Today’s Lecture

• Background: documents (SGML/HTML) and databases (structured and semistructured data)
• XML Basics and Document Type Descriptors
• These slides were adapted from slides developed at the University of Pennsylvania (by Peter Buneman and Susan Davidson)

Part I: Background

What’s the difference between documents and databases?
80% of the world’s data does NOT reside in a database!
**HTML**

- Lingua franca for publishing hypertext on the World Wide Web
- Designed to describe how a Web browser should arrange text, images and push-buttons on a page.
- Easy to learn, but does not convey structure.
- Fixed tag set.

```
<HTML>
  <HEAD><TITLE>Welcome to the XML course</TITLE></HEAD>
  <BODY>
    <H1>Introduction</H1>
    <IMG SRC="dragon.jpg" WIDTH="200" HEIGHT="150">
  </BODY>
</HTML>
```

```html
Opening tag

Text (PCDATA)

Closing Tag

Attribute name

Attribute value

"Bachelor" tag
```

**Documents vs Databases**

**Document world**
- plenty of small documents
- usually static
- implicit structure
  - section, paragraph, toc,
  - tagging
- human friendly
- content
  - form/layout, annotation
- Paradigms
  - "Save as", wysiwyg
- meta-data
  - author name, date, subject

**Database world**
- fewer large databases
- usually dynamic
- explicit structure (schema)
- records
- machine friendly
- content
  - schema, data, methods
- Paradigms
  - Atomicity, Concurrency, Isolation, Durability
- meta-data
  - schema description

Winter 2003
What to do with them

<table>
<thead>
<tr>
<th>Documents</th>
<th>Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>• editing</td>
<td>• updating</td>
</tr>
<tr>
<td>• printing</td>
<td>• cleaning</td>
</tr>
<tr>
<td>• spell-checking</td>
<td>• querying</td>
</tr>
<tr>
<td>• counting words</td>
<td></td>
</tr>
<tr>
<td>• retrieving (IR)</td>
<td></td>
</tr>
<tr>
<td>• searching</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• composing/transforming</td>
</tr>
</tbody>
</table>

Thin red line

• The line between the document world and the database world is not clear.
• In some cases, both approaches are legitimate.
• An interesting middle ground is data formats -- of which XML is an example

• Examples
  – Personal address book
  – SWISS-PROT
  – ASN.1
Personal address book over 20 years

1977
N Achison, Malcolm
F Dr, M.P, Achison
A Dept, of Computer Science
A University of Edinburgh
A Kings Buildings
A Edinburgh E12 8QQ
A Scotland
T 01-123-4565 ext - 459 (work)
T 01-124-7570 (home)

P Albani, Pietro
F Prof. Pietro Albani
A Dip. Informatica e Sistemistica
A Università di Roma La Sapienza

1980
N Achison, Malcolm
F Dr, M.P, Achison
A Dept, of Computer Science
... T 031-667-7570 (home)
C mpa@uk.ac.edi

1997
N Achison, Malcolm
F Prof. M.P, Achison
A Department of Computing Science...
... T 01-1-667-7570 (home)
C mpa@dcs.gla.ac.uk
W http://www.dcs.gla.ac.uk/mpa

1990
N Achison, Malcolm
F Prof. M.P, Achison
A Dept, of Computing Science
A University of Glasgow
A Llybank Gardens
A Glasgow G2 8QQ
A Scotland
T 014-339-8855 ext. 4359
T 041-357-3787 (private)
T 01-667-7570 (home)
X 041-339-0090
C mpa@uk.ac.glas.

Swissprot

ID 11SB_CUCMA STANDARD; PRT; 480 AA.
AC P13744;
DT 01-JAN-1990 (REL. 13, CREATED)
DT 01-JAN-1990 (REL. 13, LAST SEQUENCE UPDATE)
DT 01-NOV-1990 (REL. 16, LAST ANNOTATION UPDATE)
DE 115 GLOBULIN BETA SUBUNIT PRECURSOR.
OS CUCURBITA MAXIMA (PUMPKIN) (WINTER SQUASH).
OC EUKARYOTA; PLANTA; EMBRYOPHYTA; ANGIOSPERMAE; Dicotyledoneae;
OC VIOLALES; CUCURBITACEAE.
RN [1]
RP [SEQUENCE FROM N.A.]
RC STRAIN =CV. KUROKAWA AMAKURI NANKIN;
RX MEDLINE: 88156744.
RA HAYASHI M., MORI H., NISHIMURA M., AKAZAWA T., HANANISHIMURA I.;
RN [2]
RA OHMIYA M., HARA I., MASTUBARA H.;
Swissprot (cont’d)

