XML
Semistructured Data
Extensible Markup Language
Document Type Definitions

Framework
1. Information Integration: Making databases from various places work as one.
2. Semistructured Data: A new data model designed to cope with problems of information integration.
3. XML: A standard language for describing semistructured data schemas and representing data.

The Information-Integration Problem
- 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.
A Mediator

Semistructured Data
- 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.

Graphs of Semistructured Data
- 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.

Example: Data Graph

XML
- 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 and Valid XML
- 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.
Well-Formed XML

- Start the document with a declaration, surrounded by `<? ... ?>`.
- Normal declaration is:
  ```xml
  <? XML VERSION = "1.0" STANDALONE = "yes" ?>
  • "Standalone" = "no DTD provided."
  • Balance of document is a root tag surrounding nested tags.
  ```

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.

Example: Well-Formed XML

```xml
<? XML VERSION = "1.0" STANDALONE = "yes" ?>
<BEER><NAME>Miller</NAME>
  <PRICE>3.00</PRICE></BEER>
<BAR>...
```

XML and Semistructured Data

- Well-Formed XML with nested tags is exactly the same idea as trees of semistructured data.
- We shall see that XML also enables nontree structures, as does the semistructured data model.

Example

- The `<BARS>` XML document is:

```
<BEER>
  <NAME>Bud</NAME>
  <PRICE>2.50</PRICE>
</BEER>
<BEER>
  <NAME>Miller</NAME>
  <PRICE>3.00</PRICE>
</BEER>
```

Document Type Definitions

- 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.
DTD Structure

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

DTD 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: DTD

```xml
<!DOCTYPE Bars [
  <ELEMENT NAME (A BARS object has zero or more BAR's nested within.)
  <ELEMENT A BAR has one NAME and one or more BEER subobjects.
  <ELEMENT A BEER has a NAME and a PRICE.
] >
```

Element Descriptions

- 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.

Example: Element Description

- 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:
  ```xml
  <!ELEMENT NAME (TITLE?, FIRST, LAST) | IPADDR)
  ```

Use of DTD’s

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.
Example (a)

<!-- XML VERSION = "1.0" STANDALONE = "no" -->

The DTD

The document

Example (b)

- Assume the BARS DTD is in file bar.dtd.
  <!-- XML VERSION = "1.0" STANDALONE = "no" -->

  <BARS>
    <BAR> <NAME>Joe's Bar</NAME>
    <BEER><NAME>Sapporo</NAME>
    <PRICE>5.00</PRICE></BEER>
  </BAR>
  <BAR> ... </BAR>
</BARS>

Attributes

- 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.

Example: Attributes

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

  <!ELEMENT BAR (NAME BEER*)>
  <!ATTLIST BAR kind = "sushi" | "sports" | "other">
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: ID’s and IDREF’s

◆ 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

◆ Bar objects have name as an ID attribute and have one or more PRICE subobjects.
◆ PRICE objects have a number (the price) and one reference to a beer.
◆ Beer objects have an ID attribute called name, and a soldBy attribute that is a set of Bar names.

Example Document

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