XML

Tuesday, April 27, 2004
8:43 AM

Motivation:

Relational Data Model is quite rigid. … powerful, but rigid.

With the explosive growth of the Internet, electronic information is all around us, and tends not to be warehoused, centrally located, or conforming to a rigid set of relations along with their interrelationships.

Information is more dynamic, and *information interchange* (data exchange between applications) becomes the focus. The goal is to allow an apparent integration of data and/or databases from multiple sources.

When we wish to exchange data between entities, the data forms and formats may not be identical. The medium for information exchange becomes what is termed "semistructured data" -- a new data model designed to cope with problems of information integration.

XML is a standard language for describing semistructured data schemas and representing data.

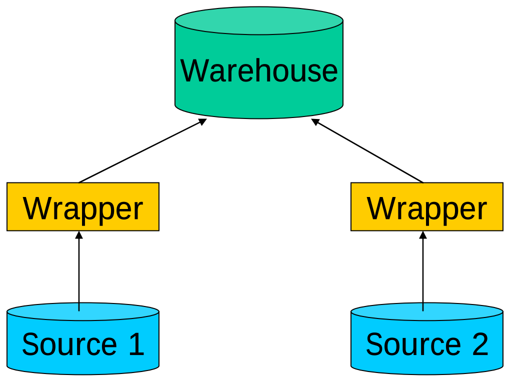
Some of the problems

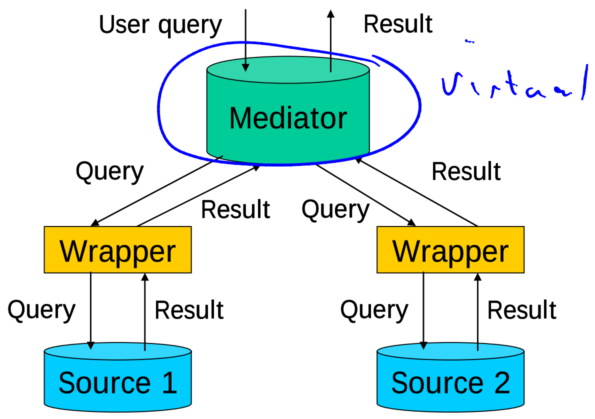
- Related data exists in many places and could, in principle, work together.
- But different databases differ in:
  1. Model (relational, object-oriented?).
  2. Schema (normalized/unnormalized?).
  3. Terminology: are consultants employees? Retirees? Subcontractors?
  4. Conventions (meters versus feet?).

Example:
- Every bar has a database.
  - One may use a relational DBMS; another keeps the menu in an MS-Word document.
  - One stores the phones of distributors, another does not.
  - One distinguishes ales from other beers, another doesn't.
  - One counts beer inventory by bottles, another by cases.

Two Approaches to Integration

1. *Warehousing* : Make copies of the data sources at a central site and transform it to a common schema.
   - Reconstruct data daily/weekly, but do not try to keep it more up-to-date than that.
2. *Mediation* : Create a view of all sources, as if they were integrated.
   - Answer a view query by translating it to terminology of the sources and querying them.



