmlns:tns="http://simplest.ws.sr.agh/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/wsdl/"
  name="SimplestHelloImplService">
  ▼<types>
    ▼<xsd:schema>
      <xsd:import namespace="http://simplest.ws.sr.agh/" schemaLocation="http://localhost:8181/SimplestHelloService?xsd=1"/>
    </xsd:schema>
  </types>
  ▼<message name="sayHello">
    <part name="parameters" element="tns:sayHello"/>
  </message>
  ▼<message name="sayHelloResponse">
    <part name="parameters" element="tns:sayHelloResponse"/>
  </message>
  ▼<portType name="SimplestHello">
    ▼<operation name="sayHello">
      <input wsam:Action="http://simplest.ws.sr.agh/SimplestHello/sayHelloRequest" message="tns:sayHello"/>
      <output wsam:Action="http://simplest.ws.sr.agh/SimplestHello/sayHelloResponse" message="tns:sayHelloResponse"/>
    </operation>
  </portType>
  ▼<binding name="SimplestHelloImplPortBinding" type="tns:SimplestHello">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    ▼<operation name="sayHello">
      <soap:operation soapAction=""/>
      ▼<input>
        <soap:body use="literal"/>
      </input>
      ▼<output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  ▼<service name="SimplestHelloImplService">
    ▼<port name="SimplestHelloImplPort" binding="tns:SimplestHelloImplPortBinding">
      <soap:address location="http://localhost:8181/SimplestHelloService"/>
    </port>
  </service>
</definitions>
```

Exercise 1

The schema can be simply accessed from the browser in the same way as the WSDL file

localhost:8181/SimplestH × localhost:8181/SimplestH ×

localhost:8181/SimplestHelloService?xsd=1

This XML file does not appear to have any style information associated witl

```
▼<!--
    Published by JAX-WS RI at http://jax-ws.dev.java.net. RI's versi
 -->
▼<xs:schema xmlns:tns="http://simplest.ws.sr.agh/" xmlns:xs="http://
   <xs:element name="sayHello" type="tns:sayHello"/>
   <xs:element name="sayHelloResponse" type="tns:sayHelloResponse"/>
  ▼<xs:complexType name="sayHello">
    ▼<xs:sequence>
       <xs:element name="arg0" type="xs:string" minOccurs="0"/>
       <xs:element name="arg1" type="xs:int"/>
       <xs:element name="arg2" type="xs:float"/>
     </xs:sequence>
   </xs:complexType>
  ▼<xs:complexType name="sayHelloResponse">
    ▼<xs:sequence>
       <xs:element name="return" type="xs:string" minOccurs="0"/>
     </xs:sequence>
   </xs:complexType>
 </xs:schema>
```

Exercise 2

- The simplest hello world with two methods as arguments using RPC Literal Binding
- Copy Exercise 1 and change SOAPBinding
- Looking into online WSDL
  - What's the difference from the Document Literal style
- TCP/IP Monitor
  - What is the difference in the transmitted SOAP message with Excercise 1
- Hint:

http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

Exercise 2

SimplestHello.java

```java
package agh.sr.ws.simplest;

import javax.jws.WebService;

@WebService
@SOAPBinding(style = Style.RPC, use = Use.LITERAL)
public interface SimplestHello {

    public String sayHello(String sender, int age, float salary);

