GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

Data Binding with JAXB and JAX:RS

Data externalisation or marshalling is the process of converting a graph of objects into a format suitable for transmission between running processes. There are numerous forms of data exchange formats, typically specific to a given domain area, e.g. finance, medicine, biology. In the context of inter-process communication (IPC) however, there are really only three modes of externalisation that we are concerned with:

- **Binary data formats**, e.g. the CDR format used by CORBA. These have the benefit of being highly structured, highly compact and highly efficient. However, the scoping and development of binary standards also highly laborious and difficult to implement for a heterogeneous distributed system.

- **Semi-compiled formats**, e.g. the serialized forms used in Java and C#. Serialization is highly efficient and flexible, but only works in a homogenous distributed system, i.e. a J2EE or .NET platform.

- **Unicode Formats**, e.g. CSV, XML and JSON. These forms are the least efficient but most flexible forms of data externalization and are ideal as "glueware" in heterogeneous systems. In particular, JSON and XML formats combine the flexibility and portability of Unicode formats with a very fine-grained degree of structure.

In this lab we will see how an XML schema document can be used as a DDL (Data Definition Language) for creating a platform and language neutral mechanism for data exchange. We are going to use the JAXB API (Java Architecture for XML Processing, part of the core Java SDK) to generate a suite of Java classes from an XML schema. We will then use the framework to convert objects to XML and JSON and vice versa. XML Schema 1.1. is a recommended standard from the W3C consortium (www.w3c.org). A schema allows us to create a grammar for a legal XML document, i.e. to specify the sequence and multiplicity of elements and the legal types to use.You should familiarise yourself with the standard at https://www.w3.org/XML/Schema, in particular, the Specifications and Development section (especially XML Schema Part 2: Datatypes).

1. **Open the XML Schema po.xsd in a text editor** and examine the structure of the document. Much like a programming language, the XML schema standard allows us to build complex types from a small set of simple types. These simple types can be translated into programming types use a set of language bindings. Note that the namespaces in the schema point at http://sw.gmit.ie/ds/. Also note that the complex type *Items* has **1 : *n*** relationship with an *Item*.

2. Open a command prompt, navigate to the directory containing the schema po.xsd and **compile the schema** using the XML Java Converter tool (XJC) as follows:

   ```
   xjc -npa po.xsd
   ```

   The schema compiler tool generates a Java class for each complexType defined in the po.xsd document. The members of each generated Java class are the same as the

elements inside the corresponding complexType, and the class contains getter and setter methods for these fields. The generated classes include an object factory for manufacturing instances of *PurchaseOrder*.

3. Open the file PurchaseOrder.java and **add the annotation @XmlRootElement** to the generated class and the import *javax.xml.bind.annotation.XmlRootElement*.

4. Open the class *JAXBPOExample* in a text editor and examine the import statements and code in the main() method. The class creates an instance of the generated class *PurchaseOrder* and then marshals and unmarshals the state of the object graph in XML and JSON format.

5. **Compile all the source code**, including the class *JAXBPOExample* in the current directory. Note that the Apache Jersey, Jackson and ASM libraries are part of the JAX-RS (Java API for RESTful Web Services) API and are only required for JSON:

   Linux / Unix / OSX:

   ```
   javac -cp .:./jersey-bundle-1.17.1.jar:./jackson-all-1.9.11.jar:./asm-3.3.jar
   $(find ./* | grep .java)
   ```

   MS Windows:

   ```
   javac -cp .;./jersey-bundle-1.17.1.jar;./jackson-all-1.9.11.jar;./asm-3.3.jar
   ie/gmit/sw/ds/*.java

   javac -cp .;./jersey-bundle-1.17.1.jar;./jackson-all-1.9.11.jar;./asm-3.3.jar
   JAXBPOExample.java
   ```

6. **Run the application** by executing the following command from the terminal:

   Linux / Unix / OSX:

   ```
   java -cp .:./jersey-bundle-1.17.1.jar:./jackson-all-1.9.11.jar:./asm-3.3.jar
   JAXBPOExample
   ```

   MS Windows:

   ```
   java -cp .;./jersey-bundle-1.17.1.jar;./jackson-all-1.9.11.jar;./asm-3.3.jar
   JAXBPOExample
   ```

   Examine the content of the screen output and the XML and JSON files generated in the process.

7. **Execute the Python script** *JAXBPOExample.py* to prove that the JSON format is language neutral:

   ```
   python JAXBPOExample.py
   ```