1 Document Instances and Grammars

Fundamentals of hierarchical document structures, or
Computer Scientist's view of XML

1.1 XML and XML documents
1.2 Basics of document grammars
1.3 Basics of XML DTDs
1.4 XML Namespaces

2.1 XML and XML documents

- XML - Extensible Markup Language, W3C Recommendation, February 1998
  - not an official standard, but a stable industry standard
  - 2nd Ed 10/2000, 3rd Ed 2/2004
    - » editorial revisions, not new versions of XML 1.0
- a simplified subset of SGML, Standard Generalized Markup Language, ISO 8879:1987
  - what is said later about *valid* XML documents applies to SGML documents, too

What is XML?

- Extensible *Markup Language* is not a markup language!
  - does not fix a tag set nor its semantics (like markup languages like HTML do)
- XML documents have **no inherent** (processing or presentation) **semantics**
  - Implementing those semantics is the topic of this course!

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 (Document Type Definitions) or Schemas
    - » E.g. XHTML a reformulation of HTML using XML
- Often "XML" ≈ XML + XML technology
  - that is, processing models and languages we're studying (and many others ...)

How does it look?

```
<?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
  - Learn to consult original sources (specifications, documentation etc) for details!
    - » The XML specification is easy to browse
- First of all, XML is a textual or character-based way to represent data

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)
  - Unicode 2.0 defines almost 39,000 distinct characters
- Characters have three different aspects:
  - their identification as numeric code points
  - their **representation** by bytes
  - their **visual presentation**

External Aspects of Characters

- Documents are stored/transmitted as a sequence of bytes (of 8 bits). An **encoding** determines how characters are represented by bytes.
  - UTF-8 (≈7-bit ASCII) is the XML default encoding
  - `encoding="KOI8R"` should be OK for Cyrillic texts
    - » (but I cannot comment on parser support)
- A **font** (collection of character images called **glyphs**) determines the visual presentation of characters

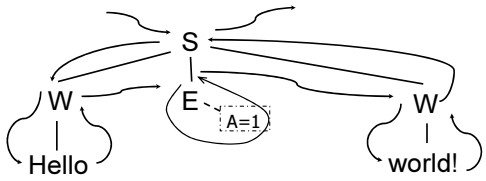
XML Encoding of Structure 1

- XML is, essentially, a textual encoding scheme of **labelled**, **ordered** and **attributed trees**:
  - 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)
- This view is shared by several XML techniques (DOM, XPath, XSLT, XQuery, ...)

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 $attr_1$="$value_1$" … $attr_n$="$value_n$">
    - » names must be unique: $attr_k \neq attr_h$ when $k \neq h$
  - text nodes written as their string value

XML Encoding of Structure: Example



<S><W> Hello</W> <E A='1'></E> <W> world! </W> </S>

XML: Logical Document Structure

- **Elements**
  - indicated by matching (case-sensitive!) tags `<ElementTypeName> … </ElementTypeName>`
  - can contain text and/or subelements
  - can be **empty**:
    `<elem-type></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
  - in start-tag after the element type name
    `<div class="preface" date='990126'>` …
  - forms "…" and '…' are interchangeable
- Also:
  - `<!-- **comments** outside other markup -->`
  - `<?note this would be passed to the application as a **processing instruction** named 'note'?>`

CDATA Sections

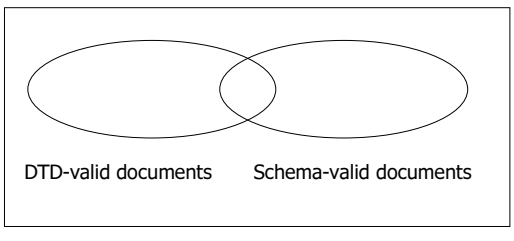
- "CDATA Sections" to include XML markup characters as textual content

```
<![CDATA[
  Here we can easily include markup
  characters and, for example, code
  fragments:
  <example>if (Count < 5 && Count > 0)
  </example>
]]>
```


Two levels of correctness (1)

- **Well-formed** documents
  - roughly: follows the syntax of XML, markup correct (elements properly nested, tag names match, attributes of an element have unique names, ...)
  - violation is a fatal error
- **Valid** documents
  - (in addition to being well-formed) obey an associated grammar (DTD/Schema)

XML docs and valid XML docs



XML documents = well-formed XML documents

An XML Processor (Parser)

- Reads XML documents and reports their contents to an application
  - relieves the application from details of markup
  - XML Recommendation specifies:
  - recognition of characters as markup or data; what information to pass to applications; how to check the correctness of documents;
  - validation based on comparing document against its grammar

Next: Basics of document grammars

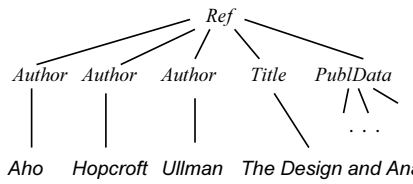
1.2 Basics of document grammars

- DTDs are variations of **context-free grammars** (CFGs), which are widely used to syntax specification (programming languages, XML, …) and to parser/compiler generation (e.g. YACC/GNU Bison)
  - No knowledge of them is necessary, but connections with CFGs may be informative for those that know about them

DTD/CFG Correspondence

DTD	CFG
XML document	parse/syntax tree
element type	nonterminal
element type declaration	production
`#PCDATA`	terminal

Example: Three Authors of a Ref



Ref –> Author Title PublData* ∈ P,
Author Author Author Title PublData ∈ L(*Author* Title PublData*)

Extended Productions

- Notice the **regular expressions** in productions
  - to describe (potentially infinite) sequences
- That is, we are using **extended** CFGs
  - content models (of a DTD) correspond to regular expressions (in an ECFG production)

1.3 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):

