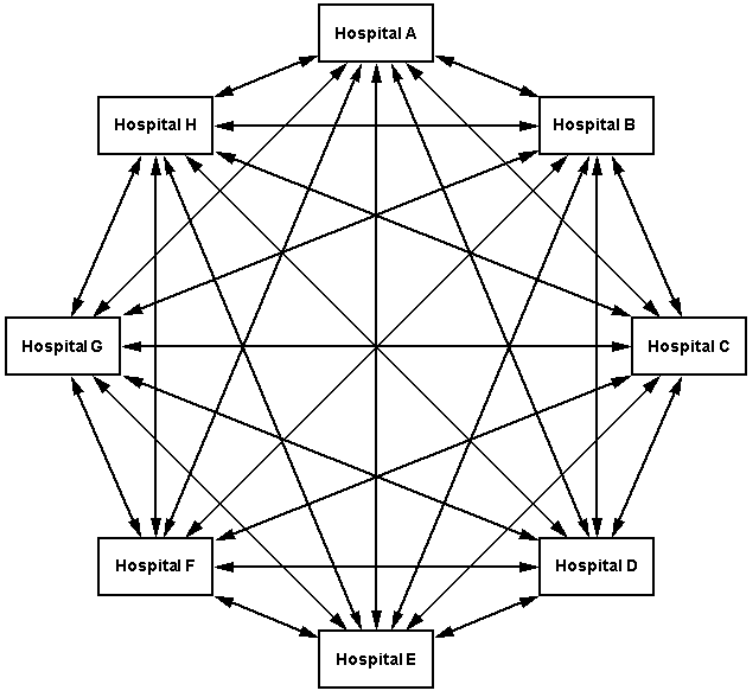
II. Data and Documents

University of California Extension
Sunnyvale, June 10, 1999

Jon Bosak
Sun Microsystems

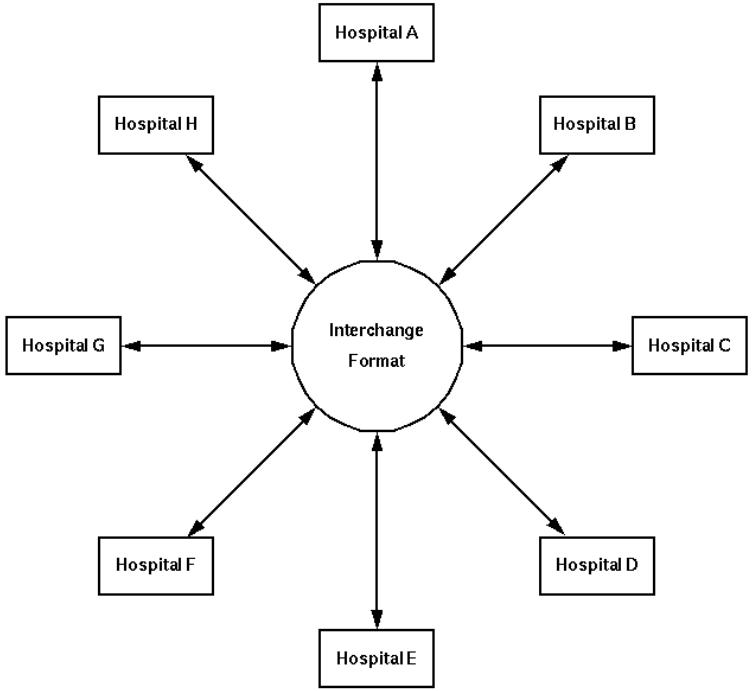
XML Data Scenarios

1: Loosely-coupled data exchange



Every organization uses a different database schema. The interchange problem is combinatorial.

