Fundamentals of Web Programming [a]

Introduction to XML

Teodor Rus

rus@cs.uiowa.edu

The University of Iowa, Department of Computer Science

Markup-Languages

A meta-markup language is a mechanism used to define markup languages.

Example: The Standard Generalized Markup Language (SGML) is a meta-markup language standardized by ISO in 1986.

- In 1990 SGML was used for the development of the HTML as a standard markup language for Web documents;
- In 1996 the World Wide Web Consortium (W3C) used SGML to develop a simpler meta-markup language, namely The eXtensible Markup Language (XML);
- The first XML standard, XML 1.0, was published in February 1998;
- The second XML standard, XML 1.1, was published in 2004. However, only XML 1.0 is used in these notes.

Rationale for XML

Part of the motivation for XML development was the deficiencies of the HTML.

Problems: HTML defines a collection of tags and attributes to be used to describe the layout on information in Web documents.

- Using HTML one can describe the layout of information without considering its meaning;
- HTML lacks a formal syntax and consequently HTML documents cannot be easily validated.

Potential Solutions

1. Define a specific set of tags and attributes for each group of users with common document needs and then use SGML standard to define a new markup language to meet those needs.
   Hence, each application area would have its own markup language.

   Feasibility: although having a domain specific markup language is a good idea, complexity of SGML makes it difficult to implement it.

2. Define a simplified version of SGML and allow users to develop their own markup languages.

   Feasibility: XML was designed to implement this goal. In this context users are organizations with common data description processing needs.

Observation

If we replace the goal of data description with the goal of problem solving then we obtain the idea of developing a markup language used to describe a domain specific problem solving process.

Facts

1. XML was not designed as a replacement of HTML. XML was designed as a tool to be used for the design of markup languages. XHTML is one of its first applications.
2. XML provide a simple and universal way of storing textual data. Data stored in XML documents can be electronically distributed and processed by any number of different applications.
3. XML is a universal data interchange language.
   XML documents represent data not processes.
4. XML is not a markup language, XML is a meta-markup language that specify rules for creating markup language. Consequently XML includes no tags. XML user must uses tags specific to her application domain.

XML Applications

A markup language designed with XML is called an XML application.

XML Tag Set: to avoid confusion, an XML-based markup language is called a tag set;

XML Document: a document that use an XML-based markup language is called an XML document.

XML Tools

- XML documents can be written by hand using simple text editors or by dedicated programs.
- Browsers have default presentation styles for every XHTML elements. However, one cannot expect a browser to have default presentation styles for elements it has never seen.

  Consequence: XML documents can be displayed by browsers only if presentation rules are provided by style sheets.

XML Processors

- An XML processor parses XML documents, decompose them into components (such as tags, attributes, data strings) and provide them to applications.
- XML documents are plain text, which are readable by both people and machines, although there is no compelling reason for people to read XML documents.
- Currently majority of Web clients use such browsers as IE6, IE7, FX, FX2 which support basic XML.

Syntax of XML

The syntax of XML is defined at two levels:

1. General low-level syntax that imposes rules on all XML documents, independent of their tag set;
2. Application dedicated syntax defined by the collection of tags, attributes, and entities available for XML documents.

Note: application dedicated syntax is defined by either Document Type Definition (DTD) or XML schema.

General Rules

An XML document is a sequence of well-nested XML elements.

- An XML element is specified as string that has the following structure:

  ```
  <tag attributes > Contents </tag>
  ```

  where each attribute is a string of the form *name = "value"*.

- An element is well-nested if its open and closing tag are not interleaved with other tags.

XML Document Format

1. An XML document begins with an XML declaration which identifies the XML standard of the document. Example:

   ```
   <?xml version = "1.0" encoding = "utf-8"?>
   ```

2. Each document has a single root element; all other elements of the XML document must be well-nested in the document root.