CC ← FUNCTION: THIS IS A SEED STORAGE PROTEIN.
CC ← SUBUNIT: HEXAMER; EACH SUBUNIT IS COMPOSED OF AN ACIDIC AND A
CC BASIC CHAIN DERIVED FROM A SINGLE PRECURSOR AND LINKED BY A
CC DISULFIDE BOND.
CC ← SIMILARITY: TO OTHER 11S SEED STORAGE PROTEINS (GLOBULINS).
DR EMBL: M36407; G167492: -.
DR PIR: S00366; FWUL1B.
DR PROSITE: PS00305; 11S_SEED_STORAGE: 1.
KW SEED STORAGE PROTEIN; SIGNAL.
FT SIGNAL 1 21
FT CHAIN 22 480 11S GLOBULIN BETA SUBUNIT.
FT CHAIN 22 295 GAMMA CHAIN (ACIDIC).
FT CHAIN 297 480 DELTA CHAIN (BASIC).
FT MOD_RES 22 22 PYRROLIDONE CARBOXYLIC ACID.
FT DISULFID 124 303 INTERCHAIN (GAMMA-DELTA) (POTENTIAL).
FT CONFLICT 27 27 S -> E (IN REF. 2).
FT CONFLICT 30 30 E -> S (IN REF. 2).
SQ SEQUENCE 480 AA; 54625 MW; DS15DD6E CRC32;
MARSSFTFL CLAVFINGCL SQEQOQPWE PQGSEVWDQH RYQSPRACRL ENLRAQDPVR RAEEAIITE WDOQNDNEFO CAGVNMHRHT IRPKGLLPG FONAPKLFV AOGFCIRGIA IPGCAETYQT DLRSSQ6AGS AFKDQH6QXKIR PREGDLVLV PAGSHIPWYMN RGQSDLV1LV

Winter 2003

II. XML Basics and Document Type Descriptors

Winter 2003
The Structure of XML

- XML consists of \textit{tags} and \textit{text}

- Tags come in pairs \texttt{<date> \ldots </date>}

- They must be properly nested
  \texttt{<date> <day> \ldots </day> \ldots </date> --- good}
  \texttt{<date> <day> \ldots </day>... </date> --- bad}

(You can't do \texttt{<i> \ldots <b> \ldots </i> \ldots <b> in HTML})

XML text

XML has only one "basic" type -- text.

It is bounded by tags e.g.
\texttt{<title> The Big Sleep </title>}
\texttt{<year> 1935 </year> --- 1935 is still text}

XML text is called PCDATA (for parsed character data). It uses a 16-bit encoding
Later we shall see how new types are specified by XML-data
XML structure

Nesting tags can be used to express various structures. E.g. A tuple (record):

```xml
<person>
  <name> Malcolm Atchison </name>
  <tel> (215) 898 4321 </tel>
  <email> mp@dcs.gla.ac.sc </email>
</person>
```

XML structure (cont.)

- We can represent a list by using the same tag repeatedly:

```xml
<addresses>
  <person> ... </person>
  <person> ... </person>
  <person> ... </person>
  ...
</addresses>
```
**Terminology**

The segment of an XML document between an opening and a corresponding closing tag is called an *element*.

```xml
<person>
  <name>Malcolm Atchison</name>
  <tel>(215) 898 4321</tel>
  <tel>(215) 898 4321</tel>
  <email>mp@dcs.gla.ac.sc</email>
</person>
```

**XML is tree-like**

```
person
   /--- name
   |     Malcolm Atchison
   
   /--- tel
   |     (215) 898 4321
   
   /--- tel
   |     (215) 898 4321
   
   /--- email
   |     mp@dcs.gla.ac.sc
```

*Semistructured data models typically put the labels on the edges*
Mixed Content

An element may contain a mixture of sub-elements and PCDATA

```xml
<airline>
  <name> British Airways </name>
  <motto>
    World’s <dubious> favorite</dubious> airline
  </motto>
</airline>
```

Data of this form is not typically generated from databases. It is needed for consistency with HTML.

