CS352 Lecture: XML

Last revised Oct 17, 2006

Objectives:

1. To introduce XML
2. To discuss the relationship between XML and databases
3. To introduce DTD's

Materials:

1. Handout: Example XML representation of a book plus DTD
2. Projectables: Mapping XML to relational tables and to objects
3. Demonstration of Circuit Sandbox (to show data storage in XML)
4. Projectable showing a simple example of using namespaces
5. Projectable showing queries in XQuery (3 pp)
6. Demos/Handout of SAX Events and DOM Structure
7. Handout/Demo: Example of Parsing XML data using SAX and DOM

I. Introduction

A. If you look at articles in the trade press or on the web, you will likely see a lot of mention of XML.

ASK Class for examples

To begin our discussion of XML, it is appropriate to consider several key questions:

1. What is XML?

2. Why is there so much interest in it? (I.e. what problem(s) does it solve?)

3. How does XML relate to databases?

B. What is XML?

1. It is perhaps best to begin by saying what XML is <u>not</u>. XML is not another data model (like the hierarchical, network, relational, and OO models). That is, we do not currently have - nor are we likely to have - full-scale database systems based on actually storing all information in a format like the example I am about to hand out.

2. Rather, XML is a language. The name XML is an abbreviation for eXstensible Markup Language.

   a) XML is, then, a specification for how one goes about constructing a representation of information using a tagged representation.

      Distribute example of an XML representation for a book. Ignore <!DOCTYPE ...> at the top and DTD at the bottom for now.

   b) You may notice some similarity between the example and HTML. That is not accidental.

      (1) XML and HTML share a common ancestor - SGML (Standard Generalized Markup Language).

         (a) Markup languages originated in the world of publishing.

         (b) Historically, markup was used to convey formatting information - e.g. markup might denote that some text was to be printed in boldface or in a specific font or the like.

         (c) In XML, markup is primarily used to convey information about semantics - what does some item mean.

         (d) By way of comparison, HTML markup can be of either type.

            Example: `<b> ... </b>` specifies that some text is to be displayed in boldface - formatting markup

            Example: `<title> .. </title>` specifies that some text is the title of the document - a semantic specification (which may be significant when doing a search), though it has formatting implications (e.g. it is displayed as a window title).

      (2) There is a variant of HTML - called XHTML - that it is also valid XML (i.e. it is both).

         (a) Every XHTML document is also an HTML document.

         (b) But not every HTML document is a valid XHTML document.

         (c) Example: All my ATM Example system pages are actually valid XHTML.

3. Actually, XML is not just a language - it is a meta-language - i.e. a language that can be used for defining other languages.

   a) For example, the handout I passed out presupposes that one has defined a language for talking about library books, in which words like "book", "call_number", and "author" have special meanings.

   b) It is also possible to create an XML language for talking about, say, computer-aided design of automobiles, in which words like "engine" and "chassis" and "piston" would have special meanings.

   c) XHTML that I alluded to earlier is a language in which words like "head", "body", "table", and "ol" have special meanings.

4. Note that a key concept in XML is the use of tags (like "author"). This attaches semantic information to data - so that we know what the data actually means. The personal name information is not just information about any person - it is about a person who is related to the book we are describing in a specific way.

   Example: Suppose we had a book description in which we recorded binding colors. Now the word "Black" would have very different meanings if it occurred as a "binding_color" rather than as the "last_name" value of an "author" or as or perhaps as part of a "title". This facilitates semantic searches - i.e. if we are interested in a book written by someone whose name is Black, we only care about occurrences of "Black" as an author; on the other hand, if we are interested in books about "Black Bears" we are only interested if the word occurs in the title, and if we are writing a detective mystery we may care about books whose binding color is black.

   Contrast this with how web search engines typically handle queries!

C. Why is there so much interest in it? (I.e. what problem(s) does it solve?)

1. One reason for interest in XML is to facilitate the *interchange* of information between diverse sources.

   a) In a networked world, it is common to want to combine information from several different sources.

