Chapter 4
XML (Extensible Markup Language)

Introduction

- SGML *very* expressive and flexible
  HTML very specialized.
- Summer 1996: John Bosak (Sun Microsystems) initiates the XML Working Group (SGML experts), cooperation with the W3C.
  Development of a subset of SGML that is simpler to implement and to understand
  `http://www.w3.org/XML/`: the homepage for XML at the W3C

⇒ XML is a "stripped-down version of SGML".

- for understanding XML, it is not necessary to understand everything about SGML ...

HTML

let's start the other way round: HTML ... well known, isn't it?

- tags: pairwise opening and closing: <TABLE> ... </TABLE>
- "empty" tags: without closing tag
, <HR>
- <P> is *in fact* not an empty tag (it should be closed at the end of the paragraph)!
- attributes: <TD colspan = "2"> ... </TD>
- empty tags with attributes:
  <IMG SRC="http://www.informatik.uni-goettingen.de/photo.jpg" ALIGN="LEFT">
- content of tag structures: <TD>123456</TD>
- nested tag structures: <TH><B>Name</B></TH>

  <A href="http:www.ifi.informatik.uni-goettingen.de">
  <B>Homepage of the IFI</B></A>

⇒ hierarchical structure

- Entities: ä = ä ß= ß

HTML

- browser must be able to interpret tags
  → semantics of each tag is fixed for all (?) browsers.
- fixed specifications how tags can be nested
  (described by a DTD (Document Type Definition))

```
<body><H1>...</H1><H2>...</H2>
                  <P> ... </P>
                  <H2>...</H2>
                  <P> ... </P>
      <H1>...</H1><H2>...</H2>
                  <P> ... </P>
</body>
```

- analogously for tables and lists ...
- reality: people do in general not adhere to this structure
  - closing tags are omitted
  - structuring levels are omitted

  → parser has to be fault-tolerant and auto-completing

Knowledge of HTML for XML?

- intuitive idea – but only of the *ASCII representation*
- this is *not a data model*
- no query language
- only a very restricted viewpoint:
  HTML is a markup language for browsers
  (note: we don't "see" HTML in the browser, but only what the browser makes out of the HTML).

  Not any more.

GOALS OF THE DEVELOPMENT OF XML

- XML must be directly usable and transmitted in the internet (Unicode-Files),
- XML must support a wide range of applications,
- XML must be compatible with SGML,
- XML documents must be human-readable and understandable,
- XML documents must be easy to create,
- it must be easy to write programs that evaluate/process/parse XML documents.

DIFFERENCES BETWEEN XML AND HTML?

- Goal: *not browsing*, but representation/storage of (semistructured) data (cf. SGML)
- SGML allows the definition of new tags according to the application semantics; each SGML application uses its own *semantic tags*.
  These are defined in a DTD (Document Type Definition).
- HTML is *an* SGML application (cf. `<HTML> at the beginning of each document </HTML>`), that uses the DTD "HTML.dtd".
- In XML, (nearly) arbitrary tags can be defined and used:

```
<country> ... </country>
<city> ... </city>
<province> ... </province>
<name> ... </name>
```

- These *elements* represent objects of the application.

XML as a Meta-Language for Specialized Languages

- For each application, it can be chosen which “notions” are used as element names etc.:
  ⇒ document type definition (DTD)
- the set of allowed element names and their allowed nesting and attributes are defined in the DTD of the document (type).
- the DTD describes the *schema*
- XML is a *meta-language*, each DTD defines an own language
- for an application, either a new DTD can be defined, or an existing DTD can be used
  → standard-DTDs
- HTML has (as an SGML application) a DTD

Example: Mondial

```
<mondial>
    :
  <country code="D" capital="city-D-Berlin" memberships="EU NATO UN ...">
    <name>Germany</name>
    <encompassed continent="europe">100</encompassed>
    <population year="1995">83536115</population>
    <ethnicgroup name="German">95.1</ethnicgroup>
    <ethnicgroup name="Italians">0.7</ethnicgroup>
    <religion name="Roman Catholic">37</religion>
    <religion name="Protestant">45</religion>
    <language name="German">100</language>
    <border country="F" length="451"/>
    <border country="A" length="784"/>
    <border country="CZ" length="646"/>
      :
```

Example: Mondial (Cont'd)

```
      :
    <province id="prov-D-berlin" capital="city-D-berlin">
      <name>Berlin</name>
      <population year="1995">3472009</population>
      <city id="city-D-berlin">
        <name>Berlin</name> <population year="1995">3472009</population>
      </city>
    </province>
    <province id="prov-D-baden-wuerttemberg" capital="city-D-stuttgart">
      <population year="1995">10272069</population>
      <name>Baden Wuerttemberg</name>
      <city id="city-D-stuttgart">
        <name>Stuttgart</name> <population year="95">588482</population>
      </city>
      <city id="cty-D-mannheim"> ... </city>
      :
    </province>
    :
  </country>
  :
</mondial>
```


CHARACTERISTICS:

- hierarchical “data model”
- subelements, attributes
- references
- ordering? documents – yes, databases – no

