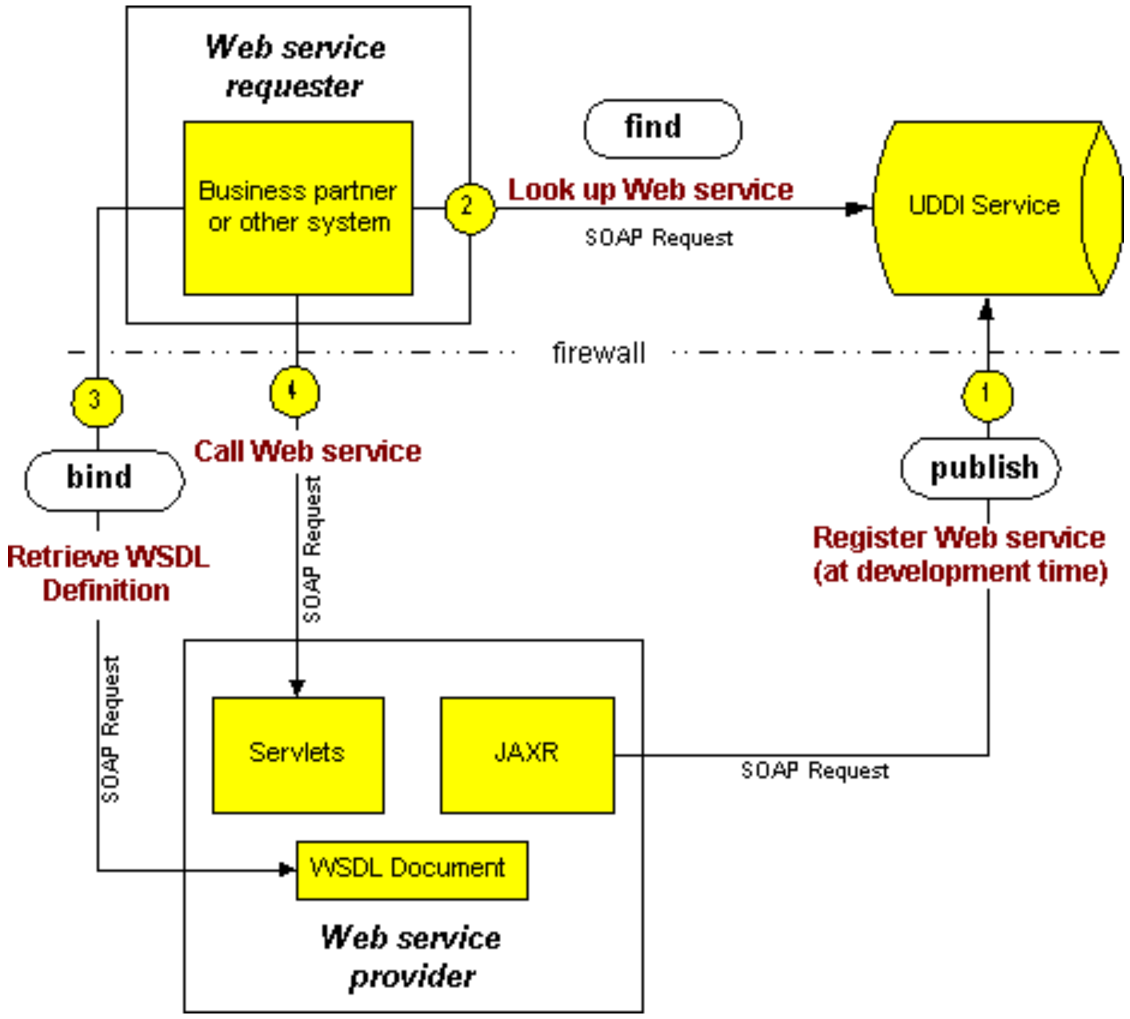
# WSDL Essential

# Working of WSDL (with Java)



## Background

- WSDL stands for Web Service Description Language
- A specification defining how to describe Web services in a common XML grammar
- Before WSDL, service providers used their own way to describe service
- Description files are inconsistent and incompatible to each other
- Microsoft and IBM then proposed to combine their technologies SCL and NASSL to WSDL
- With the contribution from Ariba, WSDL ver 1.1 was submitted to W3C in March 2001. Not yet an official standard (its status is “submission acknowledged”)
- WSDL ver 2.0 Part I was submitted in July 2007. Recommended by W3C.

- WSDL represents a contract between the service requestor and the service provider
- Using WSDL, a client can locate a Web service and invoke any of its publicly available function
- With WSDL-aware tools, the whole process can be done automatically
- WSDL describes four critical pieces of data
  - Interface information describing all publicly available functions
  - Data type information for all messages and message responses
  - Binding information about the transport protocol to be used
  - Address information for locating the specified service

- A WSDL document can be divided into six major elements

`<definitions>: Root WSDL Element`

**`<types>:`** **What data types will be transmitted?**

**`<message>:`** **What messages will be transmitted?**

**`<portType>:`** **What operations will be supported?**

**`<binding>:`** **How will the messages be transmitted on the wire?**

**`<service>:`** **Where is the service located?**

- `definitions`
  - Must be the root element
  - Define the name of the service
  - Declare the namespaces used in the document
- `types`
  - Describe all the data type used by the Client and Server
  - Can be omitted if only simple data types are used
- `message`
  - Define the name of the request/response messages
  - Define also the message part elements
- `portType`
  - Define the combination of message elements to form a complete one-way or round-trip operation

- `binding`
  - Provide specific details on how a portType operation will actually be transmitted over the wire
  - SOAP specific information can be defined here. WSDL includes built-in extensions for defining SOAP services
- `service`
  - Define the address for invoking the specified service
- `documentation (less commonly used)`
  - Provide human-readable documentation
  - Similar to making comments in a program
- `import (not all WSDL tools support)`
  - Allow importing other WSDL documents or XML Schemas into a WSDL document
  - Enable a more modular WSDL document

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions … >
   <wsdl:types … >
      :
   </wsdl:types>
   <wsdl:message … >
      :
   </wsdl:message>
   <wsdl:portType … >
      :
   </wsdl:portType>
   <wsdl:binding … >
      :
   </wsdl:binding>
   <wsdl:service … >
      :
   </wsdl:service>
</wsdl:definitions>
```

Can be omitted if only simple data types, e.g. int, String are used

**A Sample WSDL file**

## An Example: NameAndAge.wsdl

```
<definitions>: NameAndAge

<types>: JavaBean Record
         – two variables Name and Age

<message>: 1. showRecordResquest
           2. showRecordResponse

<portType>:showRecord that consists of
           a request/response service

<binding>: Direction to use the SOAP
           HTTP transport protocol

<service>: Service available at
http://localhost:8080/axis/services/
NameAndAge
```

## a. definitions

**targetNamespace is the logical namespace for information about this service. WSDL documents can import other WSDL documents, and setting targetNamespace to a unique value ensures that the namespaces do not clash**

```
<?xml version="1.0" encoding="UTF-8" ?>
<wsdl:definitions
 targetNamespace=
     "http://localhost:8080/axis/services/NameAndAge"
 xmlns="http://schemas.xmlsoap.org/wsdl/"
          :
          :
>
```

**Default namespace. All the WSDL elements, such as <definitions>, <types> and <message> reside in this namespace.**

## Define the namespaces that will be used in the later part of the document

```
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl=
    "http://localhost:8080/axis/services/NameAndAge"
xmlns:intf=
    "http://localhost:8080/axis/services/NameAndAge"
xmlns:soapenc=
    "http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="enpklun:polyu.edu.hk:soap"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap=
    "http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

## b. types – give details of complex data type

The qName of our JavaBean, its namespace is defined by targetNameSpace

Default namespace, apply to unspecified tags, e.g. schema, sequence, complexType, element

```
<wsdl:types>
 <schema targetNamespace="enpklun:polyu.edu.hk:soap”
  xmlns="http://www.w3.org/2001/XMLSchema">
  <complexType name="Record">
   <sequence>
    <element name="age" type="xsd:int" />
    <element name="name" nillable="true"
                              type="xsd:string" />

   </sequence>
  </complexType>
 </schema>
</wsdl:types>
```

can be a null string

Two parameters of Record to be sent. The element names are derived from the get/set functions of the JavaBean

- Different programming languages have different ways to declare data types, e.g. int, double, String
- One of the greatest challenges in building Web services is to create a common data type system that every programming language can understand
  - E.g. a JavaBean cannot be understood by C++ program
- WSDL by default follows the data typing system defined by W3C XML Schema Specification

```
<schema targetNamespace="enpklun:polyu.edu.hk:soap”
  xmlns="http://www.w3.org/2001/XMLSchema">
      :
</schema>
```

- XML Schema specification includes a basic type system for encoding most simple data types
- Include a long list of built-in simple types, e.g. string, float, double …. Details can be found in *http://www.w3.org/TR/2000/WD=xmlschema=0=20000407/*
- If only these data types are used in a Web service, the WSDL document does not have the "types" section to further explain them
- When converting from a service or a request to XML messages, the implementation platform, e.g. AXIS, should know how to encode these simple type data based on the specifications as defined in XML Schema

- For complex data types, e.g. JavaBean, XML Schema does not have their specifications
- If a Web service wants to use them, need to be explained in the “types” section of its WSDL file

```
<complexType name="Record">
 <sequence>
  <element name="age" type="xsd:int" />
  <element name="name" nillable="true"
                              type="xsd:string" />
 </sequence>
</complexType>
```

- **Define that the Record type in fact comprises only two variables in sequence**
- **Quite different from the original JavaBean specification**
- **But can be understood by most languages**

## c. message

- When the data type is defined, specify the kind of messages that make use of that data type
- The message element defines two kinds of messages in this example
  - showRecordRequest
  - showRecordResponse
- The showRecordRequest message only uses one kind of data type: Record
- The showRecordResponse message uses the same kind of data type: Record

