Chapter 11
Implementation and Algorithms

- XML as a data structure:
  - *abstract datatype* with API: DOM
  - (mainly main-memory) implementations; used e.g. in Java applications
  - low-level API with variable-based access
- Databases?
  - high-level API: XPath, XQuery
  - mapping to relational model (Oracle, IBM DB2) or ObjectTypes (Oracle, DB2)
  - “Native” storage: Software AG-Tamino
  - database functionality: multiuser, transactions, recovery
- Stream Processing:
  - XML data transfer as sequence of events
  - SAX (Simple Application Interface for XML)

11.1 DOM

- DOM (Document Object Model) defines a platform- and language-independent object-oriented *interface* (i.e., an *abstract datatype*) for generating, processing and manipulating XML data.
- language base of the DOM specification: OMG-IDL
- DOM is a specification of an interface/abstract datatype, *not a* data model and *not a* programming language!
- implementations in Java, C++, etc; usually main-memory-based;
  specialized Java interface definition: JDOM
- Main-memory-based: only for small application programs
  (most of the “lightweight”-tools used in the course are internally based on DOM)

DOM: PRINCIPLES

- only one document in a DOM
- step-by-step-access to the data:
  based on variable assignments in the surrounding imperative/object-oriented programming language and on iterators (cf. proceeding in the network data model):
  - document: represents the complete document,
    - Query-Methods, e.g. NodeList getElementsByTagName(string)
  - class "Node": getNodeType(), getChildren(), getFirstChild(), getNextSibling(), getParentNode(), ...
  - class "Element": getTagName(), getAttributes(), ...
  - class "Attribute": getName(), getValue(), ...
  - corresponding methods for generating and changing nodes.
- additionally, XPath and XSLT can be applied to instances of Document and Element;
- based on DOM, XPath and XQuery can be implemented (cf. Apache Xerces (XML/DOM)/Xalan (XSLT)/Xindice (DB))
- often inefficient (no indexes, query optimization)

DOM – sample code fragment: Stepwise access

(taken from LanguageElement.java from MARS)

```
// given: Element element;

protected Set<InputVariableDefinition> getInputVariableDefinitions(
          boolean includeJoinVariables)
{ Set<InputVariableDefinition> definitions = new HashSet<InputVariableDefinition>();

  @SuppressWarnings("unchecked")
  List<Element> elements = element.getChildren();
  for (Element e : elements)
  { String elementName = e.getName();
    if (!elementName.equals("Opaque"))
    { String name = e.getAttributeValue("name", "");
      InputVariableDefinition variable = null;
      if (elementName.equals("has-input-variable"))
        variable = new InputVariableDefinition(name, InputVariableDefinition.INPUT);
      else if (elementName.equals("uses-variable") && includeJoinVariables)
        variable = new InputVariableDefinition(name, InputVariableDefinition.USE);
      if (variable != null)
      { String use = e.getAttributeValue("use", "");
        if (use.length() > 0) variable.setUse(use);
        definitions.add(variable);
      }
    }
  }
  return definitions;
}
```

DOM – sample code fragment: XPath

(taken from ServiceRegistry.java from MARS)

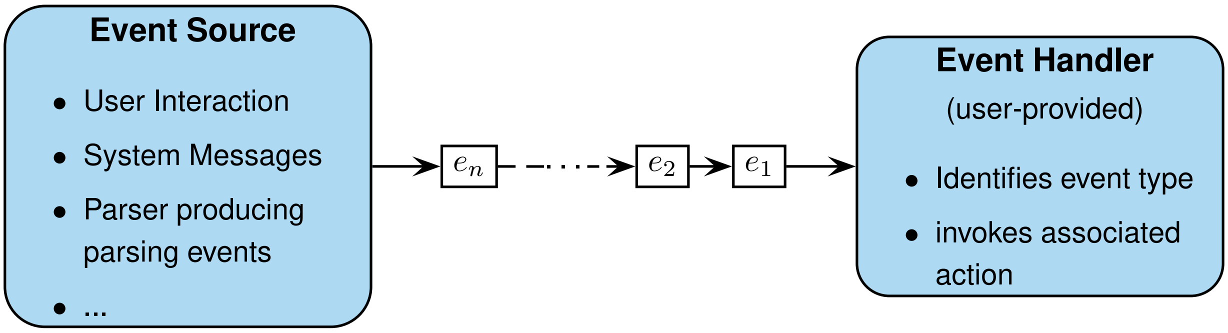
- similar to the JDBC statement concept for SQL in Java:

```
public Element getTaskDescr(Element serviceDescr, String task)
{
  Element taskDescr = null;
  try
  {  XPath xpath = XPath.newInstance(
        "./lsr:has-task-description/lsr:TaskDescription[" +
        "contains(lsr:describes-task/@rdf:resource,$task)]");
     xpath.addNamespace(Namespaces.RDF_NS);
     xpath.addNamespace(Namespaces.MARS_NS);
     xpath.addNamespace(Namespaces.LSR_NS);
     xpath.setVariable("task", task);
     taskDescr = (Element) xpath.selectSingleNode(serviceDescr);
  }
  catch (Exception e) {...}
}
```


