Axis2 recap and XPATH

Introduction to Service Design and Engineering 2013/2014.

Lab session #3

University of Trento

Cristhian Parra: Lab Lectures for IntroSDE-2013/2014
Some slides from LabSDE-2012/2013

Outline

- Recap axis2 web service sample of the last session
- Practice how to create a simple axis2 web service by doing lab2/Exercise3
- Example of XML reading using XPATH and java (covered in Lecture 3)

Lab 2 Recap

Before starting with this lab, we will review the the example 2 of Lab #2 step by step.

```
cd lab2/Example2/quickstart
open resources/META-INF/services.xml
```

1. Check the service.xml file

```
<!-- The name of the service -->
<!-- Scope and nameSpace must be later matched inthe build.xml -->
<service name="StockQuoteService"
  scope="application"
  targetNamespace="http://quickstart.samples/"
>
  <!-- a description of what the service does -->
  <description>Stock Quote Service</description>
  <!-- the classes that will process incoming messages to the service -->
  <!-- we are using standar message receivers already bundled with axis2 -->
  <messageReceivers>
    <messageReceiver
      mep="http://www.w3.org/2004/08/wsdl/in-only"
      class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver"
    />
    <messageReceiver
      mep="http://www.w3.org/2004/08/wsdl/in-out"
      class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"
    />
  </messageReceivers>
  <!-- definition of the xml schema used by the service -->
  <schema schemaNamespace="http://quickstart.samples/xsd"/>
  <!-- specification of what Java class is being exposed as a service -->
  <parameter
    name="ServiceClass">samples.quickstart.service.pojo.StockQuoteService
  </parameter>
</service>
```

2. Check/Adjust the build.xml file

First, replace the properties AXIS2_HOME and AXIS2_HOME_TOMCAT to make it fit your local configuration

```
<!-- replace them appropiately to match the places where these folders are
    in your computer -->
  <property name="AXIS2_HOME_TOMCAT"
          value="/opt/apache-tomcat-7.0.39/webapps/axis2"/>
  <property name="AXIS2_HOME"
         value="/opt/axis2-1.6.2"/>
```

For example, in your windows PC it might look like

```
  <property name="AXIS2_HOME_TOMCAT"
        value="C:\apache-tomcat-7.0.39\webapps\axis2"/>
  <property name="AXIS2_HOME"
        value="C:\axis2-1.6.2"/>
```

Appart from the typical "compile" you have *generate.wsdl* and a *generate.service*.

```
<!-- generates the wsdl of the service using java2wsdl and the service.xml -->
<target name="generate.wsdl" depends="compile.service">
      <taskdef name="java2wsdl"
               classname="org.apache.ws.java2wsdl.Java2WSDLTask"
               classpathref="axis2.classpath"/>
      <!-- This is the key: java2wsdle creates the wsdle for a class -->
      <java2wsdl className="samples.quickstart.service.pojo.StockQuoteService"
                 outputLocation="${build.dir}"
                 <!-- remember, these two must be the same as in service.xml -->
                 targetNamespace="http://quickstart.samples/"
                 schemaTargetNamespace="http://quickstart.samples/xsd">
          <classpath>
              <pathelement path="${axis2.classpath}"/>
              <pathelement location="${build.dir}/classes"/>
          </classpath>
      </java2wsdl>
  </target>
<!-- packages the compiled classes in a file and deploys the service -->
<target name="generate.service" depends="compile.service">
      <!--Copy all the classes and the service.xml file to classes folder -->
      <copy toDir="${build.dir}/classes" failonerror="false">
          <fileset dir="${basedir}/resources">
              <include name="**/*.xml"/>
          </fileset>
      </copy>
      <!-- Create a packaged version of the service named "StockQuoteService" -->
      <jar destfile="${build.dir}/StockQuoteService.aar">
          <fileset excludes="**/Test.class" dir="${build.dir}/classes"/>
      </jar>
      <!-- this part is about already deploying the service to Axis2 -->
      <copy file="${build.dir}/StockQuoteService.aar"
            toDir="${AXIS2_HOME}/repository/services"
            overwrite="yes">
      </copy>
  </target>
```

