XML Data Management

Peter Wood

BBK

Outline

Chapter 1

Introduction

What is XML?

- The eXtensible Markup Language (XML) defines a generic syntax used to mark up data with simple, human-readable tags
- Has been standardized by the World Wide Web Consortium (W3C) as a format for computer documents
- Is flexible enough to be customized for domains as diverse as:
  - Web sites
  - Electronic data interchange
  - News feeds (RSS, e.g., BBC World News)
  - Vector graphics
  - Mathematical expressions
  - Microsoft Word documents
  - Music libraries (e.g., iTunes)
  - ...

What is XML? (2)

- Data in XML documents is represented as strings of text
- This data is surrounded by text markup, in the form of *tags*, that describes the data
- A particular unit of data and markup is called an *element*
- XML specifies the exact syntax of how elements are delimited by tags, what a tag looks like, what names are acceptable, and so on

Which is Easier to Understand?

```
TCP/IP
Stevens
Foundations of Databases
Abiteboul
Hull
Vianu
The C Programming Language
Kernighan
Ritchie
Prentice Hall
...
```

```
<bib>
  <book>
    <title>TCP/IP</title>
    <author>Stevens</author>
  </book>
  <book>
    <title> ... </title>
    ...
  </book>
</bib>
```

XML vs. HTML

- Markup in an XML document looks similar to that in an HTML document
- However, there are some crucial differences:
  - XML is a meta-markup language: it doesn't have a *fixed* set of tags and elements
  - To enhance interoperability, people may agree to use only certain tags (as defined in a DTD or an XML Schema — see later)
  - Although XML is flexible in regard to elements that are allowed, it is strict in many other respects (e.g., closing tags are required)
  - Markup in XML only describes a document's structure; it doesn't say anything about how to display it

Very Brief Review of HTML

- A document structure and hypertext specification language
- Specified by the World Wide Web Consortium (W3C)
- Designed to specify the *logical structure* of information
- Intended for presentation as *Web pages*
- Text is marked up with *tags* defining the document's logical units, e.g.
  - title
  - headings
  - paragraphs
  - lists
  - ...
- The displayed properties of the logical units are determined by the browser (and stylesheet, if present)

HTML Example

- The following is a (very simple) complete HTML document:

```
<html>
  <head>
    <title>A Title</title>
  </head>
  <body>
    <h1>A Heading</h1>
  </body>
</html>
```

- When loaded in a browser (example.html)
  - “A Title” will be displayed in the title bar of the browser
  - “A Heading” will be displayed big and bold as the page contents

HTML, XHTML and XML

- These days, most web pages use *XHTML* rather than HTML
- XHTML uses the syntax of XML
- XHTML corresponds to a particular XML *vocabulary* or *document type*
- A document type is specified using a *Document Type Definition* (*DTD*) — see later
- HTML is essentially a less strict form of XHTML

Limitations of (X)HTML

So why not use XHTML rather than XML?

- (X)HTML defines a *fixed set* of elements (XHTML is *one* XML vocabulary)
- elements have *document* structuring semantics
- for presentation to human readers
- organisations want to be able to define their own elements
- applications need to exchange structured *data* too
- applications cannot consume (X)HTML easily
- use XML for *data* exchange and (X)HTML for document representation

XML versus Relational Data

- Why not use data from relational databases for exchange?
- XML is more flexible:
  - XML data is *semi-structured* rather than structured
  - Conformance of the data to a schema is not mandatory
  - XML schemas, if used, allow for more varied structures
- Relational data can always be (and often is) wrapped as XML

Motivating Example

- Say we want to store information about a personal CD library
- The CDs are all of classical music
- Some CDs contain simply solo (e.g., piano) works
- Some CDs have orchestral works (with orchestra, conductor)
- Some CDs contain performances of works by different composers
- We want to avoid repeating information in the descriptions
- We have only 4 CDs (see the next few slides)!

