XML for Java Developers
G22.3033-002

Session 5 - Main Theme
XML Information Processing (Part I)

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences

Agenda

- Summary of Previous Session
- Introduction to XML Processing
- SAX Processing
- XML SAX Parsers
- SAX and the JAXP APIs
- XML application Development using the XML Java APIs
- Java-based XML application support frameworks
Summary of Previous Session

- Advanced Logical Structuring and XML Schemas
- XML-Based Software Development
- Business Engineering Methodology
- XML Metadata Management
- XML Linking/Pointer Language, XML Base, XML Inclusions
- XML Data Binding
- Industry Specific Markup Languages
- Parsing / Generating / Serializing XML Documents
- XML Metadata Management Tools
- Assignment 2b (due next week)

XML-Based Software Development

- Business Engineering Methodology
  - Language + Process + Tools
  - e.g., Rational Unified Process (RUP)
- XML Application Development Infrastructure
  - Metadata Management (e.g., XMI)
  - XML APIs (e.g., JAXP, JAXB)
  - XML Tools (e.g., XML Editors, XML Parsers)
- XML Applications:
  - Application(s) of XML
  - XML-based applications/services (markup language mediators)
    - MOM & POP
    - Other Services (e.g., persistence, transaction, etc.)
  - Application Infrastructure Frameworks
More on XML Information Modeling

- Using UML use cases to support the development of DTDs and XML Schemas
- Establish linking relationship
  - See Family tree application of XML (under demos)

Part I

Introduction to XML Processing
Common XML APIs

- Document Object Model (DOM) API
  - Tree structure based API
  - Issued as a W3C recommendation (10/98)
  - See Session 6 Sub Topic 1 Presentation (next week)
- Simple API for XML (SAX)
  - Event driven API
  - Developed by David Megginson
- ElementHandler API
  - Event driven proprietary API provided by IBM’s XML4J
- Pull Parsing (http://www.xmlpull.org/) – J2ME applications
  - incremental (streaming) parsing where application is in control
  - parsing can be interrupted at any given moment and resumed when application is ready to consume more input.
- Pure Java APIs: JDOM (Open Source) and JAXP

XML APIs Characteristics

- DOM API: (See http://www.developerlife.com/domintro/default.htm)
  - In DOM, an XML document is represented as a tree, which becomes accessible via the API
  - The XML processor generates the whole tree in memory and hands it to an application program
- SAX API: (See http://java.sun.com/xml/docs/tutorial/sax/index.html)
  - Does not generate a data structure
  - Scans an XML document and generate events as elements are processed
  - Events can be trapped by an application program via the API
- ElementHandler:
  - Event-driven like SAX, but also creates a DOM tree
- Open Source Pure Java API (JDOM)
Simple API for XML (SAX)

- The SAX specification is an event-based interface developed by members of the XML-DEV mailing list hosted by OASIS
- SAX allows an application to interact with XML data as a series of events via a set of APIs
- SAX is best for applications that need to access a specific piece of data on a one-time basis without its relationships to surrounding elements
- SAX is faster when you do not need to access all the data in an XML document
  - Document is viewed as a data stream instead of an in-memory data structure
  - Allows to access a small number of elements at one time rather than an entire document
  - Applications have better control over parsing of specific information needed

Document Object Model (DOM)

- The DOM specification is an object-based interface developed by W3C that builds an XML document as a tree structure on memory
- An application interacts with XML data via a set of DOM APIs through an in-memory tree, which replicates the way the data is structured
- DOM allows you to dynamically traverse and update an XML document, and is ideal to manage XML data or access a complete data structure repeatedly
- DOM does the parsing up front and preserves the structure of the document
  - XML document is parsed at one time and represented as a tree structure in memory
  - Applications may make dynamic updates to the tree structure in memory
DOM vs. SAX

