Chapter 11
Implementation and Algorithms

- XML as a data structure:
  - *abstract datatype* with API: DOM
  - (mainly main-memory) implementations
  - low-level API with variable-based access
- Databases?
  - high-level API: XPath, XQuery
  - mapping to relational model (Oracle, IBM 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
- implementations in Java, C++, etc; usually main-memory-based; specialized Java interface definition: JDOM
- DOM *specifies* a *logical* data model (see “XML as data model”), where XML data is represented as trees:
  - 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.

DOM: FUNCTIONALITY LEVELS

- DOM is a specification of an interface/abstract datatype, *not a* data model and *not a* programming language!

Definition of DOM "Levels" with increasing functionality:

- DOM Level 1: Basics
- DOM Level 2: + namespaces
- DOM Level 3: + handling of DTDs, XML Schema, validation ...

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)
- no declarative access
- based on DOM, XPath and XQuery can be implemented (cf. Apache Xerces (XML/DOM)/Xalan (XSLT)/Xindice (DB)
- Main-memory-based: only for small application programs
  (most of the "lightweight"-tools (see ap34) are internally based on DOM)
- often inefficient (no indexes, query optimization)

⇒ databases must follow other ideas: what *physical data model* is best suited for XML?

11.2 SAX

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

Serialized approach to XML:

- 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,int)
- for SAX-based programming, an event handler is written that reacts on the incoming events and does something (application-dependent)
- SAX was the first API for XML – every XML application at that time was programmed as a state machine.

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 168) 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: 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.