Standards for language encoding: XML

Tomaž Erjavec
Dept. of Knowledge Technologies
Jožef Stefan Institute

ESSLLI 2011

Overview of the lecture

1. History – SGML and HTML
2. XML
3. XML Schemas
4. Selection, Trasformations and Querying
5. Stand-off markup

Language data

- How to define a language for the representation of texts that will be processed by computer programs?

- text editors: very loose encoding, too oriented to the visual appearance of text
- databases: too rigid encoding, does not allow for mixture of content (text) and structure (markup)
- ISO 8879 SGML (Standard Generalised Markup Language), 1986
  - defined a language for the representation of texts that will be processed by computer programs

SGML

it defined an encoding which is:

- very general, as it is a “metalanguage” (a language for describing other languages) and lets you design your own customised markup languages for different types of documents
- interchangeable between computer platforms
- resistant to changes in technology
- enables the use of documents for various purposes
- enables automatic validation whether a certain document is compliant with the standard

Problems with SGML

- the standard was very complex
- software for using it was either very expensive or used only in academia
- the conversion of existing documents into SGML was expensive
- so, the use of SGML was limited

The Web

- HTML was an application of SGML
- but SGML compliant HTML is used by very few web pages..
- HTML is also not expressive enough for the encoding of arbitrary web data
- the need for a new standard for encoding web data that would have all the advantages of SGML without its weaknesess:
- **eXtended Markup Language, XML** (1998)

XML now

- XML became very popular, and is has become the universal medium for interchange of (language) data
- many related standards
- many freely available tools for processing XML
- many programs support import and export of XML data

What is XML?

- XML is a definition of device-independent, system-independent methods of storing and processing texts in electronic form
- XML is a project of W3C; hence, it is an open and non-proprietary specification
- XML is a subset of SGML
- XML is a “metalanguage” -- a language for describing other languages -- which lets you design your own customised markup languages for different types of documents

W3C

- The World Wide Web Consortium
- first recommendation was HTML (1992)
- best known versions of HTML: 3.2, 4.1
- XML 1.0 released February 1998
- Many XML related standards:
  - DOM Level 1 V1.0 (October 1998)
  - XML Namespaces V1.0 (January 1999)
  - XPath V1.0 (November 1999)
  - XSLT V1.0 (November 1999)
  - XHTML V1.0 (January 2000)
  - XML Schema V1.0 (May 2001)
  - XLink V1.0 (June 2001)
  - XPointer V1.0 (September 2001)
  - XSL V1.0 (October 2001)
  - XML Information Set V1.0 (October 2001)
  - XPath 2.0 WD (April 2002)

What is a Markup Language?

- *markup* (equivalently, *encoding*)
  - making explicit an interpretation of text
- *markup language*
  - a set of markup conventions used together for encoding texts.
- A markup language must specify:
  - how markup is to be distinguished from text,
  - what the markup means,
  - what markup is allowed,
  - what markup is required.

Structure of XML documents

```
<poem>
  <title>The SICK ROSE</title>
  <stanza>
    <line>O Rose thou art sick.</line>
    <line>The invisible worm,</line>
    <line>That flies in the night</line>
    <line>In the howling storm:</line>
  </stanza>
  <stanza>
    <line>Has found out thy bed</line>
    <line>Of crimson joy:</line>
    <line>And his dark secret love</line>
    <line>Does thy life destroy.</line>
  </stanza>
</poem>
```

- document = text + mark-up
- element = start tag + content + end tag
- generic identifier = name of the tag
- element contains text or elements or both (or nothing)