A Complete XML Document

```xml
<?xml version="1.0"?>
<person>
  <name> Malcolm Atchison </name>
  <tel> (215) 898 4321 </tel>
  <email> mp@dcs.gla.ac.sc </email>
</person>
```
Two ways of representing a DB

projects:

<table>
<thead>
<tr>
<th>title</th>
<th>budget</th>
<th>managedBy</th>
</tr>
</thead>
</table>

employees:

<table>
<thead>
<tr>
<th>name</th>
<th>ssn</th>
<th>age</th>
</tr>
</thead>
</table>

Project and Employee relations in XML

Projects and employees are intermixed

```
<db>
  <project>
    <title> Pattern recognition </title>
    <budget> 10000 </budget>
    <managedBy> Joe </managedBy>
  </project>
  <employee>
    <name> Joe </name>
    <ssn> 345556 </ssn>
    <age> 34 </age>
  </employee>
  <employee>
    <name> Sandra </name>
    <ssn> 2234 </ssn>
    <age> 35 </age>
  </employee>
  <project>
    <title> Auto guided vehicle </title>
    <budget> 70000 </budget>
    <managedBy> Sandra </managedBy>
  </project>
  ...
</db>
```
Project and Employee relations in XML (cont’d)

Employees follow projects

```
<db>
  <projects>
    <project>
      <title>Pattern recognition</title>
      <budget>10000</budget>
      <managedBy>Joe</managedBy>
    </project>
    <project>
      <title>Auto guided vehicles</title>
      <budget>70000</budget>
      <managedBy>Sandra</managedBy>
    </project>
  </projects>
</db>
```

Project and Employee relations in XML (cont’d)

Or without "separator" tags ...

```
<db>
  <projects>
    <title>Pattern recognition</title>
    <budget>10000</budget>
    <managedBy>Joe</managedBy>
    <title>Auto guided vehicles</title>
    <budget>70000</budget>
    <managedBy>Sandra</managedBy>
  </projects>
</db>
```
Attributes

An (opening) tag may contain attributes. These are typically used to describe the content of an element

```
<entry>
  <word language = "en"> cheese </word>
  <word language = "fr"> fromage </word>
  <word language = "ro"> branza </word>
  <meaning> A food made ... </meaning>
</entry>
```

Attributes (cont’d)

Another common use for attributes is to express dimension or type

```
<picture>
  <height dim = "cm"> 2400 </height>
  <width dim = "in"> 96 </width>
  <data encoding = "gif" compression = "zip"> M05-.+C@O2!G96YE<FEC ... 
</data>
</picture>
```

A document that obeys the “nested tags” rule and does not repeat an attribute within a tag is said to be well-formed.
When to use attributes

It's not always clear when to use attributes

```
<person ssn="123 45 6789">
  <name> F. MacNeil </name>
  <email> fmacn@dcs.barra.ac.sc </email>
</person>

<person>
  <ssn> 123 45 6789 </ssn>
  <name> F. MacNeil </name>
  <email>
    fmacn@dcs.barra.ac.sc
  </email>
</person>
```

Using IDs

```
<family>
  <person id="jane" mother="mary" father="john">
    <name> Jane Doe </name>
  </person>
  <person id="john" children="jane jack">
    <name> John Doe </name>
  </person>
  <person id="mary" children="jane jack">
    <name> Mary Doe </name>
  </person>
  <person id="jack" mother="mary" father="john">
    <name> Jack Doe </name>
  </person>
</family>
```
ODL schema

class Movie {
    ( extent Movies, key title )
    {
        attribute string title;
        attribute string director;
        relationship set<Actor> casts
            inverse Actor::acted_In;
        attribute int budget;
    };

class Actor {
    ( extent Actors, key name )
    {
        attribute string name;
        relationship set<Movie> acted_In
            inverse Movie::casts;
        attribute int age;
        attribute set<string> directed;
    };

An example

<db>
<movie id="m1">
    <title>Waking Ned Divine</title>
    <director>Kirk Jones III</director>
    <cast idrefs="a1 a3"></cast>
    <budget>100,000</budget>
</movie>
<movie id="m2">
    <title>Dragonheart</title>
    <director>Rob Cohen</director>
    <cast idrefs="a2 a9 a21"></cast>
    <budget>110,000</budget>
</movie>
<movie id="m3">
    <title>Moondance</title>
    <director>Dagmar Hirz</director>
    <cast idrefs="a1 a8"></cast>
    <budget>90,000</budget>
</movie>
<actor id="a1">
    <name>David Kelly</name>
    <acted_In idrefs="m1 m3 m78" />
</actor>
<actor id="a2">
    <name>Sean Connery</name>
    <acted_In idrefs="m2 m9 m11" />
</actor>
<actor id="a3">
    <name>Ian Bannen</name>
    <acted_In idrefs="m1 m35" />
</actor>
</db>
Document Type Descriptors

- Document Type Descriptors (DTDs) impose structure on an XML document.
- There is some relationship between a DTD and a schema, but it is not close -- hence the need for additional “typing” systems.
- The DTD is a syntactic specification.