      Example: Searching the card catalog of several different libraries for books on a given topic, and then combining the results into a single list.

b) It is also common to find information being transmitted from one organization to another.

(1) Example: B2B applications

(2) Example: an item may come with descriptive information - e.g. our library has access to standardized catalog entries (MARC records) for books so that the cataloging librarian doesn't have to figure out all the information to be entered into the catalog for a newly-received book.

c) Raw data is not particularly useful unless it is interpreted somehow.

(1) Example: Suppose we retrieved the following data on a book:
QL737.123 Black Bears

Is it a book whose title is "Black Bears", or a book by an author named "Black" whose title is "Bears", or perhaps a book with a black binding whose title is "Bears"?

That, of course, is the role of headings in a table. But, in a large table, the headings may be far removed from the data they describe, and it may be hard to associate the correct heading with the data, either for a human or for a computer system.

(2) Life is further complicated by differences in conventions between sites - e.g. two different sites might record an individual named "Alexander George" as

Alexander George

or

George Alexander

(3) Finally, even if we have headings associated with an object, different information sources may use different names.

Example: Suppose we are trying to combine information from three sources, one of which calls the name attributes of a person "last_name" and "first_name", while another calls them "lname" and "fname" and another calls them "last" and "first".

2. Another reason for interest in XML is to separate <u>semantics</u> from <u>presentation.</u>

a) Example: The html <table>, <tr>, and <td> tags specify how data is to be displayed, but tell nothing about what it means. On the other hand, the <title> tag tells us what the data means. Html mixes the two - sometimes even offering alternatives (e.g. <b> specifies boldface presentation while the tag that most browsers render as boldface is more semantic, though still not very much so.)

b) An XML document uses semantic tags, and is typically associated with a style sheet that specifies how various elements are to be displayed. Different systems can use different style sheets - e.g. one sort of style may be appropriate for a computer web browser and another for a cell phone.

3. A third reason for interest in XML is to facilitate handling information that doesn't neatly fit the relational paradigm.

a) Information in which the order of items is important.

b) Information whose structure may change over time, or may vary from item to item (e.g. in a medical records system, one patient may have been seen once in 5 years, while another may have had a major illness requiring numerous visits to the doctor, tests, hospitalization etc.)

c) (Some of these concerns sound similar to those driving development of an OODBMS model - I wonder if perhaps some of the energy that once went toward the latter is now going into XML instead.)

D. How does XML relate to databases?

1. As I noted earlier, XML is not a data model. Though XML data can be stored in a database, one wouldn't think of building a full-scale database system whose architecture is XML.

a) A key difference between XML and data models used for databases is how meta-data is handled.

(1) In a relational DBMS, the meta-data is stored in a catalog, separate from the actual data. For example, a SQL statement like

```
create table borrower(
     id char(10) primary key,
     last_name char(20),
     first_name char(20)
)
```

creates a single entry in the system catalog. Any number of rows may be inserted into the table, but only one copy of the meta-data is kept; in fact, the meta-data will continue to exist even if all rows are deleted from the table.

(2) Conversely, XML stores meta-data with the data in the form of tags. Thus, if borrower information were stored in XML, there would be two copies of the strings "borrower", "id", "last_name", and "first_name" appearing in start and end tags for each borrower.

(3) This high level of redundancy in the storage of the meta-data makes pure textual XML an unlikely candidate for the primary means of storing data in a DBMS.

2. OTOH, DBMS's may offer a mechanism for *converting* between the internal form used for storing data and XML.

a) Data retrieved from the database can be converted to XML to facilitate integrating it with other data from other sources or use on systems storing data in a different sort of database. (This is called publishing the data)

b) Arriving XML data can be converted to the format required by the database. (This is called shredding the data)

Example: the book example I gave you earlier could be converted to several relational tuples: a row to insert into the book table, a row to insert into the book_title table, and three rows to insert into the book_author table.

PROJECT

Book

accession_number	call_number	copy_number
...		
42	QA76.9 D3 S5737 2002	1
...		

