3. XML Processor APIs

- How applications can manipulate structured documents?
  - An overview of document parser interfaces

3.1 SAX: an event-based interface

3.2 DOM: an object-based interface

Document Parser Interfaces

(See, e.g., Leventhal, Lewis & Fuchs: Designing XML Internet Applications, Chapter 10, and D. Megginson: What is an Event-Based Interface)

- Every XML application contains some kind of a parser
  - editors, browsers
  - transformation/style engines, DB loaders, ...
- XML parsers are becoming standard tools of application development frameworks

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 grammar (DTD; seldom supported)

Document Parser Interfaces

I: Event-based interfaces
- Command line and ESIS interfaces
  - traditional interface to stand-alone SGML parsers
- Event call-back interfaces: SAX

II: Tree-based (object model) interfaces
- W3C DOM Recommendation

Command Line and ESIS Interfaces

- Simplest of parser interfaces; immediately usable for simple applications
- ESIS: Element Structure Information Set
  - canonical output format for SGML parsers
  - stream of parsing events (start element, end element, attribute, ...) in "document order"
  - can be used as input to other programs
- Output format of, e.g., sgmls and nsgmls (SGML) parsers

ESIS Example: Input document

```xml
<!DOCTYPE article SYSTEM 'article.dtd'>
<article>
  <para>Written by the lecturer. (See, <fig-ref refid="fig1"]/.)</para>
  <fig id="fig1" file="pekka.jpg" caption="The Lecturer"></fig>
</article>
```
ESIS example: ASCII output of nsgmls

```xml
<ARTICLE>
  <PARA>
    Written by the lecturer. In this.
  </PARA>
  <FIG-REF>
    FIG-REF
  </FIG-REF>
  <FIG-TOKEN>
  </FIG-TOKEN>
  <FILE CDATA>pekka.jpg</FILE>
  <ACAPTION CDATA>The Lecturer</ACAPTION>
</ARTICLE>
```

Command-line parser interface

```
Command line call

ESIS Stream

SGML/XML Parser

<A i="1">Hi!</A>
```

Event Call-Back Interfaces

- Application implements a set of callback methods for handling parse events
  - parser notifies the application by method calls
    - parameters qualify events further (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

```
Application Main Routine

startDocument()
startElement()
characters()
endElement()
```

An event callback application

```
<?xml version='1.0'?>
<A i="1">Hi!</A>
```

Object Model Interfaces

- Application interacts with an object-oriented representation of
  - the parser
    - the document parse tree consisting of objects like document, element, attribute, text, ...
- Abstraction level higher than in event based interfaces; more powerful access (e.g. to descendants, following siblings)
- Drawback: Higher memory consumption

```
Application
Parser
Object
Build
```

An Object-Model Based Application

```
<A i="1">Hi!</A>
```
3.1 The SAX Event Callback API

- A de-facto industry standard
  - Not an official standard or W3C Recommendation
  - 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 many different parsers (like JDBC is a common interface to various RDBs)

- Supported directly by major XML parsers
  - most Java based and free: Sun JAXP, IBM XML4J, Oracle’s XML Parser for Java, Apache Xerces; MSXML (in IE 5), James Clark’s XP

SAX 2.0 Interfaces

- Interplay between an application and a SAX-conformant parser specified in terms of interfaces (i.e., collections of abstract methods)

- Classification of SAX interfaces:
  - Parser-to-application (or call-back) interfaces
    - to attach special behaviour to parser-generated events
  - Application-to-parser
    - to use the parser
    - Auxiliary
    - to manipulate parser-provided information

Call-Back Interfaces

- Implemented by application to override default behaviour (of ignoring any event quietly)
  - ContentHandler
    - methods to process document parsing events
  - DTDHandler
    - methods to receive notification of unparsed external entities and their notations declared in the DTD
  - ErrorHandler
    - methods for handling parsing errors and warnings
  - EntityResolver
    - methods for customised processing of external entity references

Application-to-Parser Interfaces

- Implemented by parser (or its SAX driver):
  - XMLReader
    - methods to allow the application to invoke the parser and to register objects that implement call-back interfaces
  - XMLFilter
    - interface to connect several XMLReaders in a row as a sequence of filters