The role of a hub format



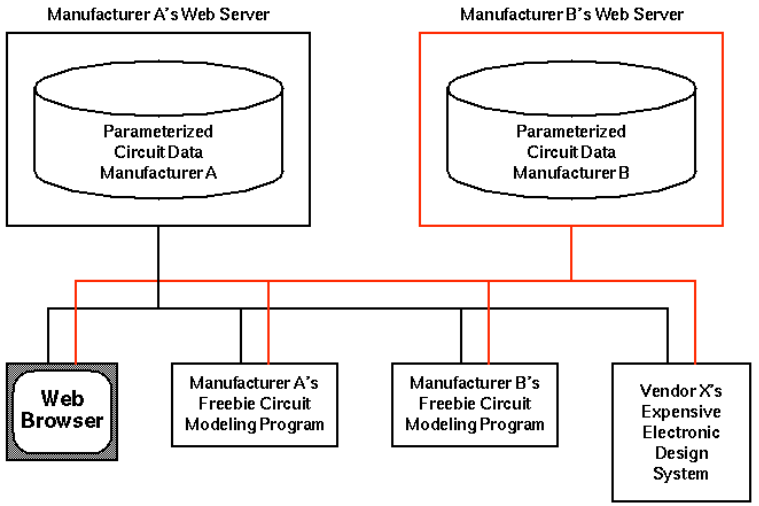
Conversion is reduced to one input program and one output program for each organization regardless of the number of organizations.

Other loosely-coupled data scenarios

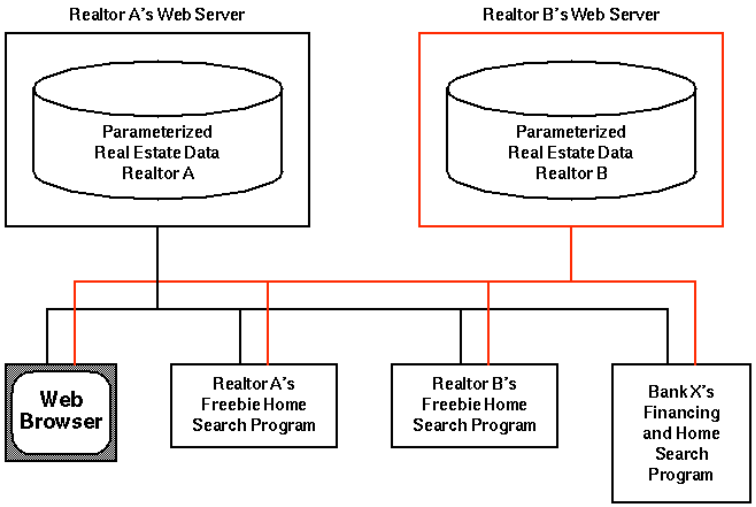
- Financial services (banking, securities, insurance)
- Government
- Legal publishing
- Pharmaceuticals (drug approval process)
- Collaborative design efforts
- **Electronic commerce**

2: Client-side processing

Example: semiconductor data



This is a widely applicable model!



Other client-side processing applications

- Design applications where the a designer considers various alternatives: electronics, engineering, architecture, etc.
- Scheduling applications where a customer explores various possibilities: airlines, trains, buses, and subways; restaurants, movies, plays, and concerts
- Commercial applications that allow consumers to explore sales alternatives: real estate, automobiles, appliances, etc.

There is an obvious connection with Java here!

3: User views of the data

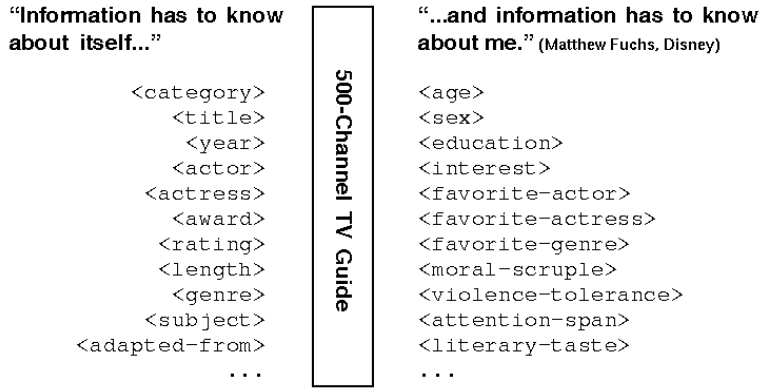
- Different views of whole documents
  - Novice view vs. expert view
  - Outline vs. content
  - Generated Tables of Contents
- Different views of document components
  - Bar graph vs. pie chart
  - Totals by region vs. totals by business unit

