8.1 Introduction
- SGML is a meta-markup language
- Developed in the early 1980s; ISO std. in 1986
- HTML was developed using SGML in the early 1990s - specifically for Web documents
- Two problems with HTML:
  1. Fixed set of tags and attributes
     - User cannot define new tags or attributes
     - So, the given tags must fit every kind of document, and the tags cannot connote any particular meaning
  2. There are no restrictions on arrangement or order of tag appearance in a document
- One solution to the first of these problems:
  Let each group of users define their own tags (with implied meanings)
  (i.e., design their own “HTMLs” using SGML)

8.1 Introduction (continued)
- Problem with using SGML:
  - It’s too large and complex to use, and it is very difficult to build a parser for it
- A better solution: Define a lite version of SGML
- XML is not a replacement for HTML
  - HTML is a markup language used to describe the layout of any kind of information
  - XML is a meta-markup language that can be used to define markup languages that can define the meaning of specific kinds of information
  - XML is a very simple and universal way of storing and transmerring data of any kind
  - XML does not redefine any tags
  - XML has no hidden specifications
  - All documents described with an XML-derived markup language can be parsed with a single parser

Problem with using SGML:
- It’s too large and complex to use, and it is very difficult to build a parser for it
- A better solution: Define a lite version of SGML
- XML is not a replacement for HTML
  - HTML is a markup language used to describe the layout of any kind of information
  - XML is a meta-markup language that can be used to define markup languages that can define the meaning of specific kinds of information
  - XML is a very simple and universal way of storing and transmerring data of any kind
  - XML does not redefine any tags
  - XML has no hidden specifications
  - All documents described with an XML-derived markup language can be parsed with a single parser

8.2 The Syntax of XML
- General XML Syntax
  - XML documents have data elements, markup declarations (instructions for the XML parser), and processing instructions (for the application program that is processing the data in the document)
  - All XML documents begin with an XML declaration:
    ```xml
    <?xml version = "1.0"?>
    ```
  - XML comments are just like HTML comments
  - XML names:
    - Must begin with a letter or an underscore
    - They can include digits, hyphens, and periods
    - There is no length limitation
    - They are case sensitive (unlike HTML names)

8.2 The Syntax of XML (continued)
- Syntax rules for XML (similar to those for XHTML)
  - Every XML document defines a single root element, whose opening tag must appear as the first line of the document
  - Every element that has content must have a closing tag
  - Tags must be properly nested
  - All attribute values must be quoted
  - An XML document that follows all of these rules is well formed
    ```xml
    <?xml version = "1.0"?>
    <ad>
      <year> 1960 </year>
      <make> Cessna </make>
      <model> Centurian </model>
      <color> Yellow with white trim </color>
      <location>
        <city> Gulfport </city>
        <state> Mississippi </state>
      </location>
    </ad>
    ```
- Attributes are not used in XML the way they are in HTML
  - In XML, you often define a new nested tag to provide more info about the content of a tag
  - Nested tags are better than attributes, because attributes cannot describe structure and the structural complexity may grow
  - Attributes should always be used to identify numbers or names of elements (like HTML <i> and <strong> attributes)
8.2 The Syntax of XML (continued)

- A tag with one attribute:
  ```
  <patient name = "Maggie Dee Magpie">
  ```

- A tag with one nested tag:
  ```
  <patient>
    <name> Maggie Dee Magpie </name>
  </patient>
  ```

- A tag with one nested tag, which contains three nested tags:
  ```
  <patient>
    <name>
      <first> Maggie </first>
      <middle> Dee </middle>
      <last> Magpie </last>
    </name>
  </patient>
  ```

8.3 XML Document Structure

- An XML document often uses two auxiliary files:
  - One to specify the structural syntactic rules
  - One to provide a style specification
- An XML document has a single root element, often consists of one or more entities
- Entities range from a single special character to a book chapter
- An XML document has one document entity
- All other entities are referenced in the document entity
- Reasons for entity structure:
  1. Large documents are easier to manage
  2. Repeated entities need not be literally repeated
  3. Binary entities can only be referenced in the document entities (XML is all text!)