- The namespace of tns1 as defined in “definition” is enpklun:polyu.edu.hk:soap
- The same as the targetNameSpace in “types”
- Hence we are talking about the "Record” described in “types”

```
<wsdl:message name="showRecordRequest">
  <wsdl:part name="in0" type="tns1:Record" />
</wsdl:message>
<wsdl:message name="showRecordResponse">
  <wsdl:part name="showRecordReturn"
                                    type="tns1:Record" />
</wsdl:message>
```

The name of the parameter used in these two messages. Only one in each message

## d. portType

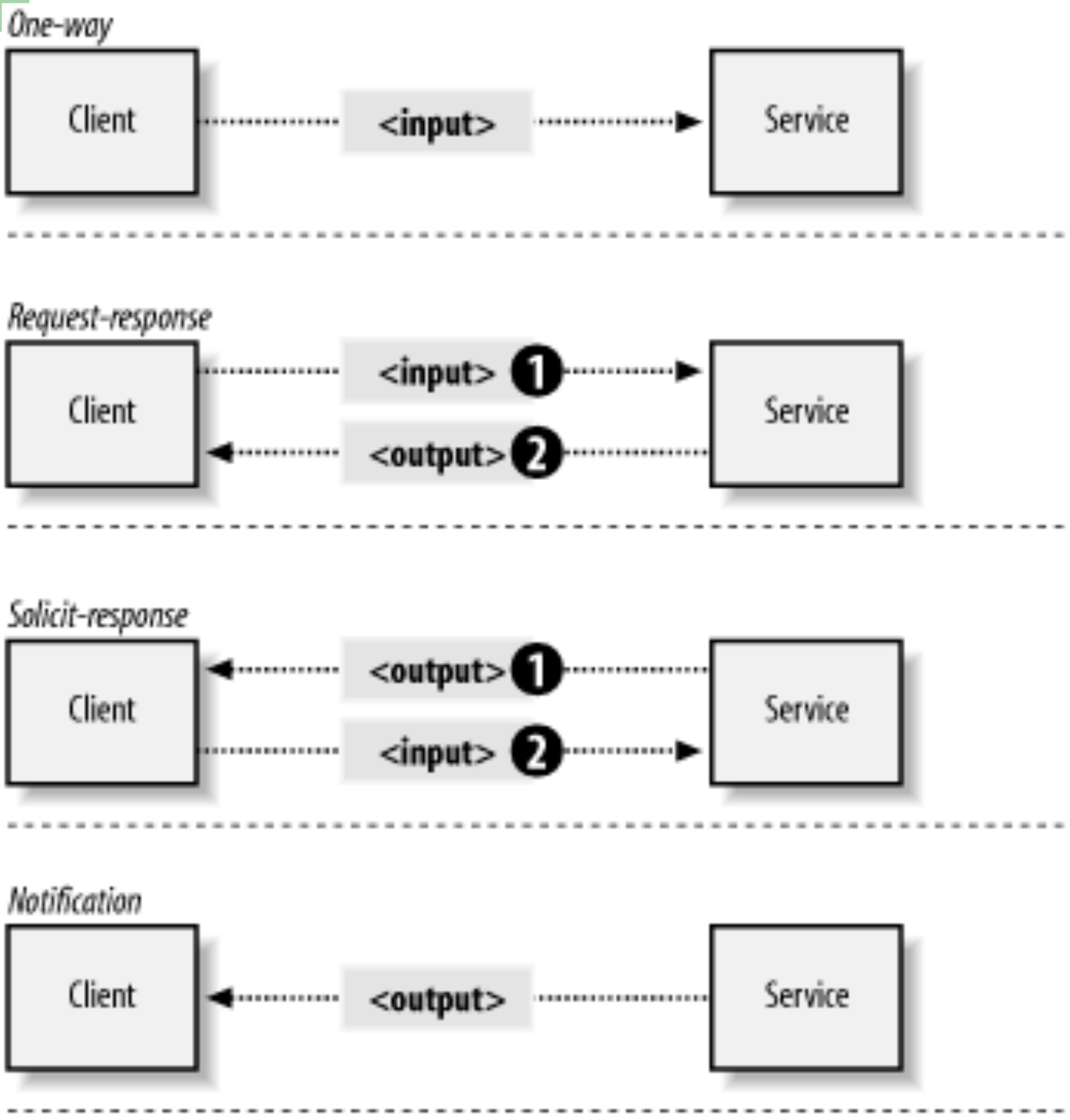
- Define how the messages are transmitted for the method: showRecord

```
<wsdl:portType name="RecordService">
  <wsdl:operation name="showRecord"
                                  parameterOrder="in0">
    <wsdl:input message="impl:showRecordRequest"
                              name="showRecordRequest" />
    <wsdl:output message="impl:showRecordResponse"
                              name="showRecordResponse" />
  </wsdl:operation>
</wsdl:portType>
```

**The sequence of the input/output message is matter. The example above means that the input message should go first and followed by the output message**

## Four operation patterns supported by WSDL 1.1

1. One-way
2. Request-response
3. Solicit-response
4. Notification



```
<wsdl:operation name="showRecord"
                                     parameterOrder="in0">
```

- A message can have more than one *"parts"*
  - E.g. if showRecord() requires three input parameters, then the input message for calling the service will have three parts
- For message that has more than one *"parts"*, need to indicate their order, e.g. which part is the first parameter and which part is the second
- Assume the input message of showRecord() has three *"parts"* – in0, in1 and in2, and in0 is the first, in1 is the second and in2 is the third, then

```
<wsdl:operation name="showRecord"
                              parameterOrder="in0 in1 in2">
```

## e. binding

- The binding element provides specific details on how a portType operation will actually be transmitted over the wire
- A single portType can have multiple bindings using different transports e.g. HTTP or SMTP
- Contain the following parts:
  - binding type
  - soap operation
    - function name to be called
    - details about the input parameters
    - details about the return parameters

Talking about the showRecord() of RecordService

```
<wsdl:binding name="NameAndAgeSoapBinding"
                                type="impl:RecordService">
 <wsdlsoap:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http" />
 <wsdl:operation name="showRecord">
   <wsdlsoap:operation soapAction="" />
   <wsdl:input name="showRecordRequest">
      :
   </wsdl:input>
   <wsdl:output name="showRecordResponse">
      :
   </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
```

**using HTTP**

**Referring to the same operation as in the portType, since same namespace**

- Provide more specific details to the input and output messages with respect to the kind of messaging protocol (soap in this case) used

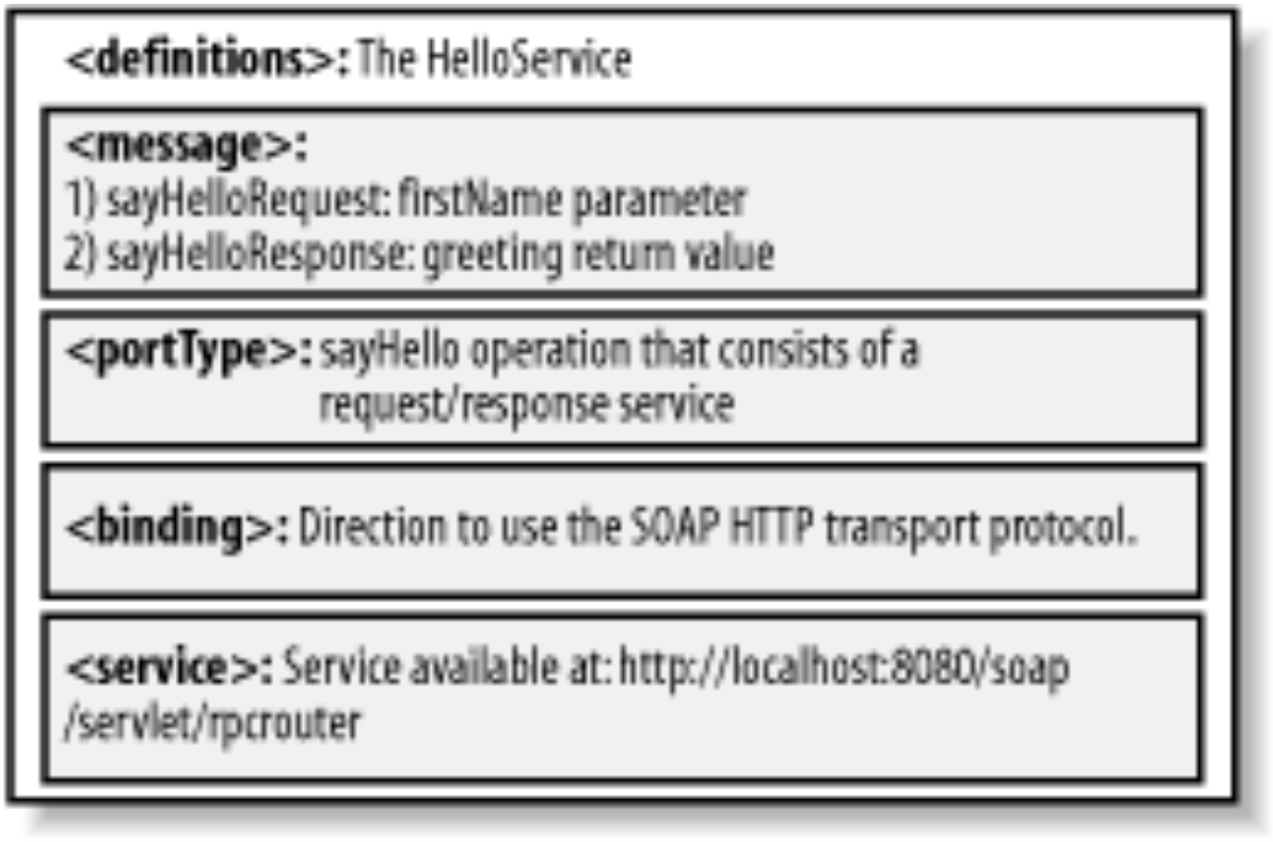
```
<wsdl:input name="showRecordRequest">
   <wsdlsoap:body encodingStyle=
           "http://schemas.xmlsoap.org/soap/encoding/"
     namespace="http://HelloBean" use="encoded" />
</wsdl:input>

<wsdl:output name="showRecordResponse">
   <wsdlsoap:body encodingStyle=
           "http://schemas.xmlsoap.org/soap/encoding/"
     namespace=
      "http://localhost:8080/axis/services/NameAndAge"
           use="encoded" />
</wsdl:output>
```

