Ch. 3: XML (eXtensible Markup Language)

- used to describe content rather than presentation
- Differs from HTML in at 3 different ways.
  - New tags may be defined at will
  - Structures may be nested to arbitrary depth
  - XML document may contain an optional description of its grammar
- Hopes to become a major standard for data exchange on the Web

3.1 Basic Syntax

3.1.1 XML Elements

element: piece of text bounded by user-defined matching tags:

```
<person>
  <name>Alan</name>
  <age>42</age>
  <email>agb@abc.com</email>
</person>
```

Note: - Element includes the start and end tag
- Quotation marks around strings have disappeared in XML! Because XML treats all data as text. This is referred to as PCDATA (Parsed Character Data).
- Empty elements: <married></married> can be abbreviated to <married/>

Collections are expressed using repeated structures.
Ex. The collection of all persons on the 4th floor:

```
<table> <description>People on the 4th floor</description>
 <people>
  <person>
   <name>Alan</name><age>42</age<<email>agb@abc.com</email>
  </person>
  <person>
   <name>Patsy</name><age>36</age><email>ptn@abc.com</email>
  </person>
 <person>
  <name>Ryan</name><age>58</age><email>rgz@abc.com</email>
 </person>
 </people>
</table>
```

3.1.2 XML Attributes

- Attributes define some properties of elements;
- Expressed as a name-value pair

```
<product>
  <name language="French">trompette six trous</name>
  <price currency="Euro">420.12</price>
  <address format="XLB56" language="French">
    <street>31 rue Croix-Bosset</street>
    <zip>92310</zip>
    <city>Sevres</city>
    <country>France</country>
  </address>
</product>
```

- As with tags, user may defiine any number of attributes;
- Attribute values must be enclosed within quotation marks.

Differences between attributes and tags:

A given attribute can occur only once within a tag
Its value is always a string;

On the other hand tags defining elements/sub-elements can repeat any number of times and their values may be string data or sub-elements.

In data exchange applications there is ambiguity on whether
to use attributes or elements:

```
<person name="Alan" age="42">
  <email>agb@abc.com</email>
</person>
```

or other combinations of attributes/elements such as

```
<person name="Alan" >
  <age>42</age>
  <email>agb@abc.com</email>
</person>
```

3.1.3 Well-formed XML documents

- tags must nest properly
- attributes must be unique within an element

Well formed XML documents will parse into a labeled tree always.

3.2 XML and semi-structured data

The basic XML syntax is perfectly suited to
describe semi-structured data.

```
<person>
  <name>Alan</name>
  <age>42</age>
  <email>agb@abc.com</email>
</person>
```

corresponding ssd-expression:

```
 {person: {name: "Alan", age: 42, email: "agb@abc.com"}}
```

Tree structured structures:

one-to-one correspondence.

The following transformation from ssd-expressions to XML is straightforward:

T(atomicvalue) = atomicvalue
T({l1:v1, ..., ln:vn}) = <l1>T(v1)</l1>...<ln>T(vn)</ln>

Beyond this simple analogy, however, XML and semi-structured data are not easy to reconcile.

3.2.1 XML Graph Model

For tree data, XML element and ssd-expressions can be expressed as trees with the following distinction:

- XML element: labeled nodes
- ssd-expressions: labeled edges

Easy to transform XML tree into ssd tree.

Simply "lift" the node edgesup one level; create a new root node. Figure 3.3

3.2.2 XML references

Use id attribute to defined a reference (similar to oids);

Use idref attribute (in an empty element) to refer to a
previously defined reference.

example:

```
  <state id="s2">          -- defines an id or a reference
    <scode>NE</scode>
    <sname>Nevada</sname>
  </state>

 <city id="c2">
  <ccode>CCN</ccode>
  <cname>Carson City</cname>
  <state-of idref="s2"/>    -- refers to object called s2;
                            -- this is an empty element
 </city>
```

Detailed example:

```
<geography>
  <states>
    <state id="s1">
      <scode>ID</scode>
      <sname>Idaho</sname>
      <capital idref="c1"/>
    <cities-in idref="c1"/><cities-in idref="c3"/> ...
    </state>
   <state id="s2">
      <scode>NE</scode>
      <sname>Nevada</sname>
      <cities-in idref="c2"/>...
    </state>
  ...
  ...
  </states>
```

```
 <cities>
  <city id="c1">
   <ccode>BOI</ccode>
   <cname>Boise</cname>
   <state-of idref="s1"/>
  </city>
<city id="c2">
   <ccode>CCN</ccode>
   <cname>Carson City</cname>
   <state-of idref="s2"/>
  </city>
<city id="c3">
   <ccode>MOC</ccode>
   <cname>Moscow</cname>
   <state-of idref="s1"/>
  </city>
   ...
   ...
 </cities>
</geography>
```

3.2.3 Order

The semi-structured data model is based on unordered collections, whereas XML is ordered.