8.3 XML Document Structure (continued)

- When the XML parser encounters a reference to a non-binary entity, the entity is merged in
- Entity names:
  - No length limitation
  - Must begin with a letter, a dash, or a colon
  - Can include letters, digits, periods, dashes, underscores, or colons
- A reference to an entity has the form:
  ```
  &entity_name;
  ```
- One common use of entities is for special characters that may be used for markup delimiters
- These are predefined (as in XHTML):
  ```
  <           &lt;
  >           &gt;
  &           &amp;
  "           &quot;
  '           &apos;
  ```
- The user can only define entities in a DTD

8.3 XML Document Structure (continued)

- If several predefined entities must appear near each other in a document, it is better to avoid using entity references
- Character data section
  ```
  <![CDATA[ content ]]]>
  ```
  e.g., instead of
  ```
  Start &gt; &gt; &gt; &gt; HERE
  ```
  ```
  use
  ```
  <![CDATA[Start >>>> HERE <<<<]]>
  ```
- If the CDATA content has an entity reference, it is taken literally

8.4 Data Type Definitions

- A DTD is a set of structural rules called declarations
- These rules specify a set of elements, along with how and where they can appear in a document
- Purpose: provide a standard form for a collection of XML documents
- Not all XML documents have or need a DTD
- The DTD for a document can be internal or external
- Errors in DTD: Find them early!
- All of the declarations of a DTD are enclosed in the block of a DOCTYPE markup declaration
- DTD declarations have the form:
  ```
  <!ELEMENT element_name (list of child names)>
  ```
- There are four possible declaration keywords:
  ELEMENT, ATTLIST, ENTITY, AND NOTATION
8.4 Data Type Definitions (continued)
- Declaring Elements (continued)
  - Child elements can have modifiers, +, *, ?
  
  e.g.,
  
  ```
  <!ELEMENT person
  (parent+, age, spouse?, sibling*)>
  ```

- Leaf nodes specify data types, most often
  PCDATA, which is an acronym for parsable
  character data
  
  - Data type could also be
  #PCDATA (no content)
  and
  #REQUIRED (can have any content)

- Example of a leaf declaration:
  
  ```
  <!ELEMENT name (#PCDATA)>
  ```

Declaring Attributes

- General form:
  
  ```
  <!ATTLIST el_name at_name at_type [default]>n
  ```

  - Attribute types: there are many possible, but we
  will consider only CDATA
  
  - Default values:
  
  a value
  #FIXED value (every element will have
  this value),
  #REQUIRED (every instance of the element must
  have a value specified), or
  #IMPLIED (no default value and need not specify
  a value)

  - e.g.,

  ```
  <!ATTLIST car doors CDATA "4">
  <!ATTLIST car engine_type CDATA #REQUIRED>
  <!ATTLIST car price CDATA #IMPLIED>
  <!ATTLIST car make CDATA #FIXED "Ford">
  
  <car doors = "2" engine_type = "V8">
  ...n
  </car>
  ```

8.5 Namespaces

- A markup vocabulary is the collection of all of the
  element types and attribute names of a markup
  language (a tag set)

- An XML document may define its own tag set and
  also use that of another tag set - CONFLICTS!

- An XML namespace is a collection of names used
  in XML documents as element types and attribute
  names

  - The name of an XML namespace has the form of
  a URI

  - A namespace declaration has the form:

  ```
  <element_name xmlns[=prefix] = "URI">
  ```

  - The prefix is a short name for the namespace,
  which is attached to names from the
  namespace in the XML document

  - In the document, you can use <ge:geometric>

  - Purposes of the prefix:
    1. Shorthand
    2. URI includes characters that are illegal in XML

8.6 XML Schemas

- Problems with DTDs:
  1. Syntax is different from XML - cannot be parsed
     with an XML parser
  2. It is confusing to deal with two different syntactic
     forms
  3. DTDs do not allow specification of particular
     kinds of data
8.6 XML Schemas (continued)

- XML Schemas is one of the alternatives to DTD

- Two purposes:
  1. Specify the structure of its instance XML documents
  2. Specify the data type of every element and attribute of its instance XML documents

- Schemas are written using a namespace:
  ```
  http://www.w3.org/2001/XMLSchema
  ```

- Every XML schema has a single root, `schema`
  The `schema` element must specify the namespace for schemas as its `xmlns:xsd` attribute

- Every XML schema itself defines a tag set, which must be named
  ```
  targetNamespace = "http://cs.uccs.edu/planeSchema"
  ```

- If we want to include nested elements, we must set the `elementFormDefault` attribute to `qualified`

