Using XML in the real world

Lou Burnard and Sebastian Rahtz

July 2001

Using XML in the Real World

1. What is XML *for*?
2. How is it best used?
3. What tools are available?

What is XML for?

1. exchanging information
   (a) between people
   (b) between people and machines
   (c) between machines
2. preserving information
   (a) without usage-dependency
   (b) without medium-dependency
   (c) independent of time, space, and language

Data reuse

Digital data can be reusable only to the extent that it is independent

1. of application
2. of platform
3. of software environment

XML therefore is the key to data reuse

Delivering information

XML is a good way of representing information.
But how about

1. delivering XML content on the web
2. ... and on paper
3. storing and managing XML documents
4. ... and virtual documents

Can we get the best of both worlds?

What tools do we need?

1. Appropriately expressive languages (eg TEI XML)
2. Syntax-checking document creation tools (aka Editors)
3. Document transformation tools
4. Document delivery tools
5. Document storage and management tools
6. Programming interfaces for a variety of languages

Generic languages

- DOM: Document Object Model Level 2;
- XML Schema (description of structures and data types);
- XPath: addressing parts of an XML document;
- XSLT: transforming XML documents for use with XSL;
- XSL: extensible stylesheet language;
- XLink: XML Linking Language;
- XPointer: XML Pointer Language.

... specialised (but generic) languages ...

1. SVG: scalable vector graphics;
2. MathML: Mathematical Markup Language;
3. RDF: Resource Description Framework;
4. SMIL: Synchronised Multimedia Integration Language

... etc etc etc

Document creation and editing

There's an ever expanding choice of XML editing tools:

1. Plain text editors, typing < and > by hand (e.g. Notepad)
2. Customised plain text editors, with built in tagging (e.g. Notetab)
3. Customised programming editors (notably GNU Emacs)
4. Word processors with XML add-ons (e.g. WordPerfect)
5. Data-oriented XML editors (eg XML Spy)
6. Document-oriented XML editors (eg XMetal)

And there's also the XML that gets generated without anyone noticing...

Document transformation tools

A *stylesheet* allows you to define how XML elements are to be transformed.

1. Extensible Style Language/Transformation (XSLT): fully-featured transformation language
2. Cascading Style Sheets (CSS): allows you to add formatting styles (only) to your document;
3. A variety of proprietary stylesheet languages also exists, tied to specific software;
4. Or you can use whatever software you like to map XML into something else (e.g. LaTeX, nroff, RTF, Framemaker)

Transformation tools

XSLT-based

1. Many, but varying in implementation level: we currently recommend saxon

proprietary

2. Legacy SGML systems like Balise, Omnimark; new scripting schemes like XML Script

generic software

3. easier to develop with XML-aware libraries, written to a standard API such as DOM

Typical transformation jobs

1. Render `<foo>` elements in italics
2. Render `<foo>` elements within `<bar>` elements in italics
3. Insert `Foo number` and the value of its number attribute in front of every foo
4. Indent every `<p>` element by 1 em, except for the first one in a `<div>`
5. Take the first `<head>` element inside each `<div>` and add it to a table of contents

XML parsers and validators

Embedded or free standing, validation is an integral part of XML document processing. There are lots of products, both free and commercial:

1. in Java from Sun, Oracle, and IBM as well as individuals
2. in C, embedded in Perl and various applications like Netscape
3. in C++ from IBM
4. something in more or less any language you like, from Python to Dylan

...plus all the existing SGML software

Processing strategies

An XML document is a serialized tree structure.
How should it be processed?
There are three currently favoured approaches:

