Practical Training XML
(September / October 2005)

(c) Prof Dr. Wolfgang May
Universität Göttingen, Germany
may@informatik.uni-goettingen.de

Practical Training ("Praktikum") in Informatics;
4 weeks full time block course, 6 ECTS Credit Points

Chapter 3
XML Processing with Java with JAXP and JAXB

- ***JAXP*** ⇒ **the *Java API for XML Processing***
  - contains the SAX (Simple API for XML) and the DOM (Document Object Model) API
  - Further information on XML processing with Java can be found under:
    ⇒ `http://java.sun.com/xml/index.jsp`
    ⇒ `http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial`
- ***JAXB*** ⇒ **the *Java API for XML Binding***
  - Part of the Java Web Services Developer Pack
  - Further information on JAXB can be found under:
    ⇒ `http://java.sun.com/webservices/jaxb`
    ⇒ `http://java.sun.com/webservices/docs/1.5/tutorial/doc`

3.1 SAX - The Simple API for XML

The simplest way of retrieving information from an XML document into a Java program is by parsing the document with *SAX*, the *Simple API for XML*.

XML PARSING WITH SAX: BASIC IDEAS

- a generic XML Parser is parameterized with a *Content Handler* (plus *Error Handler*,*DTD Handler*, and *EntityResolver*) implementation.
- On parsing the XML document, the parser produces events like `startDocument` / `endDocument`, `startElement` / `endElement` etc, describing the logical content of the XML document which is obtained during the parsing process.
- With the content handler implementation, the user provides “actions” in shape of Java code, associated with a specific event.
- if during parsing of the XML document, a specific event occurs, the code of the associated action from the content handler is invoked (*“callback”*).
- Since the content handler (and also DTD and error handler plus entity resolver) implementation is supplied by the application programmer, the programmer can specify the behavior of the parser for each specific XML event.

SAX EXAMPLE CODE

Consider a very simple application that

- detects all elements with attributes
- for each element, output the element's name
- for each element, output the name-value pairs of its attributes

```
>java PrintAttributes mondial.xml > blah.out
>less blah.out
element: country
 - attribute: 'car_code' value: 'AL' type: 'ID'
 - attribute: 'area' value: '28750' type: 'CDATA'
 - attribute: 'capital' value: 'cty-cid-cia-Albania-Tirane' type: 'IDREF'
 - attribute: 'memberships' value: 'org-BSEC org-CE org-CCC org-ECE org-EBRD org-EU ...' type: 'IDREFS'
element: encompassed
 - attribute: 'continent' value: 'europe' type: 'IDREF'
 - attribute: 'percentage' value: '100' type: 'CDATA'
element: ethnicgroups
 - attribute: 'percentage' value: '3' type: 'CDATA'
element: ethnicgroups
 - attribute: 'percentage' value: '95' type: 'CDATA'
element: religions
 - attribute: 'percentage' value: '70' type: 'CDATA'
...
```


Class "ContentHandlerPrintAttributes.java":

```
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class ContentHandlerPrintAttributes extends DefaultHandler {
    public ContentHandlerPrintAttributes(String message) {
        System.out.println(message);
    }

    public void startElement(String url, String localName, String qName,
            Attributes attrs) throws SAXException {
        if (attrs.getLength() > 0) {
            String elementName;
            if(qName == null || qName.equals("")) elementName = localName;
            else elementName = qName;
            System.out.println("element: " + elementName);
            for (int i = 0; i < attrs.getLength(); i++) {
                System.out.println(" - attribute: '" + attrs.getQName(i)
                        + "' value: '" + attrs.getValue(i) + "' type: '"
                        + attrs.getType(i)+"'");
            }
            System.out.println();
        }
    }
}
```

[see codesnippets/SAX/ContentHandlerPrintAttributes.java]

Class “PrintAttributes.java”:

```
import java.io.IOException;
import javax.xml.parsers.*;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class PrintAttributes {
    public static void main(String[] args) {
        if (args.length != 1) {
            System.err.println("usage: PrintAttributes <url>");
            System.exit(1);
        }
        String url = args[0];
        DefaultHandler handler = new ContentHandlerPrintAttributes(
                "printing attributes of document at url '" + url + "'");
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            SAXParser parser = factory.newSAXParser();
            parser.parse(url, handler);
        } catch (IOException e1) {
            e1.printStackTrace();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        }
    }
}
```

[see codesnippets/SAX/PrintAttributes.java]

WRITING A SAX PARSER:SUMMARY

- There exists a *Generic SAX Parser*.
- The Generic Parser must be parameterized with (at least) an *EventHandler* implementation.
- The most trivial Event Handler is the *DefaultHandler* that does nothing: document is parsed, events are detected, but no action is performed.
- Normally, the user-provided Event Handler extends the DefaultHandler, overwriting (some of) its Event Methods.
- even with only the unextended DefaultHandler:
  DTD / XML Schema validation is still performed (if switched on).