Example (1)

```
<CD-library>
   <CD number="724356690424">
      ...
   </CD>

   <CD number="419160-2">
      ...
   </CD>

   <CD number="449719-2">
      ...
   </CD>

   <CD number="430702-2">
      ...
   </CD>
</CD-library>
```

Example (2)

```
<CD number="724356690424">
  <performance>
    <composer>Frederic Chopin</composer>
    <composition>Waltzes</composition>
    <soloist>Dinu Lipatti</soloist>
    <date>1950</date>
  </performance>
</CD>
```

Example (3)

```
<CD number="419160-2">
  <composer>Johannes Brahms</composer>
  <soloist>Emil Gilels</soloist>
  <performance>
    <composition>Piano Concerto No. 2</composition>
    <orchestra>Berlin Philharmonic</orchestra>
    <conductor>Eugen Jochum</conductor>
    <date>1972</date>
  </performance>
  <performance>
    <composition>Fantasias Op. 116</composition>
    <date>1976</date>
  </performance>
</CD>
```

Example (4)

```
<CD number="449719-2">
  <soloist>Martha Argerich</soloist>
  <orchestra>London Symphony Orchestra</orchestra>
  <conductor>Claudio Abbado</conductor>
  <date>1968</date>
  <performance>
    <composer>Frederic Chopin</composer>
    <composition>Piano Concerto No. 1</composition>
  </performance>
  <performance>
    <composer>Franz Liszt</composer>
    <composition>Piano Concerto No. 1</composition>
  </performance>
</CD>
```

Example (5)

```
<CD number="430702-2">
  <composer>Antonin Dvorak</composer>
  <performance>
    <composition>Symphony No. 9</composition>
    <orchestra>Vienna Philharmonic</orchestra>
    <conductor>Kirill Kondrashin</conductor>
    <date>1980</date>
  </performance>
  <performance>
    <composition>American Suite</composition>
    <orchestra>Royal Philharmonic</orchestra>
    <conductor>Antal Dorati</conductor>
    <date>1984</date>
  </performance>
</CD>
```

Future of XML

- XML offers the possibility of truly cross-platform, long-term data formats:
  - Much of the data from the original moon landings is now effectively lost
  - Even reading an older Word file might already be problematic
- XML is a very simple, well-documented data format
- Any tool that can read text files can read an XML document
- XML may be the most portable and flexible document format since the ASCII text file

Overview

- In these lectures we are going to look at
  - some basic notions of XML
  - how to define XML vocabularies (DTDs, XML schemas)
  - how to query XML documents (XPath, XQuery)
  - how to process these queries

Literature

- A. Møller and M. Schwartzbach. *An Introduction to XML and Web Technologies*. Addison Wesley, 2006.
- S. Abiteboul, I. Manolescu, P. Rigaux, M-C. Rousset and P. Senellart. *Web Data Management*. Cambridge University Press, 2012.
- E.R. Harold, W.S. Means. *XML in a Nutshell.* O'Reilly, 2004
- H. Katz (editor). *XQuery from the Experts.* Addison Wesley, 2004
- These slides ...

Chapter 2

XML Fundamentals

Elements, Tags, and Data

- A very simple, yet complete, XML document:

```
<person>
    Alan Turing
</person>
```

- Composed of a single *element* whose name is `person`
- Element is delimited by the *start tag* `<person>` and the *end tag* `</person>`
- Everything between the start tag and end tag (exclusive) is the element's *content*

Elements, Tags, and Data (2)

- Content of the above element is the text string `Alan Turing`
- Whitespace is part of the content (although many applications choose to ignore it)
- `<person>` and `</person>` are *markup*,
- The string `Alan Turing` and surrounding whitespace are *character data*

