Java XML Processing API (JAXP) I: XML, DOM, SAX

Ulrich.Schaefer@dfki.de

JAXP: Java XML Processing

- XML – A bit of history
- XML introduction
- Semi-structured markup vs. structured data
- XML and NLP
- DTD and XML Schema
- SAX parser
- DOM
- StAX parser (XPP), JAXB (Java-XML binding)

XML – A bit of history

- 1969: GML=Generalized Markup Language pioneer: IBM (Goldfarb, Mosher, Lorie)
- 1986: SGML=Standardized GML; ISO 8879
- 1992: HTML (SGML DTD)
- 1994: W3C founded (industry consortium)
- motivation for XML:
  - clean WWW – HTML mixes content and layout
  - SGML too complicated

SGML & XML: Markup can be hierarchical

- Markup elements can be embedded and form a tree (e.g. book – chapters – sections – paragraphs – sentences – words)
- Convenient to describe structure of documents
- but also convenient for storing data (cf. the `flat' structure of relational databases)

Structured Markup


```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML
V4.2//EN"
"http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd">
<book lang="en">
  <bookinfo>
    <title>The book title</title>
  </bookinfo>
  <chapter>
    <title>The first chapter</title>
    <para>This is <emphasis>running</emphasis> text.</para>
  </chapter>
</book>
```

- 1996: XML W3C working draft (SGML DTD)
- 1998: 1.0 recommendation:
  http://www.w3.org/TR/xml
- based on other standards like
  - Unicode (ISO 10646)
  - URI syntax (IETF RFC 1738)
- machine-readable and human-readable

XML document example


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE nlp SYSTEM "nlp.dtd">
<!-- A sample NLP annotation (tokenization, PoS tagging, chunking) -->
<nlp>
 <sentence id="S0">
   <w id="W0" length="3" offset="0" ttype="ccase" pos="PPER">Wir</w>
   <w id="W1" length="6" offset="4" ttype="lcase" pos="VVFIN">fuhren</w>
   <chunk id="C0" cat="PP" length="11" offset="11">
     <w id="W2" length="4" offset="11" ttype="lcase" pos="APPR">nach</w>
     <w id="W3" length="6" offset="16" ttype="ccase" pos="NE">Berlin</w>
   </chunk>
   <w id="W4" length="1" offset="22" ttype="period">.</w>
 </sentence>
</nlp>
```

XML Ingredients: 'markup'


```
<nlp:w id="W0" length="5" offset="0" ttype="&ccase;">Peter</nlp:w>
<!-- comment -->
```

- **Text** ... the marked up text
- Elements ... structure the document
- Attributes ... describe elements
- Comments ... as in programming languages
- Entities ... are abbreviations
- Namespaces ... help to name and combine
- DTD/Schema ... describes a valid document

Semi-structured markup vs. structured data

- Semi-structured markup: enrich text with additional information
  - cf. SGML history: documentation (legal, military, ...)
  - validation: make documents searchable, comparable
  - optional elements: admit variation within boundaries
- Structured data: database-like formats
  - XML as universal data exchange syntax (Unicode!)
  - validation: ensure correctness of format
  - optional elements: XML as lingua franca!

The role of XML for NLP

1. NLP as text markup process ("semi-structured")
   - hand- or automatically annotated Corpora
   - document structure (e.g., Docbook, OASIS)
   - multimedia annotation (MPEG-7), VoiceXML
2. NLP also uses data ("structured")
   - ontology and semantics representation (semantic web)
   - morphology
   - typed feature structures, parse trees
3. Application data interfaces, configuration

Input sentence: **"Wir fuhren nach Berlin"**

1. Tokenization:

```
<sentence id="S0">
  <w id="W0" length="3" offset="0" ttype="ccase">Wir</w>
  <w id="W1" length="6" offset="4" ttype="lcase">fuhren</w>
  <w id="W2" length="4" offset="11" ttype="lcase">nach</w>
  <w id="W3" length="6" offset="16" ttype="ccase">Berlin</w>
  <w id="W4" length="1" offset="22" ttype="punct">.</w>
</sentence>
```

2. Part-of-speech tagging:

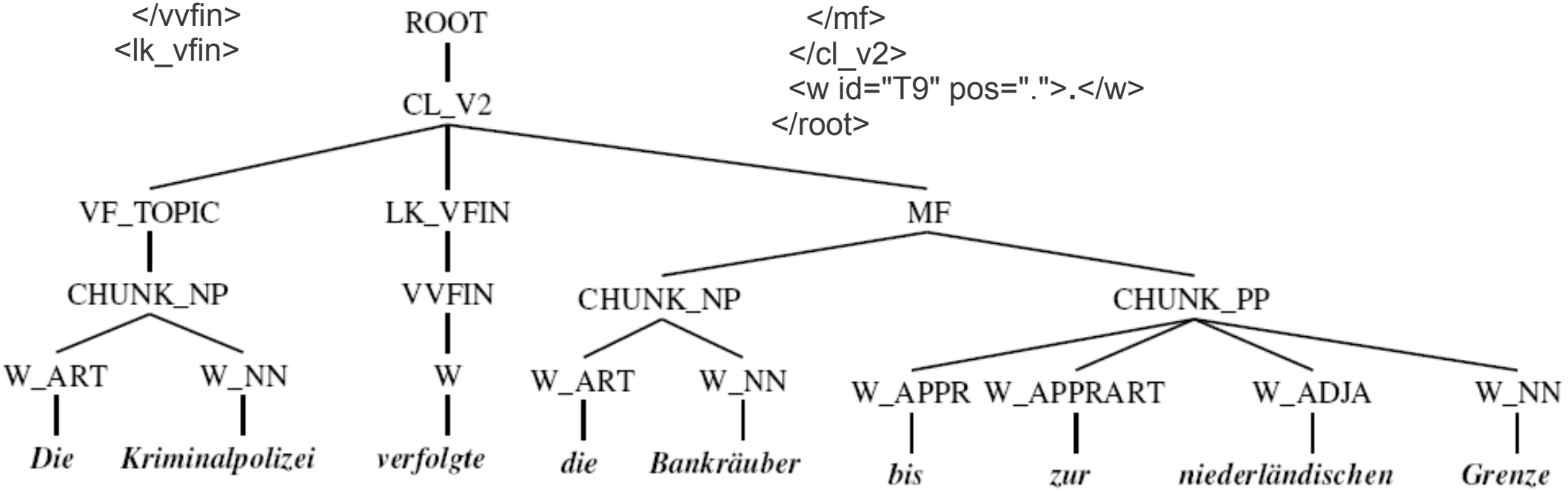
```
<sentence id="S0">
  <w id="W0" length="3" offset="0" ttype="ccase" pos="PPER">Wir</w>
  <w id="W1" length="6" offset="4" ttype="lcase" pos="VVFIN">fuhren</w>
  <w id="W2" length="4" offset="11" ttype="lcase" pos="APPR">nach</w>
  <w id="W3" length="6" offset="16" ttype="ccase" pos="NE">Berlin</w>
  <w id="W4" length="1" offset="22" ttype="punct">.</w>
</sentence>
```

3. Chunking:

```
<sentence id="S0">
  <w id="W0" length="3" offset="0" ttype="ccase" pos="PPER">Wir</w>
  <w id="W1" length="6" offset="4" ttype="lcase" pos="VVFIN">fuhren</w>
  <chunk id="C0" cat="PP" length="11" offset="11">
    <w id="W2" length="4" offset="11" ttype="lcase" pos="APPR">nach</w>
    <w id="W3" length="6" offset="16" ttype="ccase" pos="NE">Berlin</w>
  </chunk>
  <w id="W4" length="1" offset="22" ttype="period">.</w>
</sentence>
```

Encoding tree structure (PCFG)


```
<root>
 <cl_v2>
  <vf_topic>
   <chunk cat="NP">
    <w id="T0" pos="ART">Die</w>
    <w id="T1" pos="NN">Kriminalpolizei</w>
   </chunk>
  </vf_topic>
  <lk_vfin>
   <vvfin>
    <w id="T2" pos="VVFIN">verfolgte</w>
   </vvfin>
  <lk_vfin>
  <mf>
   <chunk cat="NP">
    <w id="T3" pos="ART">die</w>
    <w id="T4" pos="NN">Bankräuber</w>
   </chunk>
   <chunk cat="PP">
    <w id="T5" pos="APPR">bis</w>
    <w id="T6" pos="APPRART">zur</w>
    <w id="T7" pos="ADJA">niederländischen</w>
    <w id="T8" pos="NN">Grenze</w>
   </chunk>
  </mf>
 </cl_v2>
 <w id="T9" pos=".">.</w>
</root>
```



XML for structured data

- database-like, e.g. address book, lexicon, etc.
- cf. relational databases
- EDI (electronic data interchange)
- knowledge representation: RDF, DAML, OWL
- ...

OWL example