- Object-based interface vs. Event-based interface
- Object model created automatically vs. created by application
- Order/sequencing of the elements preserved vs. ignored in favor of single events
- Higher use of memory vs. lower use of memory
- Slower speed of initial data retrieval vs. faster speed of initial data retrieval
- Better for complex structures vs. better for simple structures
- Both support optional validation via an API in the DOMParser/SAXParser classes
- DOM has the ability to update XML documents

Part II

SAX Processing
SAX Standards
http://sax.sourceforge.net/

- SAX 2.0 Core
  - http://prdownloads.sourceforge.net/sax/sax2_r2pre2.jar
  - Includes org.xml.sax, org.xml.sax.helpers
- SAX 2.0 Extension (http://www.saxproject.org/?selected=ext)
  - http://prdownloads.sourceforge.net/sax/sax2_ext_1.0.zip
  - Includes standardized extensions
  - Anyone can define/implement other extensions using core “feature flags” and “property objects” mechanisms
- JAXP 1.2
  - Includes APIs for processing XML documents using SAX, DOM, and XSLT
  - XML Schema and XSLT compiler (XSLTC) support are new features in 1.2

Java-enabled XML Technologies

- XML provides a universal syntax for Java semantics (behavior)
  - Portable, reusable data descriptions in XML
  - Portable Java code that makes the data behave in various ways
- XML standard extension
  - Basic plumbing that translates XML into Java
    - parser, namespace support in the parser, simple API for XML (SAX), and document object model (DOM)
- XML data binding standard extension
How SAX Processing Works

- SAX analyzes an XML stream as it goes by
- Example:

```xml
<?xml version="1.0"?>
<samples>
  <server>UNIX</server>
  <monitor>color</monitor>
</samples>
```
- Events generated by SAX processor:
  - Start document
  - Start element (samples)
  - Characters (white space)
  - Start element (server)
  - Characters (UNIX)
  - End element (server)
  - Characters (white space)
  - Start element (monitor)
  - Characters (color)
  - End element (monitor)
  - Characters (white space)
  - End element (samples)

How SAX Processing Works (cont.)
SAX Processing Steps

- Create an event handler
  - Instantiate a class that implements the org.xml.sax.DocumentHandler interface
- Create the SAX parser
  - Instantiate a class that implements the org.xml.sax.Parser interface
- Assign the event handler to the parser
  - Call the parser's setDocumentHandler() method
- Parse the document, sending each event to the handler
  - Call the parse() method of the parser
- Developer can then capture the events and work on them

Advantages:
- Analysis can be started immediately rather than having to wait for all of the data to be processed
- Data does not need to be stored in memory (useful when documents are large)
- Faster processing

Disadvantages:
- Cannot make changes
- Cannot move “backward” in the data stream

Create SAX Handlers Using IDEs

- Most Java IDEs provides a SAX Handler wizard
- Typical development steps:
  - Create SAX parser
  - Edit SAX parser code to customize parsing
  - Run program to view the parsing results
  - Add attributes to the XML document and code to handle the attributes
  - Parse the document again
  - etc.
Filter Design Pattern for SAX

- e.g., take a stream of SAX events and indent tags for presentation purpose then pass massaged data to DocumentHandler, etc.
- Filter implements both the SAXParser and DocumentHandler interfaces

Filter Design Pattern for SAX (continued)

- Applications
  - Remove unwanted elements
  - Modify tags or attribute names
  - Perform validation
  - etc.
- Sample implementation: ParserFilter class
- Sample ParserFilter pipeline:
  
  ```java
  ParserFilter pipeline = new Filter3(
      new Filter2 (
        new Filter1 (
          new com.jclark.xml.sax.Driver())));
  pipeline.setDocumentHandler(outputHandler);
  ```
- Other examples: NamespaceFilter, InheritanceFilter, XLinkFilter, etc.
Rule-Based Design Pattern for SAX

Sample Switcher Implementation:

```java
import org.xml.sax.*;
import com.icl.saxon.ParserManager;
public class DisplayBookList {
    public static void main (String args[]) throws Exception {
        (new DisplayBookList()).go(args[0]);
    }
    public void go(String input) throws Exception {
        Switcher s = new Switcher();
        s.setElementHandler("books", new BooklistHandler());
        s.setElementHandler("book", new BookHandler());
        ...
        Parser p = ParserManager.makeParser();
        p.setDocumentHandler(s);
        p.parse(input);
    }
}
```

//...rest of code goes in here...