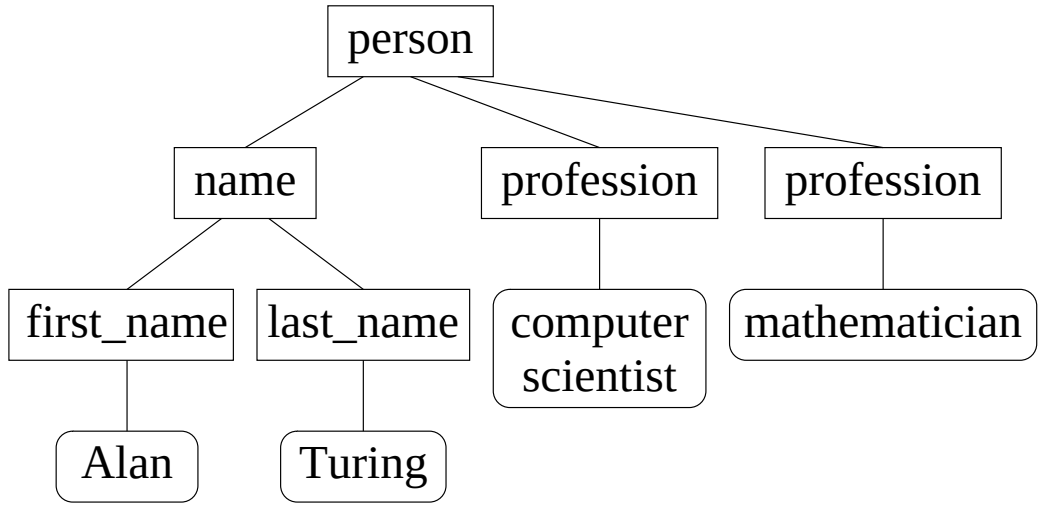
Elements, Tags, and Data (3)

- Special syntax for *empty elements*, elements without content
  - Each can be represented by a *single* tag that begins with < but ends with />
- XML is case sensitive, i.e. `<Person>` is not the same as `<PERSON>` (or `<person>`)

XML Documents and Trees

XML documents can be represented as trees

```
<person>
  <name>
    <first_name>Alan</first_name>
    <last_name>Turing</last_name>
  </name>
  <profession>
    computer scientist
  </profession>
  <profession>
    mathematician
  </profession>
</person>
```



XML Documents and Trees (2)

- The `person` element contains three *child* elements: one `name` and two `profession` elements
- The `person` element is called the *parent* element of these three elements
- An element can have an arbitrary number of child elements and the elements may be nested arbitrarily deeply
- Children of the same parent are called *siblings*
- Overlapping tags are prohibited, so the following is not possible:

```
<strong><em>
example from HTML
</strong></em>
```

XML Documents and Trees (3)

- Every XML document has one element without a parent
- This element is called the document's *root element* (sometimes called *document element*)
- The root element contains all other elements of a document

Attributes

- XML elements can have *attributes*
- An attribute is name-value pair attached to an element's start tag
- Names are separated from values by an equals sign
- Values are enclosed in single or double quotation marks
- Example:

```
<person born='1912/06/23' died='1954/06/07'>
  Alan Turing
</person>
```

- The order in which attributes appear is not significant

Attributes (2)

- We could model the contents of the original document as attributes:

```
<person>
  <name first='Alan' last='Turing'/>
  <profession value='computer scientist'/>
  <profession value='mathematician'/>
</person>
```

- This raises the question of when to use child elements and when to use attributes
- There is no simple answer

Attributes vs. Child Elements

- Some people argue that attributes should be used for metadata (about the element) and elements for the information itself
  - It's not always easy to distinguish between the two
- Attributes are limited in structure (their value is simply a string)
- There can also be no more than one attribute with a given name
- Consequently, an element-based structure is more flexible and extensible

Entities and Entity References

- Character data inside an element may not contain, e.g., a raw unescaped opening angle bracket <
- If this character is needed in the text, it has to be escaped by using the `<` *entity reference*
- `lt` is the *name* of the entity; `&` and `;` delimit the reference
- XML predefines five entities:

lt	<
amp	&
gt	>
quot	"
apos	'

CDATA Sections

- When an XML document includes samples of XML or HTML source code, all <, >, and & characters must be encoded using entity references
- This replacement can become quite tedious
- To facilitate the process, literal code can be enclosed in a *CDATA section*
- Everything between `<![CDATA[` and `]]>` is treated as raw character data
- The only thing that cannot appear in a CDATA section is the end delimiter `]]>`

Comments

- XML documents can also be commented
- Similar to HTML comments, they begin with `<!--` and end with `-->`
- Comments may appear
  - anywhere in character data
  - before or after the root element
  - However, NOT inside a tag or another comment
- XML parsers may or may not pass along information found in comments

Processing Instructions

- In HTML, comments are sometimes abused to support nonstandard extensions (e.g., server-side includes)
- Unfortunately,
  - comments may not survive being passed through several different HTML editors and/or processors
  - innocent comments may end up as input to an application
- XML uses a special construct to pass information on to applications: a *processing instruction*
- It begins with `<?` and ends with `?>`
- Immediately following the `<?` is the target (possibly the name of the application)