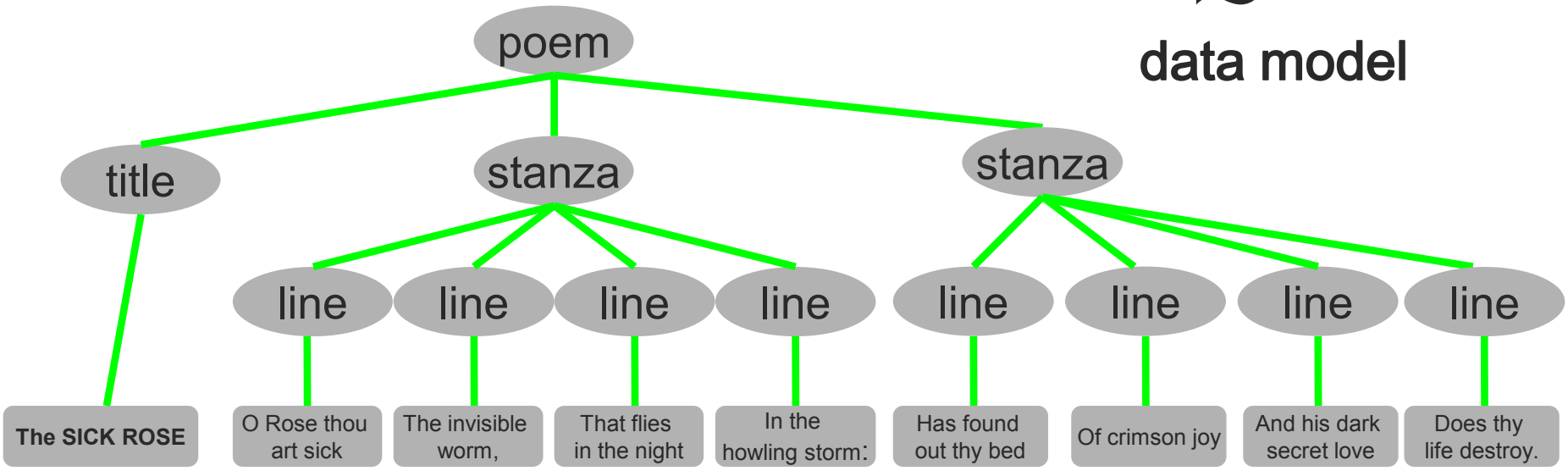
XML data model

<poem><title>The SICK ROSE</title> <stanza><line>O Rose thou art sick.</line> <line>The invisible worm,</line> <line>That flies in the night</line> <line>In the howling storm:</line></stanza> <stanza><line>Has found out thy bed</line> <line>Of crimson joy:</line> <line>And his dark secret love</line> <line>Does thy life destroy.</line></stanza></poem>

serialization

~

data model



Empty elements

- elements with content:
  <tag> … </tag>
- empty elements have no content:
  <tag/>
- used for indicating “points” in the document, for example page breaks
- formally
  <tag/> = <tag></tag>

Attributes

Attributes are used to describe properties of elements

Example:

```
<table id="P1" status='revised'> ... </table>
```

- given as *attribute-value pairs* inside the start-tag
- value must be inside matching quotation marks, single or double;
- order in which attribute-value pairs are supplied inside a tag has no significance;

Comments

- Comments can appear anywhere in text (but not in markup)
- Comments start with <!-- and end with -->
- e.g.

```
<poem>
  <title>The SLICK <!-- is this an typo? --> ROSE</title>
  <stanza>
    <line>O Rose thou art sick.</line>
    <!-- some lines missing -->
  </stanza>
  <!-- here comes the second stanza -->
</poem>
```

- Note that in XML 'meta-markup' starts with <! or <?

Example: annotated corpus

```
<s id="Osl.1.2.2.1">
  <w lemma="biti" ana="Vcps-sma">Bil</w>
  <w lemma="biti" ana="Vcip3s--n">je</w>
  <w lemma="jasen" ana="Afpmsnn">jasen</w>
  <c>,</c>
  <w lemma="mrzel" ana="Afpmsnn">mrzel</w>
  <w lemma="aprilski" ana="Aopmsn">aprilski</w>
  <w lemma="dan" ana="Ncmsn">dan</w>
  <w lemma="in" ana="Ccs">in</w>
  <w lemma="ura" ana="Ncfpn">ure</w>
  <w lemma="biti" ana="Vcip3p--n">so</w>
  <w lemma="biti" ana="Vmps-pfa">bile</w>
  <w lemma="trinajst" ana="Mcnpnl">trinajst</w>
  <c>.</c>
</s>
```

Example: dictionary

```
- <entry id="jaslo.4509">
  - <form type="hw">
      <orth type="roma">shuurisuru</orth>
      <orth type="kana">しゅうりする</orth>
      <orth type="kanji">修理する</orth>
    </form>
  - <gramGrp>
      <pos>Vs</pos>
      <subc>trans.</subc>
    </gramGrp>
  - <trans>
      <tr>popraviti</tr>
    </trans>
  - <eg>
      <q>ラジオがこわれたので修理した。</q>
      <tr>Ker se je radio pokvaril, sem ga popravil.</tr>
    </eg>
  - <eg>
      <q>そろそろ屋根（やね）を修理してもらわなければならない。</q>
      <tr>Počasi bomo morali dati popraviti streho.</tr>
    </eg>
  - <xr type="lesson" n="L1.7">
      <xref>1. letnik, lekcija 7</xref>
    </xr>
    <usg type="level">0</usg>
    <note type="admin" resp="TER">2005-07-11 Add Romaji</note>
    <note type="admin" resp="TER">2005-07-10 Add levels</note>
    <note type="admin" resp="KHS">2003-03-12 L1 (642)</note>
    <note type="admin" resp="VOJ">2005-02-22 V (342)</note>
    <note type="admin" resp="ISE">2005-02-28 Merge</note>
</entry>
```