(continued)
SAX2 Configurable Interface

- SAX2 parser implements
  - org.xml.sax.Configurableinterface
  - org.xml.sax.Parser interfaces
- Org.xml.sax.Configurableinterface
  - getFeature(featureName)
    - Ask parser whether it supports a particular feature
  - setFeature(featureName, boolean)
    - Allow application to request feature enabling/disabling
    - E.g., parser.setFeature("http://xml.org/sax/features/validation", true);
  - getProperty(featureName)
    - Allow application to request current value of some property
  - setProperty(featureName, object)
    - Allow application to set some property on the supplied value
    - SAXNotRecognizedException is thrown if feature or property name is not recognized
    - SAXNotSupportedException is thrown if feature cannot be set

Sample Applications

- XML and Java textbook samples:
  - http://pws.prserv.net/Hiroshi.Maruyama/xmlbook/samples4v2/
- “Having Good SAX with Java”:
- SAX implementations list:
  - http://www.xmlsoftware.com
  - David Megginson's original site (http://www.megginson.com/SAX/)
- Xerces2 JSamples:
  - http://xml.apache.org/xerces2-i/samples-sax.html
- JAXP Samples:
- Notes:
  - In version 2 SAX Parser:
    - com.ibm.xml.parser.SAXDriver
    - is replaced by:
    - com.ibm.xml.parsers.SAXParser
SAX, DOM, XSLT Processing Limitations

- Uniform Solution for XML transformations?
  - Express output declaratively
    - Similar to XSLT
  - Lets you include arbitrary filters and computations
    - Similar to the implementation languages underlying DOM and SAX
  - Guarantees well formedness or validity of the output
  - Compact and direct syntax
- See HaXML
  - Functional programming model for XML

Part III

XML SAX Parsers
XML Processors Characteristics

- An XML engine is a general purpose XML data processor
- An XML processor/parser is a software engine that checks the syntax (well-formedness) of XML documents
- If a schema (or DTD) is included, the parser can (optionally) validate the correctness of XML documents’ structure against it
- A parser reads the XML document’s information and makes it accessible to the XML application via a standard API

Sample XML parsers and engines

- XML parsers
  - RXP, Dan Connolly’s XML parser, XML Tolikit, LTXML, expat, TCLXML, xparse, XP, DataChannel XPLparser (DXP), XML:Parse, PyXMLTok, Lark, Microsoft’s XML parser, IBM’s XML for Java, Apache’s Xerces, Aefred, xmlproc, xmllib, Windows foundation classes, Java Project X Parser (Crimson), OpenXML Parser, Oracle XML Parser, etc.
- SGML/XML parsers
  - SGMLSpm, SP
Comprehensive List of XML Processors

- A comprehensive list of parsers is available at http://www.xmlsoftware.com/parsers
  - Includes links to latest product pages
  - Includes Version numbers, Licensing information, and Platform details
- Research work being done around MetaParsers and parallel XML parsers

Mainstream Java-Based XML Processors

- Sun’s Java Project X Parser
  - Donated on April 13, 2000 to the Apache’s XML Project under the name “Crimson”
- Apache’s Xerces2 J
  - Xerces2 J is strongly recommended for this course
  - Xerces2Parser is a standards’ compliant reference implementation of the Xerces Native Interface
  - XNI is a framework for communicating a “streaming” document information set and constructing generic parser configurations
- Oracle’s XML Parser for Java
- Expat
Xerces2Parser Components

Other Java-Based XML Processors

- Sun’s JAXP
- Jason Hunter and Brett McLaughlin’s OpenSource JDOM
- IBM Alphaworks’s XML for Java (XML4J)
  - Based on the Apache Xerces XML Parser
- DataChannel’s XJParser
Part IV

SAX and the JAXP APIs

Simple API for XML (SAX) Parsing APIs
SAX API Packages

- **org.xml.sax**
  - Defines the SAX interfaces.

