2 Basics of XML and XML documents

Survivor’s Guide to XML, or XML for Computer Scientists / Dummies

2.1 XML and XML documents
2.2 Basics of XML DTDs
2.3 XML Namespaces

2.1 XML and XML documents

  - not an official standard, but a stable industry standard
  > editorial revisions, not new versions of XML 1.0
  - what is said later about valid XML documents applies to SGML documents, too

2.2 Basics of XML

- Extensible Markup Language is not a markup language!
  - does not fix a tag set nor its semantics
  (unlike markup languages, e.g. HTML)
- XML documents have no inherent (processing or presentation) semantics
  - even though many think that XML is semantic or self-describing; See next

2.3 XML Namespaces

What is XML?

- Extensible Markup Language is not a markup language!
  - does not fix a tag set nor its semantics
  (unlike markup languages, e.g. HTML)
- XML documents have no inherent (processing or presentation) semantics
  - even though many think that XML is semantic or self-describing; See next

What is XML (2)?

- XML is
  - a way to use markup to represent information
- a metalanguage
  - supports definition of specific markup languages through XML DTDs or Schemas
- Often “XML” = XML + XML technology
  - that is, processing models and languages we’re studying (and many others ...)

Semantics of XML Markup

- Meaning of this XML fragment?

  `<student>`
  `<番号>文字データ</番号>`
  `<資格>文字データ</資格>`
  `</student>`

  - The computer cannot know it either!
  - Implementing the semantics is the topic of this course

How does it look?

```xml
<?xml version='1.0' encoding="iso-8859-1" ?>
<invoice num="1234">
  <client clNum="00-01">
    <name>Pekka Kilpeläinen</name>
    <email>kilpelai@cs.uku.fi</email>
  </client>
  <item price="60" unit="EUR">
    XML Handbook
  </item>
  <item price="350" unit="FIM">
    XSLT Programmer's Ref
  </item>
</invoice>
```

Essential Features of XML

- Overview of XML essentials
  - many details skipped
  > some to be discussed in exercises or with other topics when the need arises
  > Learn to consult original sources (specifications, documentation etc) for details!
  > The XML specification is easy to browse

XML Document Characters

- XML documents are made of ISO-10646 (32-bit) characters; in practice of their 16-bit Unicode subset (used, e.g., in Java)
- Three aspects or characters:
  - identification as numeric code points
  - representation by bytes
  - visual presentation
External Aspects of Characters

- Documents are stored/transmitted as a sequence of bytes (or octets). An encoding determines how characters are represented by bytes.
  - UTF-8 (≥ 7-bit ASCII) is the XML default encoding
  - encoding="iso-8859-1"
    → 256 Western European chars as single bytes
- A font (collection of character images called glyphs) determines the visual presentation of characters

XML Encoding of Structure 1

- XML is, essentially, just a textual encoding scheme of labelled, ordered and attributed trees, in which
  - internal nodes are elements labelled by type names
  - leaves are text nodes labelled by string values, or empty element nodes
  - the left-to-right order of children of a node matters
  - element nodes may carry attributes (= name-string-value pairs)

XML Encoding of Structure 2

- XML encoding of a tree
  - corresponds to a pre-order walk
  - start of an element node with type name A
    denoted by a start tag <A>, and its end denoted by end tag </A>
  - possible attributes written within the start tag:
    <A attr="value1" ... attr="valueN"> names must be unique: attr i ≠ attr j, when k ≠ h
  - text nodes written as their string value

XML: Logical Document Structure

- Elements
  - indicated by matching (case-sensitive!) tags
    <ElementTypeName> ... </ElementTypeName>
  - can contain text and/or subelements
  - can be empty:
    <elem-type/> or</elem-type/> (e.g. <br/> in XHTML)
  - unique root element --> document a single tree

Logical document structure (2)

- Attributes
  - name-value pairs attached to elements
    - "metadata", usually not treated as content
    - in start-tag after the element type name
      <div class="preface" date='990126'> ...
  - Also:
    - <!-- comments outside other markup -->
    - <note this would be passed to the application as a processing instruction
      named 'note'?>

CDATA Sections

- "CDATA Sections" to include XML markup characters as textual content
  <![CDATA[
  Here we could include for example code fragments:
    <example>< if (Count < 5 & & Count > 0)
      </example>
  ]]>
2. 2 Basics of XML DTDs

- A Document Type Declaration provides a grammar (document type definition, DTD) for a class of documents [Defined in XML Rec]
- Syntax (in the prolog of a document instance):
  ```xml
  <!DOCTYPE rootElement SYSTEM "ex.dtd" [ <!-- "external subset" in file ex.dtd --> | <!-- "internal subset" may come here --> ]>
  ```
- DTD is union of the external and internal subset; internal has higher precedence
  - can override entity and attribute declarations (see next)

How do Declarations Look Like?

```xml
<!ELEMENT invoice (client, item+)>  
<!ATTLIST invoice num NMTOKEN #REQUIRED>  
<!ATTLIST client num NMTOKEN #IMPLIED>  
<!ELEMENT name (#PCDATA)>  
<!ATTLIST client name (#PCDATA)>  
<!ELEMENT invoice (client, item+)>
```