}
```

Exercise 3

- Copy Exercise 1
- Generating WSDL an analyzing it (wsgen tool)
- Importing WSDL and analyzing generated classes (wsimport tool)
- Adding server and client which use generated classes and not the ones crafted from scratch (not sure if possible in server)

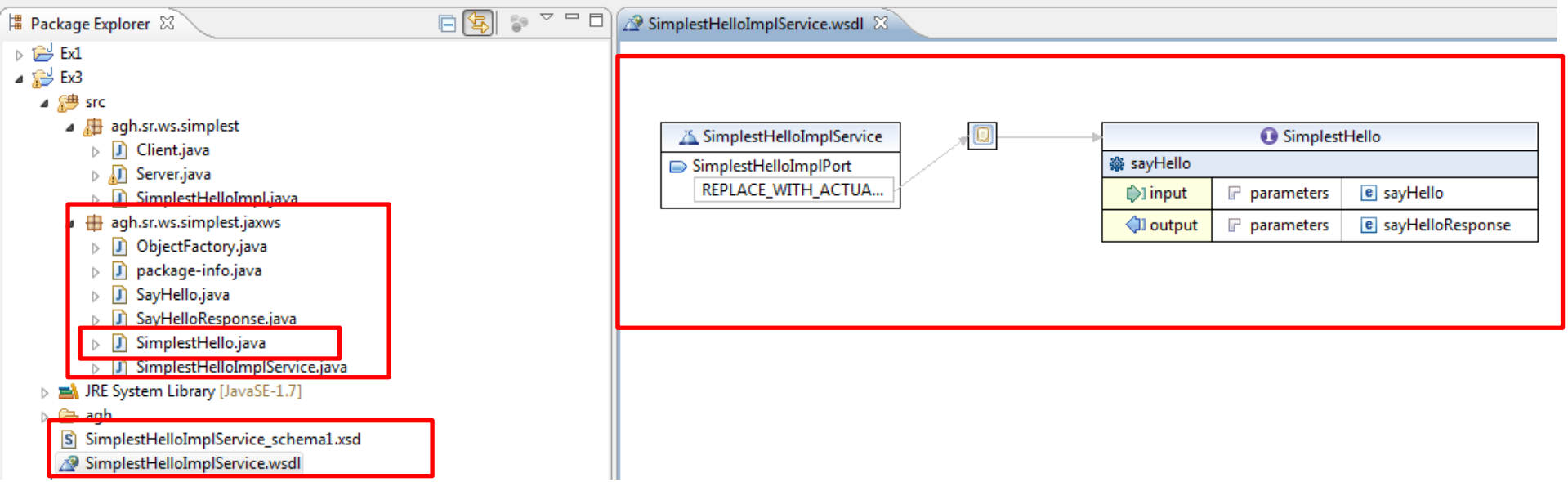
- Hints:

http://www.mkyong.com/webservices/jax-ws/jax-ws-wsgen-tool-example/
http://www.mkyong.com/webservices/jax-ws/jax-ws-wsimport-tool-example/

W katalogu zadania:

- wsgen -cp bin -wsdl agh.sr.ws.simplest.SimplestHelloImpl
- wsimport -d bin -s src -p agh.sr.ws.simplest.jaxws SimplestHelloImplService.wsdl

Exercise 3


Package Explorer
Ex1
Ex3
src
agh.sr.ws.simplest
Client.java
Server.java
SimplestHelloImpl.java
agh.sr.ws.simplest.jaxws
ObjectFactory.java
package-info.java
SayHello.java
SayHelloResponse.java
SimplestHello.java
SimplestHelloImplService.java
JRE System Library [JavaSE-1.7]
agh
SimplestHelloImplService_schema1.xsd
SimplestHelloImplService.wsdl
SimplestHelloImplService.wsdl
SimplestHelloImplService
SimplestHelloImplPort
REPLACE_WITH_ACTUA...
SimplestHello
sayHello
input
parameters
sayHello
output
parameters
sayHelloResponse

Exercise 3

Package Explorer

- Ex1
- Ex3
  - src
    - agh.sr.ws.simplest
      - Client.java
      - Server.java
      - SimplestHelloImpl.java
    - agh.sr.ws.simplest.jaxws
      - ObjectFactory.java
      - package-info.java
      - SayHello.java
      - SayHelloResponse.java
      - SimplestHello.java
      - SimplestHelloImplService.java
  - JRE System Library [JavaSE-1.7]
  - agh
  - SimplestHelloImplService_schema1.xsd
  - SimplestHelloImplService.wsdl

Client.java

```java
package agh.sr.ws.simplest;

import java.net.MalformedURLException;

public class Client {
    public static void main(String[] args) throws MalformedURLException {
        SimplestHello simplestHelloClient = new SimplestHelloImplService()
                .getSimplestHelloImplPort();
        ((BindingProvider) simplestHelloClient).getRequestContext().put(
                BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
                "http://localhost:9191/SimplestHelloService");
        System.out.println(simplestHelloClient.sayHello("The Client", 23,
                (float) 100.0));
    }
}
```

SimplestHelloImpl.java

```java
package agh.sr.ws.simplest;

import javax.jws.WebService;

