Télécom Saint-Étienne :Master Web-Intelligence
Web Services
TP n°1 - 3h00

Enseignant Amro Najjar

Remarks

– This is the subject of TD 1, for both groups A & B
– You should form groups of two students (binôme).
– A binôme should write a report containing the answers of questions and exercises listed in this document and send them to amro.najjar@telecom-st-etienne.fr.
– DEADLINE: Mardi 19/1/2016 À 13h00

Objectives

– Use XML schema to create XML files
– Write an XML-RPC
– Write a SOAP client

Part 1

1 Exercice 1: XML Schema

Suppose that we want to write an application for online bookstores. We want to model a customer, let us say Bob, buying a book from one of the famous book vendors online (e.g. Amazon.fr, Chapitre.fr, Abebooks.fr etc.)

We need to write an XML schema that defines a book purchase order. Remember that XML Schema is a language that enables us to write XML classes (similar to classes in Java). The aim of this exercise is to define a schema for BookPurchaseOrder. The book purchase order contains the two following elements:

1. Client: contains information about the client.
2. Book: contains information about the product.

1.1 Client

A client contains the following elements:

– *name:* a simple element. Type : string (xsd:string). Name is a required field (You need to impose a restriction)
– *age :* is a simple element. Type : integer. It is an optional field, but its value should be between $age \in [12,120]$
– *address:* is a complex element. It contains the following simple elements:
    1. *street*: string
    2. *city*: string
    3. *department*: string (e.g. Loire).
    4. *postal code:* xsd:decimal (e.g. 42000). Note that the child elements of address: must appear in the same order as they are declared.

1.2 Book

This element describes the book bought by the client. An order can contains 1, 2 or 3 books at most. It contains the following information:

- title: string, a simple element
- author: a complex and required element that contains
  - name: string
  - nationality: string
  - age: between 0 & 120
- vendor: a simple required element. Its type is string and its $value \in \{Amazon, Chapitre, Abebooks\}$
- price: xsd:decimal

1.3 To do

1. Write the XML Schema that describes the BookPurchaseOrder.
2. Write two XML files instantiating the BookPurchaseOrder.

1.4 Hints: un peu d'aide

First, before writing the XML Schema, you can read the W3C tutorial on XML Schema :

- XSD - The <schema> Element
- XSD Simple Elements
- XSD Restrictions/Facets
- XSD Complex Elements

You can also read the course presentation.
Second, do not forget the difference between an XML Schema and an XML file. XML Schema is equivalent to Java class and an XML fine that instantiates the schema is equivalent to an object of the class. Remember the Figure below:

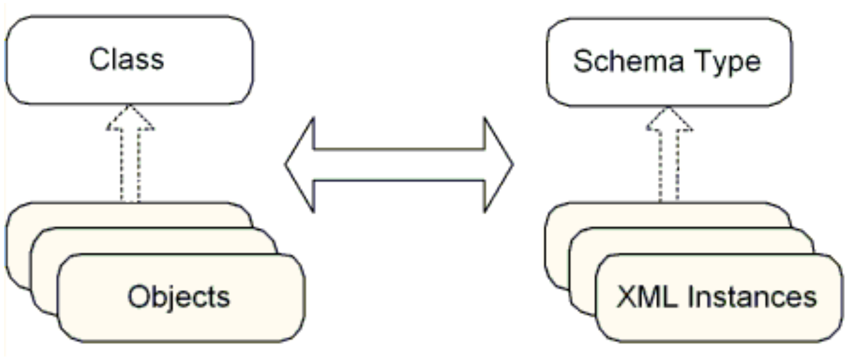

FIG. 1 – *Comparison between XML Schema and Java classes*

Remark

The solution of this exercise can be done using notepad++. Open a new file and name it as order.xsd. The instance should be called instance.xml. Compress both of them as a zip file and send them to the email address mentioned above. Please put "WS-TD" in the subject (title) of the email.

2 XML RPC

Imagine that there is a service that provides a currency converter[1]. This service is a method named currencyConvertor that takes the following arguments:

1. The currency of the source: string (e.g. euro)
2. The currency of the destination: string (e.g. dollar)
3. The amount: double

The method return the converted value as a double
Go to the slides 18 & 20 of the course on XML-RPC. write a similar request and response.
Remember that the order of argument in XML-RPC request is important.

Remark

The solution of this exercise can be done using notepad++. Open a new file and name it as xmlrpc.xml. Save it and send it to the email address mentioned above. Please put "WS-TD" in the subject (title) of the email.