Examples can be found at

`http://dbis.informatik.uni-goettingen.de/Mondial/#XML`

XML AS A DATA MODEL

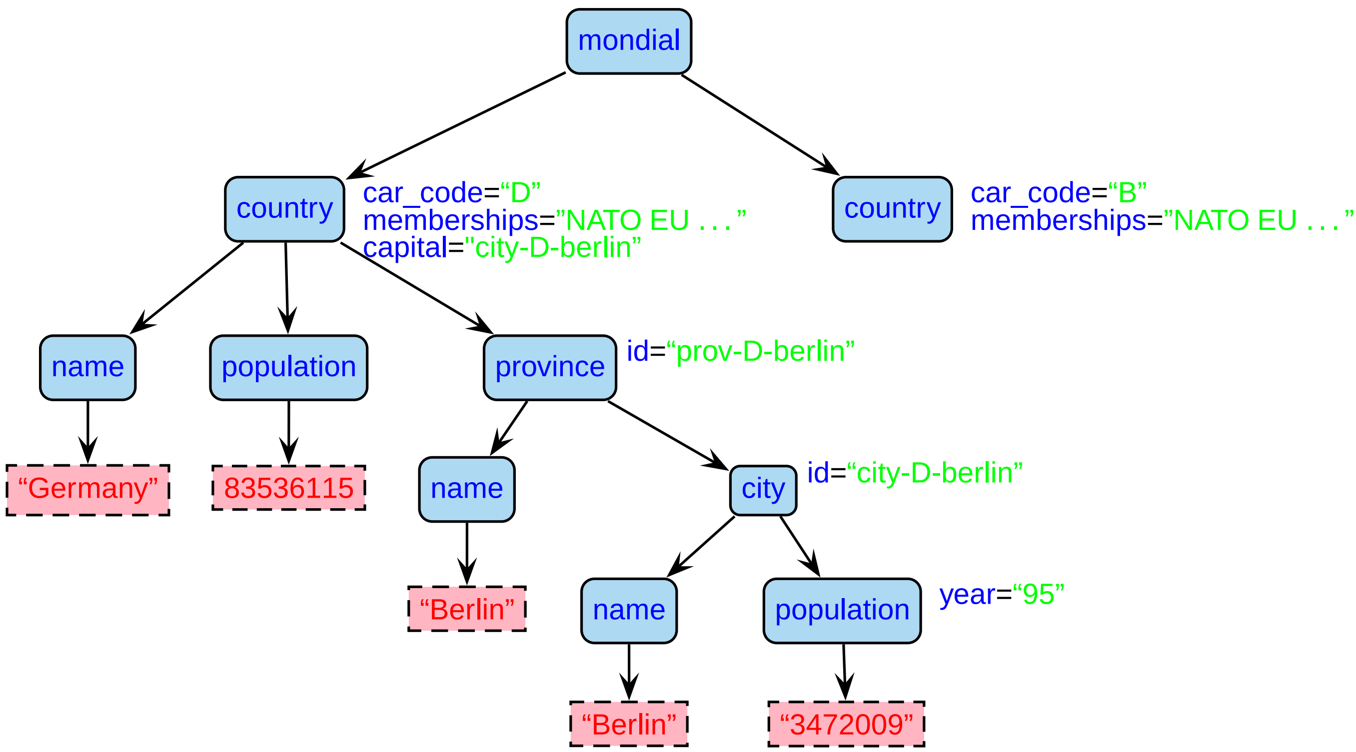
XML is much more than only the ASCII representation shown above as known from HTML (see also introductory talk)

- abstract data model (comparable to the relational DM)
- abstract datatype: DOM (Document Object Model) – see later
- many concepts around XML
  (XML is *not* a programming language!)
  - higher-level declarative query/manipulation language(s)
  - notions of “schema”