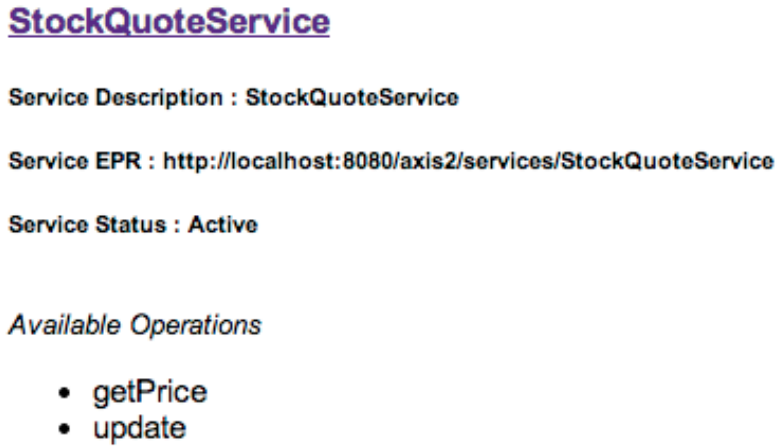
3. Generate WSDL and the packaged service (.aar)

```
ant generate.wsdl
ant generate.service
```

4. Check that the service has been deployed

```
open http://localhost:8080/axis2/services/listServices
```

You should see something like:



Remember, deploying the service is simply copying the generated .aar to the services folder of axis. In the same way *removing* the service is just deleting that file.

5. Check the service WSDL

WSDL will be covered in a future lecture:

```
open http://localhost:8080/axis2/services/StockQuoteService?wsdl
```

6. Call on the available operations on the StockQuoteService

```
open http://localhost:8080/axis2/services/StockQuoteService/getPrice?symbol=IBM
```

The answer is a soap message as follows:

```
<ns:getPriceResponse xmlns:ns="http://quickstart.samples/xsd">
  <ns:return>42.0</ns:return>
</ns:getPriceResponse>
```

The StockQuoteService class has also an update operation which if called, updates the stock value for a company symbol.

```
open http://localhost:8080/axis2/services/StockQuoteService/update?symbol=IBM&price=150
```

Calling the getPrice service now will give

```
<ns:getPriceResponse xmlns:ns="http://quickstart.samples/xsd">
  <ns:return>150</ns:return>
</ns:getPriceResponse>
```

7. Exercise

Now, take a couple of minutes to do exercise 3 from last session:

Modify the *HealthProfileReader* and expose it through an axis2 web service (solution)

Hint: you won't need a main, but you will need to initialize your database just once for the whole class (see static blocks in java)