## e. service

```
<wsdl:service name="RecordServiceService">
  <wsdl:port binding="impl:NameAndAgeSoapBinding‟
                                    name="NameAndAge">
    <wsdlsoap:address location=
   "http://localhost:8080/axis/services/NameAndAge" />
  </wsdl:port>
</wsdl:service>
```

- Specify the location of the service

## Overview of HelloService



**<definitions>:** The HelloService

**<message>:**
1) sayHelloRequest: firstName parameter
2) sayHelloResponse: greeting return value

**<portType>:** sayHello operation that consists of a request/response service

**<binding>:** Direction to use the SOAP HTTP transport protocol.

**<service>:** Service available at: http://localhost:8080/soap /servlet/rpcrouter

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="HelloService"
targetNamespace="http://www.ecerami.com/wsdl/
HelloService.wsdl" xmlns="http://schemas.xmlsoap.org/
wsdl/" xmlns:soap="http://schemas.xmlsoap.org/wsdl/
soap/" xmlns:tns="http://www.ecerami.com/wsdl/
HelloService.wsdl" xmlns:xsd="http://www.w3.org/2001/
XMLSchema">

      <message name="SayHelloRequest">
              <part name="firstName" type="xsd:string"/>
      </message>
      <message name="SayHelloResponse">
             <part name="greeting" type="xsd:string"/>
      </message>
      <portType name="Hello_PortType">
             <operation name="sayHello">
                    <input message="tns:SayHelloRequest"/>
                   <output message="tns:SayHelloResponse"/>
             </operation>
      </portType>
```