Book_title

call_number	title
...	
QA76.9 D3 S5737 2002	Database System Concepts
...	

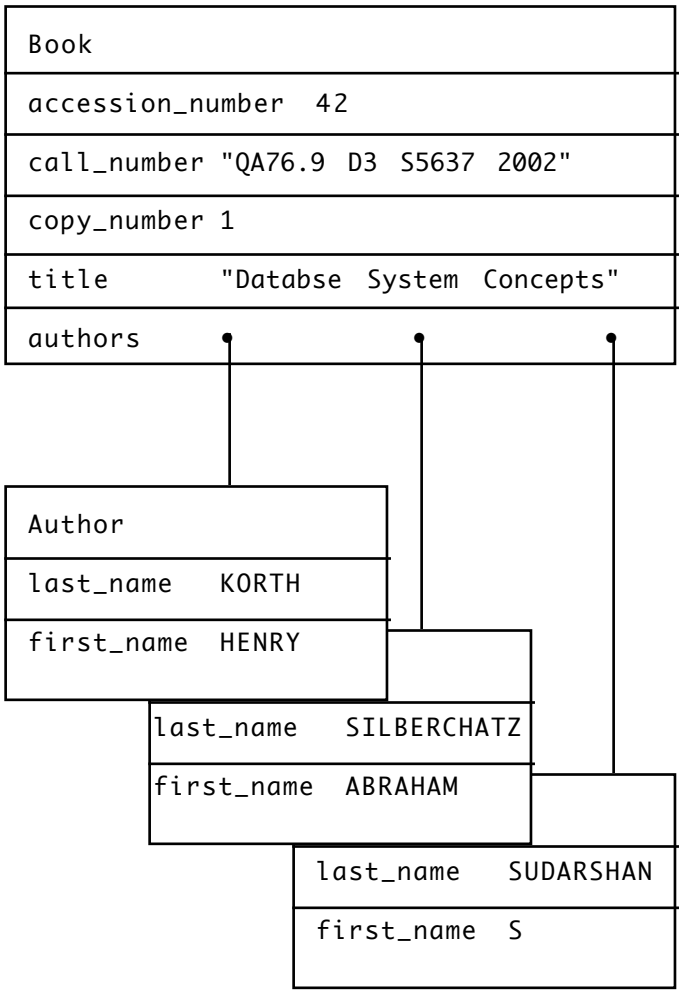
Book_author

call_number	author_last_name	author_first_name
...		
QA76.9 D3 S5737 2002	KORTH	HENRY
QA76.9 D3 S5737 2002	SILBERSCHATZ	ABRAHAM
QA76.9 D3 S5737 2002	SUDARSHAN	S
...		

MAPPING THE BOOK DESCRIPTION IN XML TO RELATIONAL TABLES

Example: the book example I gave you could also be converted to a Book object to be stored in an OO database, to be associated with several Author objects that might either already exist (and be matched by their names) or might need to be created as well.

PROJECT



MAPPING THE BOOK DESCRIPTION IN XML TO OBJECTS

3. Though XML is not the primary vehicle for data storage, SQL:2003 provides an XML data type, so that a column in a relational database may hold XML data.
   a) Data in an XML column is converted to an internal binary form, rather than being stored as text.
   b) Data in an XML column is typically associated with a schema that contains much of the meta-data to reduce redundancy.
   c) An XML column would be particularly useful for storing information that is to be distributed via the worldwide web.
   d) Recent work (which is ongoing at this time) has developed a strategy for integrating XQuery into SQL (yielding something called SQL/XML) to allow a SQL query to access XML data.

E. It is also worth noting that some applications store their data as XML (typically in flat text files), as an alternative to using a DBMS or an application-specific binary format.

Example: CircuitSandbox - a senior project from 2005 that we use in CS111 and CS311. (Some of you have used it - others will).

Create a simple circuit involving a gate, two inputs, and an output.

Save it

Show resulting file.