Part 2: SOAP (Tools needed: Eclipse)

3 Exercise: Writing a SOAP Client

The goal is to write a Java application that consumes a web service. We are not writing the service yet.

A web service client is a small program (written in Java in our case) that makes a call to a web service available over the Internet and makes use of the result.
There are a few of SOAP web services available for free. WebserviceX.net is a website that provides such web services.
In this exercise we will write a client that uses the GeoIPService[2] available on this website. The GeoIPService is a web service that returns the geographic location (i.e. country) of a given IP address. This IP address is passed as a string argument to the GeoIPService.

Write a simple Java program that:

1. Takes an IP address as a String argument (i.e. in main (args[]))
2. Calls the address and passes the string as argument.
3. Retrieves the results and displays it on the console (i.e. System.out.println())

But before, open the the web service page: GeoIPService and see the following sections: **Description, WSDL Schema Location**. Also, take a look at the WSDL.
Note that the WSDL is a long document. But do not worry about it. We have tools that generate it automatically and parse it automatically. However, there are two important tags:

1. <wsdl:service name="GeoIPService"> which gives the name of the service.
2. <wsdl:port name="GeoIPServiceSoap" binding="tns:GeoIPServiceSoap">

These two elements will be revisited later. Now, open Eclipse environment. `new` ▶`Java project` ▶`IPLocationFinder`.
Add a new Java class (called IPLocationFinder) to this project. This method should take the IP address as argument.

---

1. Convertisseur de devises.
2. If the web site webservicex is not available for the moment, you can use an equivalent service on GeoIP Service alternative. Do NOT forget use the SOAP service.

In order to call the service, we need to add the *Service End Point* (SEI) interface to our project. In the SOAP environment, each web service has a SEI that allows the clients (that maybe written in different languages) to integrate a stub. The stub is a pointer to the service method. When we want to invoke the service, we call it via the stub. For instance:
Imagine that myStub is the generated stub. Then, calling the web service will look like: myStub.getCountryName(ipAddress) where the ipAddress is the string passed as argument.

In Java we have tools that generate the stub and some other useful classes.

3.1 Stub Generation

This tool is wsimport. To use it do the following:

1. Open the commad line (cmd).
2. Create a new folder (using mkdir) called (SEI).
3. Type the command wsimport. a breif description of the functionality of this command will be shown.[3]. Do not forget to take a look.
4. Note the syntax of wsimport command. It takes the WSDL URL as argument. In the case of the GeoIPService, this is the url of the WSDL.
5. Run the wsimport command with the WSDL URL of the GeoIPService.
6. Check the content of the SEI folder. You find a folder named "net". Inside it you can find some .class files which include the stub.
7. If you want to see the .java files (the source files). Run the wsimport command with the "keep" option.

3.2 Using The Created Stub

1. Open any of the created file and see the declared package. Now, you need to go to Eclipse and add a new package to the project with the same name. Then import these files to the package you created in Eclipse. Now, we have the stub generated. We need to call it in our java application. But where is the Stub? We need to ask the WSDL.
2. Open the WSDL in the internet browser and find the element : <wsdl:service name="GeoIPService"> mentioned above. GeoIPService is the name of the class that you need to use. Create an instance of this class in your program.
3. Find a method in this instance that return an object of the same type mentioned in the port: <wsdl:port name="GeoIPServiceSoap" binding="tns:GeoIPServiceSoap">.
4. Call the method, retrieve the created object in a local variable called service.
5. This soap has a method that takes a string (the IP address) and returns the country. Find this method, call it and display the result.
6. Open the terminal again, make a "ping" to the TSE web site. Copy the IP and pass it as argument to the method. Repeat the same thing for google.com, google.fr, http://www.un.org/fr/, youtube.com etc.
   Remember that the stub is only an INTERFACE to the service invoked.
   Your program did not download the service code locally. What you've done is only downloading an interface (the stub) and using it to communicate with the server.
   You called a method that created a SOAP message, sent it and retrieved the results.

– Compress the project of Exercise 3 as a .zip file, attach it with the solutions of EX1 and EX2 and send it to the email address mentioned above. Please put "WS-TD" in the subject (title) of the email.

3. If you do not have this message displayed, then probably you have an installation problem. Please inform the instructor.

- For those who finish early, you can pick up another web service from WebserviceX.net and write a client to use it.