3. XML elements are of two kind: content and no content element.
   - An XML element that can have content has the structure"

     ```
     <tag> Content </tag>
     ```

     where the `Content` is a sequence of well-nested XML elements.
   - An XML element that has no content has the form

     ```
     <element_name attributes />
     ```

Example XML Document

```
<?xml version = "1.0" encoding = "utf-8"?>
<ad>
   <year> 1979 </year>
   <make> Cessna </make>
   <model> Centurian </centurian>
   <color> Yellow with white trim </color>
   <location>
      <city> Gulfport </city>
      <state> Mississippi </state>
   </location>
</ad>
```

Document Development

The designer of an XML document is faced with:

- The choice between adding new attributes to a tag or defining a nested element;
- No choice, as is the case when the data is an image and the reference to it can only be an attribute because images are binary data and XML document can contain only text;
- In other cases it may not matter whether an attribute or a nested element is used;
- There are however situation where choices exist and one choice is clearly better than another.

Choices

- Nested tags versus attributes: if a document needs to grow in structural complexity nested tags are better than attributes because nested tags can be added to any existing tag to describe its growing size and complexity;
- Nothing can be added to an attribute, hence attributes cannot describe structure;
- To identify an element of a set by number or by name the attributes *id* or *number* needs to be used;
- An attribute should be used when the data is a value among a given set of possibilities;
- An attribute should be used if there is no substructure of the data described by the XML element.

Example XML Choices

```
<!-- A tag with one attribute -->
<patient name = "Maggie Dee Magpie">
. . .
</patient>
!-- A tag with one nested tag -->
<patient>
   <name> Maggie Dee Magpie </name>
. . .
</patient>
<!-- A tag with multiple nested tags -->
<patient>
   <name>
      <firtst> Maggie </first>
      <middle> Dee </middle>
      <last> Magpie </last>
   </name>
. . .
</patient>
```

XML Document Structure

An XML document consists of one or more entities which represent logically related collections of information ranging in size from a single character to a book chapter.

Document entity: is always physically included in the file that represent the documents; it can be the entire document but it can also include references to the names of entities that are stored elsewhere.

XML Document Entities

Reasons for breaking an XML document into multiple entities:

- A large document is easier to manage when it is split into multiple entities;
- If the same data appears in more than one place in a document, defining it as an entity allows any number of reference to a single copy of that data, thus avoiding inconsistency among data occurrences;
- Many documents include information that cannot be represented as text, such as images, which is usually stored as binary data;
- If a binary data unit is a logical part of an XML document, it must be separate because XML documents cannot include binary data.

Entity Replacement

When an XML processor encounters the name of a non-binary entity in an XML document, it replaces that name with the value it references.

Note: Binary entities can be handled only by applications that deal with the document, such as browsers. XML processors deal only with text.

Entity Names

An entity name is a string which begins with a letter, a dash, or a colon.

- The length of an entity name is unlimited;
- After first character the entity name can have letters, digits, periods, dashes, underscores, or colons;
- A reference to an entity is the entity name prepended by ampersand (&) and appended a semicolon.

Example: if `apple_image` is the name of an entity then `&apple_image;`

is a reference to that entity.

Common Use of Entities

Entity usage allows characters that are normally used as markup delimiters to appear as themselves in a document.

- XML Entities can be user defined (using DTDs); The following XML entities are predefined for XHTML:

Character	Entity	Meaning
&	&amp;	Ampersand
<	<	Less than
>	>	Greater than
"	"	Double quote
’	'	Single quote (apostrophe)
$\frac{1}{4}$	¼	One quarter
$\frac{3}{4}$	¾	Three quarters
$\frac{1}{2}$	½	One half
°	°	Degree
(space)		Nonbreaking space

Character Data Section

When several predefined entities must appear near each other in an XML document, their references clutter the content making it difficult to read. This is prevented by using character data section instead.

Note: the content of a character data section is not parsed by the XML parser, so any tags it may include are not recognized as tags. This makes it possible to include special markup delimiter characters without using their entity references.

Data Section Specification

The form of a character data section is:

```
<![CDATA[ contents ]]>
```

Example: instead of

```
The last word of the line is &gt;&gt;&gt here &lt;&lt;&lt;
```

one can use

```
<![CDATA[The last word of the line is >>> here <<<]]>
```

Observations

1. The opening keyword of a character data section is

   `[CDATA[`

2. Consequence is that there cannot be any space between the `[` and the `C` or between `A` and the second `[` in `[CDATA[`.