II. More about XML

A. A well-formed XML document consists of a root element, which may contain other elements nested within it.

Example: Our example was an XML document describing a single book, which was the root element, with author elements nested within.

Alternately, we could have an XML document describing all the books in the catalog, where the root element is the catalog, containing book elements nested within, each of which, in turn contain author elements within them - e.g something like:

```
<catalog>
   <book> ... </book>
   <book> ... </book>
   <book> ... </book>
</catalog>
```

B. Every element in an XML document is bracketed by start and end tags (e.g. <book>, </book>; <author>, </author>.

1. In the case where nothing needs to appear between the tags, the start and end tags can be combined into one.

   Example: <author ... />

2. Start and end tags must be properly nested - e.g.

   <book> ... <author> ... </book> ... </author>

   is not properly nested and not allowed.

C. An element may incorporate any or all of the following:

1. Nested elements.

   Example: the example book element contains a call_number, copy_number, title, and three author elements within it.

2. Attributes (name/value pairs)

   Example: the book element has an accession_number attribute and the author elements have at least a last_name attribute and sometimes a first_name attribute as well.

3. Text

   Example: several of the elements (call_number, copy_number, and title) contain textual information.

4. Note that an element can contain any or all of the above.

5. A document that conforms to the basic syntax of XML is called a well-formed document.

D. XML is case sensitive, so the tags <author> and <AUTHOR> are not the same.

E. An aside on differences between XHTML and HTML:

1. Ramifications of case-sensitivity. Tags are all-lowercase in XHTML.

2. Ramifications of requiring closing tags.

   a) In some cases, end tags are optional in HTML. (E.g. </p>,). In XHTML they are required.

   b) Some HTML tags do not have an end form (e.g.
). But both HTML and XML allow a simple tag that has no content to be self-terminating - e.g.
.

III. DTD's

A. One key issue in the usefulness of XML is standardization of names for elements - e.g. we should consistently use one name like <last_name> rather than different names like <last_name>, <lname>, or <last>.

B. To facilitate this, XML allows a document to be associated with a DTD (document type description.)

1. Note the <!DOCTYPE ... reference in the example I handed out. (This must always occur first in the document.)

2. A DTD may either be part of a document, or it may be a reference to a file stored elsewhere, either in a file on the same system or at a URL on the net.

   Example: In the case of the Circuit Sandbox example I showed earlier, the DTD was actually part of the saved document.

   (PROJECT again and point out).

C. The handout contains an example of a DTD that might be referenced by the book_example document.

1. A DTD defines various elements that may occur in the document, using <!ELEMENT ... declarations.

2. The first element defined is the root element. A valid XML document that conforms to this DTD will consist of a single instance of this element with other elements nested within.

3. The structure of an element is defined by listing the things that can occur within it. Note that, in the example, a book element consists of a call_number element, a copy_number element, a title element, and 0 or more author elements - in that order. (The syntax here resembles that used for regular expressions - e.g. the use of + for "one or more" and * for "zero or more" plus the use of | for "or" - not shown here.

4. In an element declaration, #PCDATA stands for "parsed character data" - i.e. that the XML parser will not try to interpret, leaving it to the application that processes the document.

5. If an element is allowed to have attributes, these are specified in an <!ATTLIST ... declaration. Note that a given element must have an

<!ELEMENT declaration and may have an <!ATTLIST declaration (if it has attributes), If an element has only attributes, and does not contain any nested elements, then its <!ELEMENT declaration contains the word EMPTY. (As in the example for author)

D. Standardized DTD's have been developed for many fields of endeavor. By conforming to such a standardized DTD, an XML document can ensure that it is "understandable" to any audience that uses the same DTD.

EXAMPLE: There is a standard DTD for XHTML (actually three, with different levels of strictness.) An XHTML page will reference this.

```
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "DTD/xhtml11-strict.dtd">
```

E. A well-formed XML document that contains a DTD (or a reference to one) and that conforms to that DTD is said to be valid.