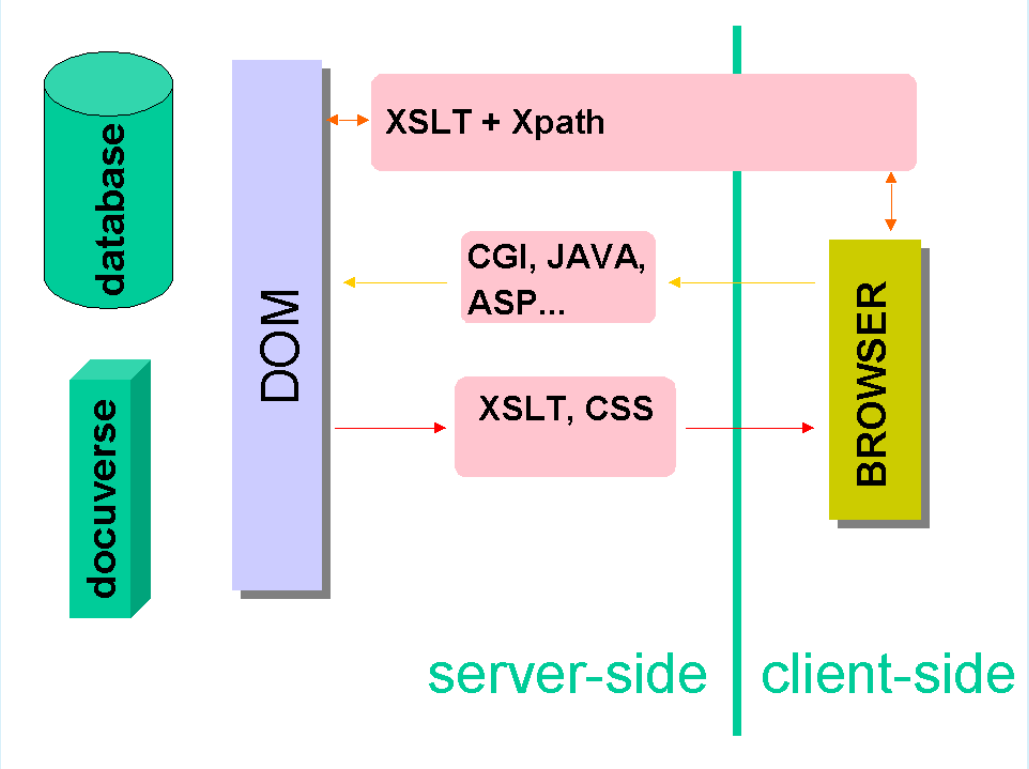
1. event-based (e.g. SAX)
2. tree-based (e.g. DOM)
3. declarative or functional (e.g. XSLT)

XML on the web

Eventually, all web user agents (browsers) will be XML aware! Until they are, we have to choose :

1. transform XML to HTML on the server (statically)
2. transform XML to HTML on the server (dynamically, using a servlet)
3. render XML on the client using CSS or dynamically with some kind of plugin

XML on the web: typical architecture



XML on paper

The combination of XML, XSL-T and a good FO-engine could do away with the need for expensive proprietary DTP and word processing systems

It hasn't happened yet, but it might...

Storage strategies

Data has to be stored somewhere. How should XML data be managed? There are several possibilities:

1. as discrete XML documents
2. within any convenient DBMS
3. within an XML repository

XML documents

In the traditional docucentric world...

1. information is stored in XML documents, somewhere, and in some form
2. entities give some degree of modularity
3. but there has to be centralized naming and management for version control, integrity, etc.

```
<!ENTITY doc1 SYSTEM "docs/frag1.xml">
<!ENTITY doc2 SYSTEM "docs/frag2.xml">
```

```
<?xml version="1.0" ?>
<!DOCTYPE theDoc SYSTEM "theDTD.dtd" [
  <!ENTITY % theDocList SYSTEM "theDocs.ent">
  %theDocList; ]>
<theDoc>
&doc1; &doc2;
</theDoc>
```

The docucentric world

Good points:

1. conceptually clear
2. robust and portable

Less good points:

1. *Everything* must be an XML entity
2. may appear inflexible or redundant

Virtual documents

Storage is a special kind of processing, like formatting, requiring a transformation in and out of some storage format. So we could

1. store information in non-XML formats (optimized for specific functions, e.g. text retrieval or relational tables)
2. recover all and only the information needed from the store in the form of a dynamically-generated XML document/fragment
3. in an XML repository, access should be in XML terms; at present, there is usually a need for some mapping process

XML databases: the options

1. Store some information as relations, and some as XML (e.g. ProtCem)
2. Store the XML structure as relations but expose only the XML view (e.g. Phelix)
3. Store and expose only XML (e.g. Meerkat and other RSS-based services)

DBMS or XML?