ADDITIONAL SAX FACTS

- SAX Parsers can be parameterized with *Event Handlers*, *DTD handlers*, *Error Handlers* and *Entity Resolvers*.
- most parsers provide only (partial) Event Handler implementations and have no Error Handler, no DTD Handler and no Entity Resolver.
- *DTD Handlers* are for dealing with unparsed entities (deprecated feature).
- *DTD Handlers* do not give access to the content models and attribute definitions of the DTD of the processed document.
- *Error Handlers* can be used to customize the parser behavior when encountering fatal errors (e.g. non-well formed documents), errors (e.g. validation errors) and warnings.
- *EntityResolver* deals with URN resolution.
- SAX parsers can be instantiated either in *validating* or *non-validating* mode (has nothing to do with whether DTD Handler is provided or not).
- For further information, the respective chapters from the Sun J2EE Tutorial are strongly recommended for studying: http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html $\Rightarrow$ Chapter 4 and 5

SAX: PRO'S AND CON'S

Plus:

- simple to understand, simple to implement
- interaction solely via Java interfaces, hence completely implementation-independent: The user's content handler implementation has to implement (e.g.) the *ContentHandler* interface (`org.xml.sax.ContentHandler`).
- fast
- well-suited also for large XML instances and XML streams

Minus:

- XML serial access only: data structure is not stored, therefore no implicit "memory"; makes detection and modification of global document properties quite difficult

3.2 DOM - The Document Object Model

A more sophisticated way of parsing XML documents is introduced with the

Document Object Model (DOM).

XML PARSING WITH DOM: BASIC IDEAS

- the Document Object Model is:
  an abstract data type, defining structures and operations for a tree representation of an XML document.
- the Document Object Model is not:
  An API. (but: numerous DOM implementations (APIs) exist for different programming languages, like Java, C++, ML/O'Caml, Haskell, Perl, Python etc)
- Result of the parsing is a DOM tree representation in memory during runtime.
- Various operations allow navigation, querying and modification of DOM trees after parsing.
- In the JAXP framework, it is possible to use DOM trees as input / output instead of ascii files, e.g. for XSL transformations.

EXAMPLE: DOMSTATS

Consider a DOM Parser for counting the nodes in the document according to their node type:
{*Element, Attribute, Text, CDATA Section, Entity Reference, Entity, Processing Instruction, Comment, Document, Document Type, Document Fragment, Notation nodes*}:

```
>parsing xml file mondial.xml
 document URI: file:C:/AndereProgramme/eclipse/workspace/xmlmusterlsgn_extern/mondial.xml
 XML encoding: utf-8
 XML version: 1.0
 doctype: mondial
 ELEMENT_NODES: 24176
 ATTRIBUTE_NODES: 25227
 TEXT_NODES: 71672
 CDATA_SECTION_NODES: 0
 ENTITY_REFERENCE_NODES: 0
 ENTITY_NODES: 0
 PROCESSING_INSTRUCTION_NODES: 0
 COMMENT_NODES: 0
 DOCUMENT_NODES: 1
 DOCUMENT_TYPE_NODES: 1
 DOCUMENT_FRAGMENT_NODES: 0
 NOTATION_NODES: 0
 total no of nodes: 121077
>
```



```
import java.io.File;
import java.io.IOException;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class DomStats {
    public static void main(String[] argv) {
        // input args processing
        if (argv.length != 1) {
            System.err.println("Usage: java DomStats <inputfilename>");
            System.exit(1);
        }

        // instantiating and configuring the builder factory
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setValidating(false);
        factory.setNamespaceAware(true);
        try {
            // instantiating the document builder
            DocumentBuilder builder = factory.newDocumentBuilder();
            // parsing the file into org.w3c.dom.Document doc
            Document document = builder.parse(new File(argv[0]));
            System.out.println("parsing '" + argv[0]+"'\n  doc URI: " + document.getDocumentURI() +
            "  \nXML encoding: " + document.getXmlEncoding() + "\n  XML version: " + document.getXmlVersion());
            if (document.getDoctype() != null)
                System.out.println("  doctype: "+ document.getDoctype().getName());
            int[] stats = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
            doStats(document, stats);
            System.out.println("  ELEMENT_NODES: " + stats[1]);
            System.out.println("  ATTRIBUTE_NODES: " + stats[2]);
            System.out.println("  TEXT_NODES: " + stats[3]);
            System.out.println("  CDATA_SECTION_NODES: " + stats[4]);
            System.out.println("  ENTITY_REFERENCE_NODES: " + stats[5]);
            System.out.println("  ENTITY_NODES: " + stats[6]);
            System.out.println("  PROCESSING_INSTRUCTION_NODES: " + stats[7]);
            System.out.println("  COMMENT_NODES: " + stats[8]);
            System.out.println("  DOCUMENT_NODES: " + stats[9]);
            System.out.println("  DOCUMENT_TYPE_NODES: " + stats[10]);
            System.out.println("  DOCUMENT_FRAGMENT_NODES: " + stats[11]);
            System.out.println("  NOTATION_NODES: " + stats[12]);
            System.out.println("  total no of nodes: " + stats[0]);
        } catch (ParserConfigurationException e) { e.printStackTrace();
        } catch (SAXException e) { e.printStackTrace();
        } catch (IOException e) { e.printStackTrace(); }
    }

    private static void doStats(Node node, int[] stats) {
        stats[0]++;
        stats[node.getNodeType()]++;
        NodeList children = node.getChildNodes();
        for (int i = 0; i < children.getLength(); i++) doStats(children.item(i), stats);
        NamedNodeMap attrs = node.getAttributes();
        if (attrs == null) return;
        for (int i = 0; i < attrs.getLength(); i++) doStats(attrs.item(i), stats);
    }
}
```

