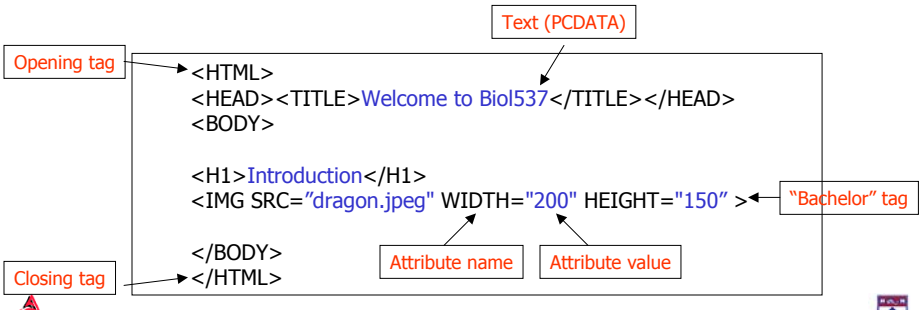
Introduction to XML

Susan B. Davidson
susan@cis.upenn.edu

HTML: The precursor of XML

- 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.



XML Anatomy

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
  <mastersthesis mdate="2002-01-03" key="ms/Brown92">
    <author>Kurt P. Brown</author>
    <title>PRPL: A Database Workload Specification Language</title>
    <year>1992</year>
    <school>Univ. of Wisconsin-Madison</school>
  </mastersthesis>
  <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
    <editor>Paul R. McJones</editor>
    <title>The 1995 SQL Reunion</title>
    <journal>Digital System Research Center Report</journal>
    <volume>SRC1997-018</volume>
    <year>1997</year>
    <ee>db/labs/dec/SRC1997-018.html</ee>
    <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
  </article>
```

Processing Instr.

Element

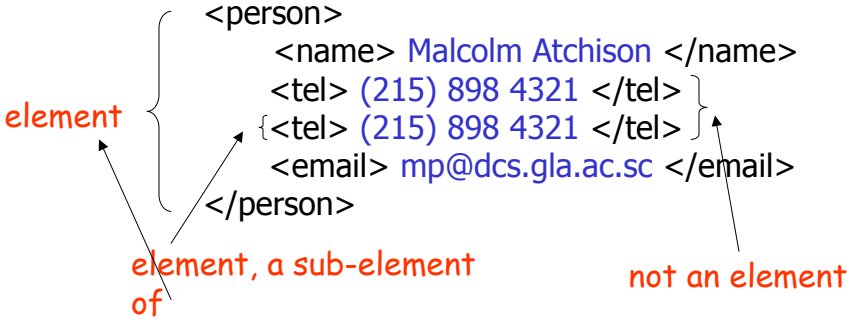
Open-tag

Attribute

Close-tag

Element Terminology

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



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>
</entry>
```

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$@02!G96YE<FEC ...
    </data>
</picture>
```

Attributes are unordered.

When to use attributes

It's not always clear when to use attributes.

They must be used to identify components of a document (IDs and IDREFs).

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

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

IDs and IDREFs

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


Well-Formed XML

A legal XML document – fully parseable by an XML parser

- All open-tags have matching close-tags (unlike so many HTML documents!), or a special:
  `<tag/>` shortcut for empty tags (equivalent to `<tag></tag>`
- Attributes (which are unordered, in contrast to elements) only appear once in an element
- There is a single root element
- XML is case-sensitive

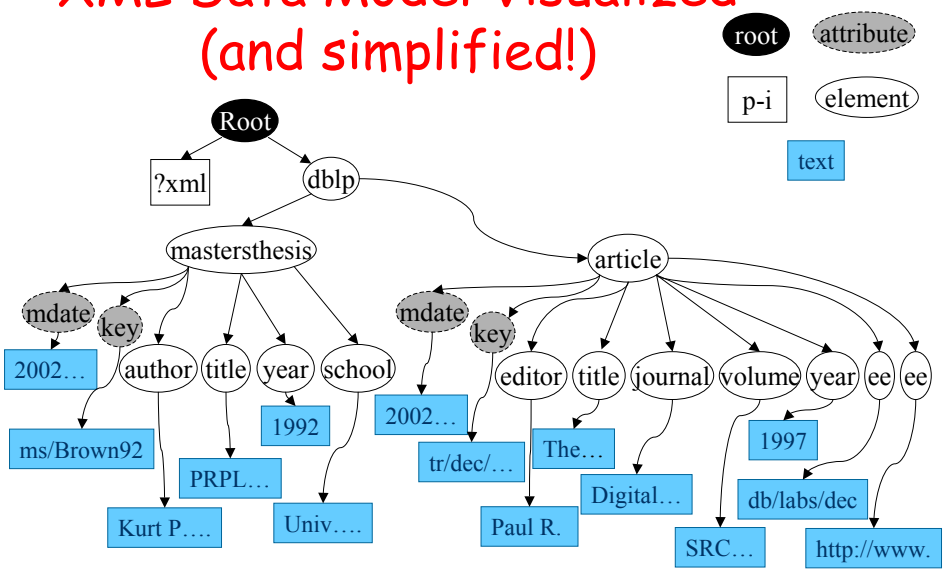
XML as a Data Model

XML "information set" includes 7 types of nodes:

- Document (root)
- Element
- Attribute
- Processing instruction
- Text (content)
- Namespace
- Comment

XML data model includes this, plus typing info, plus order info and a few other things

XML Data Model Visualized (and simplified!)



XML Easily Encodes Relations

Student-course-grade

sid	cno	exp-grade
1	570103	B
23	550103	A