```
<binding name="Hello_Binding" type="tns:Hello_PortType">
             <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
             <operation name="sayHello">
                   <soap:operation soapAction="sayHello"/>
                   <input>
                         <soap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
      namespace="urn:examples:helloservice"
             use="encoded"/>
                   </input>
                   <output>
                         <soap:body
      encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
      namespace="urn:examples:helloservice" use="encoded"/>
                   </output>
             </operation>
      </binding>
      <service name="Hello_Service">
             <documentation>WSDL File for
                           HelloService</documentation>
             <port binding="tns:Hello_Binding"
name="Hello_Port">
             <soap:address
             location="http://localhost:8080/soap/servlet/
rpcrouter"/          >
             </port>
      </service>
</definitions>
```

## WSDL Invocation Tools

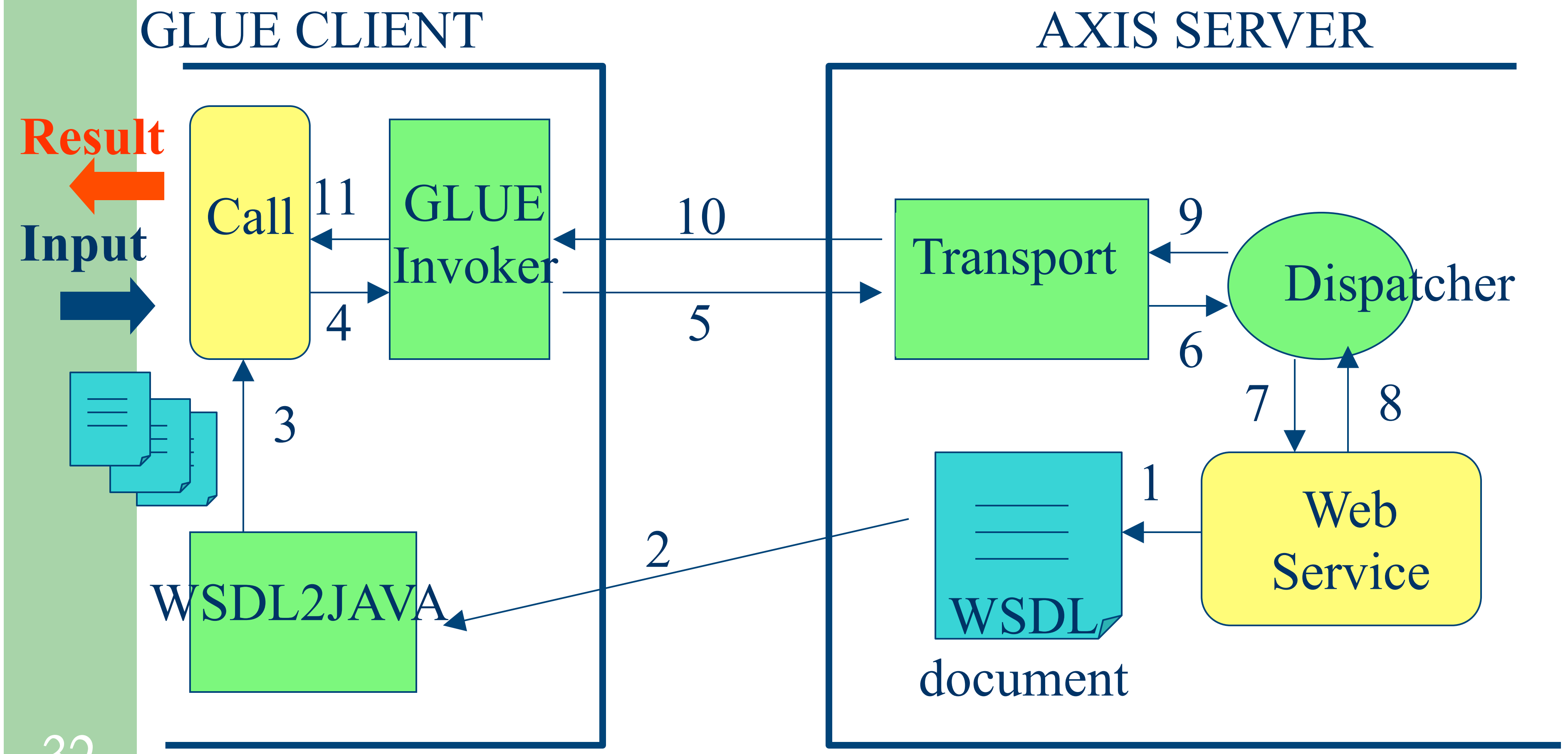
- WSDL gives a full description of a Web service
  - Define the namespaces (in definition)
  - Define the data type (in types)
  - Define the messages format (in message)
  - Define the sequence of sending messages (in portType)
  - Define the kind of the messaging system to be used, e.g. Soap, and its implementation details (in binding)
  - Define the location of the service (in service)
- By having the WSDL document of a Web service, basically we have obtained all information required to invoke this service

- Since WSDL is developed based on standardized rules (XML Schema), service providers can automatically generate the WSDL document of a Web service
- Since a WSDL document is a full description of a Web service, requestors can automatically generate requests based on WSDL
  - Hence no need for client to develop the request program, e.g. RecordClient (see SOAP Implementation)
- Different software vendors have developed tools to facilitate the above objectives
  - WebMethods 's GLUE
  - IBM's WSIF (included in its ETTK package)
  - SOAP:Lite for Perl

## Main Objectives of the Tools

- To hide away the complication of invoking the Web service from the client as much as possible
- To standardize as much as possible the procedure to client to invoke different kind of services

- For example, the following is the interaction when using AXIS and GLUE

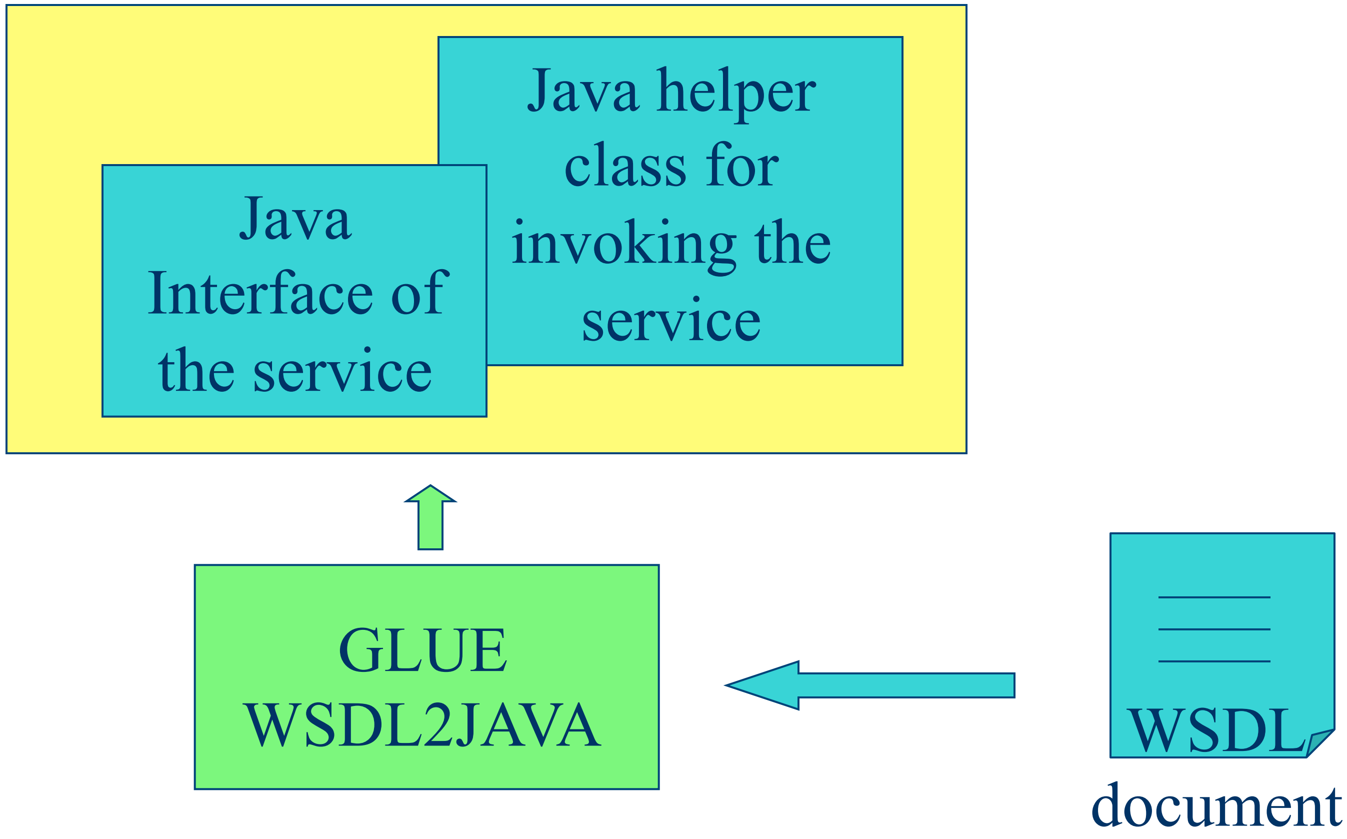


1. AXIS automatically generates the WSDL document of a Web service
2. GLUE client uses the GLUE's WSDL2JAVA tool to retrieve the WSDL document. It obtains the required info of the Web service and generates a set of Java files
3. The Info is applied to a relatively standard Web service calling program
4. A GLUE service invoker is generated to handle the problems for invoking a SOAP service

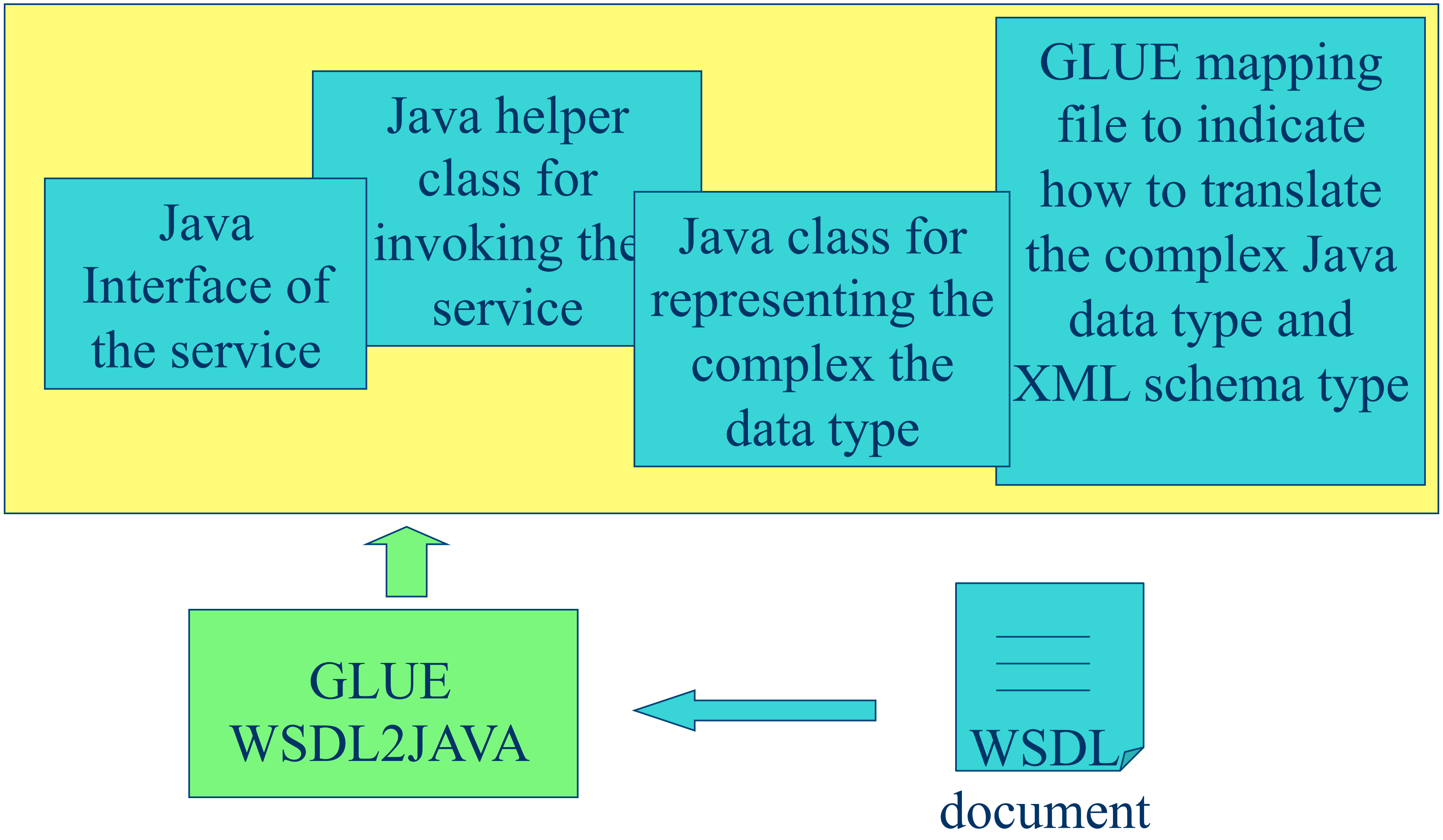
5-9. AXIS calls the method in the service and sends the result back to the GLUE service invoker

10. GLUE client finally gets the result

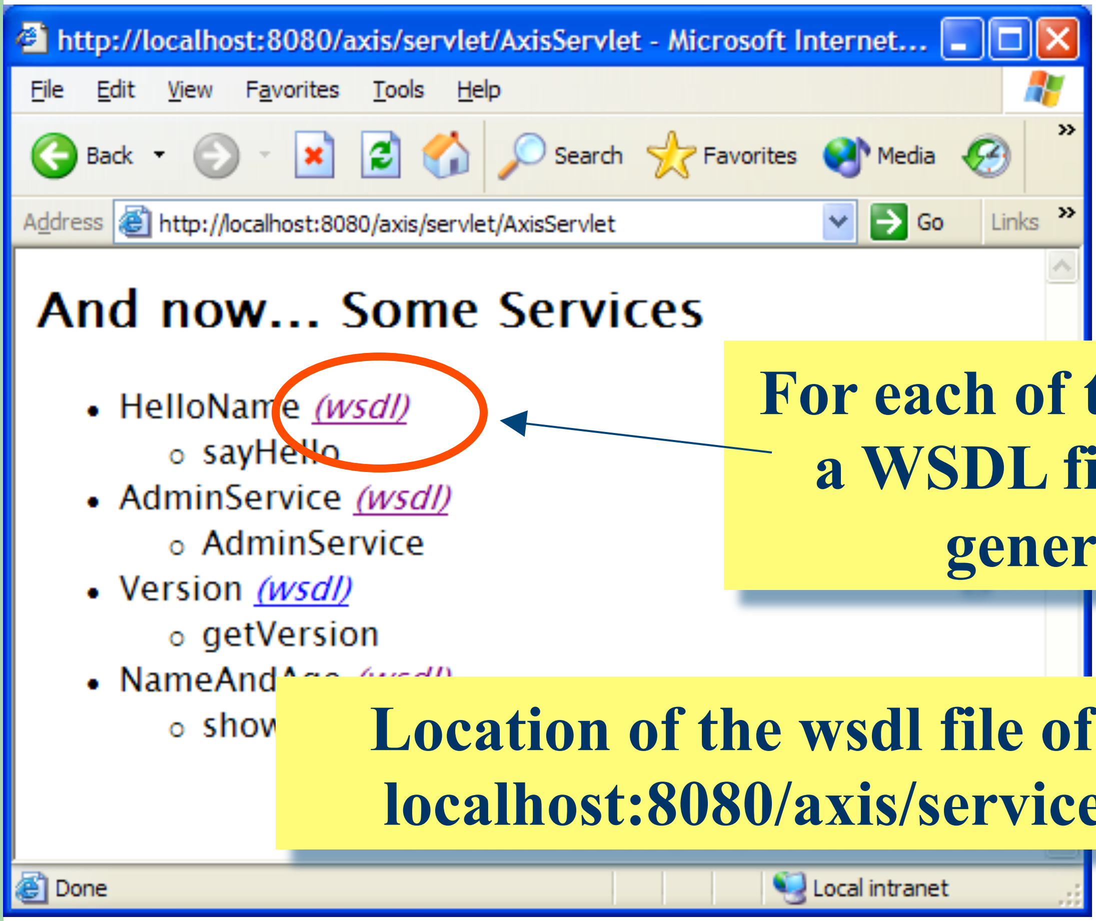
## For simple data types


Java helper class for invoking the service
Java Interface of the service
GLUE WSDL2JAVA
WSDL document

## For complex data types


Java Interface of the service
Java helper class for invoking the service
Java class for representing the complex the data type
GLUE mapping file to indicate how to translate the complex Java data type and XML schema type
GLUE WSDL2JAVA
WSDL
document

## Invoking Services using Simple Data Types


http://localhost:8080/axis/servlet/AxisServlet - Microsoft Internet...

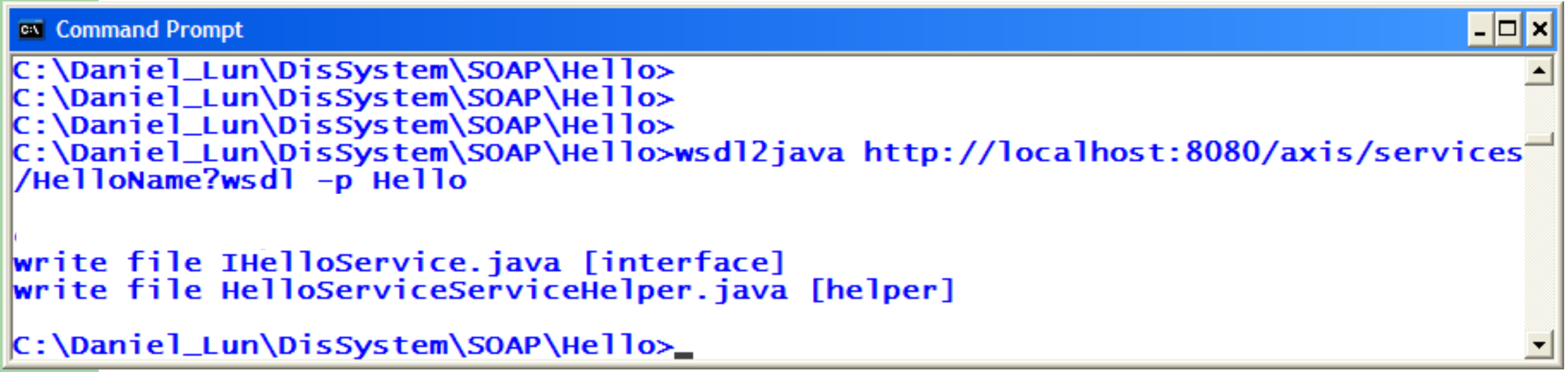
http://localhost:8080/axis/servlet/AxisServlet

And now... Some Services

- HelloName (wsdl)
  - sayHello
- AdminService (wsdl)
  - AdminService
- Version (wsdl)
  - getVersion
- NameAndAge (wsdl)
  - show

For each of the deployed service, a WSDL file is automatically generated by AXIS

Location of the wsdl file of HelloName: http://localhost:8080/axis/services/HelloName?wsdl


Command Prompt