```
<rdf:Description rdf:about="http://www.lt-world.org/ltw.owl#obj_65046">
    <rdf:type rdf:resource="http://www.lt-world.org/ltw.owl#Active_Person"/>
    <ltw:subaffiliatedWith rdf:resource="http://www.lt-world.org/ltw.owl#obj_68479"/>
    <ltw:personEmail rdf:datatype="http://www.w3.org/2001/XMLSchema#string">kay@csli.stanford.edu</ltw:personEmail>
</rdf:Description>


<rdf:Description rdf:about="http://www.lt-world.org/ltw.owl#obj_65046">
     <ltw:personLastname rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Kay</ltw:personLastname>
     <ltw:affiliatedWith rdf:resource="http://www.lt-world.org/ltw.owl#obj_72667"/>
     <ltw:personEmailAutomate rdf:datatype="http://www.w3.org/2001/XMLSchema#string">:auto-0.61:kay@coli.uni-
sb.de</ltw:personEmailAutomate>
     <ltw:dc_description rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Personal homepage; This is a personal
homepage</ltw:dc_description>
     <ltw:dc_keyword rdf:datatype="http://www.w3.org/2001/XMLSchema#string">finite-state methods in nlp</ltw:dc_keyword>
     <ltw:affiliatedWith rdf:resource="http://www.lt-world.org/ltw.owl#obj_77187"/>
     <ltw:deadlinkURL rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www-linguistics.stanford.edu/cgiw-
bin/linguistics/people/ppage.pl?id=Fmjkay</ltw:deadlinkURL>
     <ltw:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Martin Kay</ltw:name>
     <ltw:personNameVariant rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Martin Kay</ltw:personNameVariant>
     <ltw:personFirstname1 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Martin</ltw:personFirstname1>
     <ltw:dc_language rdf:resource="http://www.lt-world.org/ltw.owl#lt-world_Individual_105"/>
     <ltw:dc_rights rdf:datatype="http://www.w3.org/2001/XMLSchema#string">1999, 2000, 2001 Stanford Linguistics
Department</ltw:dc_rights>
     <ltw:homepageURL rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www.coli.uni-
sb.de/~kay/</ltw:homepageURL>
     <ltw:personTitle rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Prof.</ltw:personTitle>
     <ltw:personName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Martin Kay</ltw:personName>
     <ltw:homepageURL rdf:datatype="http://www.w3.org/2001/XMLSchema#string">http://www-
linguistics.stanford.edu/people/pages/kay.shtml</ltw:homepageURL>
     <ltw:dc_keyword rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Parsing generation</ltw:dc_keyword>
     <ltw:dc_creator rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Martin Kay</ltw:dc_creator>
</rdf:Description>
```

XML as application data and interchange format

- web application servers (Tomcat, Zope, ...)
- neutral content storage, layout generation
- XML application interfaces
- XML-RPC, SOAP
- configuration
- WSDL

- header:

  `<?xml version='1.0' encoding='...'?>` should be the first line

- reference to a DTD (optional; may be in-line):

  ```
  <!DOCTYPE annotation SYSTEM 'nlp.dtd'>
  ```

- reference to a stylesheet (optional; see later)

  ```
  <?xml-stylesheet type='text/xsl' href='style.xsl'?>
  ```

- body containing a *single* root element with nested child elements (tree structure)

- XML documents *must* be well-formed
  - a document must contain a single root element
    - e.g. <sentence>...</sentence>
  - elements must be balanced and properly nested
    - <sentence><w id="W0">**Peter**</w>...</sentence>
    - empty element: <sentence/>
  - attributes must be specified and their values must be quoted
    - <w id="W0">
  - text content must contain legal XML characters

- XML documents *may* be valid
  - document structure and content follows rules specified by a grammar, e.g.
    - DTD (Document Type Definition),
    - XML Schema, or
    - Relax NG
  - most XML parsers offer (optional) DTD validation

DTD Syntax

- DTD defines
  - admissible and required elements
  - element nesting, sequence, choice, optionality
  - required and optional ("implied") attributes
  - default values for optional attributes
- in non-XML syntax, BNF-like (easy to read)
- no classic data types, document structure in focus
- similar to SGML DTD, but less powerful

DTD example (inline DTD; incomplete)