[see DOM/DomStats.java]

DOM: PRO'S AND CON'S

Plus:

- powerful and flexible; allows for easy detection and modification of global properties; structure-recursive access $\Leftrightarrow$ sequence of events from pre/post-order document pass
- can be used as in-memory XML interchange format within JAXP, JAXB and other XML frameworks

Minus:

- larger overhead in terms of learning and programming (navigational concepts)
- (usually) slower than SAX
- not usable for *really* large XML instances and continuous XML streams

DOM FACTS

- can be used in *validating* or *non-validating* mode (but validation seems to happen anyway)
- DOM is defined as a language-independent data model
  $\Rightarrow$ JAXP DOM implementation does not take maximum advantage of OO-modeling concepts. But:
  There is a DOM-like API for Java called *JDOM*, which is more strictly object-oriented.
- The current DOM implementation in JDK 1.5 is based on the Apache project Xerces for XML processing.
- For further information, the respective chapter from the Sun J2EE Tutorial is strongly recommended for studying:
  http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html $\Rightarrow$ Chapter 6: Document Object Model

3.3 JAXB - The Java API for XML Binding

Use case: software configuration management via XML files

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="config.xsd">
  <userprofiles>
    <user name="wmay" realname="Wolfgang May" last_login="19/9/2005-17:41.02"/>
    <user name="ofritzen" realname="Oliver Fritzen" last_login="21/9/2005-13:04:27"/>
      ...
  </userprofiles>
  <cachesizes>
    <size user="wmay" value="0"/>
    <size user="ofritzen" value="65635"/>
      ...
  </cachesizes>
</configuration>
```

- software with different users
- each user has properties (name, last login etc)
- each user has cache size setting

Imagine the case you want to update wmay's settings for last login:

PROCESSING ALTERNATIVES

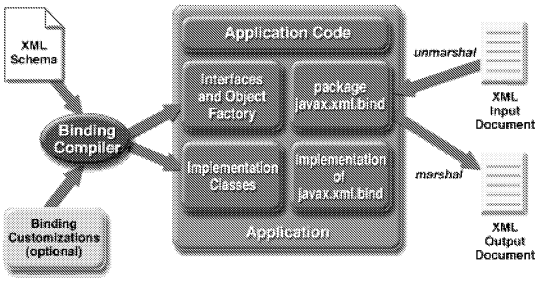
- using DOM:
  1. parsing document into DOM tree
  2. modify DOM tree (navigate through tree, find and finally update some attribute node)
  3. write back DOM tree
- using SAX:
  1. create java class "Configuration.java" by hand, containing the configuration data
  2. instantiate configuration object
  3. parse document; events "fill" configuration object with the configuration data from the configuration file
  4. modify configuration object
  5. serialize configuration object to XML file via hand-written method(s)

  $\Rightarrow$ even worse!

JAXB ADVANTAGES

- provides convenient way to bind XML schemas to Java representations.
- provides methods for unmarshalling XML instance documents into Java objects.
- provides methods for marshalling Java objects back into XML instance documents.

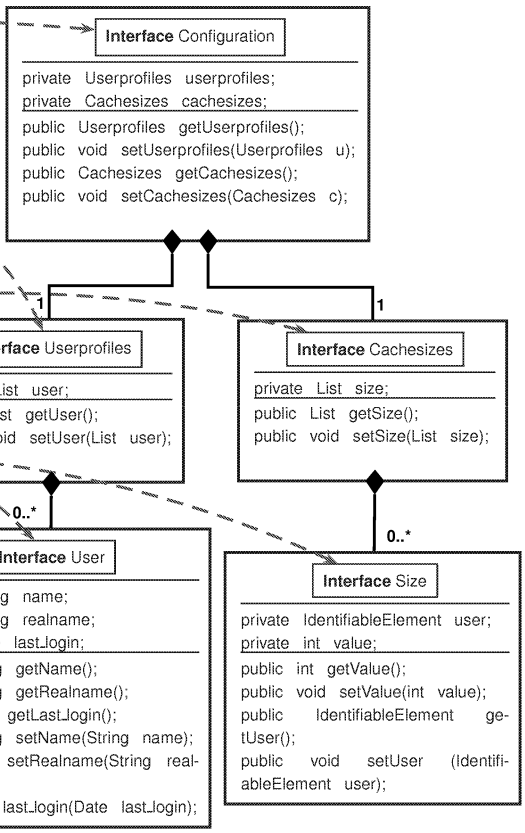
Architecture Overview



- XML Schema
- Binding Compiler
- Implementation of `javax.xml.bind`
- Schema-Derived Classes
- Java Application
- XML Input / Output Documents

JAXB: BINDING XML SCHEMA TO JAVA CLASSES / INTERFACES