SAX 2.0: Auxiliary Interfaces

- Attributes
  - methods to access a list of attributes

- 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

- Methods for receiving information of general document events. (See API documentation for a complete list):
  - setDocumentLocator(Locator locator)
    - Receive an object for locating the origin of SAX document events (e.g. for reporting semantic errors controlled by the application)
  - startDocument(); endDocument()
    - notification of the beginning and of a document
  - startElement(String namespaceURI, String localName, String qName, Attributes_atts);
    - parameters support namespaces
  - endElement( ... ) similar (without attributes)
    - parameters support namespaces
Digression: XML Namespaces

- W3C Recommendation for separating possibly overlapping “vocabularies” (sets of element type and attribute names) within a single document
  - e.g. XSLT stylesheets contain mixtures of XSLT command names and result document element names
  - separation by using a local name prefix, which is bound to a globally unique URI
  - built on top of XML 1.0
    - use attribute syntax to introduce namespaces
    - does not affect validation (i.e., namespace attributes have to be declared for DTD-validity)

XML namespaces briefly (2)

- Namespace identified by a URI associated with the local prefix
  - e.g. http://www.w3.org/1999/XSL/Tranform for XSLT
  - conventional but not required to use URLs
    - the identifying URI has to be unique, but it does not have to be an existing address
- Association inherited to sub-elements
  - see the next example

XML namespaces: Example

```xml
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/xhtml1/strict">
    <xsl:template match="/">
        <html>
            <xsl:value-of select="//total"/>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Namespaces: Example (2)

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

- A SAX invocation of startElement for xsl:template would pass following parameters:
  - namespaceURI = http://www.w3.org/1999/XSL/Transform
  - localname = template, qName = xsl:template

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.

SAX Processing Example (1)

- Input: XML representation of a personnel database:

```
<db>
  <person idnum="1234">
    <last>Kilpeläinen</last><first>Pekka</first>
  </person>
  <person idnum="5678">
    <last>Möttönen</last><first>Matti</first>
  </person>
  <person idnum="9012">
    <last>Möttönen</last><first>Maija</first>
  </person>
  <person idnum="3456">
    <last>Römppänen</last><first>Maija</first>
  </person>
</db>
```
Task: Format the document as a list like this:

- Pekka Kilpeläinen (1234)
- Matti Möttönen (5678)
- Maija Möttönen (9012)
- Maija Romppänen (3456)

Solution strategy using event-based processing:
- At the start of a person, record the idnum (e.g., 1234)
- Keep track of starts and ends of elements last and first, in order to record content of those elements (e.g., “Kilpeläinen” and “Pekka”)
- At the end of a person, output the collected data

Application: Begin by importing relevant packages:

```java
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;
// SUN JAXP used as the SAX implementation:
import javax.xml.parsers.*;
```

Define a class to implement relevant call-back methods:

```java
public class SAXDBApp extends DefaultHandler {

    // Flags to remember element context:
    private boolean InFirst = false,
    InLast = false;

    // Storage for element contents and
    // attribute values:
    private String FirstName, LastName, IdNum;
```

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

```java
public void startElement (String namespaceURI, String localName,
String rawName, Attributes atts) {

    if (localName.equalsIgnoreCase("first"))
        InFirst = true;
    if (localName.equalsIgnoreCase("last"))
        InLast = true;
    if (localName.equalsIgnoreCase("person"))
        IdNum = atts.getValue("idnum");
}
```

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

```java
public void characters (char ch[], int start, int length) {

    if (InFirst) FirstName =
        new String(ch, start, length);
    if (InLast) LastName =
        new String(ch, start, length);
}
```

Call-back methods continue:
- At an exit from person, output the collected data:

```java
public void endElement(String namespaceURI,
String localName, String qName) {

    if (localName.equalsIgnoreCase("person"))
        System.out.println(FirstName + " "+
        LastName + " (" + IdNum + ")");

    // Update the context flags:
    if (localName.equalsIgnoreCase("first"))
        InFirst = false;
    if (localName.equalsIgnoreCase("last"))
        InLast = false;
    if (localName.equalsIgnoreCase("person"))
        IdNum = atts.getValue("idnum");
}
```
SAX Processing Example (8)

```java
// Application main method:
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();  
            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);  
        } // main
```

SAX Processing Example (9)

```java
// Main method continues:
  // 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);  
  } // main
```