- **org.xml.sax.ext**
  - Defines SAX extensions that are used when doing more sophisticated SAX processing, for example, to process a document type definitions (DTD) or to see the detailed syntax for a file.

- **org.xml.sax.helpers**
  - Contains helper classes that make it easier to use SAX -- for example, by defining a default handler that has null-methods for all of the interfaces, so you only need to override the ones you actually want to implement.

- **javax.xml.parsers**
  - Defines the SAXParserFactory class which returns the SAXParser. Also defines exception classes for reporting errors.

Java API Packages

- **java.xml.parsers**
  - The JAXP APIs, which provide a common interface for different vendors' SAX and DOM parsers.
  - Two vendor-neutral factory classes: SAXParserFactory and DocumentBuilderFactory that give you a SAXParser and a DocumentBuilder, respectively. The DocumentBuilder, in turn, creates DOM-compliant Document object.

- **org.w3c.dom**
  - Defines the Document class (a DOM), as well as classes for all of the components of a DOM.

- **org.xml.sax**
  - Defines the basic SAX APIs.

- **jaxax.xml.transform**
  - Defines the XSLT APIs that let you transform XML into other forms.
DOM Parsing APIs

- **org.w3c.dom**
  - Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.

- **javax.xml.parsers**
  - Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface. The factory that is used to create the builder is determined by the javax.xml.parsers system property, which can be set from the command line or overridden when invoking the newInstance method. This package also defines the ParserConfigurationException class for reporting errors.
XSLT APIs

- **Transformer Factory**
- **Source**
- **Transformer**
- **Result**
- **Transformation Instructions**

XSLT API Packages

- See Session 3 handout on “Processing XML Documents in Java Using XPath and XSLT”
- **javax.xml.transform**
  - Defines the TransformerFactory and Transformer classes, which you use to get a object capable of doing transformations. After creating a transformer object, you invoke its transform() method, providing it with an input (source) and output (result).
- **javax.xml.transform.dom**
  - Classes to create input (source) and output (result) objects from a DOM.
- **javax.xml.transform.sax**
  - Classes to create input (source) from a SAX parser and output (result) objects from a SAX event handler.
- **javax.xml.transform.stream**
  - Classes to create input (source) and output (result) objects from an I/O stream.
**JAXP and Associated XML APIs**

- **JAXP: Java API for XML Parsing**
  - Common interface to SAX, DOM, and XSLT APIs in Java, regardless of which vendor's implementation is actually being used.

- **JAXB: Java Architecture for XML Binding**
  - Mechanism for writing out Java objects as XML (*marshalling*) and for creating Java objects from such structures (*unmarshalling*).

- **JDOM: Java DOM**
  - Provides an object tree which is easier to use than a DOM tree, and it can be created from an XML structure without a compilation step.

- **JAXM: Java API for XML Messaging**
  - Mechanism for exchanging XML messages between applications.

- **JAXR: Java API for XML Registries**
  - Mechanism for publishing available services in an external registry, and for consulting the registry to find those services.

---

**Content of Jar Files**

- **jaxp.jar (interfaces)**
  - `javax.xml.parsers`
  - `javax.xml.transform`
    - `javax.xml.transform.dom`
    - `javax.xml.transform.sax`
    - `javax.xml.transform.stream`

- **crimson.jar (interfaces and helper classes)**
  - `org.xml.sax`
    - `org.xml.sax.helpers`
    - `org.xml.sax.ext`
    - `org.w3c.dom`

- **xalan.jar (contains all of the above implementation classes)**
Related Java Bindings

- Sun’s Java API for XML Parsing (JAXP)
  - Provides a standard way to seamlessly integrate any XML-compliant parser with a Java application
  - Developers can swap between XML parsers without changing the application
  - The reference implementation uses Sun’s Java Project X as its default XML parser
- DOM 2.0 and DOM 1.0 Java binding specification (http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/java_binding.zip)