Example: The Address Book

```xml
<person>
  <name>MacNiel, John</name>
  <greet>Dr. John MacNiel</greet>
  <addr>1234 Huron Street</addr>
  <addr>Rome, OH 98765</addr>
  <tel>(321) 786 2543</tel>
  <fax>(321) 786 2543</fax>
  <tel>(321) 786 2543</tel>
  <email>jm@abc.com</email>
</person>
```

- Exactly one name
- At most one greeting
- Mixed telephones and faxes
- As many as needed (in order)
- As many as needed
Specifying the structure

The structure of a person entry can be specified by

\[ \text{name, greet?, addr*, (tel | fax)*, email*} \]

This is known as a *regular expression*. Why is it important?

---

Regular Expressions

Each regular expression determines a corresponding *finite state automaton*. Let's start with a simpler example:

\[ \text{name, addr*, email} \]

This suggests a simple parsing program
Another example

name, address*, (tel | fax)*, email*

Adding in the optional greet further complicates things

A DTD for the address book

```xml
<!DOCTYPE addressbook [ 
<!ELEMENT addressbook (project*)>
<!ELEMENT project 
  (name, greet?, address*, (fax | tel)*, email*)>
<!ELEMENT name   (#PCDATA)>
<!ELEMENT greet  (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT tel    (#PCDATA)>
<!ELEMENT fax    (#PCDATA)>
<!ELEMENT email  (#PCDATA)>
]>```
Our relational DB revisited

projects:

<table>
<thead>
<tr>
<th>title</th>
<th>budget</th>
<th>managedBy</th>
</tr>
</thead>
</table>

employees:

<table>
<thead>
<tr>
<th>name</th>
<th>ssn</th>
<th>age</th>
</tr>
</thead>
</table>

Two DTDs for the relational DB

```xml
<!DOCTYPE db [
  <!ELEMENT db     (projects,employees)*>
  <!ELEMENT projects  (project)*>
  <!ELEMENT employees (employee)*>
  <!ELEMENT project   (title, budget, managedBy)*>
  <!ELEMENT employee  (name, ssn, age)*>

...]
```

```xml
<!DOCTYPE db [
  <!ELEMENT db     (project | employee)*>
  <!ELEMENT project   (title, budget, managedBy)>  
  <!ELEMENT employee  (name, ssn, age)>  

...]
```
Recursive DTDs

<DOCTYPE genealogy [
  <!ELEMENT genealogy (person*)>
  <!ELEMENT person ( 
    name, 
    dateOfBirth, 
    person, -- mother 
    person )> -- father
  ...
  ]>
What is the problem with this?

Recursive DTDs cont’d.

<DOCTYPE genealogy [
  <!ELEMENT genealogy (person*)>
  <!ELEMENT person ( 
    name, 
    dateOfBirth, 
    person?, -- mother 
    person? )> -- father
  ...
  ]>
What is now the problem with this?
Some things are hard to specify

Each employee element is to contain name, age and ssn elements in some order.

```xml
<!ELEMENT employee
  ( (name, age, ssn) | (age, ssn, name) |
   (ssn, name, age) | ...
 )>
```

Suppose there were many more fields!

Summary of XML regular expressions

- `A` The tag A occurs
- `e1,e2` The expression e1 followed by e2
- `e*` 0 or more occurrences of e
- `e?` Optional -- 0 or 1 occurrences
- `e+` 1 or more occurrences
- `e1 | e2` either e1 or e2
- `(e)` grouping
Specifying attributes in the DTD

```xml
<!ELEMENT height (#PCDATA)>
<!ATTLIST height
dimension CDATA #REQUIRED
accuracy CDATA #IMPLIED >
```

The dimension attribute is required; the accuracy attribute is optional.

CDATA is the “type” of the attribute -- it means string.

Specifying ID and IDREF attributes

```xml
<!DOCTYPE family [ 
<!ELEMENT family (person)*>
<!ELEMENT person (name)> 
<!ELEMENT name (#PCDATA)>
<!ATTLIST person
 id ID #REQUIRED
 mother IDREF #IMPLIED
 father IDREF #IMPLIED
 children IDREFS #IMPLIED>
]
```
Some conforming data