```
<!DOCTYPE rootElemType SYSTEM "ex.dtd"
<!-- "external subset" in file ex.dtd -->
[ <!-- "internal subset" may come here -->
]>
```

- DTD is the union of the external and internal subset

Markup Declarations

- DTD consists of **markup declarations**
  - **element type declarations**
    - » similar to productions of ECFGs
  - **attribute-list declarations**
    - » for declared element types
  - **entity declarations** (see later)
  - **notation declarations**
    - » to pass information about external (binary) objects to the application

How do Declarations Look Like?

```
<!ELEMENT invoice (client, item+)>
<!ATTLIST invoice num NMTOKEN #REQUIRED>
<!ELEMENT client (name, email?)>
<!ATTLIST client num NMTOKEN #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT item (#PCDATA)>
<!ATTLIST item
        price     NMTOKEN #REQUIRED
        unit      (FIM | EUR) "EUR" >
```

Element Type Declarations

- General form:
  `<!ELEMENT` elementTypeName `(`*E*`)>`
  where *E* is a **content model**
- ≈ regular expression of element names
- Content model operators:
  E | F : choice E, F: concatenation
  E? : optional E* : zero or more
  E+ : one or more (E) : grouping
- Must group: (A,B)|C or A,(B|C), but A,B|C forbidden

Attribute-List Declarations

- Can declare attributes for elements:
  - Name, data type and possible default value
- Example:

```
<!ATTLIST FIG
     id    ID    #IMPLIED
     descr CDATA #REQUIRED
     class (a | b | c)    "a">
```

- Semantics mainly up to the application
  - processor checks that `ID` attributes are unique and that targets of `IDREF` attributes exist

Mixed, Empty and Arbitrary Content

- **Mixed content**:

```
<!ELEMENT P     (#PCDATA | I | IMG)*>
```

  - may contain text (`#PCDATA`) and elements
- **Empty content**:

```
<!ELEMENT IMG         EMPTY>
```

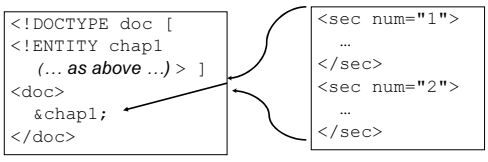
- Unrestricted content: `<!ELEMENT HTML ANY>`
  (= `<!ELEMENT HTML (#PCDATA |` *choice-of-all-declared-element-types*`)*>`)

Entities (1)

- Named storage units or fragments of XML documents (~ macros in some languages)
- Multiple uses:
  - **character entities**:
    - `<` `&#60;` and `&#x3C;` all expand to '<' (treated as data, not as start-of-markup)
    - other **predefined entities**: `&` `>` `&apos;` `&quote;` expand to `&` `>` `'` and `"`
  - **general entities** are shorthand notations:
    `<!ENTITY UKU  "University of Kuopio">`

Entities (2)

- physical storage units comprising a document
  - **parsed entities**
    `<!ENTITY chap1 SYSTEM "http://myweb/ch1">`
  - **document entity** is the starting point of processing
  - entities and elements must nest properly:

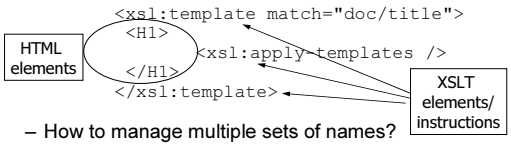


Unparsed Entities and Parameter Entities

- Unparsed entities allow XML documents refer to external binary objects like graphics files
  - XML processor handles only text
  - I've rarely used these
- Parameter entities are used in DTDs
  - useful for modularizing declarations
- We skip these

1.4 XML Namespaces

- Documents often comprise parts processed by different applications (and/or defined by different grammars)
  - for example, in XSLT scripts:



  - How to manage multiple sets of names?

XML Namespaces (2/5)

- Solution: XML Namespaces, W3C Rec. 14/1/1999 for separating possibly overlapping "vocabularies" (sets of element type and attribute names) within a single document
- by introducing (arbitrary) local name **prefixes**, and binding them to (fixed) globally unique URIs
  - For example, the local prefix "`xsl:`" conventionally used in XSLT scripts

XML Namespaces briefly (3/5)

- Namespace identified by a URI (through the associated local prexif)
  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 (4/5)

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">

  <!-- XHTML is the 'default namespace' -->
     <xsl:template match="doc/title">
      <H1>
          <xsl:apply-templates />
      </H1>
     </xsl:template>

</xsl:stylesheet>
```


XML Namespaces briefly (5/5)

- Mechanism built on top of basic XML
  - overloads attribute syntax (`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