```
<?xml version="1.0" encoding="UTF-8"?>
 <!DOCTYPE   nlp [    <!-- sample XML document with inline DTD -->
   <!ELEMENT nlp (sentence)* >
   <!ELEMENT sentence (w|chunk)* >
   <!ATTLIST   sentence id  ID  #REQUIRED>
   <!ELEMENT chunk (w|chunk)* >
   <!ATTLIST   chunk    id  ID  #REQUIRED
                        cat  CDATA  #IMPLIED
                        length  NMTOKEN  #REQUIRED ... >
   <!ELEMENT w (#PCDATA) >
   <!ATTLIST   w    ttype  "ccase|ucase|lcase|number" ...>
   ...
   <!ENTITY % finiteVerb "VVFIN" >
 ]>
 <nlp>
  <sentence id="S0">
   <w id="W0" length="3" offset="0" ttype="ccase" pos="PPER">Wir</w>
   <w id="W1" length="6" offset="4" ttype="lcase" pos="&finiteVerb;">fuhren</w>
   <chunk id="C0" cat="PP" length="11" offset="11">
    <w id="W2" ...
```

DTD Design Principles

- When you design a DTD, ask yourself
  - what should be elements: structure!
  - what should be attributes: atomic "modifiers"
  - what is text
  - elements are extensible (by adding child elements or attributes), attributes are not!
  - do not encode structure in attribute values!!!

1. parameter entities for use in DTDs (RHS)
2. general entities
   - HTML's ä etc. are NOT defined in XML! (use UTF-8 characters instead)
   - predefined entities in XML:
     - & = &
     - ' = '
     - > = >
     - < = <
     - " = "

- W3C Standard (http://w3.org/TR/xmlschema11-1)
- Replacement for DTD
- finer-grained constraints
- rich datatype repository predefined
- user-definable data types (including complex types)
- XML Syntax

XML Schema example: user-defined data type

Integer with value (1800, 1899]:

```
<xsd:simpleType name='NineteenthCentury'>
  <xsd:restriction base='xsd:integer'>
    <xsd:minExclusive value='1800'/>
    <xsd:maxInclusive value='1899'/>
  </xsd:restriction>
</xsd:simpleType>
```

XML Schema vs. DTD

DTD

```
<!ELEMENT order (item)+>
<!ELEMENT item (name,price)>
<!ATTLIST item code NMTOKEN
          #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price currency
          NMTOKEN 'USD'>
```

XML Schema

```
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>

 <xsd:element name='order' type='Order'/>
 <xsd:element name='item'  type='Item'/>
 <xsd:element name='name'  type='Name'/>
 <xsd:element name='price' type='Price'/>

 <!-- <!ELEMENT order (item)+> -->
 <xsd:complexType name='Order'>
  <xsd:sequence>
   <xsd:element ref='item' minOccurs='1'
          maxOccurs='unbounded'/>
  </xsd:sequence>
 </xsd:complexType>

 <!-- <!ELEMENT item (name,price)> -->
 <xsd:complexType name='Item'>
  <xsd:sequence>
   <xsd:element ref='name'/>
   <xsd:element ref='price'/>
  </xsd:sequence>
  <!-- <!ATTLIST item code NMTOKEN #REQUIRED> -->
  <xsd:attribute name='code'>
   <xsd:simpleType>
    <xsd:restriction base='xsd:string'>
     <xsd:pattern value='[A-Z]{2}\d{3}'/>
    </xsd:restriction>
   </xsd:simpleType>
  </xsd:attribute>
 </xsd:complexType>

 <!-- <!ELEMENT name (#PCDATA)> -->
 <xsd:simpleType name='Name'>
  <xsd:restriction base='xsd:string'/>
 </xsd:simpleType>

 <!-- <!ELEMENT price (#PCDATA)> -->
 <xsd:complexType name='Price'>
  <xsd:simpleContent>
   <xsd:extension base='NonNegativeDouble'>
    <!-- <!ATTLIST price currency NMTOKEN 'USD'> -->
    <xsd:attribute name='currency' default='USD'>
     <xsd:simpleType>
      <xsd:restriction base='xsd:string'>
       <xsd:pattern value='[A-Z]{3}'/>
      </xsd:restriction>
     </xsd:simpleType>
    </xsd:attribute>
   </xsd:extension>
  </xsd:simpleContent>
 </xsd:complexType>

 <xsd:simpleType name='NonNegativeDouble'>
  <xsd:restriction base='xsd:double'>
   <xsd:minInclusive value='0.00'/>
  </xsd:restriction>
 </xsd:simpleType>