```
<student-course-grade>
  <tuple><sid>1</sid><cno>570103</cno>
    <exp-grade>B</exp-grade></tuple>
  <tuple><sid>23</sid><cno>550103</cno>
    <exp-grade>A</exp-grade></tuple>
</student-course-grade>
```


But XML is More Flexible… "Non-First-Normal-Form" (NF^2)

```
<parents>
 <parent name="Jean" >
  <son>John</son>
  <daughter>Joan</daughter>
  <daughter>Jill</daughter>
 </parent>
 <parent name="Feng">
  <daughter>Felicity</daughter>
 </parent>
...
```

XML encodes Object Oriented Databases

Sample 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;
};
```

OODB instance in XML

```
<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 Hirtz</director>
    <cast idrefs="a1 a8"></cast>
    <budget>90,000</budget>
  </movie>
    :
```

```
<actor id="a1">
  <name>David Kelly</name>
  <acted_In idrefs="m1 m3 m78" >
  </acted_In>
</actor>
<actor id="a2">
   <name>Sean Connery</name>
   <acted_In idrefs="m2 m9 m11">
   </acted_In>
   <age>68</age>
</actor>
<actor id="a3">
    <name>Ian Bannen</name>
    <acted_In idrefs="m1 m35">
    </acted_In>
</actor>
  :
</db>
```

Document Type Definitions (DTDs)

A DTD is an EBNF grammar defining XML structure

- XML document specifies an associated DTD, plus the root element
- DTD specifies children of the root (and so on)

- There is *some* relationship between a DTD and a schema, but it is not close -- hence the need for additional "typing" systems as proposed in XMLSchema.
- The DTD is a *syntactic* specification.

Example: An Address Book

```
<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

As many address lines as needed (in order)

Mixed telephones and faxes

As many as needed

Specifying the structure

The structure of a person entry could be specified by the regular expression

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

Regular expressions are important because they have efficient parsers.

A DTD for an address book

```
<!DOCTYPE addressbook [
  <!ELEMENT addressbook (entry*)>
  <!ELEMENT entry
      (name, greet?, address*, (fax | tel)*, email*)>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT greet   (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT tel     (#PCDATA)>
  <!ELEMENT fax     (#PCDATA)>
  <!ELEMENT email   (#PCDATA)>
]>
```

DTDs aren't enough

DTDs capture grammatical structure, but have some drawbacks:

- Not themselves in XML – inconvenient to build tools for them
- Don't capture database datatype domains
- IDs aren't a good implementation of keys
  - Why not?
- No way of defining OO-like inheritance
- No way of expressing FDs

XML Schema

Aims to address the shortcomings of DTDs

… But an "everything including the kitchen sink" spec

Features:

- XML syntax
- Better way of defining keys using XPath
- Type subclassing that's more complex than in a programming language
  - Programming languages don't consider order of member variables!
  - Subclassing "by extension" and "by restriction"
- Domains and built-in datatypes

Simple XMLSchema example

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="mastersthesis" type="ThesisType"/>
<xsd:complexType name="ThesisType">
    <xsd:attribute name="mdate" type="xsd:date"/>
    <xsd:attribute name="key" type="xsd:string"/>
    <xsd:attribute name="advisor" type="xsd:string"/>
    <xsd:sequence>
        <xsd:element name="author" type="xsd:string"/>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="year" type="xsd:integer"/>
        <xsd:element name="school" type="xsd:string"/>
        <xsd:element name="committeemember"
          type="CommitteeType" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
```


Whither XML Schema?

- XMLSchema has syntax for keys (a set of attributes) and foreign keys
- Adopted by a few tools, but…
- The jury is still out on this spec
  - Considered too big, bulky, inconsistent
  - Still no way of expressing FDs, and the key mechanism isn't that elegant
    - Other people have defined more elegant schema languages, FD languages, and key languages (much of the work was here at Penn)
- But it's also at the core of the data model of the XQuery language and the other W3C specs…

Summary (XML)

- Serves as a document format that's better than HTML
  - Allows custom tags (e.g., used by MS Word, openoffice)
  - Supplement it with stylesheets (XSL) to define formatting
- Provides an exchange format for data (still need to agree on terminology)
- Basis for RDF format for describing libraries and the Semantic Web