3. The only thing that cannot appear in a character data section is the closing delimiter `]] >`.

4. Because the content of a character data section is not parsed by the XML parser, entity references included are not expanded.

Document Type Definition

A DTD is a set of structural rules called *declarations* that specify a set of elements, attributes, and entities and how they can appear in an XML document.

Note:

- Not all XML documents need a DTD;
- Use of DTD-s is related to the use of external style sheets to impose a uniform style over a set of documents;
- DTD-s are used when the same tag set definition is used for a collection of documents which must have a consistent and uniform structure.

Facts

1. A DTD can be internal or external the XML document whose syntax it describes. An internal DTD is embedded in the XML document while an external DTD is stored in a separate file;
2. External DTD-s describe collections of XML documents and avoid two different notations (XML and DTD) from appearing in the same document;
3. An XML document can be tested against its DTD to determine if it conforms to the specification rules;
4. Application programs that process XML data can be written to assume a particular document form;

Reasons for External DTD

Correct DTD-s separately developed prevent earlier errors in XML applications. Therefore:

- Fixing a DTD and all copies of it should be the first and simplest step in an XML document development;
- After the correction of DTD is completed, all XML documents it describes must be tested against it;
- Changes to the associated style sheets may be necessary during DTD correction process.

DTD Syntax

Syntactically, a DTD is a sequence of declarations of the form:

```
<!keyword BNF rule >
```

where `keyword` is one of:

1. `ELEMENT` if the BNF rule describes the structure of an XML element;
2. `ATTLIST` if the BNF rule defines a list of one or more tag attributes used by the XML documents;
3. `ENTITY` if the BNF rule defines an entity used by the XML documents;
4. `NOTATION` if the BNF rule defines data type notation.

ELEMENT Declaration

Each element declaration in a DTD specifies the structure of one category of elements used in XML documents using the form

```
<!ELEMENT elementName (list of names of child elements) >
```

where:

- `elementName` is name of the element (the tag);
- `list of names of child elements` is the comma separated sequence of element components.

Note: although an XML element is a string of characters its structure is described by a tree whose root is the element tag, whose interior nodes are element components, and whose leafs are strings of characters.

Example Element Declaration

- An XML element representing a memo may be declared by:

```
<!ELEMENT memo (from, to, date, re, body) >
```

- The tree structure of the element `memo` is in Figure 1

Element Tree Representation

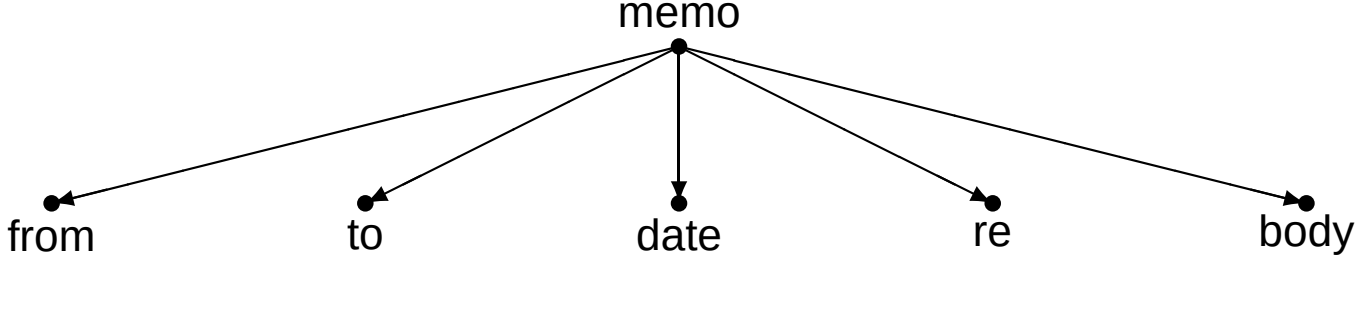

Figure 1: Example Element Tree Representation

Facts

1. In many cases it is necessary to specify the number of times that an element may appear. This can be done by using regular operators as seen in the following table:

Operator	Meaning
+	One or more occurrences
*	Zero or more occurrences
?	Zero or one occurrences

2. Example:

```
<!ELEMENT person (parent+, age, spouse?, siblings*) >
```

Here the `person` element is specified as having the children: one or more `parent` elements, one `age` element, possible a `spouse` element, and zero or more `sibling` elements.