</xsd:schema>
```

- History
  - JDK 1.3 and below: external JAR, since then part of standard Java
  - JDK 1.4 (JAXP 1.1): "Crimson" reference implementation, XSLT 1.0
  - JDK 5 (JAXP 1.3): Apache "Xerces" implementation, +DOM3, +XSLTC
  - JDK 6 (JAXP 1.4): + StaX
  - next step: XPath/XSLT 2.0 ?

JAXP Packages

- good idea (history!): provide interchangable implementations ("pluggability layer")
- vendor-independent packages:
  - javax.xml.{parsers,transform}
  - org.xml.sax
  - org.w3c.dom
  - JDK 5:

    + javax.xml.{datatype,namespace,validation,xpath}

XML Processing: SAX parser

- Simple API for XML (http://www.saxproject.org/)
- 'Event-driven', programming language-independent interface for XML parsing (read-only, callback)
- SAX in JDK (package org.xml.sax):
  - SAXParser encapsulates implementations
  - SAX1: SAXParser wraps Parser
  - SAX2: SAXParser wraps XMLReader

- Three steps to parse:
  1. spfi = **SAXParserFactory.newInstance()**: creates new Factory instance (e.g., one per Thread)
  2. sp **= spfi.newSAXParser()**: creates new SAX parser
  3. sp**.parse(InputSource, DefaultHandler)**: parses an XML InputSource (InputStream, Reader, URI)
  - DefaultHandler already implements DTDHandler, ContentHandler, ErrorHandler and EntityResolver interfaces

SAX: Where to put the parser code?

Write your own class that extends DefaultHandler, reacting and dispatching on these callback events:

- startDocument
- startElement (elementname, attributes)
- endElement (elementname)
- endDocument
- characters (text)

... and SAX does the rest for you!

SAX: DefaultHandler (ContentHandler)

input document:

```
<?xml version="1.0"?>
<sentence id="S0">
  <w id="W0" length="3" offset="0">Wir</w>
  <w id="W1" length="6" offset="4">fuhren</w>
  <w id="W2" length="4" offset="11">nach</w>
  <w id="W3" length="6" offset="16">Berlin</w>
  <w id="W4" length="1" offset="22">.</w>
</sentence>
```

You have to keep track of the parsed elements (states) in your code!!! -> Exercise

ContentHandler callback calls:

```
startDocument ()
startElement ("sentence", ["id"])
startElement ("w", ["id", "length", "offset"])
characters ("Wir")
endElement ("w")
startElement ("w", ["id", "length", "offset"])
characters ("fuhren")
endElement ("w")
startElement ("w", ["id", "length", "offset"])
characters ("nach")
endElement ("w")
startElement ("w", ["id", "length", "offset"])
characters ("Berlin")
endElement ("w")
startElement ("w", ["id", "length", "offset"])
characters (".")
endElement ("w")
endElement ("sentence")
endDocument ()
```

XML Processing: DOM interface

- DOM (Document Object Model)
- XML tree traversal
- programming language-independent API and object model for dynamic XML document access
- read and write access
- http://w3.org/DOM

- Interface org.w3c.dom.Node
  - Sub-interfaces: Attr, CDATASection, CharacterData, Comment, Document, DocumentFragment, DocumentType, Element, Entity, EntityReference, Notation, ProcessingInstruction, Text
  - typical access methods: set/getAttribute, getTagName, getChildNodes, setNodeValue, getNextSibling, hasAttributes, appendChild, ...

DocumentBuilder Example

- Parse an XML document into a DOM tree:
  - XML input as String (in xmlString)
  - DOM output as Document node (in doc)

```
import org.xml.sax.InputSource;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