Entities

- XML documents can also contain entity references, which are, when processing the document, substituted by their interpretation (the entity)
- an entity reference starts with the character ampersand and ends with the semicolon: &…;
- character references: a number (character code) is substituted by its Unicode character, e.g. Ђ → Ђ
- a few „proper“ entities are predefined in XML:
  < → < > → >
  & → &
  ' → ' " → "
- < and & are “magic” characters and must always be escaped when using them in the text:
  - 1 < 2 must be written as 1 < 2
  - Procter & Gamble must be written as Procter & Gamble
- entities can also be used for other purposes

XML declaration

Every XML document must begin with an XML declaration which does two things:

- specifies that this *is* an XML document,
- version of the XML standard used:
  - XML 1.0: 1998, (<ascii>)
  - XML 1.1: 2006 (<čačka>)
- character encoding of the document:
  - <?xml version="1.0" encoding="iso-8859-1" ?>
  - <?xml version="1.0" ?>
    default, and recommended, encoding is UTF-8

Minimal requirements

- the document starts with the XML declaration
- tags and entities are correctly written
  Wrong: 1 &lt 2
- the document must be a tree:
  - every start tag has a matching end-tag
    (<name> ≠ <Name> ≠ <NAME>)
  - elements are correctly nested
    Wrong: <a>………
  - the document has a single top-level element

→ This is then a well-formed XML document

Splot the mistake

<greeting>Hello world!</greeting>
<greeting>Hello world!</Greeting>

<greeting><grunt>Ho</grunt> world!</greeting>
<grunt>Ho <greeting>world!</greeting></grunt>
<greeting><grunt>Ho world!</greeting></grunt>

<grunt type=loud>Ho</grunt>
<grunt type="loud"></grunt>

<grunt type= "loud">
<grunt type ="loud"/>

Another bad XML document

```
<HTML>
<HEAD><TITLE>Links</TITLE></HEAD>
<BODY>
<H1 align=center>Interesting<BR>WWW links</H1>
<UL>
<LI><A HREF="http://www.w3.org/XML">W3C XML</A>
<LI><A HREF="http://xml.coverpages.org/">Cover's pages</A>
</UL>
<FORM action="http://www.google.com/search" method=get>
<A href="http://www.google.com/">Google</A>
<input type=text name=q size=28 maxlength=256>
<input type=hidden name=meta value="lr=&hl=en">
</FORM>
</BODY>
</HTML>
```

Exercise: mark-up a recipe

- Have a look at http://nl.ijs.si/et/teach/esslli11/mikuni.htm
- Copy to your computer the „equivalent“ XML document http://nl.ijs.si/et/teach/esslli11/mikuni.xml
- Now open your local XML file with some simple text editor (e.g. Wordpad)
- Also open it in your Web browser
- Now start marking up XML elements; use your best judgement on how to name and nest elements
  - Don‘t worry too much about the tag names, the point of the excercise is not to get some „perfect“ element set, but just to give you a feeling for XML syntax and markup
- Every once in a while check if it is still well-formed by reloading it in your browser

Some possible insights

- When encoding a textual source in XML you will almost invariably loose information
  - you have to be realistic: what is important for you?
- The XML element names express meaning, not visual apperance
- There is more than one way to skin a cat – the envisioned use of the resources (and the time/budget!) influences what to encode and how

Defining the rules

- A valid XML document conforms to rules which are stated in an (external) schema (“element grammar”) of some sort.
- A schema specifies:
  - names for all elements used
  - names and datatypes and (occasionally) default values for their attributes
  - rules about how elements can nest
  - and a few other things, depending on the schema language
- n.b. A schema does *not* specify anything about what elements mean – this is the job of the documentation!