XML Recommendation specifies, partly, the behaviour
- relieves the application from details of markup
- XML Recommendation specifies, partly, the behaviour
- how to check the correctness of documents;
- information about external (binary) objects

Element type declarations

- The general form is
  ```xml
  <!ELEMENT elemType (E)>  
  ```
  where E is a content model (regular expr.);
  - ECFG production
  - similar to productions of ECFGs
- Content model operators:
  - E | F: alternation
  - E*: optional
  - E+: one or more
  - E: grouping
- No priorities: either (A,B)|C or A,(B|C), but no A,B|C

Attribute-List Declarations

- Can declare attributes for elements:
  - Name, data type and possible default value:
    ```xml
    <!ATTLIST F elemType (E)>  
    ```
  - may contain text (#PCDATA) and elements
- Empty content
  ```xml
  <!ELEMENT IMG EMPTY>  
  ```
- Arbitrary content:
  ```xml
  <!ELEMENT HTML ANY>  
  ```
  (= <ELEMENT HTML (#PCDATA | choice-of-all-declared-element-types) *>)

Mixed, Empty and Arbitrary Content
Entities (1)

- Named storage units or XML fragments (macros in some languages)
  - character entities:
    `&lt;`, `&amp;`, `&gt;` and `&apos;` (expanding to `<`, `&`, `>`, `apos;`
  - predefined entities:
    `&amp;`, `&apos;`, `&quot;` and `&gt;`
  - general entities are shorthand notations:
    `<ENTITY chap1 SYSTEM "http://myweb/ch1"">` `</ENTITY chap1>`

Unparsed Entities

- For connecting external binary objects to XML documents; (XML processor handles only text)
  `<!ENTITY fig123 SYSTEM "figs/f123.tif" NDATA TIFF>
  <!ATTLIST IMG file ENTITY REQUIRED>`

Parameter Entities

- to parameterize and modularize DTDs:
  `<!ENTITY % tableDTD SYSTEM "dtds/tab.dtd">` `tableDTD; <!-- include sub-ddt -->` `<!ENTITY % table.dtd SYSTEM "dtds/tab.dtd" %table.dtd; -->` `<!ATTLIST CHAP % %stAttr stAttr; >` `<!ATTLIST SECT % %stAttr stAttr; >` `<!ENTITY chap1 SYSTEM "http://myweb/ch1">` `<!ENTITY chap2 SYSTEM "http://myweb/ch2">` `<!ENTITY chap3 SYSTEM "http://myweb/ch3">` `<!DOCTYPE doc [<!ENTITY UKU "University of Kuopio">]>`

Speculations about XML Parsing

- Parsing involves two things:
  1. Pulling the entities together, and checking the syntactic correctness of the input
  2. Building a parse tree for the input (a’la DOM), or otherwise informing the application about document content and structure (e.g. a’la SAX)
- Task 2 is simple (like simplicity of XML markup; see next)
- Checking well-formedness is straightforward; implementing validation is a bit more challenging

Building an XML Parse Tree

- Uses DOM like tree operations
  `C ← new Document(); // current node
  Scan document text:
  case:
  "&lt;Type&gt;": N ← new ElementNode(Type);
  C.addChildNode(N);
  C ← N;
  "&lt;Txt&gt;": C.addChildNode(new TextNode(Txt));
  "&lt;/Type&gt;": C ← C.parentNode();
  return C;`
Sketching XML validation

- Treat the document as a tree $d$
- Document is valid w.r.t. a grammar (DTD/Schema) $G$ iff $d$ is a Syntax tree over $G$
  - Check that the root is labelled by the start symbol of $G$
  - For each element node $n$ of the tree, check that its
    - attributes match attributes declared for its type
    - content matches the content model of its type. That is, if $n$ is
      of type $A$ and its children of type $B_1, \ldots, B_n$, check that the
      grammar has a production $A \rightarrow E$ for which
      \[ B_1 \ldots B_n \in L(E) \quad (1) \]

2.3 XML Namespaces

- Document often comprise parts processed by different applications (and/or defined in different grammars)
  - for example, in XSLT scripts:
    \[
    \begin{XML}
    \begin{element}
    \end{element}
    \end{XML}
    \]
  - How to manage multiple sets of names?

XML Namespaces briefly (3/5)

- Namespace identified by a URI (through the associated local prefix)
  - e.g. $http://www.w3.org/1999/XSL/Transform$ for XSLT
    - conventional but not required to use URLs
    - the identifying URI has to be unique, but it does not
      have to be an existing address
- Association inherited to sub-elements
  - see the next example (of an XSLT script)

XML Namespaces briefly (5/5)

- Mechanism built on top of basic XML
  - overloads attribute syntax ($\text{xmlns:}$) to introduce namespaces
  - does not affect validation
    - namespace attributes have to be declared for DTD validity
    - all element type names have to be declared (with their initial prefixes?)
  - Other schema languages (XML Schema, Relax NG) better for validating documents with Namespaces