Data Type Specification

The leaf nodes of a DTD specify the data types of the content of their parent nodes, which are elements.

There are three kinds of data specified by the leaf elements of a DTD:

1. Parsable character data, which is a string of any printable characters except $<$, $>$, and $\&$, specified by

   ```
   <!ELEMENT elementName (#PCDATA)>;
   ```

2. Empty content element, specified by

   ```
   <!ELEMENT elementName (EMPTY)>
   ```

3. Unrestricted content element, which removes any syntax checking:

   ```
   <!ELEMENT elementName (ANY)>
   ```

Attribute Declaration

The general form of an attribute declaration is:

```
<!ATTLIST elementName attributeName attributeType [defaultValue]>
```

If more than one attribute is declared for an element the declaration is:

```
<!ATTLIST elementName
          attributeName_1 attributeType [defaultValue_1]
          attributeName_2 attributeType [defaultValue_2]
          . . .
          attributeName_n attributeType [defaultValue_n] >
```

Attribute Types

The `attributeType` in an attribute declaration can have the following values:

Value	Meaning
CDATA	The value is character data
(eval|eval|..)	The value must be an enumerated value
ID	The value is an unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is predefined

Default Value

The `defaultValue` in attribute declarations can be:

Value	Meaning
#DEFAULT value	The attribute has a default value
#REQUIRED	The attribute value must be included in the element
#IMPLIED	The attribute value does not have to be included
#FIXED value	The attribute value is fixed and quoted

Example Attribute Declarations

- An empty element with the attribute `width` of type CDATA.

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">
```

XML example:

```
<square width="100" />
```

- Default attribute value

```
<!ATTLIST elementName attributeName CDATA "default-value">
```

DTD example:

```
<!ATTLIST payment type CDATA "check">
```

XML example:

```
<payment type = "check">
```

Note: specifying a default value for an attribute, assures that the attribute will get a value even if the author of the XML document didn't include it.

More Attribute Examples

- Implied attribute

```
<!ATTLIST elementName attributeName attributeType #IMPLIED>
```

DTD example:

```
<!ATTLIST contact fax CDATA #IMPLIED>
```

XML example:

```
<contact fax = "555-667788">
```

Note: an implied attribute is used when the author is not forced to include an attribute and there is no option for a default value.

More Attribute Examples

- Required attribute

```
<!ATTLIST elementName attributeName attributeType #REQUIRED>
```

DTD example:

```
<!ATTLIST person number CDATA #REQUIRED>
```

XML example:

```
<person number = "5677">
```

Note: Use a required attribute if you don't have an option for a default value, but still want to force the attribute to be present.

More Attribute Examples

- Fixed attribute value

```
<!ATTLIST elemName attrNname attrTtype #FIXED "value">
```

DTD example:

```
<!ATTLIST sender company CDATA #FIXED "Microsoft">
```

XML example:

```
<sender company = "Microsoft">
```

Note: Use a fixed attribute value when you want an attribute to have a fixed value without allowing the author to change it. If an author includes another value, the XML parser will return an error.

More Attribute Examples

- Enumerated attribute values

```
<!ATTLIST elementName attributeName (eval|eval|..) default-value>
```

DTD example:

```
<!ATTLIST payment type (check|cash) "cash">
```

XML example:

```
<payment type = "check">
<payment type = "cash">
```

Note: Use enumerated attribute values when you want the attribute values to be one of a fixed set of legal values.

Entity Declaration

Entities are defined in DTD and are referenced in XML document. There are two kind of entities:

1. General entities: are entities that can be referenced anywhere in an XML document and are defined by:

   ```
   <!ENTITY entityName "entityValue">
   ```

   Example:

   ```
   <!ENTITY jfk "John Fitzgerald Kenedy">
   ```

2. Parametric entities: are entities that can be referenced only in DTD-s and are defined by:

   ```
   <!ENTITY % entityName "entityValue">
   ```

Fact

1. When the entity value is longer than a few words, such as a section of a technical article, the entity is defined outside of the DTD and is called *external entity*.
2. The declaration of an external entity has the form:

   ```
   <!ENTITY entityName SYSTEM "entityLocation">
   ```

   The keyword `SYSTEM` tells that the entity definition is in a different file whose location is specified by "entityLocation".

Entity Reference

- Internal entity: DTD Example:

```
<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">
```

XML example:

```
<author> &writer; &copyright;</author>
```

- External entity: DTD Example:

```
<!ENTITY writer SYSTEM "http://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM "http://www.w3schools.com/entities.dtd">
```

XML example:

```
<author>&writer;&copyright;</author>
```

NOTATION Declaration

The purpose of a notation declaration is to define the format of some external non-XML file, such as a sound or image file, so you can refer to such files in your document.

see `http://www.google.com/, DTD NOTATION`.

Observations: the general form of a notation declaration can be either of:

```
<!NOTATION nname PUBLIC std>
<!NOTATION nname SYSTEM url>
```

where

- `name` is the name given to the notation;
- `std` is the published name of a public notation;
- `url` is a reference to a program that can render a file in the given notation.

NOTATION Use

There are four steps to connecting an attribute to a notation:

1. Declare the notation. For example:

```
<!NOTATION jpeg PUBLIC "JPG 1.0">
```

2. Declare the entity. For example:

```
<!ENTITY bogiePic SYSTEM "http://stars.com/bogart.jpg" NDATA jpg>
```

3. Declare the attribute as type ENTITY. For example:

```
<!ATTLIST star-bio pin-shot ENTITY #REQUIRED>
```

4. Use the attribute:

```
<star-bio pin-shot = "bogiePic"> ... </star-bio>
```

A Sample DTD

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!-- planes.dtd a DTD for planes.xml document -->

<!ELEMENT planes4Sale (ad+)>
<!ELEMENT ad (year, make, model, color, description, price?,
              seller, location)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT make (#PCDATA)>
<!ELEMENT modesl (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT seller (#PCDATA)>
<!ELEMENT location (city, state)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
```

Sample DTD, Continuation

```
<!-- Attribute declarations -->

<!ATTLIST seller phone CDATA #REQUIRED>
<!ATTLIST seller email CDATA #IMPLIED>

<!ENTITY c "Cessna">
<!ENTITY p "Piper">
<!ENTITY b "Beechcraft">
```

DTD Use

- An internal DTD is included in the XML file it describes using declaration

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE rootName
  [ <!-- DTD --> ]
  <!-- XML file -->
```

- An external DTD is included in the XML file it describes using the declaration:

```
<?xml version = "1.0" encoding = "utf-8" ?>
<!DOCTYPE XMLrootName SYSTEM "DTDfileNameLocation">
  <!-- XML file -->
```

Example DTD Use

```
<?xml version = "1.0" encoding = "utf-8">
<!DOCTYPE planes4Sale SYSTEM ExamplesDTD/planes4Sale.dtd>
<planes4Sale>
  <ad>
    <year> 1977 </year>
    <mkae> &c   </make>
    <model> Skyhawk </model>
    <color> Light blue with white </color>
    <description> New paint, nearly new interior, 685 hours SMOH,
                  full IFK King avionics </description>
    <price> 23,495 </price>
    <seller phone = "555-222-3333"> Skyway Aircraft </seller>
    <location>
       <city> Rapid City, </city>
       <state> South Dakota </state>
    </location>
  </ad>
```

Example DTD, continuation

```
  <ad>
    <year> 1965 </year>
    <make> &p   </make>
    <model> Cherokee </model>
    <color> Gold </color>
    <description> 240 hours SMOH, dual NAVCOMs, DME, new
                  breaks, great shape </description>
    <seller phone = "555-333-22222" email = "jseller&www.axl.com">
       John Seller
    </seller>
    <location>
       <city> Cedar Rapids, </city>
       <state> Iowa </state>
    </location>
  </ad>
</planes4Sale>
```