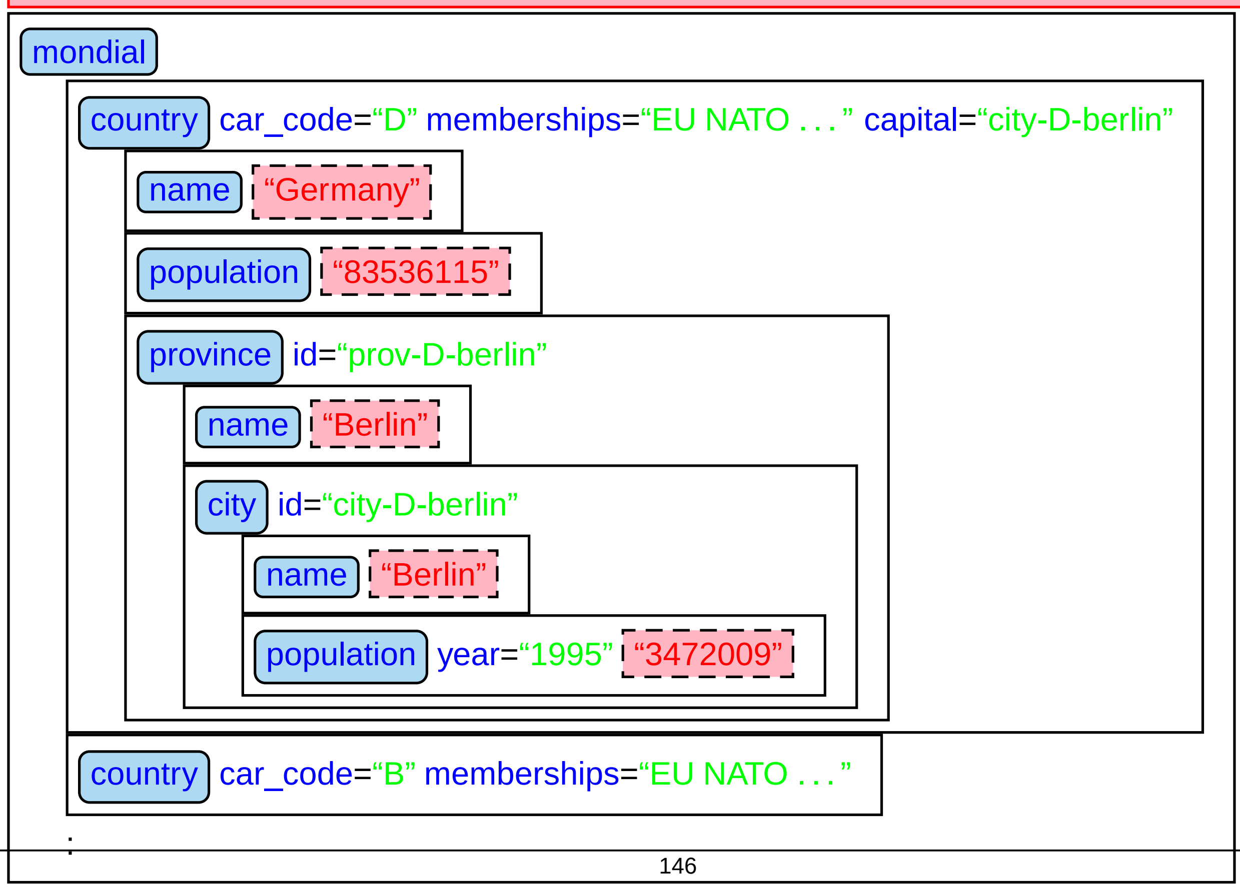
4.1 Structure of the Abstract XML Data Model (Overview)

- for each document there is a document node which “is” the document, and which contains information about the document (reference to DTD, doctype, encoding etc).
- the document itself consists of nested *elements* (tree structure),
- among these, exactly one *root element* that contains all other elements and which is the only child of the document node.
- elements have an element type (e.g. Mondial, Country, City)
- element content (if not empty) consists of text and/or *subelements*.
  These *child nodes* are ordered.
- elements may have *attributes*.
  Each attribute node has a name and a value (e.g. (car_code, “D”)).
  The attribute nodes are unordered.
- *empty elements* have no content, but can have attributes.
- a *node* in an XML document is a logical unit, i.e., an element, an attribute, or a text node.
- the allowed structure can be restricted by a schema definition.

EXAMPLE: MONDIAL AS A TREE



EXAMPLE: MONDIAL AS A NESTED STRUCTURE



OBSERVATIONS

- there is a global order (preorder-depth-first-traversing) of all element- and text nodes, called *document order*.
- actual text is only present in the text-nodes
  Documents: if all text is concatenated in document order, a pure text version is obtained.
  Exercise: consider an HTML document.
- element nodes serve for structuring (but do not have a “value” for themselves)
- attribute nodes contain values whose semantics will be described in more detail later
  - attributes that describe the elements in more detail
    (e.g. td/@colspan or population/@year)
  - IDs and references to IDs
  - can be used for application-specific needs

4.2 XML ASCII Representation

- Tree model and nested model serve as abstract datatypes (see later: DOM)

data exchange? how can an XML document be represented?

- a relational DB can be output as a finite set of tuples (cf. relational calculus)
  country(“Germany”, “D”, 83536115, 356910, “Berlin”, “Berlin”)
  or
  country(Name: “Germany”, Code: “D”, Population: 83536115, Area: 356910,
  Capital: “Berlin”, CapitalProvince: “Berlin”)
- object-oriented databases: OIF (Object Interchange Format)
- OEM-tripels, F-Logic-frames
- XML?
  Exporting the tree in a *preorder-depth-first-traversing*.
  The node types are represented in a specified syntax:
  ⇒ XML as a representation language

ASCII: XML as a Representation Language

- elements are limited by
  - opening <Country> and
  - closing *tags* </Country>,
  - in-between, the *element content* is output recursively.
- Element content consists of text

  <Name> United Nations </Name>
- and *subelements*: <Country> <City> ... </City>

  <City> ... </City>

  </Country>
- *attributes* are given in the opening tag:

  <Country car_code="D"> . . . </Country>

  where attribute values are always given as strings, they do not have further structure. The difference between value- and reference attributes is not visible, but is only given by the DTD.
- *empty elements* have only attributes: <border country="F" length="451"/>

XML as a Representation Language: Grammar

The language "XML" defined as above can be given as an EBNF grammar:

Document	::=	Element
Element	::=	"<" ElementName Attribute* ">" Content "</" ElementName ">"
		| "<" ElementName Attribute* "/>"
Content	::=	(Element | Text)+
Text	::=	*characters including whitespace*
Attribute	::=	AttributeName "='" AttributeValue "'"
ElementName, AttributeName	::=	*character string with some restrictions*
AttributeValue	::=	*characters including whitespace*

- note that this grammar does not guarantee that the opening and closing tags match!
- instead of ', also the usual " are allowed
- strict adherence to these rules (closing and empty elements) is required.
- an XML instance given as ASCII is *well-formed*, if it satisfies these rules.
- "XML parsers" process this input.