Do you have to choose?

1. The argument from history
   (a) flatfiles gave way to network DBMS
   (b) network DBMS gave way to relational
   (c) will relational DBMS give way to oodbs?
2. Getting the best of both worlds
   (a) DBMS are good at storing and managing *relations*
   (b) the equivalent XML technologies are not yet mature
   (c) but DBMS can be cajoled into presenting their contents in XML terms

Delivery strategies

1. Our goal is fast and efficient access to any subtree of the docuverse, of any size
2. XPATH has an adequately rich semantics
3. XSL-T has an adequately rich syntax (we think)
4. The rest is a Simple Matter of Programming...

Delivery strategies (today)

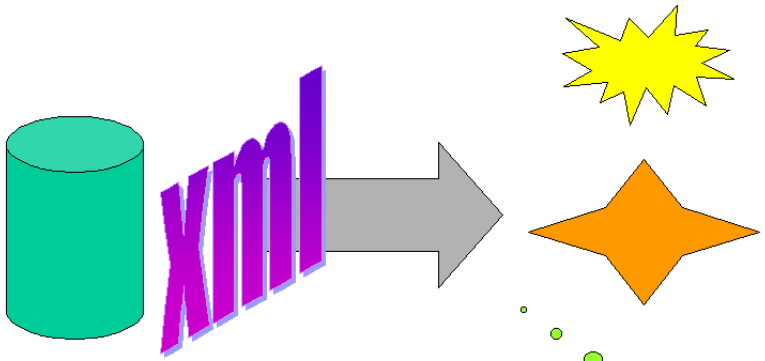
Small-scale solution Use XSLT and XPATH expressions

Untidy solution Store XML in conventional database and do textual search

High-tech solution Pre-index all text in all elements, and provide one-off front-end application

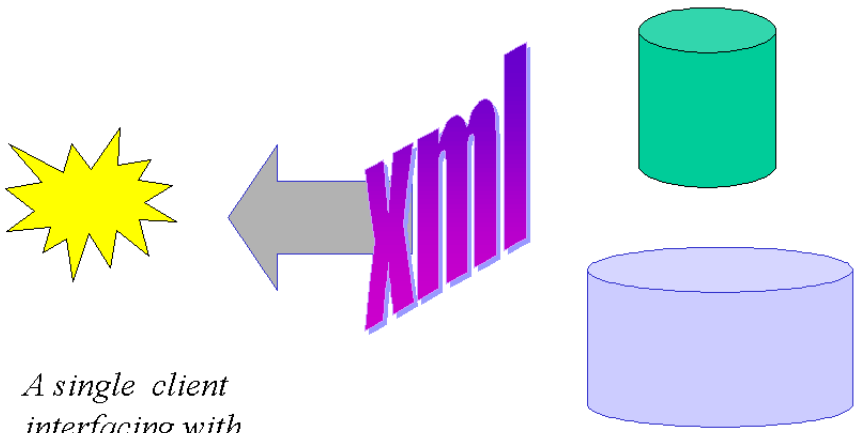
Low-tech solution Use XML-ified grep-like utility to search documents (LTXML tools)

Heterogeneity one way ...



Heterogenous clients interfacing with a single database

... or another



A single client interfacing with heterogenous databases

Development strategies

1. XML began as a way of smuggling SGML onto the web...
2. ... but seems to have taken over as the industry's driving force
3. Where will XML have taken us in the next few years?
4. What should we expect to be able to do?