```
C:\Daniel_Lun\DisSystem\SOAP\Hello>
C:\Daniel_Lun\DisSystem\SOAP\Hello>
C:\Daniel_Lun\DisSystem\SOAP\Hello>
C:\Daniel_Lun\DisSystem\SOAP\Hello>wsdl2java http://localhost:8080/axis/services
/HelloName?wsdl -p Hello


write file IHelloService.java [interface]
write file HelloServiceServiceHelper.java [helper]

C:\Daniel_Lun\DisSystem\SOAP\Hello>_
```

**Command:**

wsdl2java http://localhost:8080/axis/services/HelloName?wsdl –p Hello

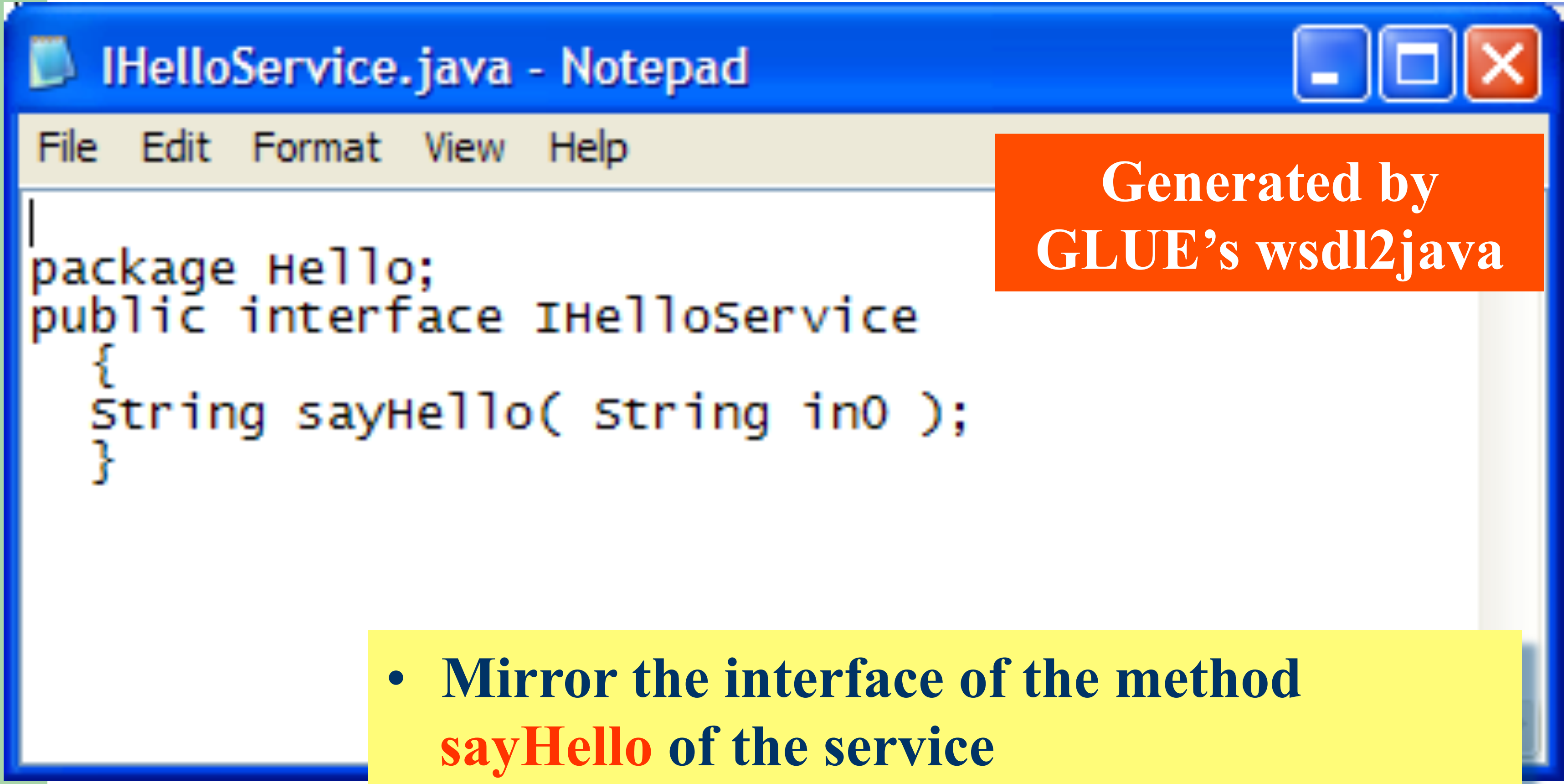
The files generated should be placed in the Hello package

Location of the wsdl file

**File generated:**

**IHelloService.java** – exposes the method interface

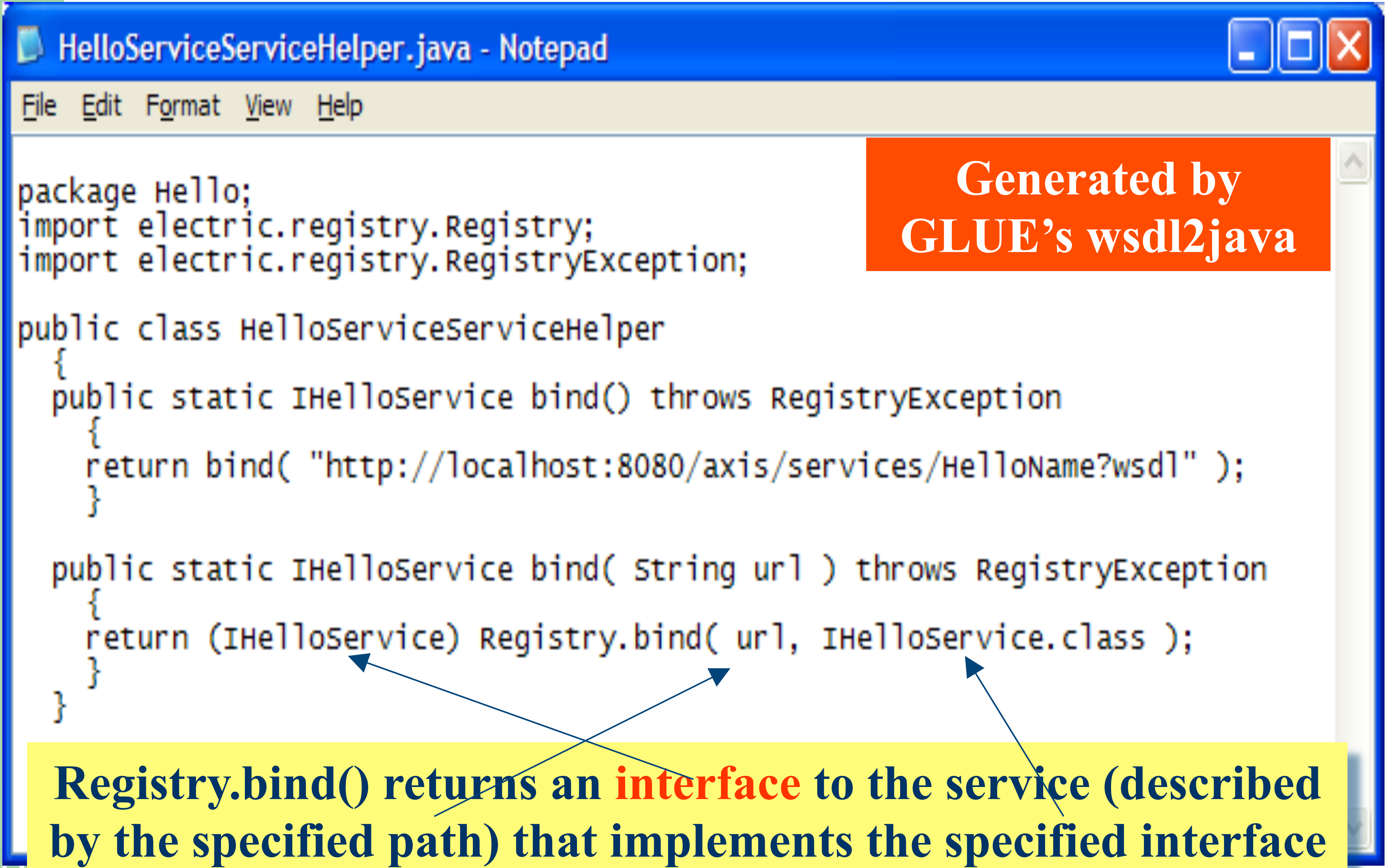
**HelloServiceServiceHelper.java** – dynamically bind to the service specified by the WSDL file