In XML a schema is optional

- XML allows you to make up your own tags, and doesn't *require* a schema...
- The XML concept is dangerously powerful:
  - XML elements are light in semantics
  - one man's <p> is another's <para> (or is it?)
  - the appearance of interchangeability may be worse than its absence
- But XML is too good to ignore
  - mainstream software development
  - proliferation of tools
  - the language of the web

What can a schema do for you?

- ensure that your documents use only predefined elements, attributes, and entities
- enforce structural rules such as ‘every chapter must begin with a heading’ or ‘recipes must include an ingredient list’
- make sure that the same thing is always called by the same name
- schema languages vary in the amount of validation they support

Schema languages

- Schemas can be written in:
  - XML DTD Language (inherited from SGML)
  - The W3C schema language (main successor of DTDs)
  - The ISO Relax NG schema language (used by TEI)

A simple DTD

XML document:

```
<city>
    <name>Graz</name>
    <inhabitants>285,470</inhabitants>
    <country>Austria</country>
</city>
```

DTD:

```
<!ELEMENT city (name, inhabitants, country)>
<!ELEMENT name          (#PCDATA)>
<!ELEMENT inhabitants   (#PCDATA)>
<!ELEMENT country       (#PCDATA)>
```

A more complex DTD

```
<!ELEMENT anthology (poem+)>
<!ELEMENT poem (title?, stanza+)>
<!ELEMENT title (#PCDATA) >
<!ELEMENT stanza (line+) >
<!ELEMENT line (#PCDATA) >
```

An element definition gives:

- the name of the element
- its content model

```
<anthology>
 <poem>
  <title>The SICK ROSE</title>
  <stanza>
   <line>O Rose thou art sick.</line>
   <line>The invisible worm,</line>
   <line>That flies in the night</line>
   <line>In the howling storm:</line>
  </stanza>
  <stanza>
   <line>Has found out thy bed</line>
   <line>Of crimson joy:</line>
   <line>And his dark secret love</line>
   <line>Does thy life destroy.</line>
  </stanza>
 </poem>
</anthology>
```

Content Model Operators

- (open bracket for grouping
-) close bracket
- , follows
- | or
- ? maybe
- * repeated 0 or more times
- + repeated once or more times

```
<!ELEMENT poem
                    (title?,
                              (line+
                               |
                               (refrain?, (stanza, refrain?)+)
                              )
                    )
>
```

Mixed content

<title>The <hi>SICK</hi> ROSE</title>

If an element contains #PCDATA and element content, #PCDATA must always appear as the first option in an alternation; the group containing it must use the star operator; it may appear once only, and in the outermost model group.

```
<!ELEMENT Item1 (#PCDATA | para)*>          <!-- OK -->
<!ELEMENT item2 (#PCDATA | para | note)*> <!-- OK -->

<!ELEMENT item3 (#PCDATA , para)*> <!-- WRONG! -->
<!ELEMENT item4 (para | #PCDATA)*> <!-- WRONG! -->
<!ELEMENT item5 (#PCDATA | para)+> <!-- WRONG! -->
<!ELEMENT item6 (para | (#PCDATA | note)*)> <!-- WRONG! -->
```

Empty Content

Empty elements do not have content. To distinguish them from those with content in well-formed XML documents, they have a special form: the tag ends with a slash.

- In the DTD:
  <!ELEMENT pageBreak EMPTY>
- In the document:
  ... The page ends here. <pageBreak/> Here starts a new one. ...

Attributes

- In the DTD:

attribute name; type default

```
<!ATTLIST table
    type        CDATA       #IMPLIED        allowed
    id          ID          #REQUIRED       necessary
    status      (draft|
                 revised |
                 final )     "draft"         default value
>
```

- In the XML document:

```
<table id="tab.12" type= "summary" status= "revised">
```

A Complete Valid XML Document

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE anthology [
        <!ELEMENT anthology (poem+)>
        <!ELEMENT poem (title?, stanza+)>
        <!ELEMENT title (#PCDATA) >
        <!ELEMENT stanza (line+) >
        <!ELEMENT line (#PCDATA) >
]>
<anthology>
  <poem>
    <title>The SICK ROSE</title>
    <stanza>
      <line>O Rose thou art sick.</line>
      <line>The invisible worm,</line>
      <line>That flies in the night</line>
      <line>In the howling storm:</line>
    </stanza>
    <stanza>
      <line>Has found out thy bed</line>
      <line>Of crimson joy:</line>
      <line>And his dark secret love</line>
      <line>Does thy life destroy.</line>
    </stanza>
  </poem>
</anthology>
```

Exercise: make a DTD for your XML recipe

- Put the DTD directly into your XML file; open it with a browser to see if it validates

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE recipe [
    <!ELEMENT recipe (…)>
      …
]>
<recipe>
…
</recipe>
```