InputSource source = new InputSource(new StringReader(xmlString));
try {
      DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
      factory.setValidating(true);
      DocumentBuilder builder = factory.newDocumentBuilder();
      Document doc = builder.parse(source);  //newDocument() creates empty d
      Element root = doc.getDocumentElement();
    }
catch ...
```

DOM Level 3 in JDK 5 (JAXP 1.3)

- powerful DOM navigation through org.w3c.dom.traversal, org.w3c.dom.ranges
- TreeWalker
- NodeFilter
- NodeIterator
- XPath

SAX vs. DOM

SAX:

- suitable even for huge input documents
- strictly sequential proc.
- only parsing framework, no data structure for XML content provided
- read only

DOM:

- builds tree structure of input document in memory as node objects
- resulting tree can be traversed flexibly
- read and write access

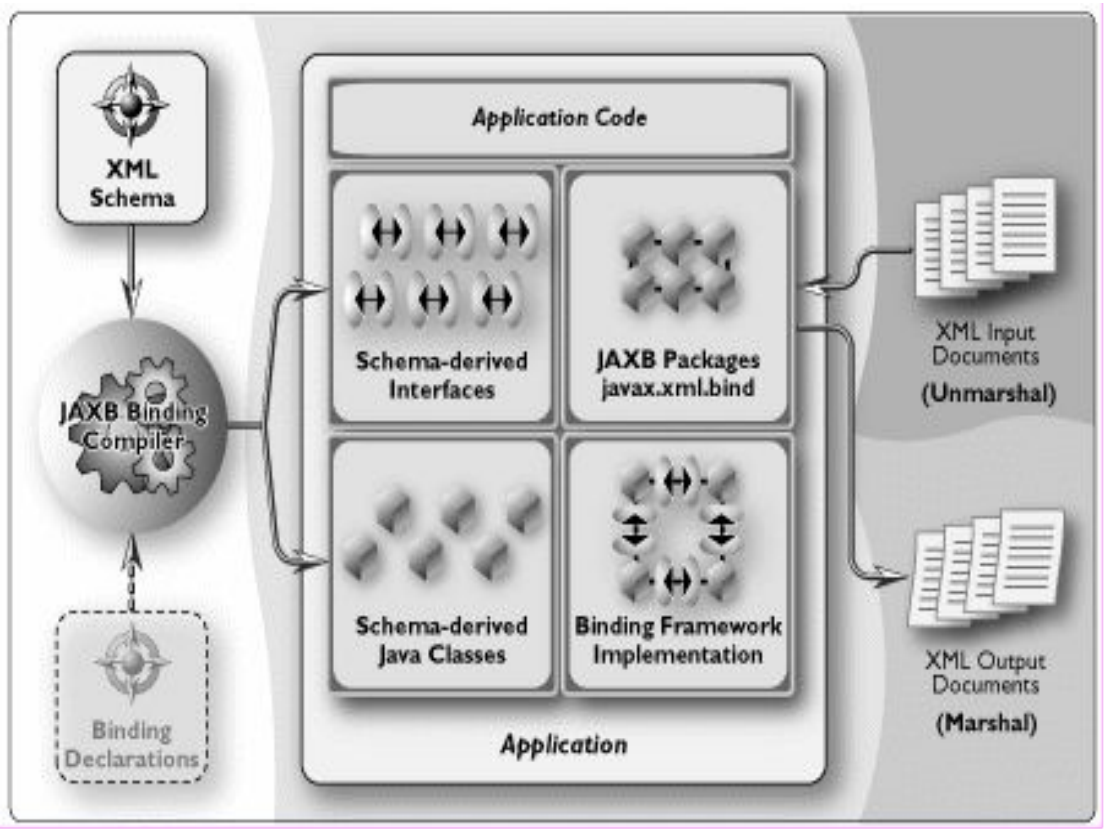
"- don't use SAX unless your document is huge, don't use DOM unless someone is putting a gun to your head" - Istvan Albert

StAX: Streaming API for XML

- more recent than SAX and DOM (JSR 173; part of JAXP 1.4 / JDK 6)
- similar to SAX (also in efficiency), but Pull API (XPP = XML Pull Parser)
  - XMLStreamReader (Cursor; simple)
  - XMLEventReader (Iterator; flexible)
  - StreamFilter + EventFilter to read only partially
- support for XML output (unlike SAX)

Java XML Bindings

- Apache Commons Digester (see Apache lecture)
- JAXB: automatic serialization based on XML Schema



stolen from http://java.sun.com/xml/jaxb/about.html

Literature

- W3C XML 1.0 recommendation: http://www.w3.org/TR/xml/
- Elements vs. Attributes: http://www.oasis-open.org/cover/elementsAndAttrs.html
- XML Tutorial: http://de.selfhtml.org/xml/
- DTD Syntax quickref: http://www.mulberrytech.com/quickref/XMLquickref.pdf
- JAXP, SAX, DOM Tutorial: Chapters 2-7 in J2EE 1.4 Tutorial (PDF, HTML): http://java.sun.com/xml/tutorial_intro.html
- StAX: Chapter 18 in J2EE 1.5 Tutorial (PDF, HTML), Chapter 14 in "Java ist auch eine Insel"