```
package Hello;
public interface IHelloService
   {
   String sayHello( String in0 );
   }
```

Generated by GLUE's wsdl2java

- **Mirror the interface of the method sayHello of the service**
- **Based on this interface, a calling program should know the method to be called, the input and output parameters**


HelloServiceServiceHelper.java - Notepad
File Edit Format View Help

```
package Hello;
import electric.registry.Registry;
import electric.registry.RegistryException;

public class HelloServiceServiceHelper
  {
  public static IHelloService bind() throws RegistryException
    {
    return bind( "http://localhost:8080/axis/services/HelloName?wsdl" );
    }

  public static IHelloService bind( String url ) throws RegistryException
    {
    return (IHelloService) Registry.bind( url, IHelloService.class );
    }
  }
```

**Generated by GLUE's wsdl2java**

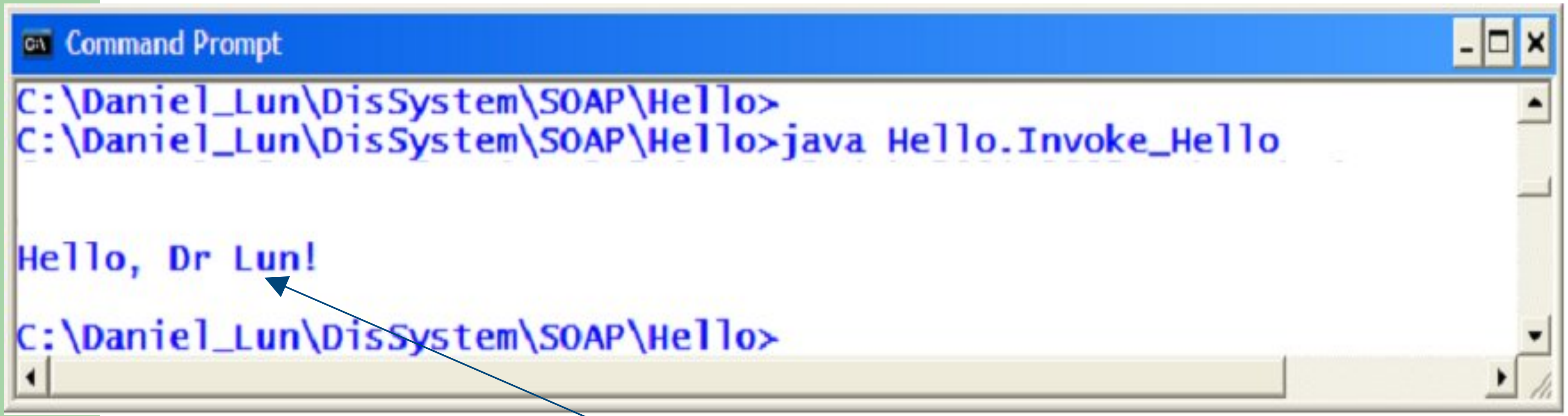
**Registry.bind() returns an interface to the service (described by the specified path) that implements the specified interface**

- By using the helper files, a relatively standard service calling file can be used
- Need no knowledge about SOAP hence enables automatic service invocation
- To enable full automated Web service, need an automatic process to
  - extract the method name and the class type of the input and output parameters
  - provide the input parameter and
  - interpret the semantic meaning of the return result

```
public class Invoke_Hello {

  public String say (String name)
      throws Exception {
    IHelloService Service =
           HelloServiceServiceHelper.bind();
    return Service.sayHello(name);
  }
  public static void main (String[] args)
      throws Exception {
    Invoke_Hello invoker = new Invoke_Hello();
    String result = invoker.say("Dr Lun");
    System.out.println(result);
  }
}
```

**Can be extracted from the interface**

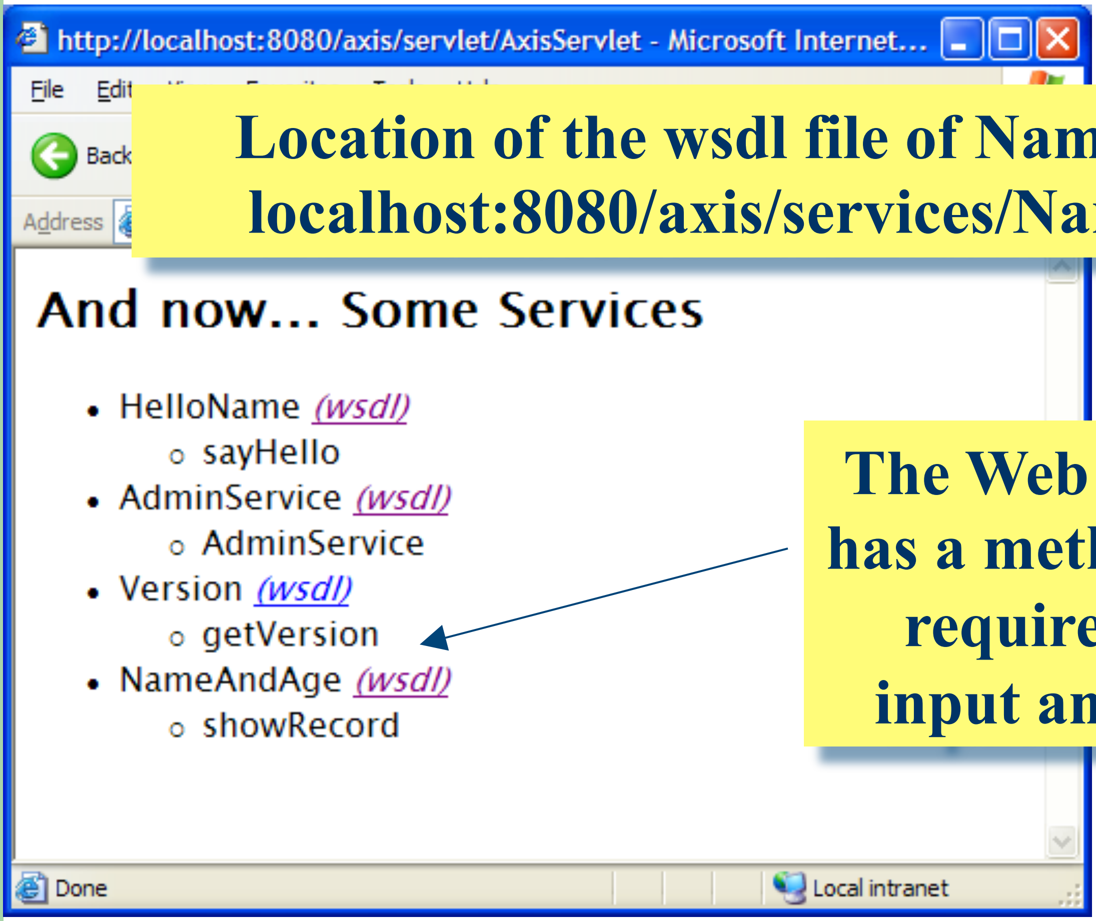
**Only need to provide the required input and interpret the return result**



```
C:\Daniel_Lun\DisSystem\SOAP\Hello>
C:\Daniel_Lun\DisSystem\SOAP\Hello>java Hello.Invoke_Hello


Hello, Dr Lun!

C:\Daniel_Lun\DisSystem\SOAP\Hello>
```

Result received from the remote service

## Invoking Services using Complex Data Types

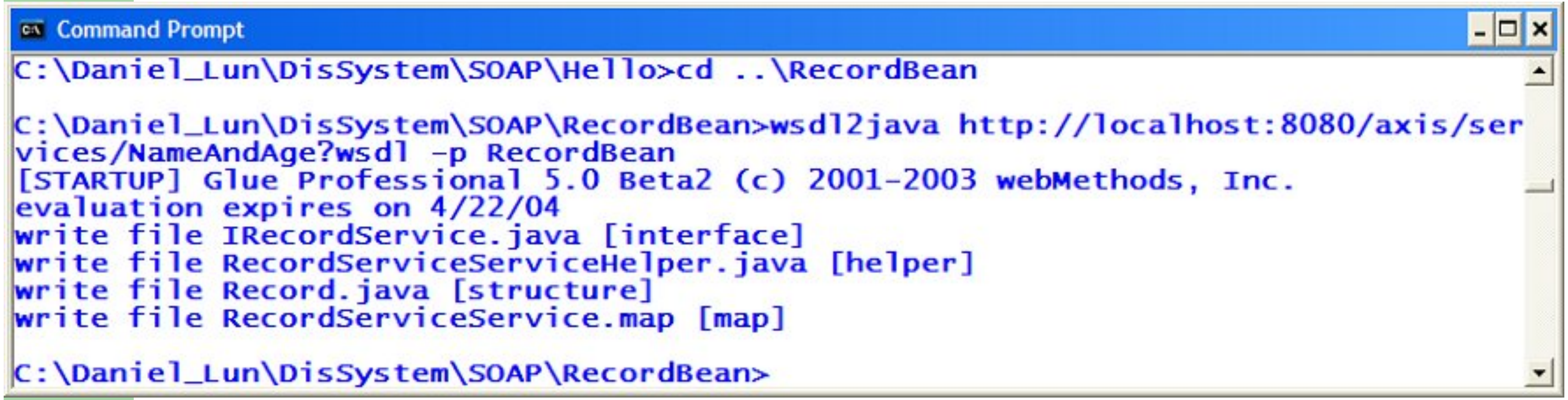

http://localhost:8080/axis/servlet/AxisServlet - Microsoft Internet...
File
Edit
Back
Address
Done
Local intranet

And now... Some Services

- HelloName *(wsdl)*
  - sayHello
- AdminService *(wsdl)*
  - AdminService
- Version *(wsdl)*
  - getVersion
- NameAndAge *(wsdl)*
  - showRecord

**Location of the wsdl file of NameAndAge: http://localhost:8080/axis/services/NameAndAge?wsdl**

**The Web service NameAndAge has a method showRecord() that requires a JavaBean as the input and return a JavaBean**