  ```
  xmlns = "http://cs.uccs.edu/planeSchema"
  ```

- The default namespace must also be specified

- A complete example of a schema element:
  ```
  <xsd:schema
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
    targetNamespace = "http://cs.uccs.edu/planeSchema"
    xmlns = "http://cs.uccs.edu/planeSchema"
    elementFormDefault = "qualified">
  ```

8.6 XML Schemas (continued)

- Defining an instance document

  ```
  <planes
    xmlns = "http://cs.uccs.edu/planeSchema"
    xmlns:xsi = http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://cs.uccs.edu/planeSchema planes.xsd" >
  ```

8.6 XML Schemas (continued)

- Defining a simple type:

  ```
  <xsd:element name = "bird"
    type = "xsd:string" />
  ```

- An instance could have:

  ```
  <bird> Yellow-bellied sap sucker </bird>
  ```

- Element values can be constant, specified with the `fixed` attribute

  ```
  fixed = "three-toed"
  ```

- User-Defined Types

  ```
  <xsd:simpleType name = "middleName" >
    <xsd:restriction base = "xsd:string" >
      <xsd:maxLength value = "20" />
    </xsd:restriction>
  </xsd:simpleType>
  ```

8.6 XML Schemas (continued)

- Categories of Complex Types

  1. Element-only elements
  2. Text-only elements
  3. Mixed-content elements
  4. Empty elements

- Element-only elements

  ```
  <elem name = "MiddleName">
    <middleName> Yellow-bellied sap sucker </middleName>
  </elem>
  ```

- Use the `sequence` tag for nested elements that must be in a particular order

- Use the `all` tag if the order is not important
8.6 XML Schemas (continued)

```xml
<xsd:complexType name="sports_car">
  <xsd:sequence>
    <xsd:element name="make" type="xsd:string"/>
    <xsd:element name="model" type="xsd:string"/>
    <xsd:element name="engine" type="xsd:string"/>
    <xsd:element name="year" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

- Nested elements can include attributes that give the allowed number of occurrences (minOccurs, maxOccurs, unbounded).

SHOW planes.xsd and planes.xml

- We can define nested elements elsewhere.

```xml
<xsd:element name="year">
  <xsd:simpleType>
    <xsd:restriction base="xsd:decimal">
      <xsd:minInclusive value="1990"/>
      <xsd:maxInclusive value="2003"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

8.6 XML Schemas (continued)

- The global element can be referenced in the complex type with the ref attribute.

```xml
<xsd:element ref="year"/>
```

8.7 Displaying Raw XML Documents

- There is no presentation information in an XML document.

- An XML browser should have a default style sheet for an XML document that does not specify one.

- You get a stylized listing of the XML.

SHOW Figure 8.2 and 8.3

8.8 Displaying XML Documents with CSS

- A CSS style sheet for an XML document is just a list of its tags and associated styles.

- The connection of an XML document and its style sheet is made through an xml-stylesheet processing instruction.

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

SHOW planes.css and Figure 8.4

8.9 XSLT Style Sheets

- XSLT began as a standard for presentations of XML documents.

- Split into two parts:
  - XSLT - Transformations
  - XSL-FO - Formatting objects

- XSLT uses style sheets to specify transformations.

8.8 XSLT Style Sheets (continued)

- An XSLT processor merges an XML document into an XSLT style sheet.

- This merging is a template-driven process.

- An XSLT style sheet can specify page layout, page orientation, writing direction, margins, page numbering, etc.

- The processing instruction we used for connecting a CSS style sheet to an XML document is used to connect an XSLT style sheet to an XML document.

```xml
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Format">
  <xsl:template match="/">
    ...</xsl:template>
</xsl:stylesheet>
```

- An example:

```xml
<xml version="1.0"/>
<xsl:stylesheet type="text/xsl" href="xslplane.xsl"/>
</xsl:stylesheet>
</plane>
```

- A template can match any element, just by naming it (in place of /).

- XSLT elements include two different kinds of elements, those with content and those for which the content will be merged from the XML document.

- Elements with content often represent HTML elements.

```xml
<open style="font-size: 14">Happy Easter!</open>
```

8.8 XML Transformations and Style Sheets (continued)

- XSLT elements that represent HTML elements are simply copied to the merged document.

- The XSLT value-of element.

```xml
<xsl:value-of select="CAR/ENGINE"/>
```

- The value of select can be any branch of the document tree.

SHOW xslplane.xsl and Figure 8.5

- The XSLT for-each element.

SHOW xslplanes.xml

SHOW xslplanes.xsl & Figure 8.6