Standard schemas

- There are by now many standard data formats, which are expressed in XML schemas + associated documentation
- Docbook: software manuals
- SVG: Scalable vector graphics
- MathML: Mathematical Markup Language
- MusicXML: Music Notation

DocBook example

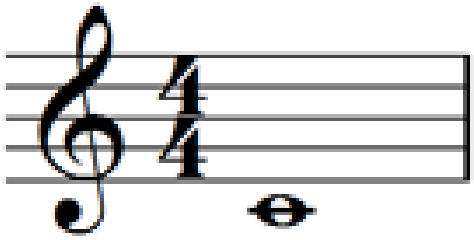
```
<?xml version="1.0" encoding="UTF-8"?>
<book xml:id="simple_book" xmlns="http://docbook.org/ns/docbook" version="5.0">
  <title>Very simple book</title>
  <chapter xml:id="chapter_1">
    <title>Chapter 1</title>
    <para>Hello world!</para>
    <para>I hope that your day is proceeding <emphasis>splendidly</emphasis>!</para>
  </chapter>
  <chapter xml:id="chapter_2">
    <title>Chapter 2</title>
    <para>Hello again, world!</para>
  </chapter>
</book>
```

MathML

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
<mrow>
  <mi>x</mi>
  <mo>=</mo>
  <mfrac>
    <mrow>
      <mrow>
        <mo>-</mo>
        <mi>b</mi>
      </mrow>
      <mo>&#xB1;<!--PLUS-MINUS SIGN--></mo>
      <msqrt>
        <mrow>
          <msup>
            <mi>b</mi>
            <mn>2</mn>
          </msup>
          <mo>-</mo>
          <mrow>
            <mn>4</mn>
            <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
            <mi>a</mi>
            <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
            <mi>c</mi>
          </mrow>
        </mrow>
      </msqrt>
    </mrow>
    <mrow>
      <mn>2</mn>
      <mo>&#x2062;<!--INVISIBLE TIMES--></mo>
      <mi>a</mi>
    </mrow>
  </mfrac>
</mrow>
```

MusicXML



```
<part id="P1">
  <measure number="1">
    <attributes>
      <divisions>1</divisions>
      <key>
        <fifths>0</fifths>
      </key>
      <time>
        <beats>4</beats>
        <beat-type>4</beat-type>
      </time>
      <clef>
        <sign>G</sign>
        <line>2</line>
      </clef>
    </attributes>
    <note>
      <pitch>
        <step>C</step>
        <octave>4</octave>
      </pitch>
      <duration>4</duration>
      <type>whole</type>
    </note>
  </measure>
</part>
```

XML Namespaces

- A XML document could usefully contain elements and attributes that are defined for and used by multiple software modules.
- Such documents pose problems of recognition and collision.
- Therefore document constructs should have universal names, whose scope extends beyond their containing document;
- Such universal names are defined by the XML Namespaces specification
- Namespaces make use of the notion of a Uniform Resource Identifier, (URI), which identifies a resource by meta-information of any kind; in contrast, an URL locates a resource on the net, which means if you have a URL and the appropriate protocol you can retrieve the resource.

xmlns

```
<?xml version="1.0" ?>
<html xmlns="http://www.w3.org/HTML/1998/html4"
      xmlns:nms="http://www.names.net/address">
 <head><title>Addresses</title></head>
 <body xml:lang="en">
   <nms:addresses nms:version="1.0">
   <hr/>
   <nms:person>
     <nms:title>Mr.</nms:title>
     <nms:first>Simon</nms:first>
     <nms:last>Schuster</nms:last>
   </nms:person>
   <hr/>
<!-- ... -->
 </body>
</html>
```

- Two-part naming system for element types and attributes
- The xmlns prefixed attributes give the URI and the local prefix of the namespaces
- „Qualified names“ consist of prefix, colon and local part of the name
- Note: The URI is not a URL - it does not need to refer to a DTD or to be accessible

xmlns