@WebService(endpointInterface = "agh.sr.ws.simplest.jaxws.SimplestHello")
public class SimplestHelloImpl implements SimplestHello {

    @Override
    public String sayHello(String sender, int age, float salary) {
        String ret = "Saying hello to: " + sender + " at age: " + age
                + "with salary: " + salary;
        System.out.println("Processing request with the following result:"
                + ret);
        return ret;
    }
}
```

Exercise 4

- Copy Exercise 2 – RPC LITERAL Style
- Generating WSDL an analyzing it in eclispe
- Importing WSDL and analyzing generated classes
  - What is the difference between the WSDL from Exercise 3
  - What is the difference between generated classes?

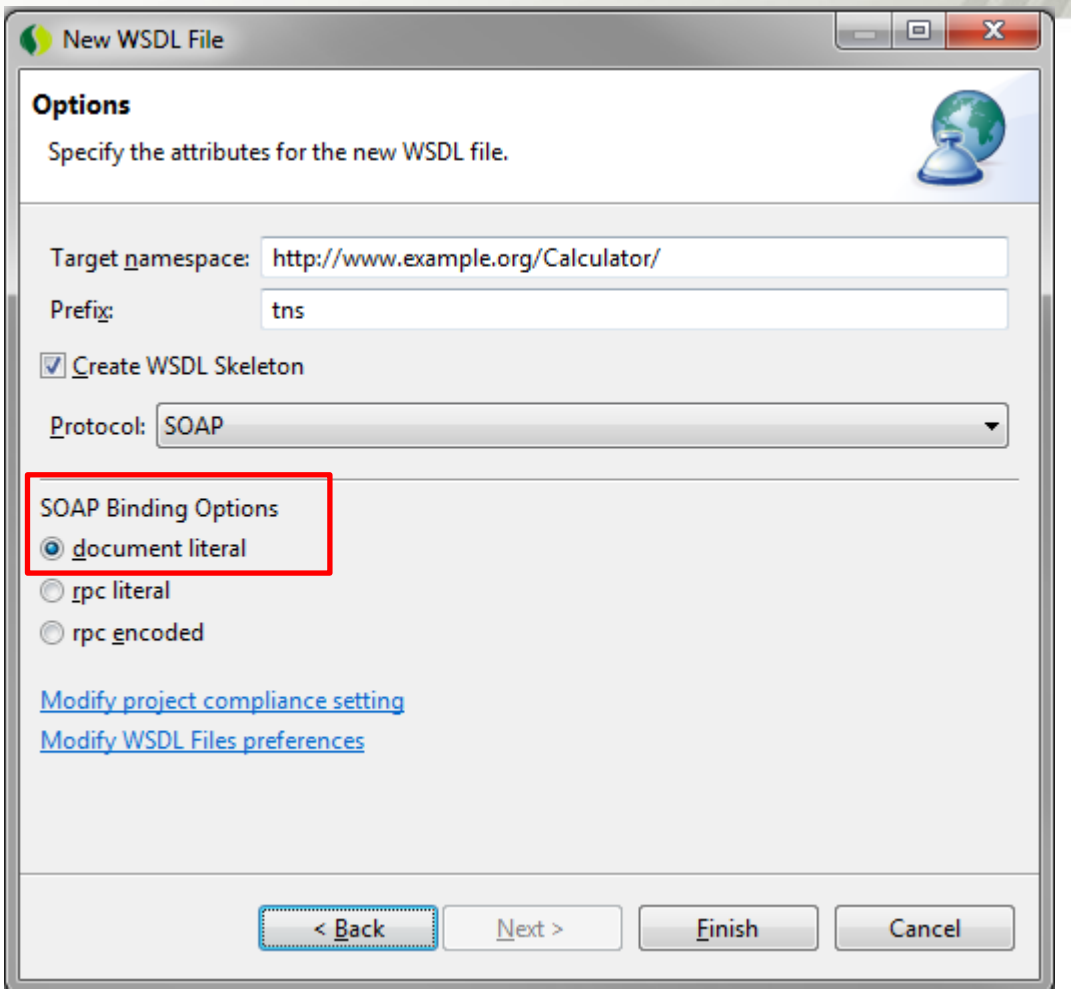
http://www.mkyong.com/webservices/jax-ws/jax-ws-wsgen-tool-example/

http://www.mkyong.com/webservices/jax-ws/jax-ws-wsimport-tool-example/

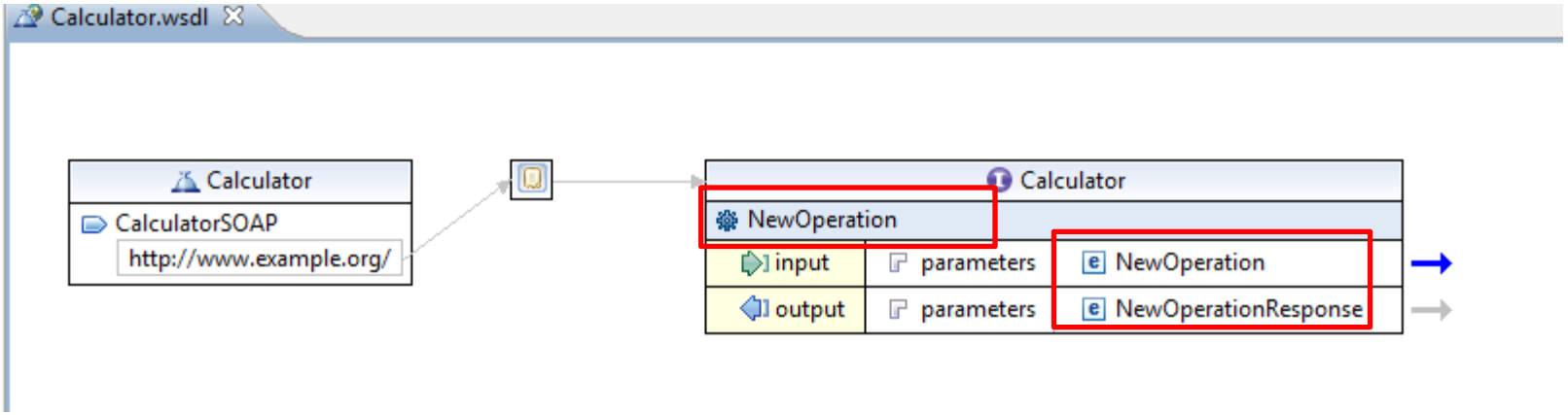
Exercise 5

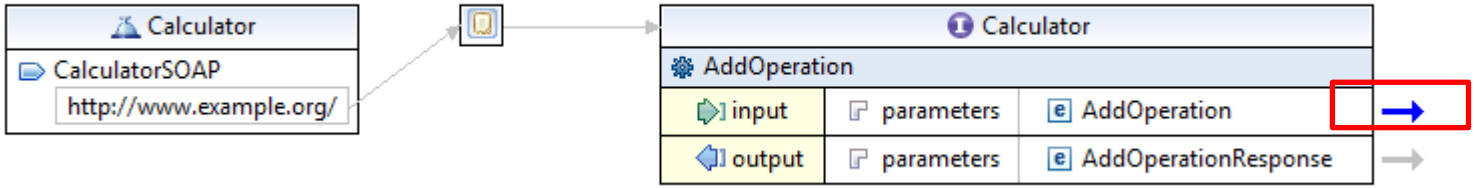