```
<xs:element name="configuration">
 <xs:complexType >
  <xs:sequence>
   <xs:element name="userprofiles">
    <xs:complexType><xs:sequence>
     <xs:element name="user" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
       <xs:attribute name="name" type="xs:ID"/>
       <xs:attribute name="realname" type="xs:string"/>
       <xs:attribute name="last_login" type="xs:date"/>
      </xs:complexType>
     </xs:element>
    </xs:sequence></xs:complexType>
   </xs:element>
   <xs:element name="cachesizes">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="size" minOccurs="0" maxOccurs="unbounded">
       <xs:complexType>
        <xs:attribute name="user" type="xs:IDREF"/>
        <xs:attribute name="value" type="xs:integer"/>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```



JAXB: EXAMPLE CODE PIECE

```
        ...
        try {
            JAXBContext jc = JAXBContext.newInstance("configuration.binding");
            ObjectFactory objFactory = new ObjectFactory();
            Unmarshaller u = jc.createUnmarshaller();
            Configuration conf = (Configuration) u.unmarshal(new FileInputStream(infilename));
Userprofiles uprofs = (Userprofiles) conf.getUserProfiles();
            List users = uprofs.getUsers();
            User user = null;
            for(int i=0; i<users.getLength(); i++) {
                user = (User) users.item(i);
                if("wmay".equals(user.getName())) break;
            }
            if(user != null) user.last_login(timer.getDateTime());

            Marshaller m = jc.createMarshaller();
            // put result into DOM tree
            DOMResult domResult = new DOMResult();
            m.marshal(s, domResult);
            Document doc = (Document) domResult.getNode();
            // transformer stuff is only for writing DOM tree to file/stdout
            TransformerFactory factory = TransformerFactory.newInstance();
            Source docSource = new DOMSource(doc);
            StreamResult result = new StreamResult(outfilename);
            Transformer transformer = factory.newTransformer();
            transformer.transform(docSource, result);
        } catch (TransformerConfigurationException e) { e.printStackTrace();
        } catch (TransformerException e) { e.printStackTrace();
        } catch (FileNotFoundException e) { e.printStackTrace();
        } catch (JAXBException e) { e.printStackTrace();
        }
        ...
```


JAXB: PRO'S AND CON'S

Plus:

- allows for easy and lightweighted unmarshaling, bean-based manipulation and marshaling of XML data
- integrates well with XML files and with DOM trees
- many (otherwise annoying) things are performed automatically:
  parsing, marshalling, providing data structures
- $\Rightarrow$ higher level of abstraction from XML representation compared to DOM and SAX

Minus:

- still wasting much time fiddling around with lists of lists of lists ... of items
  (But: seems more like a problem inherent to Java than to JAXB technology)
- ...

JAXB FACTS

- Part of the Java Web Services Developer Pack (JWSDP, current version: 1.6)
- Strongly recommended for studying: SUN's Java WebService Tutorial
  http://java.sun.com/webservices/docs/1.5/tutorial/doc/index.html
- at /afs/informatik.uni-goettingen.de/course/xml-prakt/xml/JAXB-README
  you find an explanation on how to install and use JAXB in the CIP Pool.