3. XML Processor APIs

- How can (Java) applications manipulate structured (XML) documents?
  - An overview of XML processor interfaces

3.1 SAX: an event-based interface
3.2 DOM: an object-based interface
3.3 JAXP: Java API for XML Processing

Tasks of a Parser

- Document instance decomposition
  - elements, attributes, text, processing instructions, entities, ...
- Verification
  - well-formedness checking
    » syntactical correctness of XML markup
  - validation (against a DTD or Schema)
- Access to contents of the DTD (if supported)
  - SAX 2.0 Extensions provide info of declarations:
    element type names and their content model expressions

Event Call-Back Interfaces

- Application implements a set of call-back methods for handling parse events
  - parser notifies the application by method calls
  - parameters qualify events further, with
    » element type name
    » names and values of attributes
    » values of content strings, ...
- Idea behind “SAX” (Simple API for XML)
  - an industry standard API for XML parsers
  - could think as “Serial Access XML”

Object Model Interfaces

- Application interacts with
  - a parser object, which builds ...
  - a document object consisting of document, elements, attributes, text, ...
- Abstraction level higher than in event based interfaces; more powerful access
  - to descendants, following siblings, ...
- Drawback: Higher memory consumption
  - > used mainly in client applications
    (to implement document manipulation by user)
An Object-Model Based Application

Application

Parse

Build

Parser Object

Access/Modify

In-Memory Document Representation

Document

A

=x=1

"Hi!"

SDPL 2007 3: XML Processor Interfaces 9

3.1 The SAX Event Callback API

- A de-facto industry standard
  - Developed by members of the xml-dev mailing list
  - Version 1.0 in May 1998, Vers. 2.0 in May 2000
  - Not a parser, but a common interface for different parsers (like, say, JDBC is a common interface to various DBs)
  - Supported directly by major XML parsers
    - many Java based, and free: Examples: Apache Xerces, Oracle's XML Parser for Java; MSXML (in IE 5), James Clark's XP

SAX 2.0 Interfaces

- Co-operation of an application and a parser specified in terms of interfaces (i.e., collections of methods)
- My classification of SAX interfaces:
  - Application-to-parser interfaces
    - to use the parser
  - Parser-to-application (or call-back) interfaces
    - to act on various parsing events
  - Auxiliary interfaces
    - to manipulate parser-provided information

Call-Back Interfaces

- Implemented by application to act on parse events
  (A DefaultHandler quietly ignores most of them)
  - ContentHandler
    - methods to process document parsing events
  - DTDHandler
    - methods to receive notification of unparsed external entities and notations declared in the DTD
  - ErrorHandler
    - methods for handling parsing errors and warnings
  - EntityResolver
    - methods for customised processing of external entity references

SAX 2.0: Auxiliary Interfaces

- Attributes
  - methods to access a list of attributes, e.g:
    - int getLength()
    - String getValue(String attrName)
- Locator
  - methods for locating the origin of parse events
    (e.g. systemID, line and column numbers, say, for reporting semantic errors controlled by the application)

The ContentHandler Interface

- Information of general document events. (See API documentation for a complete list):
  - setDocumentLocator(Locator locator)
    - Receive a locator for the origin of SAX document events
  - startDocument(); endDocument()
    - notify the beginning/end of a document.
  - startElement(String nsURI, String localName, String rawName, Attributes atts);
  - endElement( ... ); same params, w.o. attributes

Namespaces in SAX: Example

<xs1:stylesheet version="1.0"
xmlns:xsl=http://www.w3.org/1999/XSL/Transform
xmlns:xalan=http://www.apache.org/xml/xalan/images/"
<x:template match="/">
  <html>
    <xsl:value-of select="/total"/>
  </html>
</x:template>
</xs1:stylesheet>

- startElement for this would pass following parameters:
  - nsURI= http://www.w3.org/1999/XSL/Transform
  - localname = template, rawName = xsl:template
Namespaces: Example (2)

<xml:stylesheet version="1.0" xmlns="http://www.w3.org/TR/xhtml1/strict">
    <xml:template match="/html">
        <html> ... </html>
    </xml:template>
</xml:stylesheet>

ContentHandler interface (cont.)