```xml
<family>
  <person id="jane" mother="mary" father="john">
    <name> Jane Doe </name>
  </person>
  <person id="john" children="jane jack">
    <name> John Doe </name>
  </person>
  <person id="mary" children="jane jack">
    <name> Mary Doe </name>
  </person>
  <person id="jack" mother="mary" father="john">
    <name> Jack Doe </name>
  </person>
</family>
```

Consistency of ID and IDREF attribute values

- If an attribute is declared as ID
  - the associated values must all be distinct (no confusion)
- If an attribute is declared as IDREF
  - the associated value must exist as the value of some ID attribute (no dangling “pointers”)
- Similarly for all the values of an IDREFS attribute
- ID and IDREF attributes are not typed
An alternative specification

<!DOCTYPE family [ 
  <!ELEMENT family (person)> 
  <!ELEMENT person (mother?, father?, children, name)> 
  <!ATTLIST person id ID #REQUIRED> 
  <!ELEMENT name (#PCDATA)> 
  <!ELEMENT mother EMPTY> 
  <!ATTLIST mother idref IDREF #REQUIRED> 
  <!ELEMENT father EMPTY> 
  <!ATTLIST father idref IDREF #REQUIRED> 
  <!ELEMENT children EMPTY> 
  <!ATTLIST children idrefs IDREFS #REQUIRED> ]>

Winter 2003

The revised data

<family>
  <person id = "jane"> 
    <name> Jane Doe </name> 
    <mother idref = "mary"></mother> 
    <father idref = "john"></father> 
  </person> 
  <person id = "john"> 
    <name> John Doe </name> 
    <children idrefs = "jane jack"> </children> 
  </person> 
  ... 
</family>
A useful abbreviation

When an element has empty content we can use

\[
\text{<tag blah blah blah/> for \<tag blah blah blah\> </tag>}
\]

For example:

\[
\text{<family>}
\text{\<person id = "jane"/>}
\text{\<name> Jane Doe \</name>}
\text{\<mother idref = "mary"/>}
\text{\<father idref = "john"/>}
\text{\</person>}
\text{...}
\text{\</family>}
\]

Winter 2003

ODL schema

\[
\text{class Movie}
\text{\{ extent Movies, key title \}}
\text{\{}
\text{\attribute string title;}
\text{\attribute string director;}
\text{\relationship set<Actor> cast}
\text{\\\\\inverse Actor::acted_In;}
\text{\attribute int age;}
\text{\attribute set<string> directed;}
\text{\}}
\]

\text{class Actor}
\text{\{ extent Actors, key name \}}
\text{\{}
\text{\attribute string name;}
\text{\relationship set<Movie> acted_In}
\text{\\\\\inverse Movie::cast;}
\text{\attribute int age;}
\text{\}}
\]

Winter 2003
<!DOCTYPE db [ 
<!ELEMENT db (movie+, actor+)>
<!ELEMENT movie (title,director,cast,budget)>
<!ATTLIST movie id ID #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT director (#PCDATA)>
<!ELEMENT cast EMPTY>
<!ATTLIST cast idrefs IDREFS #REQUIRED>
<!ELEMENT budget (#PCDATA)>

Schema.dtd (cont’d)

<!ELEMENT actor (name, acted_In, age?, directed*)>
<!ATTLIST actor id ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT acted_In EMPTY>
<!ATTLIST acted_In idrefs IDREFS #REQUIRED>
<!ELEMENT age (#PCDATA)>
<!ELEMENT directed (#PCDATA)>
]>

Constraints on IDs and IDREFs

- ID stands for identifier. No two ID attributes with the same name may have the same value (of type CDATA)
- IDREF stands for identifier reference. Every value associated with an IDREF attribute must exist as an ID attribute value
- IDREFS specifies several (0 or more) identifiers

Connecting the document with its DTD

In line:
```xml
<?xml version="1.0"?>
<!DOCTYPE db [<!ELEMENT ...> ... ]>
<db> ... </db>
```

Another file:
```xml
<!DOCTYPE db SYSTEM "schema.dtd">
```

A URL:
```xml
<!DOCTYPE db SYSTEM "http://www.schemaauthority.com/schema.dtd">
```
Well-formed and Valid Documents

- *Well-formed* applies to any document (with or without a DTD): proper nesting of tags and unique attributes
- *Valid* specifies that the document conforms to the DTD: conforms to regular expression grammar, types of attributes correct, and constraints on references satisfied

DTDs v.s Schemas (or Types)

- By database (or programming language) standards DTDs are rather weak specifications.
  - Only one base type -- PCDATA
  - No useful “abstractions” e.g., sets
  - IDREFs are untyped. You point to something, but you don’t know what!
  - No constraints e.g., child is inverse of parent
  - No methods
  - Tag definitions are *global*
- Some of the XML extensions impose something like a schema or type on an XML document. This has been taken up in XMLSchema.
Summary

- XML is a new data format. Its main virtues are widespread acceptance and the (important) ability to handle semistructured data (data without schema)
- DTDs provide some useful syntactic constraints on documents. As schemas they are weak
- How to query XML document?
- How to store large XML documents?
- How to map between XML and other representations?