XML Parser

- an *XML parser* is a program that processes an XML document given in ASCII representation according to the XML grammar, and generates a result:
  - correctness: check for well-formedness (and adherence to a given DTD)
  - DOM-parser: transformation of the XML instance into a DOM model (implementation of the abstract datatype; see later).
  - SAX-parser: traversing the XML tree and generation of a sequence of "events" that *serialize* the document (see later).
- XML parsers are required to accept only well-formed instances.
  - simple grammar, simple (non-fault-tolerant) parser
  - HTML: fault-tolerant parsers are much more complex
    (fault tolerance wrt. omitted tags is only possible when the DTD is known)
- each XML application must contain a parser for processing XML instances in ASCII representation as input.

XML Parsing in the General Case

- ElementName is a separate production and

  Element ::= "<" ElementName Attribute* ">" Content "</" ElementName ">"
  | "<" ElementName Attribute* "/>"

  does not guarantee matching tags

⇒ not context-free!

- Nevertheless, context-free-style parsing with push-down-automaton *without fixed stack alphabet* possible:
  - for every opening tag, put ElementName on the stack
  - for every closing tag, compare with top of stack, pop stack.

⇒ linear-time parsing

- Exercise: give an automaton for parsing XML and describe the handling of the stack (solution see Slide 179).

Viewing XML Documents?

- as a file in the editor
  - emacs with xml-mode
  - Linux/KDE: kxmleditor
- browser cannot "interpret" XML
  (in contrast to HTML)
- with "show source" in a browser:
  current versions of most browsers show XML in its ASCII representation with indentation and allow to open/close elements/subtrees.
- but, in general, XML is not intended for viewing:
  → transformation to HTML by XSLT stylesheets
  (see later)

4.3 Datatypes and Description of Structure for XML

- relational model: atomic data types and tuple types
- object-oriented model: literal types and object types, reference types

Data Types in XML

- data types for text content
- data types for attribute values
- element types (as "complex objects")
- somewhat different approaches in DTD (document-oriented, coarse) and XML Schema (database-oriented, fine)

DOCUMENT TYPE DEFINITION – DTD

- the set of allowed tags and their nestings and attributes are specified in the DTD of the document (type).
- the idea of the DTD comes from the SGML area
  - meets the requirements for describing document structure
  - does not completely meet the requirements of the database area
    $\rightarrow$ XML Schema (later)
  - simple, and easy to understand.
- the DTD for a document type $doctype$ is given by a grammar (context-free; regular expression style) that characterizes a class of documents:
  - what elements are allowed in a document of the type $doctype$,
  - what subelements they have (element types, order, cardinality)
  - what attributes they have (attribute name, type and cardinality)
  - additionally, “entities” can be defined (they serve as constants or macros)

DATA TYPES OF DTDS

- text content: PCDATA – parsed character data
  it is up to the application to distinguish between string data and numerical data
- data types for attribute values:
  - CDATA: simple strings
  - NMTOKEN: string without blanks
  - NMTOKENS: a list of tokens, separated by blanks
  - ID: like NMTOKEN, each value must be unique in the document
  - IDREF: like NMTOKEN, each value must occur in the same document as an ID value
  - IDREFS: the same, multivalued
- element types: definition of structure in the style of regular expressions.

DTD: Element Type Definition – Structure of the Element Contents

<!ELEMENT *elem_name struct_spec*>

- EMPTY: empty element type,
- (#PCDATA): text-only content
- (*expression*): expression over element names and combinators (same as for regular expressions). Note that the expression must be deterministic.
  - ",": sequence,
  - "|": (exclusive-)or (choice),
  - "*": arbitrarily often,
  - "+": at least once,
  - "?": optional
- (#PCDATA|*elem_name*$_1$|...|*elem_name*$_n$)*
  mixed content, here, only the types of the subelements that are allowed to occur together with #PCDATA can be specified; no statement about order or cardinality.
- ANY: arbitrary content

Element Type Definition: Examples

- from HTML: images have only attributes and no content
  <!ELEMENT img EMPTY >
- from Mondial:

```
<!ELEMENT country (name, encompassed+, population*,
                   ethnicgroup*, religion*, border*,
                   (province+ | city+))>
<!ELEMENT name (#PCDATA)>
```

- for text documents:

```
<!ELEMENT Section (Header,
                   (Paragraph|Image|Figure|Subsection)+,
                   Bibliography?)>
```

- Element type definitions by regular expressions
  ⇒ can be checked by finite state automata

DTD: Attribute Definitions

- General: an element contains at most one attribute of every attribute name.
- details about allowed attribute names and their types are specified in the DTD.

<!ATTLIST	*elem_name*		
	$attr\_name_1$	$attr\_type_1$	$attr\_constr_1$
	:	:	:
	$attr\_name_n$	$attr\_type_n$	$attr\_constr_n$>

- $attr\_type_i$: value/reference attribute and scalar/multi-valued
  - CDATA: arbitrary text.
  - NMTOKEN: scalar, token-content (text without blanks).
  - NMTOKENS: multi-valued, token-content.
  - ($const_1 | \ldots | const_k$): scalar, from a given domain.
  - ID: distinguished scalar attribute, token-content, unique in the whole document.
  - IDREF: scalar, its value is a token that occurs as a value of an ID attribute in the same document (reference).
  - IDREFS: multi-valued reference attribute.