Parser Independence

- SAX parser may be provided as a command line option
  - Could use Xerces SAXParser as default parser
  - Parser must implement either of the following interfaces:
    - org.xml.sax.Parser
    - org.xml.sax.XMLReader
- JAXP can be used instead
  - Need to specify JVM option to specify SAX parser factory
    - -Djavax.xml.parsers.SAXParserFactory=(…)
XML Data Binding Standard Extension

- Aims to automatically generate substantial portions of the Java platform code that processes XML data
- A Sun project, codenamed “Adelard”
- See JSR 31 XML Data Binding Specification
- See [http://java.sun.com/xml/jaxp-1.0.1/docs/binding/DataBinding.html](http://java.sun.com/xml/jaxp-1.0.1/docs/binding/DataBinding.html)

Part V

XML Application Development
Using the XML Java APIs
Typical XML Processor Installation

- Pick a processor based on the features it provides to match your requirements
- Download and install the latest (or supported) version of the JDK from http://www.javasoft.com
- Install the XML processor
- Update the PATH and CLASSPATH variables as needed, and test the processor

Reading/Parsing XML Documents

- Use Apache’s XercesJ or Alphaworks’ XML Parser for Java
  - The applications provided in section 2.4 of “XML and Java” may need to be adapted to support the latest version of the parsers
  - We suggest looking at the source for the sample applications located on the CD/Web
  - For initial testing, use XML and Java’s sample documents provided or the “personal.xml” sample XML document provided with XML4J’s sample application
Generating XML Documents

- Hand-coded serialization to file output stream
- SAX + Xerces serialization to file output stream
- JAXP + SAX serialization to servlet output stream
  - [http://www.javazoom.net/services/newsletter/xmlgeneration.html](http://www.javazoom.net/services/newsletter/xmlgeneration.html)

Presenting XML Documents
Using Java Tools

- Presenting an XML document requires processing of the XML document by accessing its internal structure
- An XML document’s structure can be accessed using the various XML APIs
- Various third party tools have been implemented using such APIs to apply XSL style sheets to XML documents and generate HTML output (e.g., Xalan, LotusXSL)
- TraX API is now included in the JAXP APIs
XML Data Exchange Protocols

- Message format alternatives
  - Text-based (e.g., EDI, RFC822, SGML, XML)
  - Binary (e.g., ASN.1, CORBA/IIOP)
- An API that provide a common interface to work with EDI or XML/EDI objects is supported by OpenBusinessObjects
- Guidelines for using XML for EDI are provided at http://www.geocities.com/WallStreet/Floor/5815/guide.htm and http://www.xmledi-group.org/

XML Fragment Interchange

- Defines a way to send fragments of an XML document without having to send all of the containing document up to the fragment
- Fragments are not limited to predetermined entities
- The approach captures the context that the fragment had in the larger document to make it available to the recipient
- See http://www.w3.org/TR/WD-xm1_fragment
XML Data Processing Examples

- Sections 2, 3, and 5 of “XML and Java” cover various examples of XML document processing using the DOM, and SAX APIs.
- Sessions 6-8 of “Processing XML with Java” cover additional examples of XML document processing using the SAX API.

Part VI

Java-Based Application Support Frameworks
XML MOM and POP Frameworks

- An XML support framework must include:
  - XML Parser (conformity checker)
  - XML applications that use the output of the Parser to achieve unique objectives
- See subsection 2.3.2 of the weekly notes on “XML MOM Application Server Frameworks” for a complete description of a general purpose XML MOM framework.