```
C:\Daniel_Lun\DisSystem\SOAP\Hello>cd ..\RecordBean

C:\Daniel_Lun\DisSystem\SOAP\RecordBean>wsdl2java http://localhost:8080/axis/ser
vices/NameAndAge?wsdl -p RecordBean
[STARTUP] Glue Professional 5.0 Beta2 (c) 2001-2003 webMethods, Inc.
evaluation expires on 4/22/04
write file IRecordService.java [interface]
write file RecordServiceServiceHelper.java [helper]
write file Record.java [structure]
write file RecordServiceService.map [map]

C:\Daniel_Lun\DisSystem\SOAP\RecordBean>
```

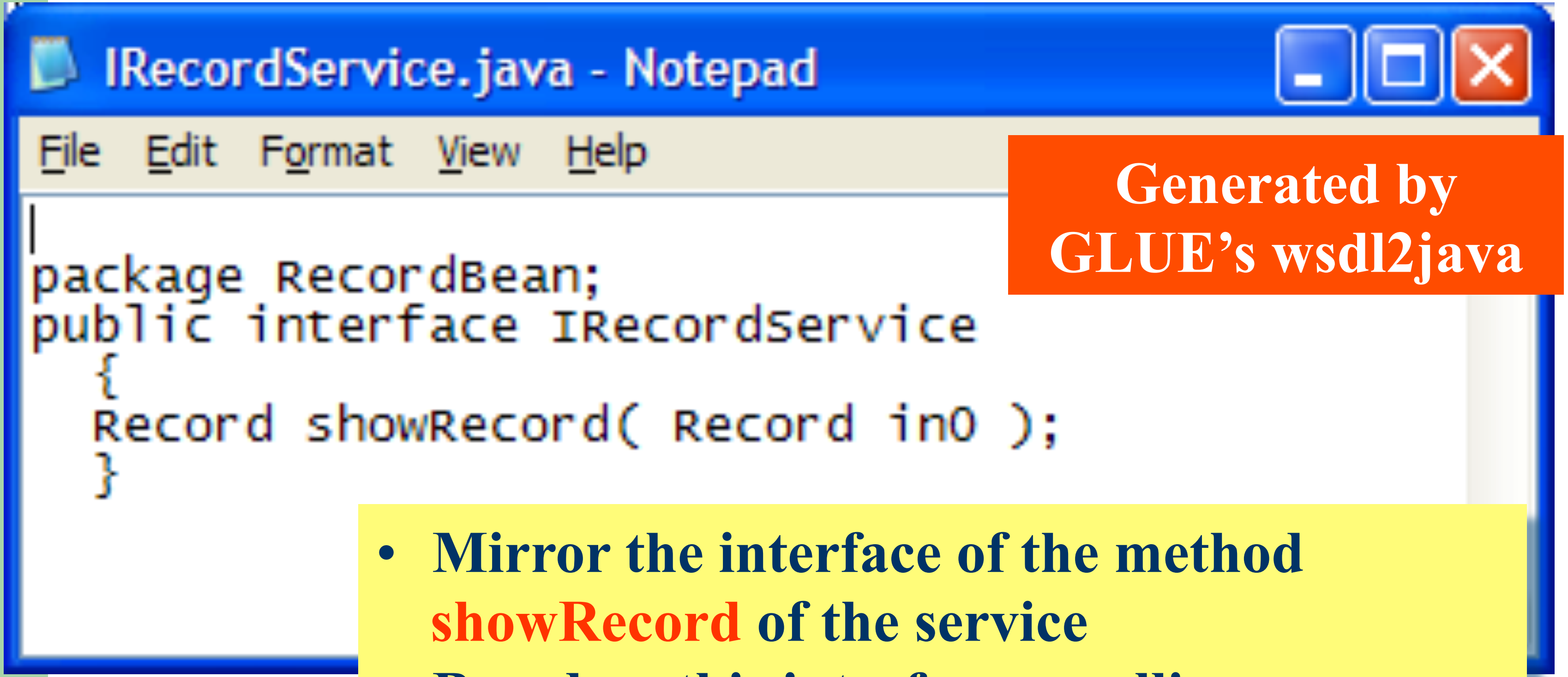
**File generated:**

**IRecordService.java** – exposes the method interface

**RecordServiceServiceHelper.java** – dynamically bind to the service specified by the WSDL file

**Record.java** – specify the structure of the class that can represent the complex data type used in the service

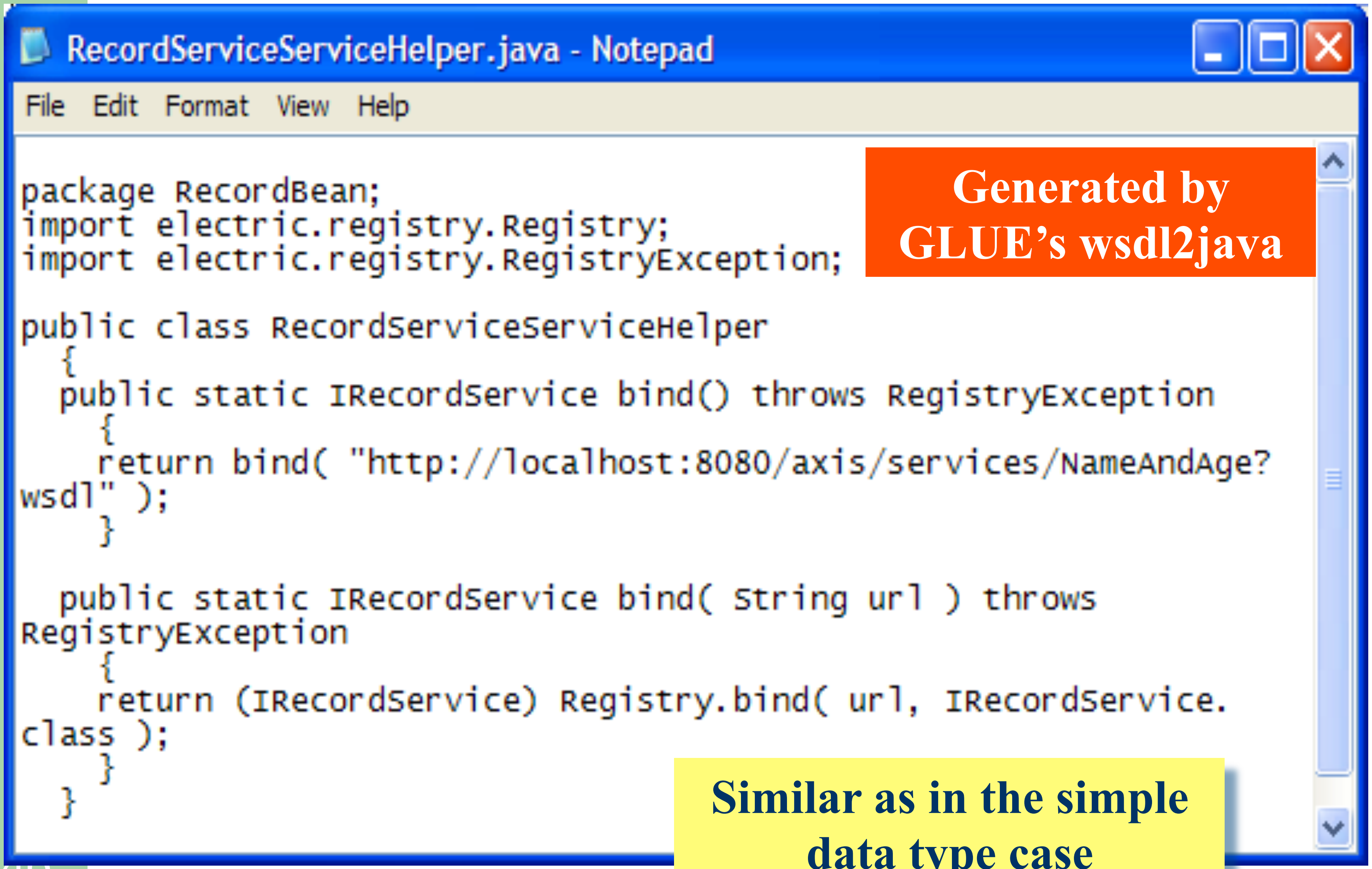
**RecordServiceService.map** – specify how to map between the data types in Record.java and the complex data type