DTD: Attribute Definitions (cont'd)

<!ATTLIST	*elem_name*		
	$attr\_name_1$	$attr\_type_1$	$attr\_constr_1$
	:	:	:
	$attr\_name_n$	$attr\_type_n$	$attr\_constr_n$>

- $attr\_constr_i$: minimal cardinality
  - #REQUIRED: attribute must be present for each element of this type.
  - #IMPLIED: attribute is optional.
  - *default*: Default-value (non-monotonic value inheritance).
  - #FIXED *value*: attribute has the same (given) value for each element of this type (monotonic value inheritance).

DTD: Attribute-Definitions (Examples)

```
<!ATTLIST Country
    Code         ID           #REQUIRED
    Capital      IDREF        #REQUIRED
    Memberships  IDREFS       #IMPLIED
    Products     NMTOKENS     #IMPLIED >
<!ATTLIST desert
    id           ID           #REQUIRED
    Type         (sand,rocks,ice) 'sand'
    Climate      NMTOKENS     #FIXED 'dry' >
```

- when an XML parser reads an XML instance and its DTD, it fills in default and fixed values.

DTD and XML Instances

- Each DTD defines an own markup language (i.e., an XML application – HTML is one, Mondial is another).
- an XML instance has a *document node* (which is not the root node, but even “superior”) that contains among other things information about the DTD.
  (see next slides ...)
- the root element of the document must be of an element type that is defined in the DTD.
- an XML instance is *valid* wrt. a DTD if it satisfies the structural constraints specified in the DTD.
  Validity can be checked by an extended finite state automaton in linear time.
- XML-instances can exist without a DTD (but then, it is not explicitly specified what their tags “mean”).

XML Document Structure: The Prolog

The *prolog* of an XML document in ASCII-representation contains additional information about the document (associated with the document node):

- XML declaration (with optional attributes)

  <? xml version="1.0" encoding="utf-8"?>

  encoding="ISO-8859-1" allows additionally German "Umlauts".

- document type *declaration*: indication of the document type, and where the document type *definition (DTD)* can be found.
  - <!DOCTYPE *name* {SYSTEM|PUBLIC *public-id*} *url*>
    SYSTEM *url*: own document type,
    name: one of the element names given in the DTD

    <!DOCTYPE Mondial SYSTEM "mondial-2.0.dtd">

    PUBLIC *public-id url*: standard document type (e.g. XHTML), or
  - <!DOCTYPE *name* [*dtd*]>
    with DTD directly included *in* the document.

- then follows the document content (i.e., the root node with the document body as its content).

Example: Mondial

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE mondial SYSTEM "mondial-2.0.dtd">
<mondial>
 <country car_code="AL" area="28750" capital="cty-cid-cia-Albania-Tirane"
          memberships="org-BSEC org-CE org-CCC ...">
  <name>Albania</name> <population>3249136</population>
  <encompassed continent="europe" percentage="100"/>
  <ethnicgroups percentage="3">Greeks</ethnicgroups>
  <ethnicgroups percentage="95">Albanian</ethnicgroups>
  <border country="GR" length="282"/> <border country="MK" length="151"/>
  <border country="YU" length="287"/>
  <city id="cty-cid-cia-Albania-Tirane" is_country_cap="yes" country="AL">
   <name>Tirane</name>
   <longitude>10.7</longitude> <latitude>46.2</latitude>
   <population year="87">192000</population>
  </city>
  :
 </country>
 :
</mondial>
```

TOOL: XMLLINT

`xmllint` is a simple tool that allows (among other things – see later) to validate a document (belongs to libxml2):

- `man xmllint`: lists all available commands
- currently, we are mainly interested in the following:
  `xmllint -loaddtd -valid -noout mondial-europe.xml`
  validates an XML document wrt. the DTD given in the prolog.

XMLLINT: Further Functionality (see later)

XMLLINT can be used to “visit” the document, and to walk through it:

- call `xmllint -loaddtd -shell mondial-europe.xml`.

Then, one gets a “navigating shell” “inside” the XML document tree
(very similar to navigating in a UNIX directory tree):

- `validate`: validates the document
- `cd xpath-expression`: navigates into a node
  (the XPath expression must uniquely select a single node)
  relativ: `cd country[1]`
  absolut: `cd //country[@car_code="D"]`
- `pwd`: gives the path from the root to the current position
- `cat`: prints the current node
- `cat xpath-expression`
  `cat .//city/name`
- `du xpath-expression` lists the content of the node that is selected by `xpath-expression` (starting from the current node)
- `dir xpath-expression` prints the node type and attributes of the selected node

Example: “Books” from W3C XML Use Cases

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE bib SYSTEM "books.dtd">
<!-- from W3C XML Query Use Cases -->
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
  <book year="1999">
    <title>Economics of ... for Digital TV</title>
    <editor>
      <last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
```

[see XML-DTD/books.xml]

Exercise: Generate a DTD for the above XML

... do it step-by-step, using a validator:

- for all element types:
  <!ELEMENT *name* ANY>
- declare <!ATTLIST *name* ...> where needed
- validate
- stepwise refinement of content models ...
- ... blackboard demonstration ...
- solution see Slide 175

DATA-CENTERED VS. DOCUMENT-CENTERED XML DOCUMENTS

Data-Centered XML Documents

- very regular structure with “data fields”
- only some text
- no naturally induced tree structure

Document-Centered XML Documents

- tree structure with much text (text content is the text of the document)
- non-regular structure of elements
- logical markup of the documents
- annotations of the text by additional elements/attributes

Semistructured XML Documents

- combine both (e.g. medical information systems)

SUBELEMENTS VS. ATTRIBUTES

When designing an XML structure, often the choice of representing something as subelement or as attribute is up to the designer.