- Create fresh project
- Design a WSDL in **document literal style** for Calculator service with operations:
  - int AddOperation(int a, int b, int c)
  - double DivideOperation(double dividend, double divisor)
- Use wsimport to generate classes from WSDL
- Implement Server and Client
- Analyze with TCP/IP Monitor

Exercise 5

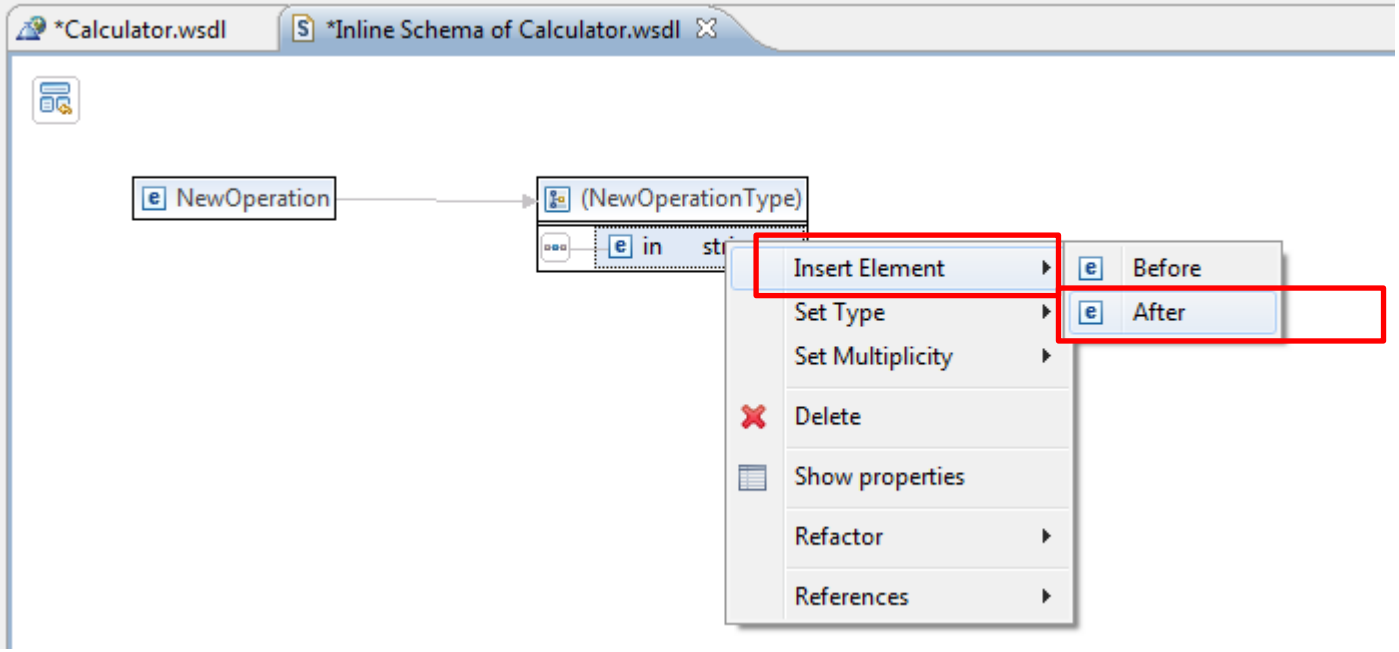

New WSDL File
Options
Specify the attributes for the new WSDL file.
Target namespace:
http://www.example.org/Calculator/
Prefix:
tns
Create WSDL Skeleton
Protocol:
SOAP
SOAP Binding Options
document literal
rpc literal
rpc encoded
Modify project compliance setting
Modify WSDL Files preferences
< Back
Next >
Finish
Cancel

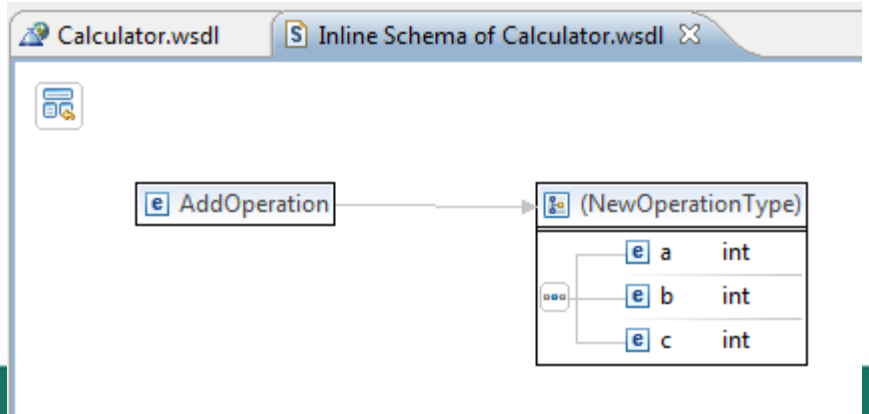
Exercise 5


Calculator.wsdl
Calculator
CalculatorSOAP
http://www.example.org/
Calculator
NewOperation
input
parameters
NewOperation
output
parameters
NewOperationResponse


Calculator
CalculatorSOAP
http://www.example.org/