```
package RecordBean;
public interface IRecordService
  {
  Record showRecord( Record in0 );
  }
```

Generated by GLUE's wsdl2java

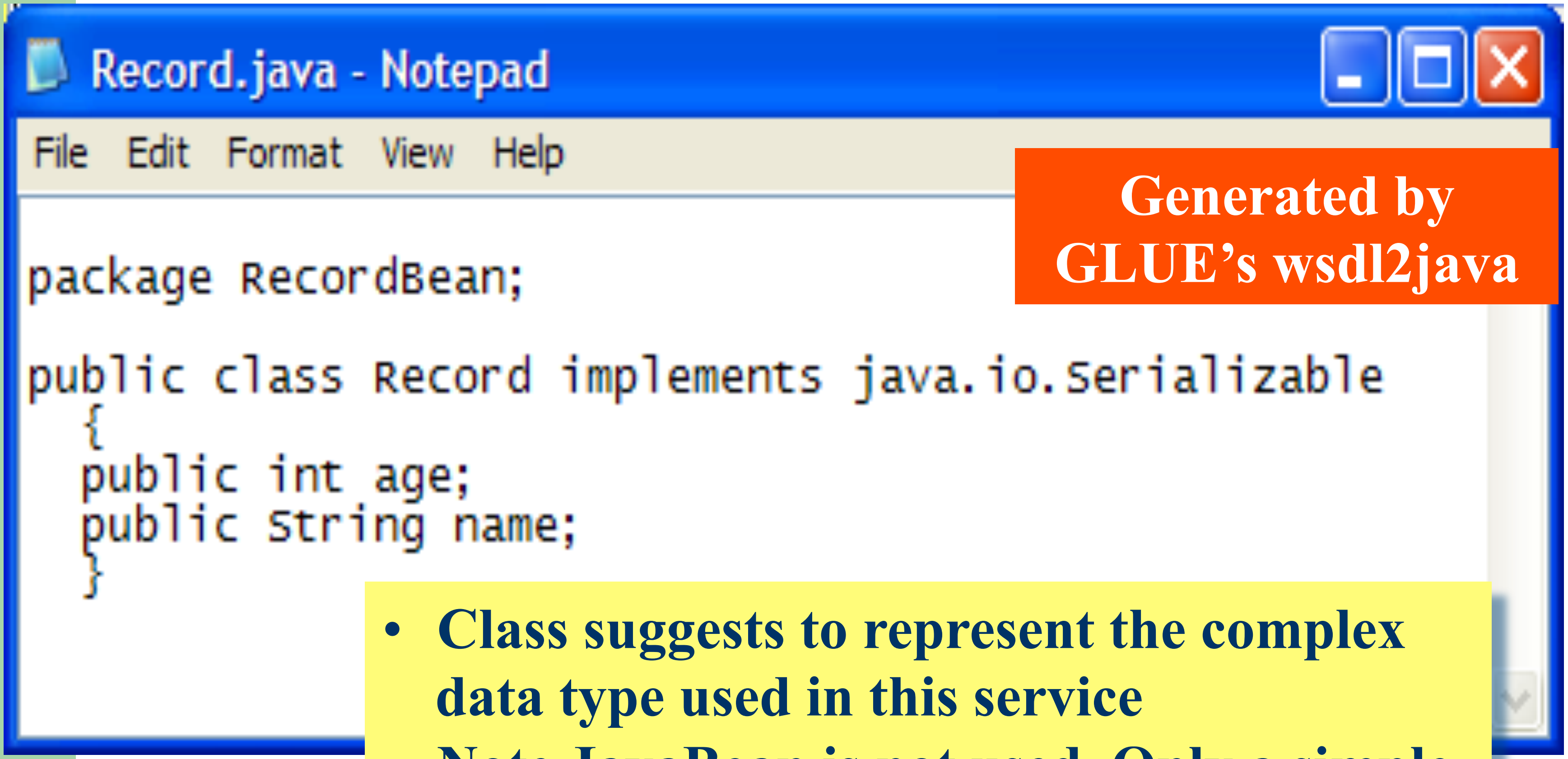
- Mirror the interface of the method showRecord of the service
- Based on this interface, a calling program should know the method to be called, the input and output parameters
- Note the complex data type required by this method


RecordServiceServiceHelper.java - Notepad
File Edit Format View Help

```
package RecordBean;
import electric.registry.Registry;
import electric.registry.RegistryException;

public class RecordServiceServiceHelper
  {
  public static IRecordService bind() throws RegistryException
    {
    return bind( "http://localhost:8080/axis/services/NameAndAge?
wsdl" );
    }

  public static IRecordService bind( String url ) throws
RegistryException
    {
    return (IRecordService) Registry.bind( url, IRecordService.
class );
    }
  }
```

**Generated by GLUE's wsdl2java**

**Similar as in the simple data type case**



```
package RecordBean;

public class Record implements java.io.Serializable
   {
   public int age;
   public String name;
   }
```

Generated by GLUE’s wsdl2java

- Class suggests to represent the complex data type used in this service
- Note JavaBean is not used. Only a simple class
- Hence can be more easily handled by a general invocation program

```
<?xml version='1.0' encoding='UTF-8'?>
<map:mappings
   xmlns:map='http://www.themindelectric.com/schema/'
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
  <xsd:schema targetNamespace=
                    'enpklun:polyu.edu.hk:soap'>
    <xsd:complexType name='Record'
                  map:class='RecordBean.Record'>
      <xsd:sequence>
        <xsd:element name='age' map:field='age'
                                type='xsd:int'/>
        <xsd:element name='name' nillable='true'
               map:field='name' type='xsd:string'/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</map:mappings>
```

**RecordServiceService.map generated by GLUE's wsdl2java**

```
<xsd:complexType name='Record'
              map:class='RecordBean.Record'>
  <xsd:sequence>
    <xsd:element name='age' map:field='age'
                            type='xsd:int'/>
    <xsd:element name='name' nillable='true'
           map:field='name' type='xsd:string'/>
  </xsd:sequence>
</xsd:complexType>
```

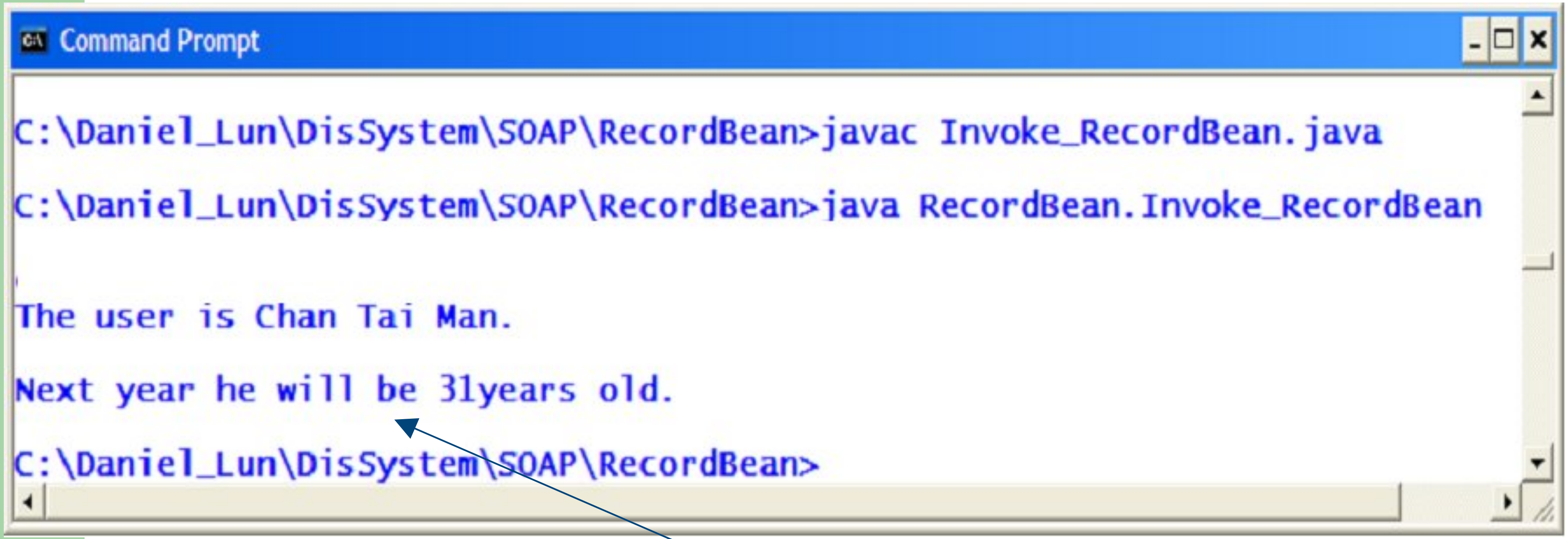
- Map the element `age` to the `age` variable in `RecordBean.Record` class and it is of type `integer` defined in XML Schema
- Map the element `name` to the `name` variable in `RecordBean.Record` class and it is of type `string` defined in XML Schema

```
package RecordBean;
public class Invoke_RecordBean {

  public Record check (Record userRecord)
      throws Exception {
    Mappings.readMappings("RecordServiceService.map");
    IRecordService Service =
        RecordServiceServiceHelper.bind();
    Record updatedRecord =
        Service.showRecord(userRecord);
    return updatedRecord;
  }

  public static void main (String[] args)
      throws Exception {
     :
  }
}
```

The only difference as compared with the simple data type case

```
public static void main (String[] args)
    throws Exception {
  Invoke_RecordBean invoker = new Invoke_RecordBean();

  Record currRecord = new Record();
    // This Record is not JavaBean, but the
    //   class generated by wsdl2java()
  currRecord.name = new String("Chan Tai Man");
  currRecord.age = 30;
    // Again need to pass the required parameters
  Record result = invoker.check(currRecord);
    // When result is received, need to interpret the
    //    the result
  System.out.println("The user is "+result.name+".
\n");
  System.out.println("Next year he will be”+
                         result.age+"years old.");
}
```

The only part that is application specific



```
C:\Daniel_Lun\DisSystem\SOAP\RecordBean>javac Invoke_RecordBean.java

C:\Daniel_Lun\DisSystem\SOAP\RecordBean>java RecordBean.Invoke_RecordBean


The user is Chan Tai Man.

Next year he will be 31years old.

C:\Daniel_Lun\DisSystem\SOAP\RecordBean>
```

Result received from the remote service

- In summary, to both the case of simple or complex data types, a very similar procedure is required to invoke the service
- No knowledge is required in the specific messaging system, e.g. SOAP
- However, the invoker program needs to know
  - the location where the wsdl file can be found (can be solved by UDDI)
  - where to get the parameters to be sent to the service (require the invoker program to have some intelligence, very often application dependent)
  - how to handle the returned results from the service (require the invoker program to have some intelligence, very often application dependent)