11.2 SAX: Event-Based XML Processing

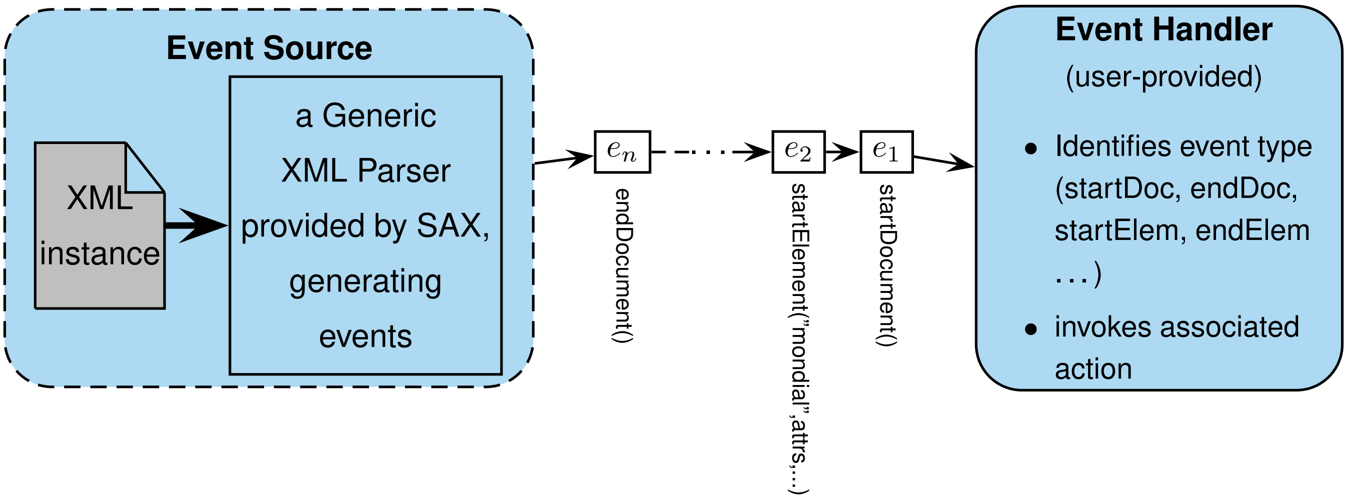
- SAX (The Simple API for XML) is an *event-based interface/model*

Event-Based Processing as a *general Design Pattern*



- The application programmer provides the Event Handler implementation, containing actions for each type of event;
- kind of *rule-based*;
- user is *not* in charge of the control flow.

Event-based XML Parsing with SAX



Represents/processes an XML document as a sequence of events (depth-first traversal), e.g.

- startDocument(), endDocument()
- startElement(Name, attributesList) – attributes not split
- endElement(Name)
- characters(string, int)

XML Parsing with SAX

- a generic XML Parser is parameterized with a *Content Handler* (plus *Error Handler*, *DTD Handler*, and *EntityResolver*) implementation.
- The most trivial Content Handler is the *DefaultHandler* that does nothing: document is parsed, events are detected, but no action is performed.
- Normally, the user-provided Content 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).
- With the content handler implementation, the user provides "actions" in form of Java code, associated with specific events (and even dependent on context information).
- if during parsing of the XML document, a specific event occurs, the code of the associated action from the content handler is invoked (*"callback"*).

SAX: Applications

Only events are signaled: linear processing based on incoming sequence of events.