Calculator
AddOperation
input
parameters
AddOperation
output
parameters
AddOperationResponse

Exercise 5


*Calculator.wsdl
*Inline Schema of Calculator.wsdl
NewOperation
(NewOperationType)
in
Insert Element
Set Type
Set Multiplicity
Delete
Show properties
Refactor
References
Before
After


Calculator.wsdl
Inline Schema of Calculator.wsdl
AddOperation
(NewOperationType)
a int
b int
c int

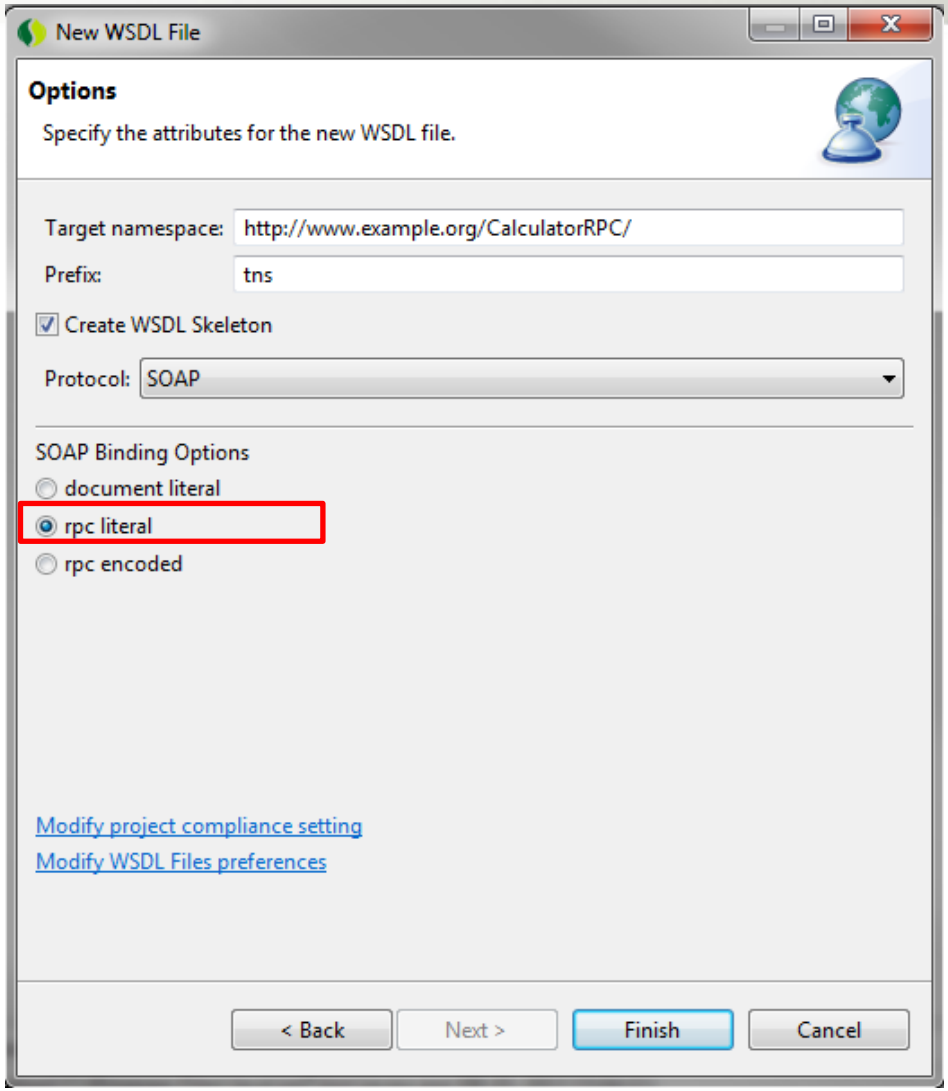
Exercise 6

- Create fresh project
- Design a WSDL in **RPC literal style** for Calculator service with operations:
  - int AddOperation(int a, int b, int c)
  - double DivideOperation(double dividend, double divisor)
- Use wsimport to generate classes from WSDL
- Implement Server and Client
- Analyze with TCP/IP Monitor

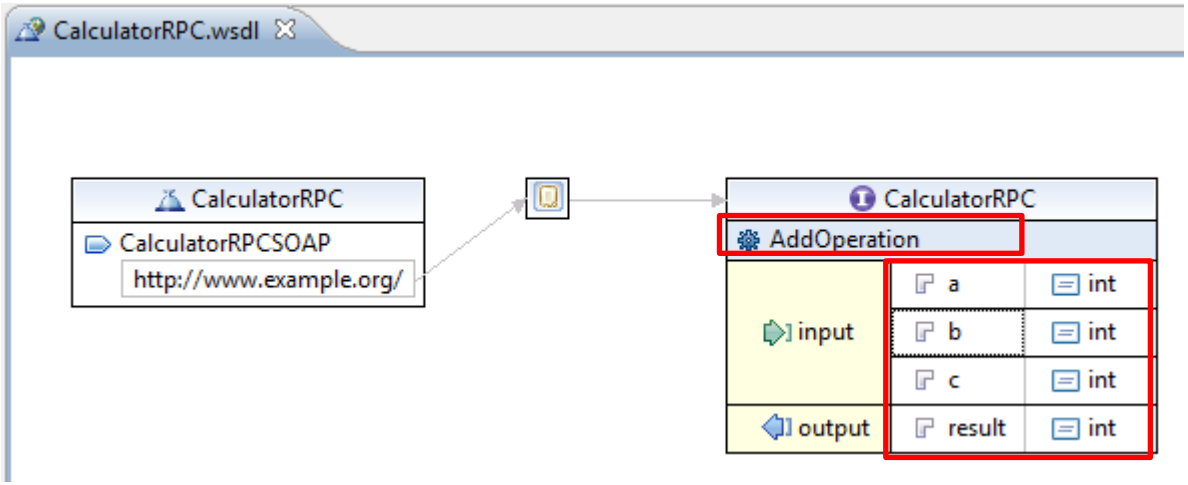
Q:

- Compare the SOAP msg with Ex. 5

Exercise 6


New WSDL File
Options
Specify the attributes for the new WSDL file.
Target namespace:
http://www.example.org/CalculatorRPC/
Prefix:
tns
Create WSDL Skeleton
Protocol:
SOAP
SOAP Binding Options
document literal
rpc literal
rpc encoded
Modify project compliance setting
Modify WSDL Files preferences
< Back
Next >
Finish
Cancel

Exercise 6


CalculatorRPC.wsdl
CalculatorRPC
CalculatorRPCSOAP
http://www.example.org/
CalculatorRPC
AddOperation
input
a
int
b
int
c
int
output
result
int

Exercise 7

- Implement web service server and client as maven projects runnable from the „mvn” command line

- Hints:
  - http://www.hascode.com/2011/08/contract-first-web-services-using-jax-ws-jax-b-maven-and-eclipse/
    - Remember to change version of jaxws-rt maven artifact to 2.2.3

Exercise 8

- Copy Ex. 7
- Add simple HTTP authentication with username and password
- Analyze SOAP messages (TCP/IP Monitor)
  - Where is the username and password sent?

- Hints:
  http://www.mkyong.com/webservices/jax-ws/application-authentication-with-jax-ws/

Exercise 9

- Copy Ex. 8
- Add transmission of binary data with the use of MTOM:
  - From client to server
  - From server to client
- Implement sending of some binary file (eg. picture file) with the use of DataHandler

- Hints:
  - http://www.mkyong.com/webservices/jax-ws/jax-ws-attachment-with-mtom/ (DataHandler not involved)
  - http://www.apache.org/dyn/closer.cgi?path=/cxf/2.7.1/apache-cxf-2.7.1.zip - samples/mtom

Exercise 10

- Copy Ex. 9
- Add asynchronous client invocations:
  - Callback: with the use of AsyncHandler
  - Pooling: with the use of Response

- Hints:
  - http://www.apache.org/dyn/closer.cgi?path=/cxf/2.7.1/apache-cxf-2.7.1.zip - samples/jaxws_async

Additional sources for exercises

- Java Web Services Tutorial: http://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/index.html
- Porównanie JAX-RPC z JAX-WS: http://www.ibm.com/developerworks/webservices/library/ws-tip-jaxwsrpc/index.html

References

- Technologia Web Service 2012 – Przemysław Dadel
- http://soadecisions.org/download/t052-zimmermann-3.0.pdf