XML Navigation: XPath

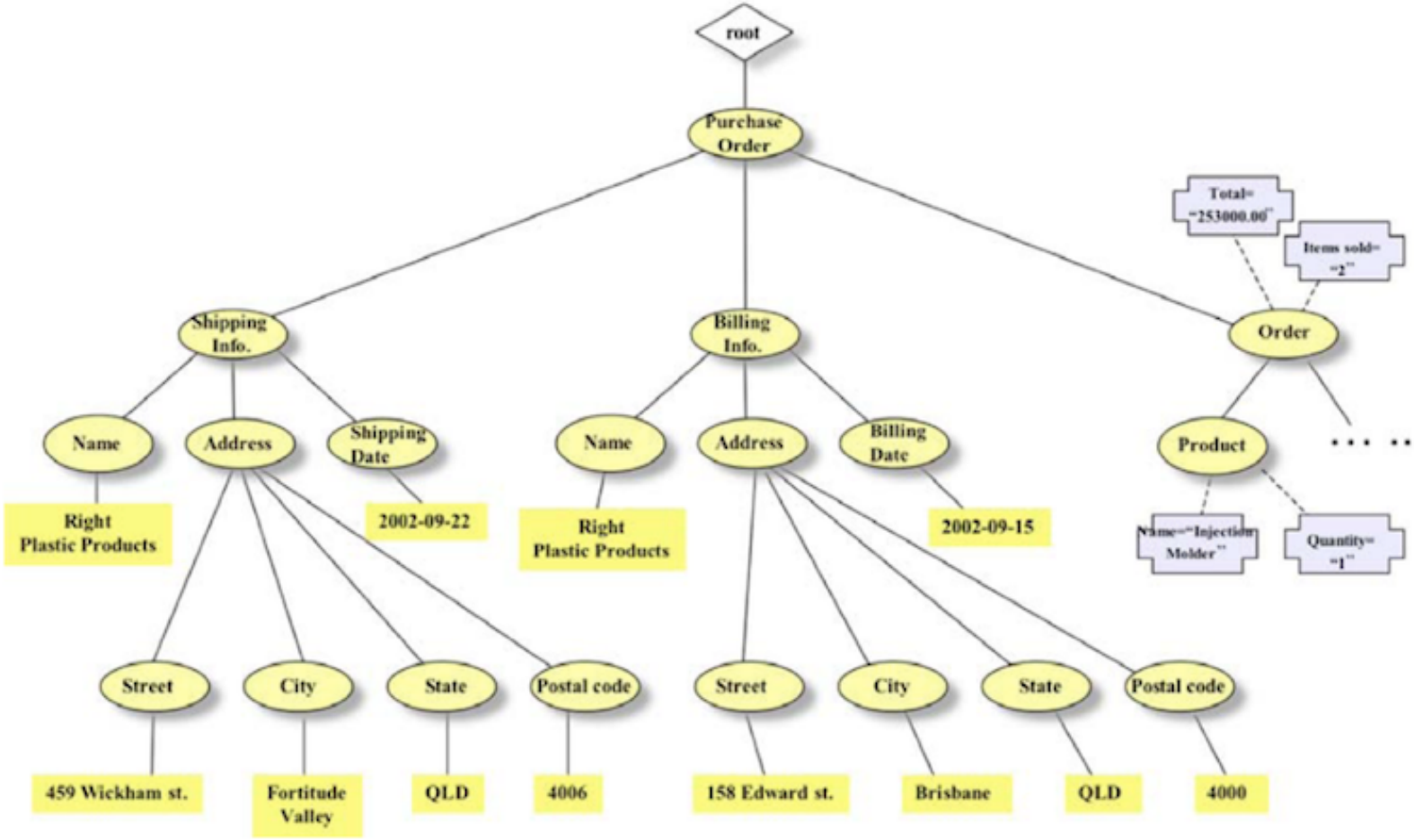
Reading/Parsing XML documents

XPATH Introduction

- XPath is a **query language** designed for querying XML documents (actually, DOM trees)
- XPath uses path expressions to navigate in XML documents
- XPath contains a library of standard functions
- XPath describes paths to elements in a similar way an operating system describes paths to files
- XPath is a W3C recommendation: http://www.w3.org/TR/xpath20/

XPATH Tree

Take a look again to the XPath tree you saw during lectures. What's the similarity with an x
document?



XML Documents: Nodes & Relationships

- bookstore is the **parent** of **book**; **book** is the **parent** of the two **chapters**
- The two chapters are the **children of book**, and the section is the **child of the second chapter**
- The two chapters of the book are **siblings** (they have the same parent)
- Library, book, and the second chapter are the **ancestors** of the section
- The two chapters, the section, and the two paragraphs are the **descendants** of the book

```
<library>
  <book>

    <chapter>
    </chapter>

    <chapter>
      <section>
        <paragraph/>
        <paragraph/>
      </section>
    </chapter>

  </book>
</library>
```

- It is also a **TREE!**

Example 1: using xpath in java

- Open lab3/Example1-XPATH/XPathTest.java
- Compile it
- Execute it

```
// we are using the library javax.xml.xpath.XPath
import javax.xml.xpath.XPath;
...
public class XPathTest {
    public static void main(String[] args)  throws ParserConfigurationException, SAXException,
      // 1. read the xml file and parse it to a in-memory DOM tree
      DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
      domFactory.setNamespaceAware(true);
      DocumentBuilder builder = domFactory.newDocumentBuilder();
      System.out.println("Loading books.xml...");
      Document doc = builder.parse("books.xml");
      // 2. create an XPath object to evaluate an XPathExpression
      XPathFactory factory = XPathFactory.newInstance();
      XPath xpath = factory.newXPath();
      XPathExpression expr = xpath.compile("/bookstore/book/title/text()");
      // 3. Operate on the result
      Object result = expr.evaluate(doc, XPathConstants.NODESET);
      NodeList nodes = (NodeList) result;
      for (int i = 0; i < nodes.getLength(); i++) {
          System.out.println(nodes.item(i).getNodeValue());
      }
    }
}
```

Exercise 1 (10 min)

- In the *lab3/my-solutions* folder, copy XPathTest.java and modify it to print also the authors

Node Selection

Think about the xml document for the bookstore

```
<bookstore>
    <book year="2000">
        <title lang="eng">Snow Crash</title>
        <author>Neal Stephenson</author>
        <publisher>Spectra</publisher>
        <isbn>0553380958</isbn>
        <price>14.95</price>
    </book>

    <book year="2005">
        <title>Burning Tower</title>
        <author>Larry Niven</author>
        <author>Jerry Pournelle</author>
        <publisher>Pocket</publisher>
        <isbn>0743416910</isbn>
        <price>5.99</price>
    </book>

    <book year="1995">
        <title>Zodiac</title>
        <author>Neal Stephenson</author>
        <publisher>Spectra</publisher>
        <isbn>0553573862</isbn>
        <price>7.50</price>
    </book>
</bookstore>
```