Document-Centered XML

- the concatenation of the whole text content should be the “text” of the document
- element structures for logical markup and annotations
- attributes contain additional information *about* the structuring elements.

Data-Centered XML

- more freedom
- attributes are unstructured and cannot have further attributes
- elements allow for structure and refinement with subelements and attributes
- using DTDs as schema language allows the following functionality only for attributes:
  – usage as identifiers (ID)
  – restrictions of the domain
  – default values
  (XML Schema and XLink allow many more things)

EXAMPLES AND EXERCISES

- The MONDIAL database is used as an example for practical experiments. See `http://dbis.informatik.uni-goettingen.de/Mondial#XML`.
- many W3C documents base on examples about a literature database (book, title, authors, etc.).
- each participant (possibly in groups) should choose an *own* application area to set up an own example and to experiment with it.
  - from the chosen branch of study?
  - database of music CDs
  - lectures and persons at the university
  - exams (better than FlexNever?)
  - calendar and diary
  - other ideas ...

Exercise: Define a DTD and generate a small XML document for your chosen application.

EXERCISES

- Validate your example document with a suitable prolog and internal DTD.
- put your DTD publicly in your public-directory and validate a document that references this DTD as an external DTD.
- take a DTD+url from a colleague and write a small instance for the DTD and validate it.

Data Exchange with XML

For *Electronic Data Interchange (EDI)*, a commonly known+used DTD is required

- producers and suppliers in the automobile industry
- health system, medical area
- finance/banking

Proceeding

Usually, XML data is exchanged in its ASCII representation.

- XML-Server make documents in the ASCII representation accessible (i.e., as a stream or as a textfile)
- applications *parse* this input (linear) and store it internally (DOM or anything else).

4.3.1 Aside: XML Parsing

... one of the objectives of this lecture is also to show the applications and connections of basic concepts of CS ...

- XML/DTD: content models are regular expressions
  ⇒ can be checked by finite state automata
  - design one automaton for each <!ELEMENT ...> declaration
  - design a combined automaton for validating documents against a given DTD
  - extension to attributes: straightforward (when processing opening tags, dictionary-based)
  - checking for well-formedness and validity in linear time
    - ⋆ with a DOM parser: during generation of the DOM
    - ⋆ with a SAX parser: streaming, on the fly
    - ⋆ using a DOM instance: depth-first traversal
- without a DTD: requires a push-down automaton
  (remembering opening tags); still linear time
  - checking well-formedness
  - generating a DOM instance, or on-the-fly (SAX)

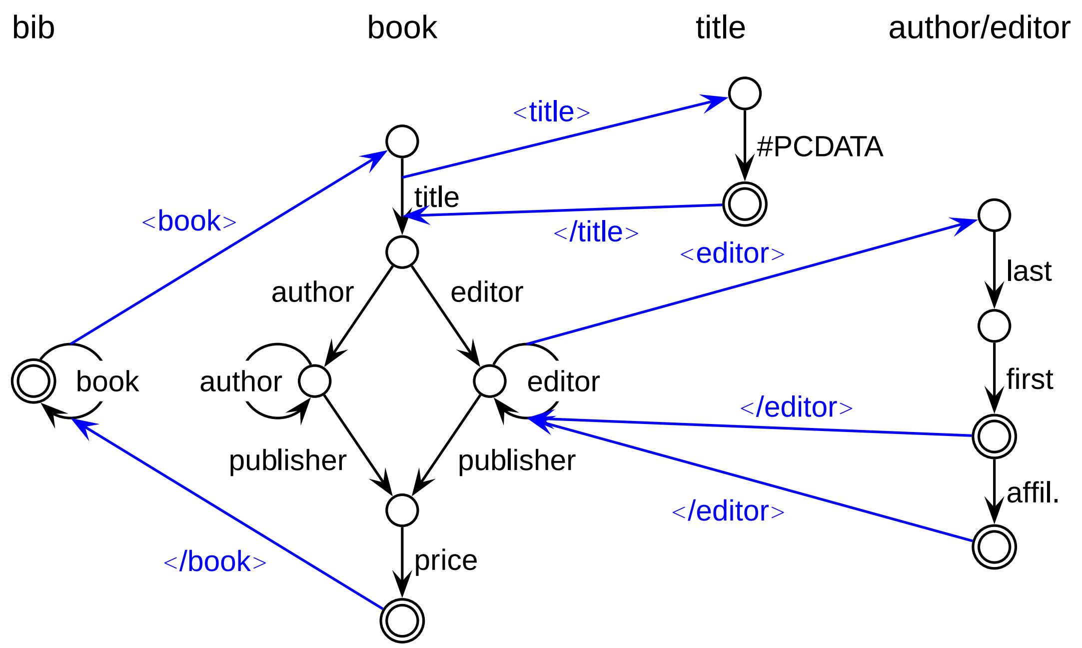
Finite State Automata for Validation
Example: books.dtd