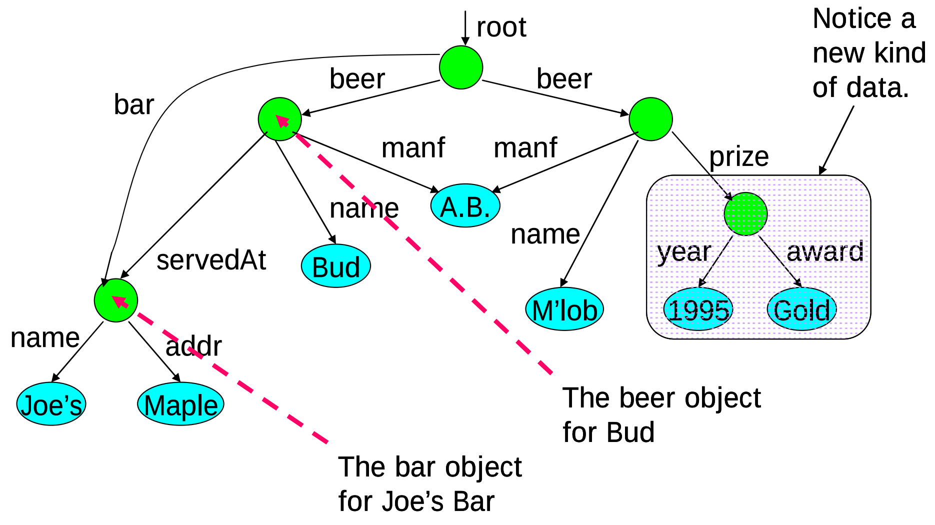


Semistructured Data Model

- Purpose: represent data from independent sources more flexibly than either relational or object-oriented models.
- Think of objects, but with the type of each object its own business, not that of its "class."
- Labels to indicate meaning of substructures.

Think of Semistructured Data as a Graph

- Nodes = objects.
- Labels on arcs (attributes, relationships).
- Atomic values at leaf nodes (nodes with no arcs out).
- Flexibility: no restriction on:
  - Labels out of a node.
  - Number of successors with a given label.



XML as a language for specifying semistructured data

- XML = Extensible Markup Language.
- While HTML uses tags for formatting (e.g., "italic"), XML uses tags for semantics (e.g., "this is an address").
- Key idea: create tag sets for a domain (e.g., genomics), and translate all data into properly tagged XML documents.

- *Well-Formed XML* allows you to invent your own tags.
  - Similar to labels in semistructured data.
- *Valid XML* involves a DTD (Document Type Definition), which limits the labels and gives a grammar for their use.

- Start the document with a *declaration*, surrounded by <? ... ?> .
- Normal declaration is:

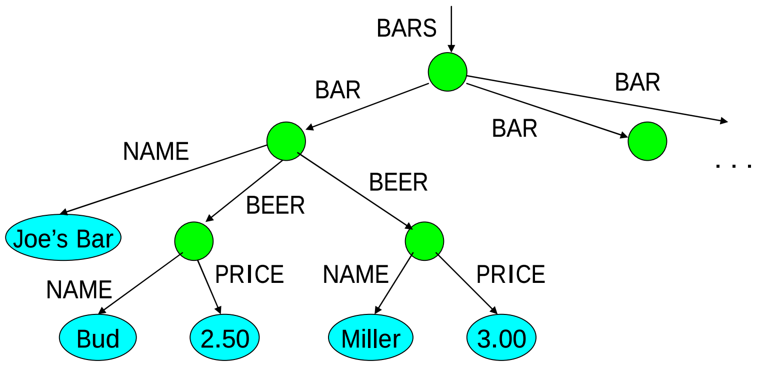
```
<? XML VERSION = "1.0"
  STANDALONE = "yes" ?>
```

  - "Standalone" = "no DTD provided."
- Balance of document is a *root tag* surrounding nested tags.

- Tags, as in HTML, are normally matched pairs, as <FOO> ... </FOO> .
- Tags may be nested arbitrarily.
- Tags requiring no matching ender, like <P> in HTML, are also permitted.

```
<? XML VERSION = "1.0" STANDALONE = "yes" ?>
<BARS>
   <BAR><NAME>Joe's Bar</NAME>
        <BEER><NAME>Bud</NAME>
               <PRICE>2.50</PRICE></BEER>
        <BEER><NAME>Miller</NAME>
               <PRICE>3.00</PRICE></BEER>
   </BAR>
   <BAR> ...
</BARS>
```

- The <BARS> XML document is:



- Another Example (from textbook)

```
<bank-1>
    <customer>
        <customer-name> Hayes </customer-name>
        <customer-street> Main </customer-street>
        <customer-city>     Harrison </customer-city>
        <account>
            <account-number> A-102 </account-number>
            <branch-name>      Perryridge </branch-name>
            <balance>              400 </balance>
        </account>
        <account>
            …
        </account>
    </customer>
      .
      .
</bank-1>
```