Why HTML can't handle multiple views:

1. Fixed tag set is not rich enough to support alternative views of the same data
2. Fixed presentation of each element makes it hard to change the way data is rendered

4: Web agents

Example: the 500-channel TV guide



Once again, a category of applications depends on the ability to standardize on a form of data representation for a particular industry or problem domain.

5: Collection management

Collection management is what originally put XML parsers in both major Web browsers.

RDF is a standardized XML language for representing web objects in a **directed labeled graph** (DLG).

- W3C initiative
- Starting to be implemented in Netscape software
- Will become increasingly important

XML Document Exchange

Documents vs. data

This is essentially a difference in purpose.

- Data
  - Order-independent
  - Shallow structure
  - Low text/tag ratio
- Documents
  - Order-dependent
  - Deep structure
  - High text/tag ratio: requires mixed content

```
<foo>This is <emph>mixed</emph> content.</foo>
```

Supporting documents is much harder than supporting data. XML is becoming ubiquitous because it can serve **both** purposes.

A brief history of documents

- **4000 BC:** inventories
- **3500 BC:** signed inventories
- **3000 BC:** signed, labeled inventories (contracts, credit and debit ledgers, labor accounting, deeds, etc.)
- Every other application of writing comes **later**

In the beginning, the typical document was a **business** document.

Today, the typical document is **still** a business document.

More recent developments in documents

- **Classical era:** Spaces between words
- **Medieval era:** Codices (i.e., books) instead of scrolls
  - Random access to contents
  - Page layout

The eternal document

- Despite many changes in media, business documents themselves haven't changed fundamentally for centuries
- Most other documents haven't changed much, either

People just like to do some basic things by exchanging documents. *They will continue to do so.*

In particular, electronic commerce will be accomplished through the exchange of documents, because only a *displayed document* can be a contract.

Requirements for documents

1. You have to be able to exchange them.

   GOT THAT (XML)

2. You have to be able to display them.

   YOU ARE HERE (CSS)

3. You have to be able to format them automatically at a level that will satisfy a graphic designer.

   THIS COMES NEXT (XSL)

4. You have to know what they mean.

   THIS IS THE REALLY HARD PART

Basic methods of XML document display

- Procedural (scripts, programs)
  - Very powerful
  - Very difficult to manage on a large scale (e.g., editing)
- Declarative (stylesheets)
  - More limited (though still very powerful)
  - Vastly more manageable and user-friendly

The evolutionary trend in display technology is away from the procedural and stream-oriented and toward the declarative and object-oriented (example: PostScript vs. PDF)

The delusion that print formatting is passé

- As displays improve, online formatting will stop being a subset of print formatting
- In fact, online formatting will be recognized as a **superset** of print formatting
  - Within the next decade, display quality will come to equal print quality
  - Given the capability, online designers will **demand** print layout control for the screen
  - This means complete control over formatting and page geometry **even when users change the aspect ratio of the page**
  - And it adds other new dimensions to traditional print requirements: animation and interactivity
- XSL must provide not only powerful transformational ability but also powerful formatting ability

Requirements for XSL

- Structural transformation to (X)HTML for online display in the short term
- Complex layout and typography for print output and online display in the long term

Current problem: lots of implementation work happening in XSL transformation (XSLT), very little happening in XSL formatting.

- XML rendering is limited to the CSS level (no knowledge of page or column boundaries, multiple text flows, etc.)
- XML is trapped in middleware, and we are **back where we started.**

Bright spots: Interleaf, FOP (James Tauber)