Node Selection (1)

- A path that begins with a / represents an absolute path, starting from the top of the document **/bookstore/book/title**
- Note: an absolute path can select more than one element
- A / by itself means “the whole document”
- A path that does not begin with a / represents a path starting from the current element
- book/title Selects all title elements that are children of book

Node Selection (2)

- A path that begins with // can start from anywhere in the document.
- //title Selects every element title, no matter where it is.
- bookstore//title Selects all title elements that are descendant of the bookstore element, no matter where they are under the bookstore element.

```
XPathExpression expr = xpath.compile("//book[price " + condition + "'" + price + "']");
```

Predicates

- Predicates are used to find a specific node or a node that contains a specific value.
- /bookstore/book[1] Selects the first book element that is the child of the bookstore element.
- /bookstore/book[last()] Selects the last book element that is the child of the bookstore element.
- /bookstore/book[position()<3] Selects the
- first two book elements that are children of the bookstore element.
- /bookstore/book[price>3] Selects all the book elements of the bookstore element that have a price element with a value greater than 3.

```
XPathExpression expr = xpath.compile("/bookstore/book[isbn='" + ISBN + "']");
```

Attributes

- You can select attributes by themselves, or elements that have certain attributes.
- To choose the attribute itself, prefix the name with @
- //@lang Selects all attributes that are named lang
- To choose elements that have a given attribute, put the attribute name in square brackets
- //title[@lang='eng'] Selects all the title elements that have an attribute named lang with a value of 'eng'

Wildcards

- "*" Matches all element node at this level
- /bookstore/* Selects all the children nodes of the bookstore element
- @* Matches all attribute node
- //title[@*] Selects all title elements which have any attribute
- node() Matches any node of any kind

Several Paths

- By using the | operator in an XPath expression you can select several paths.
- //book/title | //book/price
  - Selects all the title and price elements of all book elements
- /bookstore/book/title | //price
  - Selects all the title elements of the book element of the bookstore element and all the price elements in the document

Axes

- An axis is a set of nodes relative to a given node
- X::Y means “choose Y from the X axis”
- child::book Selects all book nodes that are children of the current node
- child::text() Selects all text child nodes of the current node
- child::node() Selects all child nodes of the current node
- ancestor::book Selects all book ancestors of the current node

```
XPathExpression expr = xpath.compile("//child::book[author='" + authorName + "']");
```

Arithmetic Operators

- "+" add
- "-" subtract
- "*"" multiply
- div (not /) divide
- mod modulo (remainder)

Boolean Operators

- = equals (Notice it's not ==)
- != not equals
- value = node-set will be true if the node-set contains any node with a value that matches value
- value != node-set will be true if the node-set contains any node with a value that does not match value
- Hence, value = node-set and value != node-set may both be true at the same time!
- And
- Or
- not()
- The following are used for numerical comparisons only:
  - <, <=, >, >=

Java and XPath types

- XPath and Java language do not have identical type systems
- XPath 1.0 has only four basic data types:
  - node-set
  - Number
  - Boolean
  - String
- The evaluate() method may return
  - org.w3c.dom.NodeList
  - java.lang.Double
  - java.lang.Boolean
  - java.lang.String
  - org.w3c.dom.Node
- When you evaluate an XPath expression in Java, the second argument specifies the return type you expect.

Example 2: xpath advanced

- Explore **lab3/Example2-XPath/XPathAdvance.java** to see the examples of what we have seen in prevous slides
- Compile it
- Run it

Exercise 2

- Replace the HashMap db in the HealthProfile Reader with a xml file as follows

```
<people>
    <person>
        <firstname>George R. R.</firstname>
        <lastname>Martin</lastname>
        <healthprofile>
            <weitgh>120</weitgh>
            <height>1.65</height>
        </healthprofile>
    </person>
    <!-- add more people to the db -->
</people>
```

- Use xpath to implement methods like getWeight and getHeight
- Make a function which prints all people in the list with detail
- A function which accepts name as parameter and print that particular person HealthProfile
- A function which accepts weight and an operator (=, > , <) as parameters and prints people that fulfill that condition.

Bonus Example: using xpath on the wild

- Get scraper (chrome only extension)
- Go to meteotrentino.it
- Inspect Element on top of the temperature
- Copy as XPATH (//div/div/span/strong)
- Scrape Similar