- ... among many other things, one can generate a DOM tree structure,
- validation according to a DTD (using the automaton as given on Slide 175) in linear time,
- stream-processing of XML input
  - start processing already when input document is not yet complete
  - filtering for elements that are relevant for a given application
  - linear search for something, e.g., names of countries
    (Exercise: sketch the behavior of the event handler on relevant events)
- if necessary: application needs to maintain context
  - usually: maintain a stack that contains all ancestor elements of the element that is currently processed
    (Exercise: describe the event-handler that maintains the stack)

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 > bla.out
>less bla.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 “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];    // ... prepare a contentHandler:
        DefaultHandler handler = new ContentHandlerPrintAttributes(
                "printing attributes of document at url '" + url + "'");
        SAXParserFactory factory = SAXParserFactory.newInstance();
        try {
            SAXParser parser = factory.newSAXParser();
            parser.parse(url, handler);  // <<<<<<<<< and now it runs ...
        } catch (IOException e1) {
            e1.printStackTrace();
        } catch (ParserConfigurationException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        }
    }
}
```

[see java/SAX/PrintAttributes.java]

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);
    }

    // react on opening elements:
    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();
        }
    }

    // all other "parsing events" are ignored in this case
}
```

[see java/SAX/ContentHandlerPrintAttributes.java]

SAX: Applications to XPath Query Answering

Forward queries

XPath-queries like //country[@car_code='D']/population can be answered very (time- and memory-)efficient,

- use the sequence of events (linear)
- maintain some context (often logspace/additional logtime sufficient)

... works only for queries, that contain only forward steps,

General queries

which XPath expressions can be *transformed* in equivalent forward-expressions (and with what efforts)?

- "XPath: Looking forward"; F. Bry et al ; 2002; LMU München
- theory: complexity, connections to linear temporal logic
  For every linear temporal logic formula that uses past and future operators, there is an equivalent formula that uses only future operators
  ... but in general of exponential size.

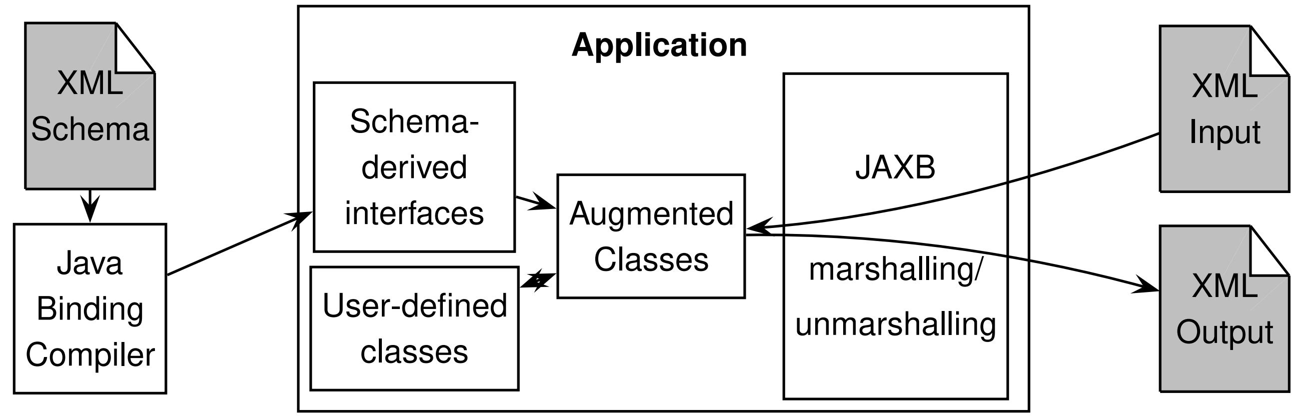
11.3 JAXB - The Java API for XML Binding

- XML elements describe objects with properties,
- correspond to classes of an application,
- derive interface with setX/getX methods (= Java Beans) as skeletons for these classes (automatically generated from XSD),
- user derives classes from these interfaces by adding behavior,
- application logics implemened by using these classes,
- import/export of XML instances of these classes via generic mappings (derived from the XSD).

JAXB Architecture

- map XML Schemas to Java interfaces (get/set methods),
- methods for *unmarshalling* XML instance documents into Java objects.
- methods for *marshalling* Java objects back into XML instance documents.