Consider the "books" example:

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, (author+ | editor+), publisher, price)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (last, first, affiliation?)>
<!ELEMENT last (#PCDATA)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT editor (last, first, affiliation?)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
```


Finite State Automata

- individual automata for element content models
  (recall that the content model must be deterministic)
- + detailed by nesting (jumping on opening/closing tags)



XML Grammar in presence of a DTD

Consider the grammar from Slide 150:

- Element names known from a DTD: context-free grammar

Document	::=	Element
Element	::=	"<bib" Attribute* ">" Content "</bib>"
Element	::=	"<book" Attribute* ">" Content "</book>"
⋮	⋮	⋮
Content	::=	(Element | Text)+
Text	::=	*characters*
Attribute	::=	AttributeName "='" AttributeValue "'"
AttributeValue	::=	*characters*

- there is even a regular grammar, see above automata, but this is not derived from the XML EBNF.

XML Grammar in General

- no DTD present/element names not known:
  Consider the grammar from Slide 150:
- ElementName is a separate production and

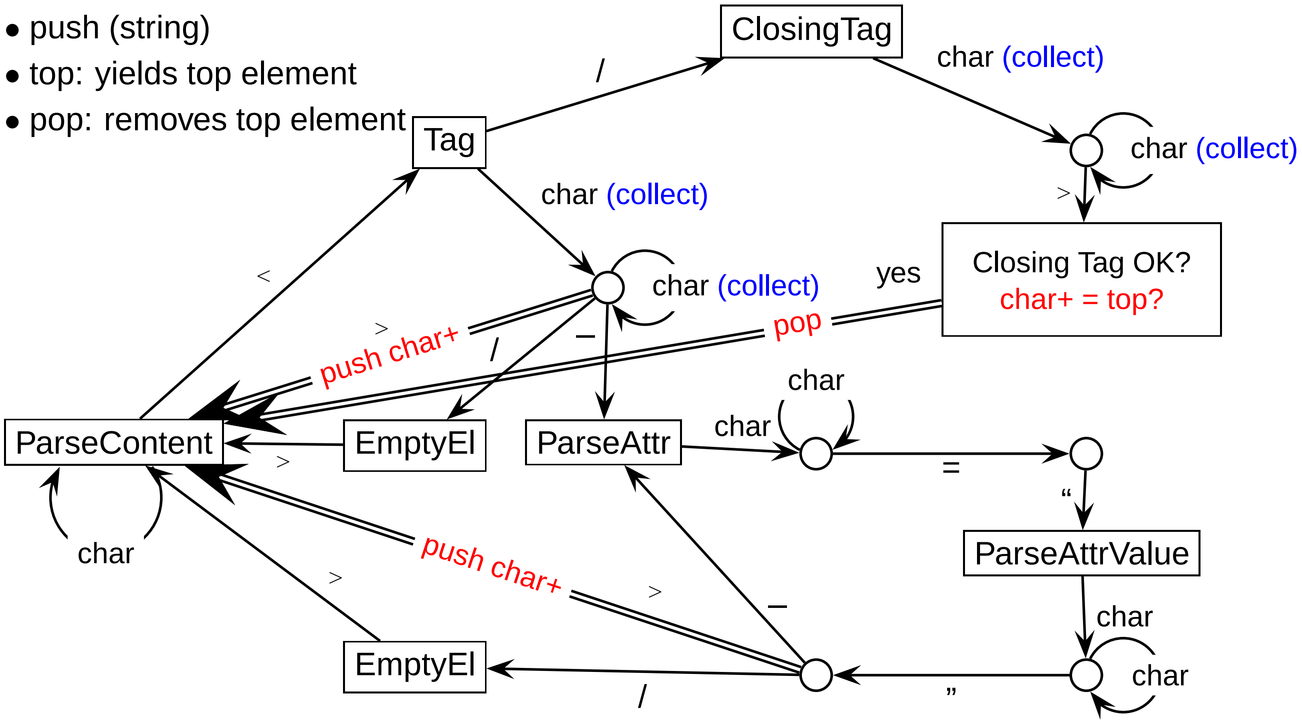
  Element ::= "<" ElementName Attribute* ">" Content "</" ElementName ">"
  | "<" ElementName Attribute* "/>"

  does not guarantee matching tags.
- Nevertheless, context-free-style parsing with push-down-automaton *without fixed stack alphabet* possible:
  - for every opening tag, put ElementName on the stack
  - for every closing tag, compare with top of stack, pop stack.
- Automaton: see next slide.

XML Grammar in General

Stack Commands:

- push (string)
- top: yields top element
- pop: removes top element



4.4 Example: XHTML

- XML documents that adhere to a strict version of the HTML DTD
- Goal: browsing, publishing
- DTD at `http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd`
  (note that the DTD requires also some entity files)
- Validator at `http://validator.w3.org/`
- Example at ... DBIS Web Pages
- only the text content is shown in the browser, all other content describes *how* the text is presented.
- no logical markup of the documents (sectioning etc), but
- only optical markup (“how is it presented”).

Exercise

Design (and validate) a simple homepage in XHTML, and put it as `index.html` in your public-directory.

4.5 Miscellaneous about XML

4.5.1 Remarks

- all letters are allowed in element names and attribute names
- text (attribute values and element content) can contain nearly all characters. Western european umlauts are allowed if the XML identification contains encoding="UTF-8" or encoding="ISO-8859-1" etc.
- comments are enclosed in <!-- ... -->
- inside XML content,

  <![CDATA[...]]>

  (*character data sequences*) can be included that are not parsed by XML parsers, but which are copied character-by-character.

4.5.2 Entities

Entities serve as macros or as constants and are defined in the DTD. They are then accessible as "&*entityname*;" in the XML instance and in the DTD:

<!ENTITY *entity_name replacement_text*>

- additional special characters, e.g. ç:

  DTD: <!ENTITY ccedilla "&#231">
  XML: president="Fran&ccedilla;ois Mitterand"

- reserved characters can be included as *references to predefined entities*:
  < = < (less than), > = > (greater than)
  & = & (ampersand), *space* = , *apostroph* = ', *quote* = "
  ä = ä, ..., Ü = Ü

  <name>Düsseldorf </name>

- characters can also be given directly as character references, e.g. &#x20 (Space), &#xD (CR).

Entities (cont'd)

- global definitions that may change can be defined as constants:

  DTD: <!ENTITY server "http://www.informatik.uni-goettingen.de">
  XML: <url> &server;/dbis <url>

- macros that are needed frequently:

  DTD: <!ENTITY european
  "<encompassed continent='europe'>100</encompassed>">
  XML: <country car_code="D">
  <name >Germany</name>
  &european; ...
  </country>

- note: single and double quotes can be nested.

Parameter Entities