- Nesting of data is useful in data transfer
  - Example: elements representing customer-id, customer name, and address nested within an order element
- Nesting is not supported, or discouraged, in relational databases
  - With multiple orders, customer name and address are stored redundantly
  - normalization replaces nested structures in each order by foreign key into table storing customer name and address information
  - Nesting is supported in object-relational databases
- But nesting is appropriate when transferring data
  - External application does not have direct access to data referenced by a foreign key
- Mixture of text with sub-elements is legal in XML.
  - Example:

```
<account>
    This account is seldom used any more.
     <account-number> A-102</account-number>
     <branch-name> Perryridge</branch-name>
     <balance>400 </balance>
</account>
```

  - Useful for document markup, but discouraged for data representation

Grammar for specifying "Valid XML"

... that is, XML conforming to a schema

Two "flavors"

Two "flavors"

- Document Type Definition (DTD)
  - Widely used
- XML Schema
  - Newer, and with increasing use

DTDs

- Essentially a context-free grammar for describing XML tags and their nesting.
- Each domain of interest (e.g., electronic components, bars-beers-drinkers) creates one DTD that describes all the documents this group will share.

```
<!DOCTYPE <root tag> [
  <!ELEMENT <name> ( <components> )
  <more elements>
]>
```

- The description of an element consists of its name (tag), and a parenthesized description of any nested tags.
  - Includes order of subtags and their multiplicity.
- Leaves (text elements) have #PCDATA in place of nested tags.

Example:

```
<!DOCTYPE Bars [
   <!ELEMENT BARS (BAR*)>
   <!ELEMENT BAR (NAME, BEER+)>
   <!ELEMENT NAME (#PCDATA)>
   <!ELEMENT BEER (NAME, PRICE)>
   <!ELEMENT PRICE (#PCDATA)>
]>
```

A BARS object has zero or more BAR's nested within.

A BAR has one NAME and one or more BEER subobjects.

A BEER has a NAME and a PRICE.

NAME and PRICE are text.

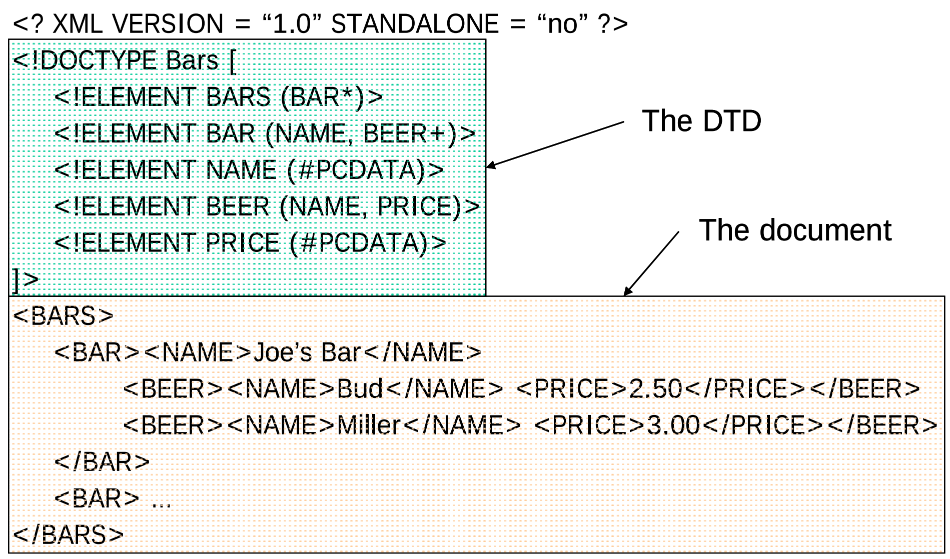
- Subtags must appear in order shown.
- A tag may be followed by a symbol to indicate its multiplicity.
  - * = zero or more.
  - + = one or more.
  - ? = zero or one.
- Symbol | can connect alternative sequences of tags.