Processing Instructions (2)

Examples:

- Associating a stylesheet with an XML document:

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

- Embedded PHP in (X)HTML:

```
<?php
  mysql_connect("database...",
                "user",
                "password");
  ...
  mysql_close();
?>
```

XML Declaration

- The *XML declaration* looks like a processing instruction, but only gives some information about the document:

```
<?xml version='1.0'
      encoding='US-ASCII'
      standalone='yes'?>
```

- *version*: at the moment 1.0 and 1.1 are available (we focus on 1.0)
- *encoding*: defines the character set used (e.g. ASCII, Latin-1, Unicode UTF-8)
- *standalone*: determines if some other file (e.g. DTD) has to be read to determine proper values for parts of the document

Well-Formedness

A *well-formed* document observes the syntax rules of XML:

- Every start tag must have a matching end tag
- Elements may not overlap
- There must be exactly one root element
- Attribute values must be quoted
- An element may not have two attributes with the same name
- Comments and processing instructions may not appear inside tags
- No unescaped < or & signs may occur in character data

Well-Formedness (2)

- XML names must be formed in certain ways:
  - May contain standard letters and digits 0 through 9
  - May include the punctuation characters underscore (_), hyphen (-), and period (.)
  - May only start with letters or the underscore character
  - There is no limit to the length
- The above list is not exhaustive; for a complete list look at the W3C specification
- A parser encountering a non-well-formed document will stop its parsing with an error message

XML Namespaces

- MathML is an XML vocabulary for mathematical expressions
- SVG (Scalable Vector Graphics) is an XML vocabulary for diagrams
- say we want to use XHTML, MathML and SVG in a single XML document
- how does a browser know which element is from which vocabulary?
- e.g., both SVG and MathML define a `set` element
- we shouldn't have to worry about potential name clashes
- we should be able to specify different *namespaces*, one for each of XHTML, MathML and SVG

The namespaces solution

- The solution is to *qualify* element names with *URIs*
- A URI (Universal Resource Identifier) is usually used for *identifying* a resource on the Web
- (A Uniform Resource Locator (URL) is a special type of URI)
- A *qualified name* then consists of two parts:
  `namespace:local-name`
- e.g., `<http://www.w3.org/2000/svg:circle ... />`
- where `http://www.w3.org/2000/svg` is a URI and namespace
- The URI does *not* have to reference a real Web resource
- URIs only disambiguate names; they don't have to define them
- In this case, the browser knows the SVG namespace and behaves accordingly

Namespace declarations

- using URIs everywhere is very cumbersome
- so namespaces are used indirectly using
  - namespace *declarations* and
  - associated *prefixes* (user-specified)

```
<... xmlns:svg="http://www.w3.org/2000/svg">
   <p>A circle looks like this
   ...
      <svg:circle ... />
   ...
</...>
```

- The `xmlns:svg` attribute
  - declares the namespace `http://www.w3.org/2000/svg`
  - associates it with prefix `svg`

Scope of namespace declarations

- the *scope* of a namespace declaration is
  - the element containing the declaration
  - and all its *descendants* (those elements nested inside the element)
  - can be overridden by *nested* declarations
- both elements and attributes can be qualified with namespaces
- unprefixed element names are assigned a *default* namespace
- default namespace declaration: `xmlns="`*`URI`*`"`

Namespaces example

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
   ...
   <p>A circle looks like this
      <svg:svg ... >
         ...
         <svg:circle ... />
         ...
      </svg:svg>
      and has
      ...
   </p>
</html>
```

- `html` and `p` are in the *default* namespace (`http://www.w3.org/1999/xhtml`)

Namespaces example (2)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:svg="http://www.w3.org/2000/svg">
   ...
   <p>A circle looks like this
      <svg:svg ... >
         ...
         <svg:circle ... />
         ...
      </svg:svg>
      and has
      ...
   </p>
</html>
```

- namespace for `svg` and `circle` is `http://www.w3.org/2000/svg`
- note that `svg` is used both as a prefix and as an element name

Summary

- This chapter gave a brief summary of XML
- Only the most important aspects (which are needed later on) were covered