```
<?xml version="1.0" ?>
<html xmlns="http://www.w3.org/HTML/1998/html4"
      xmlns:nms="http://www.names.net/address">
 <head><title>Addresses</title></head>
 <body xml:lang="en">
   <nms:addresses nms:version="1.0">
   <hr/>
   <nms:person>
     <nms:title>Mr.</nms:title>
     <nms:first>Simon</nms:first>
     <nms:last>Schuster</nms:last>
   </nms:person>
   <hr/>
<!-- ... -->
 </body>
</html>
```

- The default namespace is introduced by the attribute xmlns, without a local prefix
- The prefix xml is by definition bound to the namespace name http://www.w3.org/XML/1998/namespace
- xml:id and xml:lang are predefined attributes in XML

Non-hierarchical structures

- XML has a tree-based information model
- But not all structures are trees
- The standard case is when we want to model several hierarchies (trees) over the same text, e.g.
  - document structure +
  - linguistic structure +
  - physical structure

Crossing hierarchies

<l>Scorn not the sonnet; critic, you have frowned,</l>
<l>Mindless of its just honours; with this key</l>
<l>Shakespeare unlocked his heart; the melody</l>
<l>Of this small lute gave ease to Petrarch's wound.</l>

<seg>Scorn not the sonnet;</seg>
<seg>critic, you have frowned, Mindless of its just honours;</seg>
<seg>with this key Shakespeare unlocked his heart;</seg>
<seg>the melody Of this small lute gave ease to Petrarch's wound.</seg>

The two markups cannot be simply combined
within one document

Stand-off markup

- Many ways have been suggested how to overcome this limitation
- All have associated problems:
  - more complex processing
  - more difficult validation
- For HLT the most popular mechanism is stand-off markup:
  - the annotations are not part of the document, but only point to it

Stand-off example

```
<l>
  <w xml:id="w001">Scorn</w>
  <w xml:id="w002">not</w>
  <w xml:id="w003">the</w>
  <w xml:id="w004">sonnet</w>;
  <w xml:id="w005">critic</w>,
  <w xml:id="w006">you</w>
  <w xml:id="w007">have</w>
  <w xml:id="w008">frowned</w>,
</l>
…
<!-- elsewhere in the current document -->
<seg><xi:include xpointer="range(element(w001),element(w004))"/></seg>
<seg><xi:include xpointer="range(element(w005),element(w013))"/></seg>
```

XML related recommendations

- XML is a good development, storage and interchange format – but what can you *do* with it?
  - Transform: into HTML, PDF, DB, …
  - Search: find information in XML docs
- How do you do this?
  - Using XML related recommendations
  - XPath, XSLT, XQuery

XPath

- The primary purpose of XPath is to *address parts of an XML document*;
- XPath uses a compact, non-XML syntax similar to the *path notation* in URLs
- XPath models an XML document as a *tree of nodes*
- Expression evaluation occurs with respect to its *context node*

XPath examples

- para selects all „para“element children of the context node
- * selects all element children of the context node
- @name selects the attribute „name“ of the context node
- @* selects all the attributes of the context node
- para[1] selects the first „para“ child of the context node
- . selects the context node
- .//para selects all „para“ grandchildren of the context node
- /doc/chapter[2]/section[1]
  selects the 1st section of the 2th chapter of the doc child of the root node

XPath cont.

- You can also select the parent, ancestor, sibling, etc. nodes, e.g.
  ancestor::section
  following-sibling::para[1]
- Other constrains:
  para[@id], para[hi], para[ancestor::section]
- XPath functions:
  substring-after(para,‘Author: ‘)
  fn:replace(para,‘x*‘,‘y‘) //XPath 2.0//

XSLT

- A language in which to write transformations (stylesheets) for XML document
- XSLT stylesheets are written in XML
- Several free XSLT processors exist (e.g. xsltproc, saxon)
- Output is XML, HTML or text
- Again, takes the XML document as a tree
- Uses XPath to select nodes to process

XSLT example

```
<?xml version="1.0"?>
<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
.
.
</catalog>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
  <html>
    <body>
      <h2>My CD Collection</h2>
        <table border="1">
          <xsl:for-each select="catalog/cd">
            <tr>
              <td><xsl:value-of select="title"/></td>
              <td><xsl:value-of select="artist"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

XQuery

- For searching in XML documents
- Somewhat similar to SQL
- Uses XPath
- Used by native XML databases, e.g. eXist

Starting with XML

- all browsers show XML
- editing can be done in plain text editor, but an XML editor is better (Oxygen)
- validation to schema can be done in XML editors, by browsers (to an extent) or by stand-alone programs
- for XSLT processing, use Saxon (or via an XML editor)
- many many tutorials on the Web!