1. Validity is only possible if the document is well formed.

2. Validity requires a DTD.

F. XML documents can also make use of namespaces.

1. One problem that can easily arise is that the same name may be used for two different purposes in two different contexts.

   Example: "title" means different things in the context of courses and in the context of books. .

2. XML provides mechanisms for using names that are qualified by a namespace.

   A very simple example - PROJECT

```
<!DOCTYPE course [
     <!ELEMENT course (reg:title, text)>
     <!ELEMENT reg:title (#PCDATA)>
     <!ELEMENT text (book:title)>
     <!ELEMENT book:title (#PCDATA)>
]>
```

```
<course>
     <reg:title>Database Systems</reg:title>
     <text>
          <book:title>
               Database System Concepts
          </book:title>
     </text>
</course>
```

3. Typically, namespaces are standardized and associated with a specific URI. (That is not shown in the above).

IV. Querying XML

A. Just as SQL has emerged as an industry-standard language for querying data in a relational database, so a language known as XQuery has emerged in recent years as an industry-standard language for querying XML data.

1. XQuery incorporates a previous line of work known as XPath.

2. One site makes this statement: "The best way to explain XQuery is to say that XQuery is to XML what SQL is to database tables".

3. The current standard for XQuery is less than a year old, and currently has the status of a "W3C candidate recommendation".

4. As we noted earlier, the most recent SQL standard introduced the notion of an XML data type for a column in a relational database. The SQL:2003 standard also defined a syntax for embedding a query written in XQuery in a SQL query - e.g. to find rows in a database which contain data in an XML column that satisfies a particular query written in XQuery.

5. Note that XQuery is a language for querying data - it does not provide support for modifying data. (It performs a function analogous to that of SQL `select`, but does not have facilities analogous to SQL `insert`, `delete`, or `update`.)

B. The basic unit of XQuery is the "FLWOR" (note strange spelling) expression - an expression which consists of some or all of the following elements:

1. for (always present) - specifies the source of the data - analogous to from in SQL

2. let - binding local variables (optional)

3. where - analogous to where in SQL

4. order by - analogous to order by in SQL (optional)

5. return (always present) - specifies the value to return - analogous to select in SQL

   Example: PROJECT Examples from Book (3 pp)

V. Processing XML in a Java program

A. The standard J2SE platform includes a number of packages that provide support for working with XML.

   1. Some of these are defined by sun and are in the "javax" tree:

      SHOW IN J2SE DOCUMENTATION: javax.xml ....

   2. Some are defined by the world-wide web consortium or xml.org

      SHOW IN DOCUMENTATION org.w3c..., org.xml...

B. There's obviously a lot of material here. We will only scratch the surface. Basically, the standard packages provide support for two models of XML parsing.

   1. SAX (Simple API for XML) is an event-driven parsing model.

      a) It provides support for reading through an XML document, and firing events including

         (1) The occurrence of a start tag for an element (including information about any attributes included in the tag)

         (2) The occurrence of an end tag for an element

         (3) The occurrence of character data within an element

b) DEMO/HANDOUT: SAXEvents.java using book_example.xml

c) When SAX is used to process an XML document, no internal representation of the document is created. Rather, handlers associated with the various events must do what needs to be done when the appropriate portion of the document is read.

2. DOM (Document Object Model) allows an XML stream to be converted into a tree structure which can then be manipulated internally.

   a) The complete document is read and converted to an internal representation; only then does the program do whatever it needs to do with the data.

   b) DEMO/HANDOUT: DOMStructure.java on book_example.xml

3. HANDOUT: An example of using both approaches to parse an XML document.

   a) Note that, in the SAX example, the keyword search is being done as the document is being read; and if another keyword search is desired, the XML document must be read again.

   b) In the DOM example, the document is first read, and then any number of keyword searches can be done against the internal representation.

4. It would seem, at first, that DOM allows more flexible access to the information in the document, which is true. However, SAX has the advantage that no internal representation needs to be constructed - which is especially advantageous when the XML document being processed is large.