Architecture



JAXB - Use Case

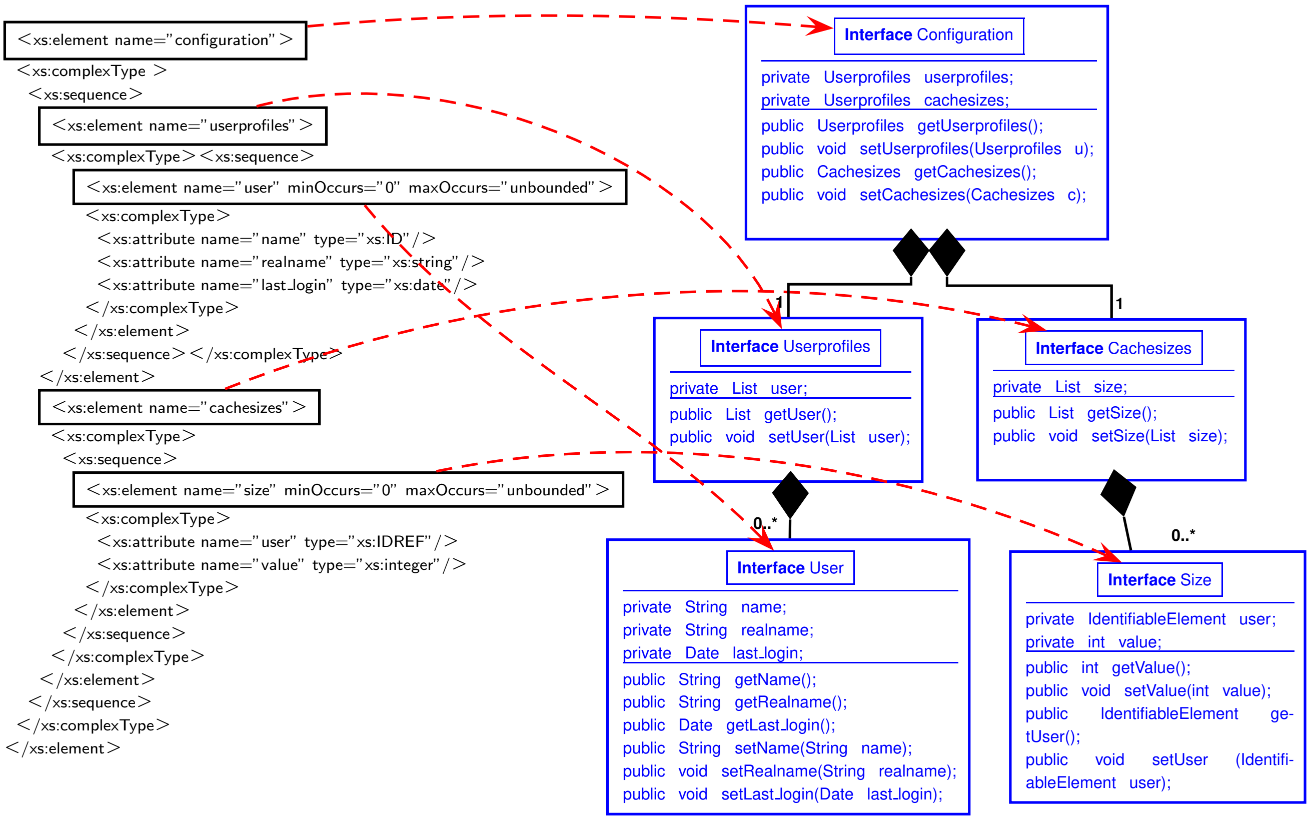
Software configuration management via XML files

```xml
<?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="19-09-2005T17:41.02"/>
    <user name="ofritzen" realname="Oliver Fritzen" last_login="21-09-2005T13:04:27"/>
      ...
  </userprofiles>
  <cachesizes>
    <size user="wmay" value="0"/>
    <size user="ofritzen" value="65535"/>
      ...
  </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:

JAXB: Binding XML Schema to Java Classes / Interfaces



JAXB: Example Code Piece

```
:
try {
    JAXBContext jc = JAXBContext.newInstance("config.binding");
    ObjectFactory objFactory = new ObjectFactory();
    Unmarshaller u = jc.createUnmarshaller();
    Configuration conf = (Configuration) u.unmarshal(new FileInputStream(infilename));

    List users = conf.getUserProfiles().getUsers();
    User user = null;
    for(int i=0; i<users.getLength(); i++) {
        user = (User) users.item(i);
        if(user.getName().equals("wmay")) break;
    }
    if(user != null) user.lastLogin(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 marshalling of XML data,
- integrates well with XML files and with DOM trees,
- higher level of abstraction from XML representation, compared with 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
- 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.