- characters(char ch[], int start, int length)
  - notification of character data
- ignorableWhitespace(char ch[], int start, int length)
  - notification of ignorable whitespace in element content

```
<!DOCTYPE A [<ELEMENT B (#PCDATA)> ]>
```

SAX Processing Example (1/9)

- Input: XML representation of a personnel database:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<db>
    <person idnum="1234">
        <first>Maija</first>
        <last>Matti</last>
        <tt idnum="9012">
            Matti Mottonen (5678)
        </tt>
    </person>
    <person idnum="3456">
        <first>Maija</first>
        <last>Rompapnen</last>
        <tt idnum="9012">
            Maija Romppanen (3456)
        </tt>
    </person>
</db>
```

SAX Processing Example (2/9)

- Task: Format the document as a list like this:

```
Pekka Kilpeläinen (1234)
Matti Mottonen (5678)
Maija Mottonen (9012)
Maija Romppanen (3456)
```

- Event-based processing strategy:
  - at the start of person, record the idnum (e.g., 1234)
  - record starts and ends of last and first to store their contents (e.g., "Kilpeläinen" and "Pekka")
  - at the end of a person, output the collected data

SAX Processing Example (3/9)

- Application:
  First import relevant interfaces & classes:

```
import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.ContentHandler;
// Default (no-op) implementation of:
// interface ContentHandler:
import org.xml.sax.helpers.DefaultHandler;
// JAXP to instantiate a parser:
import javax.xml.parsers.*;
```

SAX Processing Example (4/9)

- Implement relevant call-back methods:

```
public class SAXDBApp extends DefaultHandler {
    // Flags to remember element context:
    private boolean InFirst = false;
    private boolean InLast = false;
    // Storage for element contents and
    // attribute values:
    private String FirstName, LastName, IdNum;
```

SAX Processing Example (5/9)

- Call-back methods:
  - record the start of first and last elements,
  and the idnum attribute of a person:

```
public void startElement {
    String nsURI, String localName, String rawName, Attributes attrs; 
    if (rawName.equals("person"))
        IdNum = atta.getAttribute("idnum");
```

SAX Processing Example (6/9)

- Call-back methods continue:
  - Record the text content of elements first and last:

```
public void characters {
    char buf[], int start, int length) {
```
SAX Processing Example (7/9)

- At the end of `person`, output the collected data:

```java
public void endElement(String nsURI, String localName, String qName) {
    if (qName.equals("person"))
        System.out.println(FirstName + " " + LastName + "(" + IdNum + ")" );
    //Update the context flags:
    if (qName.equals("first"))
        InFirst = false;
    // (and the same for "last" and InLast)
}
```

SDPL 2007 3: XML Processor Interfaces 25

SAX Processing Example (8/9)

- Application `main` method:

```java
public static void main (String args[]) throws Exception {
    // Instantiate an XMLReader (from JAXP
    // SaxParserFactory):
    SaxParserFactory spf = SaxParserFactory.newInstance();
    try {
        SaxParser saxParser = spf.newSAXParser();
        XmlReader xmlReader = saxParser.getXMLReader();
        // Instantiate and pass a new
        // ContentHandler to xmlReader:
        ContentHandler handler = new SAXDBApp();
        xmlReader.setContentHandler(handler);
        for (int i = 0; i < args.length; i++) {
            xmlReader.parse(args[i]);
        }
        } catch (Exception e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
```

SDPL 2007 3: XML Processor Interfaces 26

SAX Processing Example (9/9)

- Main method continues:

```java
// Instantiate and pass a new
// ContentHandler to xmlReader:
ContentHandler handler = new SAXDBApp();
xmlReader.setContentHandler(handler);
for (int i = 0; i < args.length; i++) {
    xmlReader.parse(args[i]);
}
```

SDPL 2007 3: XML Processor Interfaces 27

SAX: Summary

- A low-level parser-interface for XML documents
- Reports document parsing events through method call-backs
  - > efficient: does not create in-memory representation of the document
  - > used often on servers, and to process LARGE documents

SDPL 2007 3: XML Processor Interfaces 28