The following two pieces of semi-structured data are equivalent:

```
  person: {firstname: "John", lastname: "Smith:}

 person: {lastname: "Smith", firstname: "John"}
```

but the following two XML data are not:

```
<person><firstname>John</firstname>
        <lastname>Smith</lastname>
</person>

<person><lastname>Smith></lastname>
        <firstname>John</firstname>
</person>
```

To make matters worse, attributes are NOT ordered in XML; Following two are equivalent:

```
<person firstname="John" lastname="Smith"/>

<person lastname="Smith" firstname="John"/>
```

3.2.4 Mixing Elements and Text

XML allows us to mix PCDATA and sub-elements within an element.

```
<person>
  This is my best friend
  <name>Alan</name>
  <age>42</age>
  I am not sure of the following email
  <email>agb@abc.com</email>
</person>
```

This seems un-natural from a database perspective,
but from a document perspective, this is quite natural!

To translate such XML data into ssd-expressions,
we need to introduce new standard tags for the PCDATA.

3.2.5 Other XML Constructs

These additional constructs in XML have little or no use for data exchange purposes.

Comments:

```
<!-- this is a comment -->
```

Processing Instruction (PI):

```
<?xml - stylesheet href="book.css" type="text/css" ?>

<?xml version="1.0" ?>
```

such instructions are passed on to applications that process XML files.

CDATA (Character Data):

used to write escape blocks containing text that
otherwise would be considered markup:

```
  <![CDATA[   <start>this is not an element</start> ]]>
```

Entities: &lt   stands for <

Users can define new entities in XML

Document Type Definition (DTD):

```
<!DOCTYPE name [markup-declarations]>
```

name of root tag followed by several markup declarations
that define the tags that are permitted in the document
and their associated structure.

```
<?xml ...?>
<!DOCTYPE name [markup-declarations]>
  <name> ... </name>
```

is the structure of an entire XML document,

3.3 Document Type Definitions (DTDs)

The DTD serves as a grammar for the underlying XML document and is part of the XML language.

The DTD may serve as the schema for the underlying data.

3.3.1 A simple DTD:

```
<db> <person>
       <name>Alan</name>
       <age>42</age>
       <email>agb@abc.com</email>
     </person>
     ...
</db>
```

A DTD for the above XML document:

```
<!DOCTYPE db [
    <!ELEMENT db (person*)>
    <!ELEMENT person (name,age,email)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT age (#PCDATA)>
    <!ELEMENT email(#PCDATA)>
  ]>
```

- db consists of 0 or more persons
- person has three sub-elements name, age, email
- name, age, email are string type

e* e+ e|e' e.e' e? (0 or one)
are some of the other regular expressions allowed.

3.3.2 DTDs as grammars

A DTD is precisely a context-free grammar for the document.
In the above example,

db --> person | person db
person --> name age email

Grammars can be recursive as in

```
<!DOCTYPE node [
  <!ELEMENT node (leaf  | (node,node))>
  <!ELEMENT leaf (#PCDATA)>
 ]>
```

This describes binary trees!
A sample XML document consistent with the DTD is

```
  <node>
    <node>
      <node> <leaf> 1 </leaf> </node>
      <node> <leaf> 2 </leaf> </node>
    </node>
    <node>
        <leaf> 3 </leaf>
    </node>
  </node>
```

3.3.3 DTDs as schemas

To a certain extent, DTDs can be used as schemas. For example,

- The db DTD requires the person element to have 3 fields:
  name, age, and email.
- The data types are limited, though.

Consider the relational database example:

```
<db>
  <r1><a>a1</a><b>b1</b><c>c1</c></r1>
  <r1><a>a2</a><b>b2</b><c>c2</c></r1>
  <r2><c>c2</c><d>d2</d></r2>
  <r2><c>c3</c><d>d3</d></r2>
  <r2><c>c4</c><d>d4</d></r2>
</db>
```

A DTD for this XML data is

```
<!DOCTYPE db [
  <!ELEMENT db (r1*, r2*)>
  <!ELEMENT r1(a,b,c)>
  <!ELEMENT r2(c,d)>
  <!ELEMENT a (#PCDATA)>
  <!ELEMENT b (#PCDATA)>
  <!ELEMENT c (#PCDATA)>
  <!ELEMENT d (#PCDATA)>
]>
```

This DTD correctly constrains r1 elements to have 3 sub-elements: a, b, and c in that order.

The order of sub-elements are fixed. We could redefine
the DTD to allow different orders:

```
<!ELEMENT r2 ((c,d) | (d,c))>
```

The DTD constrains r2 elements to follow r1 elements,
but the following change allows rt1 and r2 elements to
be interspersed:

```
<!ELEMENT db ((r1 | r2)*)>
```

More flexibility in DTDs! For example,

```
<!ELEMENT r1 (a, b?, c+)>
```

requires r1 element to have exactly one a,
followed by 0 or one b, followed by one or more c's.