Java and the Apache XML Project

- See related article at:
- List of XML Sub-Projects:
  - Xerces: XML parsers in Java, C++ (with Perl and COM bindings)
  - Xang: Rapid development of dynamic server pages, in JavaScript
  - Xalan: XSLT stylesheet processors, in Java and C++
  - SOAP: Simple Object Access Protocol
  - FOP: XSL formatting objects, in Java
  - Crimson: Java XML parser derived from the Sun Project X Parser
  - Cocoon: XML-based Web publishing, in Java
  - Batik: Java-based toolkit for Scalable Vector Graphics (SVG)
  - AxKit: XML-based Web publishing, in mod_perl
POP Applications Support Frameworks

- Objective is to “serve” XML
- HTML generation applications are provided
- Sample solutions
  - XML::Parser module with Perl
  - XML processing via Java servlets
    - e.g., IBM Alphaworks’ XMLEnabler
      - See session 2’s sub-topic 2.3.2 on “XML POP Application Server Framework”
  - Apache’s Cocoon
  - Active Server Pages (ASP) with MSXML (see “Serving XML with ASP”, and rocket)

MOM Applications Support Frameworks

- Many applications can be envisioned
  - One objective is to support application integration via XML data interchange
- Sample solutions:
  - XML::Parser module with Perl
  - XML processing via Java applications
Part VII

Conclusions

Summary

- SAX is an event-driven API for processing XML documents
- Various parser implementations are available for SAX
- Java developers should interface parsers via JAXP to ensure portability of their applications
- Mainstream MOM and POP application development tools are being supported by IBM, Sun, Oracle, and Microsoft
- Java MOM and POP applications are developed using Java bindings to the DOM, and SAX APIs
- XML provides a standard data interchange message format
Summary (continued)

- The W3C XML Fragments specification focuses on the handling of XML document fragments
- MOM and POP (Java based) application support frameworks are still emerging and are becoming common facilities in the ubiquitous Web Services Infrastructure

Something to Think About

- Business Processes are being standardized and represented using XML Markup Languages
- Both the implementations of these business processes and the associated markup languages can be manipulated as we used to manipulate data in ODSs and Data Warehouses
- Traditional Data Warehousing technology is becoming applicable to Business Process Management
  - ETL
  - Data Mining
  - etc.
More on Industry-Specific Markup Languages

(see http://www.oasis-open.org/cover/xml.html#contentsApps)

- Extensible Business Reporting Language (XBRL)
- Bank Internet Payment System (BIPS)
- Electronic Business XML (EbXML)
- Privacy-enabled Customer Data Interchange (CPExchange)
- Visa XML Invoice Specification
- Legal XML
- NewsML
- Electronic Catalog XML (eCX)
- Open eBook Publication Structure

Sample XML-Based Architecture
Readings

- Readings
  - XML and Java: Chapter 5, Appendices A and B
  - Processing XML with Java: Chapters 6-8, Appendix C
  - Developing Java Web Services: Chapters 7-8
  - Handouts posted on the course web site

- Project Frameworks Setup (ongoing)
  - Apache’s Web Server, TomCat/JRun, and Cocoon
  - Apache’s Xerces, Xalan, Saxon
  - Antenna House XML Formatter, Apache’s FOP, X-smiles
  - Visibroker 4.5 (or BES 5.2), WebLogic 6.4 81, WAS 5.0
  - POSE & KVM (See Session 3 handout)

Assignment

- Assignment #3:
  - This part of the project focuses on the application process model design/development using XML information processing technology. The design/development process should adhere to the following steps: (a) Identifying the points of data integration, (b) Defining the optimal integration approach at each point, (c) Establishing linking relationships, and (d) Considering data integration and linking issues when designing an overall application data model
  - More specific project related information, and extra credit assignments will be provided during the session
Next Session:
XML Information Processing (Part II)

- Document Object Model (DOM)
  - DOM API
  - Creating a Document Using DOM
- JDOM
  - Java-centric Document API for XML
- Advanced XML Parser Technology
  - JAXP: Java API for XML Processing (continued)
  - DOM, SAX, JDOM, and JAXP comparison
- Latest W3C APIs and Standards for Processing XML
  - XML Infoset, DOM Level 3, Canonical XML
  - XML Signatures, XBase, XInclude, XPointers
  - XML Fragments, XML Schema Adjuncts