Entities that should be usable only in the DTD are defined as *parameter entities*:

- macros that are needed frequently:
  <!ENTITY % namedecl "name CDATA #REQUIRED">
  <!ATTLIST City
  %namedecl;
  zipcode ID #REQUIRED>

- define enumeration types:
  <!ENTITY % waters "(river|lake|sea)">
  <!ATTLIST City_located_at
  type %waters; #REQUIRED
  at IDREF #REQUIRED>

ENTITIES FROM EXTERNAL SOURCES

Entity “collections” can also be used from external sources as *external entities*:

<!ENTITY *entity_name* SYSTEM “url”>

is an entity that stands for a remote resource which itself defines a set of entities by

<!ENTITY *entity_name’ replacement_text*>

e.g. a set of technical symbols:

<!ENTITY % isotech SYSTEM
“http://www.schema.net/public-text/ISOtech.pen”>
%isotech;

the reference %isotech; makes then all symbols accessible that are defined in the external resource.

This can be iterated for defining “style files” that collect a set of external resources that are used by an author.

4.5.3 Integration of Multimedia

- for (external) non-text resources, it must be declared which program should be called for showing/processing them. This is done by NOTATION declarations:

  <!NOTATION *notation_name* SYSTEM “$program\_url$”>
  <!NOTATION postscript SYSTEM “file:/usr/bin/ghostview”>

- the entity definition is then extended by a declaration which notation should be applied on the entity:

  <!ENTITY *entity_name* SYSTEM “url”
  NDATA *notation_name*>
  <!ENTITY manual SYSTEM “file:/.../name.ps”
  NDATA postscript>

- the *application program* is then responsible for evaluating the entity and the NDATA definition.

- XLink will later present another mechanism for referencing resources.

4.6 Summary and Outlook

XML: "basic version" consists of DTD and XML documents

- tree with additional cross references
- hierarchy of nested elements
- order of the subelements
  – documents: 1st, 2nd, ... section etc.
  – databases: order in general not relevant
- attributes
- references via IDREF/IDREFS
  - documents: mainly cross references
  - databases: part of the data (relationships)
- XML model similar to the network data model:
  relationships are mapped into the structure of the data model
  - the basic explicit, stepwise navigation commands of the network data model have an equivalent for XML in the DOM-API (see later), but
  - XML also provides a declarative, high-level, set-oriented language.

Requirements

- Documents: logical markup (Sectioning etc.)
  presentation on Web pages in (X)HTML? – transformation languages
- databases: structuring of data;
  several equivalent alternatives
  query languages?
  presentation on Web pages in (X)HTML? – transformation languages
- application-specific formats
  e.g. XHTML: browsing
  DTDs are induced by the application-programs
  Web-Services: WSDL, UDDI; CAD; ontology languages; ...
  transformation between different XML languages
  application-programs must "understand" XML internally

Further Concepts of the XML World

Extensions:

- namespaces: use of different DTDs in a database
  (see Slide 223)
- APIs: DOM, SAX
- theoretical foundations
- query languages: XPath, XML-QL, Quilt, XQuery
- stylesheets/transformation languages: CSS, DSSSL, XSL
- better schema language: XML Schema
- XML with inter-document handling: XPointer, XLink

4.7 Recall

- XML as an abstract *data model*
  - cf. relational DM
  - XML now has become less abstract: creation of instances in the editor, validating, viewing ...
- a data model needs ... implementation? theory?
- ... first, something else: *abstract datatype, interface(s)*
  - constructors, modificators, selectors, predicates (cf. Info I)
- here: “two-level model”
  - as an ADT (programming interface): Document Object Model (DOM):
    detailed operations as usual in programming languages (Java, C++).
  - as a database model (end user interface; declarative):
    import (parser), *queries*, updates
- theory: formal specification of the semantics of the languages, other issues are the same as in classical DB theory (transactions etc.).