- A name is an optional title (e.g., “Prof.”), a first name, and a last name, in that order, or it is an IP address:

```
<!ELEMENT NAME (
 (TITLE?, FIRST, LAST) | IPADDR
)>
```

To use DTDs

1. Set STANDALONE = "no"
2. Either
   a. Include the DTD as a preamble of the XML document, or
   b. Follow DOCTYPE and the <root tag> by SYSTEM and a path to the file where the DTD can be found.



```
<? XML VERSION = “1.0” STANDALONE = “no” ?>
<!DOCTYPE Bars [
    <!ELEMENT BARS (BAR*)>
    <!ELEMENT BAR (NAME, BEER+)>
    <!ELEMENT NAME (#PCDATA)>
    <!ELEMENT BEER (NAME, PRICE)>
    <!ELEMENT PRICE (#PCDATA)>
]>
<BARS>
    <BAR><NAME>Joe’s Bar</NAME>
          <BEER><NAME>Bud</NAME> <PRICE>2.50</PRICE></BEER>
          <BEER><NAME>Miller</NAME> <PRICE>3.00</PRICE></BEER>
    </BAR>
    <BAR> ...
</BARS>
```

◆Assume the BARS DTD is in file bar.dtd.

```
<? XML VERSION = "1.0" STANDALONE = "no" ?>
<!DOCTYPE Bars SYSTEM "bar.dtd">
<BARS>
   <BAR><NAME>Joe's Bar</NAME>
        <BEER><NAME>Bud</NAME>
                <PRICE>2.50</PRICE></BEER>
        <BEER><NAME>Miller</NAME>
                <PRICE>3.00</PRICE></BEER>
   </BAR>
   <BAR> ...
</BARS>
```

Get the DTD from the file bar.dtd

◆Opening tags in XML can have *attributes*, like <A HREF = "..."> in HTML.

◆In a DTD,

`<!ATTLIST <element name>... >`

gives a list of attributes and their datatypes for this element.

◆Bars can have an attribute `kind`, which is either sushi, sports, or "other."

```
<!ELEMENT BAR (NAME BEER*)>
  <!ATTLIST BAR kind = "sushi" |
    "sports" | "other">
```

ID's and IDREF's

◆These are pointers from one object to another, in analogy to HTML's NAME = "foo" and HREF = "#foo".

◆Allows the structure of an XML document to be a general graph, rather than just a tree.

Creating ID's

- Give an element *E* an attribute *A* of type ID.
- When using tag <*E* > in an XML document, give its attribute *A* a unique value.
- Example:

```
<E  A = “xyz”>
```

Creating IDREF's

- To allow objects of type *F* to refer to another object with an ID attribute, give *F* an attribute of type IDREF.
- Or, let the attribute have type IDREFS, so the *F*–object can refer to any number of other objects.

Example

- Let’s redesign our BARS DTD to include both BAR and BEER subelements.
- Both bars and beers will have ID attributes called `name`.
- Bars have PRICE subobjects, consisting of a number (the price of one beer) and an IDREF `theBeer` leading to that beer.
- Beers have attribute `soldBy`, which is an IDREFS leading to all the bars that sell it.

The DTD

```
<!DOCTYPE Bars [
   <!ELEMENT BARS (BAR*, BEER*)>
   <!ELEMENT BAR (PRICE+)>
        <!ATTLIST BAR name = ID>
   <!ELEMENT PRICE (#PCDATA)>
        <!ATTLIST PRICE theBeer = IDREF>
   <!ELEMENT BEER ()>
        <!ATTLIST BEER name = ID, soldBy = IDREFS>
]>
```

The Example Object

```
<BARS>
   <BAR name = “JoesBar”>
        <PRICE theBeer = “Bud”>2.50</PRICE>
        <PRICE theBeer = “Miller”>3.00</PRICE>
   </BAR> ...
   <BEER name = “Bud”, soldBy = “JoesBar,
        SuesBar,...”>
   </BEER> ...
</BARS>
```

Final Sections

12.1 – 12.7

15.1 – 15.5

16.1

10.1 – 10.3