DTDs can be stored in files and the file can be
included in the XML document as follows:

```
 <!DOCTYPE db SYSTEM "schema.dtd">
```

or even

```
 <!DOCTYPE db SYSTEM="http://tinman.cs.gsu.edu/~raj/schema.dtd">
```

where the schema file is available publicly on the Web.

3.3.4 Declaring Attributes in DTDs

Consider the XML document:

```
<product>
  <name language="French" department="music">
      trompette six trous</name>
  <price currency="Euro">420.12</price>
</product>
```

The DTD for this document includes attribute definitions as follows:

```
<!ATTLIST name language CDATA #REQUIRED
               department CDATA #IMPLIED>

<!ATTLIST price currency CDATA #IMPLIED>
```

Element name has two attributes:
language which is required and department which is optional;
Both attributes are string type.

ID, IDREF, IDREFS attributes:

ID is used to define object identifier
IDREF is used to refer to an object id
IDREFS is used to refer to a list of object ids separated by spaces

Example: Family Tree specification

```
  <!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>
  ]>
```

An XML element that conforms to the above DTD is

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

DTD for geography example:

```
<!DOCTYPE geography [
    <!ELEMENT geography (state|city)*>
    <!ELEMENT state (scode,sname,capital,cities-in*)>
            <!ATTLIST state id ID #REQUIRED>
    <!ELEMENT scode (#PCDATA)>
    <!ELEMENT sname (#PCDATA)>
    <!ELEMENT capital EMPTY>
        <!ATTLIST capital idref IDREF #REQUIRED>
    <!ELEMENT cities-in EMPTY>
        <!ATTLIST cities-in idref IDREF #REQUIRED>
    <!ELEMENT city (ccode,cname,state-of)>
        <!ATTLIST city id ID>
    <!ELEMENT ccode (#PCDATA)>
    <!ELEMENT cname (#PCDATA)>
    <!ELEMENT state-of EMPTY>
        <!ATTLIST state-of idref IDREF #REQUIRED>
]>

-- EMPTY indicates that this element will always be empty
```

The attribute type IDREFS allows an attrinute to refer to multiple entities.

The above DTD can be modified as follows to take advantage of IDREFS:

```
  <!ELEMENT state (scode,sname,capital,cities-in)>

        -- instead of cities-in*

  <!ELEMENT cities-in EMPTY>
    <!ATTLIST cities-in idrefs IDREFS #REQUIRED>

        -- instead of IDREF
```

This way cities-in element can be written as:

```
  <cities-in idrefs="c1 c2"/>
```

Another way to make the representation more compact is to make capital and cities-in attributes instead of sub-elements as in:

```
<!ATTLIST state id ID #REQUIRED
          capital IDREF #REQUIRED
          cities-in IDREFS #REQUIRED>
```

So, a state element can be described as:

```
<state id="s1" capital="c1" cities-in="c1 c3">
    <scode>ID</scode>
    <sname>Idaho</sname>
</state>
```

An example illustrating entities that refer to
external data using a URL.

```
<?xml version "1.0"?>
<!DOCTYPE report [
  <!ELEMENT report (meta,title,abstract,content)>
  <!ELEMENT meta EMPTY>
      <!ATTLIST meta
         keywords CDATA #REQUIRED
         author CDATA #REQUIRED
         date CDATA #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT abstract (#PCDATA)>
  <!ELEMENT content (#PCDATA)>
  <!ENTITY %abstract SYSTEM "/u/abiteboul/LEBOOK/abstract">
  <!ENTITY %content SYSTEM "/u/suciu/LEBOOK/lebook">
]>
```

```
<report>
  <meta keywords="xml,www,web,semistructured"
        author="abiteboul,Buneman,Suciu"
        date="25.12.98"/>
  <title>Data on the Web</title>
  %abstract
  %content
</report>
```

3.3.5 Valid XML Documents

Well-formed XML document: matching nested tags,
no duplicate attributes

Valid XML document: One which has a DTD AND the document
is consistent with the DTD.
i.e. elements must be nested only in the way the
DTD specifies and the tags used must be those
defined in the DTD. IDs must be unique and
IDREFS must refer to existing IDs.

There is no restriction on the types of objects
the IDREFS point to!

3.3.6 Limitations of DTDs as Schemas

- DTDs impose order
- No atomic types except PCDATA
- Constraints such as age between 0 and 120
  not possible
- Global names (cannot use same identifier for
  two different element types)
  ex. name of person and name of course may have
  different structures; in XML we are forced to use
  two different identifiers : personname and coursename
- DTDs do not constrain the type of IDREFs.
  We would like to constrain the idref in
  state-of to point to a state element;
  This is not possible in DTDs.

Skip 3.4, 3.5, 3.6