Chapter 3

Document Type Definitions

Document Types

- A *document type* is defined by specifying the constraints which any document which is an *instance* of the type must satisfy
- For example,
  - in an HTML document, one paragraph cannot be nested inside another
  - in an SVG document, every `circle` element must have an `r` (radius) attribute
- Document types are
  - useful for restricting authors to use particular representations
  - important for correct processing of documents by software

Languages for Defining Document Types

- There are many languages for *defining* document types on the Web, e.g.,
  - document type definitions (DTDs)
  - XML schema definition language (XSDL)
  - relaxNG
  - schematron
- We will consider the first three of these

Document Type Definitions (DTDs)

- A DTD defines a *class* of documents
- The structural constraints are specified using an *extended context-free grammar*
- This defines
  - *element* names and their allowed contents
  - *attribute* names and their allowed values
  - *entity* names and their allowed values

Valid XML

- A *valid* XML document
  - is well-formed and
  - has been validated against a DTD
  - (the DTD is specified in the document — see later)

DTD syntax

- The syntax for an element declaration in a DTD is:

  `<!ELEMENT` *name* `(` *model* `) >`

  where
  - `ELEMENT` is a keyword
  - *name* is the element name being declared
  - *model* is the element *content model* (the allowed contents of the element)
- The content model is specified using a *regular expression* over element names
- The regular expression specifies the permitted *sequences* of element names

Examples of DTD element declarations

- An `html` element must contain a `head` element followed by a `body` element:

  ```
  <!ELEMENT html     (head, body) >
  ```

  where "," is the *sequence* (or concatenation) operator
- A `list` element (not in HTML) must contain either a `ul` element or an `ol` element (but not both):

  ```
  <!ELEMENT list     (ul|ol) >
  ```

  where "|" is the *alternation* (or "exclusive or") operator
- A `ul` element must contain zero or more `li` elements:

  ```
  <!ELEMENT ul       (li)* >
  ```

  where "*" is the *repetition* (or "Kleene star") operator

DTD syntax (1)

In the table below:

- `b` denotes any element name, the simplest regular expression
- α and β denote regular expressions

DTD Syntax	Meaning
`b`	element `b` must occur
α	elements must match α
(α)	elements must match α
α , β	elements must match α followed by β
α | β	elements must match either α or β (not both)
α*	elements must match zero or more occurrences of α

DTD syntax (2)

DTD Syntax	Meaning
α+	one or more sequences matching α must occur
α?	zero or one sequences matching α must occur
`EMPTY`	no element content is allowed
`ANY`	any content (of declared elements and text) is allowed
`#PCDATA`	content is text rather than elements

- α+ is short for (`(`α`,`α`*)`
- α? is short for (`(`α`|EMPTY)`
- `#PCDATA` stands for "parsed character data," meaning an XML parser should parse the text to resolve character and entity references

RSS

- RSS is a simple XML vocabulary for use in news feeds
- RSS stands for *Really Simple Syndication*, among other things
- The root (document) element is `rss`
- `rss` has a single child called `channel`
- `channel` has a `title` child, any number of `item` children (and others)
- Each `item` (news story) has a `title`, `description`, `link`, `pubDate`, ...

RSS Example Outline

```
<rss version="2.0">
  <channel>
    <title> BBC News - World </title>
      ...
    <item>
      <title> Hollande becomes French president </title>
        ...
    </item>
    <item>
      <title> New Greece poll due as talks fail </title>
        ...
    </item>
    <item>
      <title> EU forces attack Somalia pirates </title>
    </item>
      ...
  </channel>
</rss>
```

RSS Example Fragment (channel)

```
<channel>
  <title> BBC News - World </title>
  <link>http://www.bbc.co.uk/news/world/...</link>
  <description>The latest stories from the World section of
               the BBC News web site.</description>
  <lastBuildDate>Tue, 15 May 2012 13:42:05 GMT</lastBuildDate>
  <ttl>15</ttl>
  ...
</channel>
```

RSS Example Fragment (first item)

```
<item>
  <title>Hollande becomes French president</title>
  <description>Francois Hollande says he is fully aware
    of the challenges facing France after being sworn
    in as the country's new president.</description>
  <link>http://www.bbc.co.uk/news/world-europe-...</link>
  <pubDate>Tue, 15 May 2012 11:44:17 GMT</pubDate>
  ...
</item>
```

RSS Example Fragment (second item)

```
<item>
  <title>New Greece poll due as talks fail</title>
  <description>Greece is set to go to the polls again
    after parties failed to agree on a government for
    the debt-stricken country, says Socialist leader
    Evangelos Venizelos.</description>
  <link>http://www.bbc.co.uk/news/world-europe-...</link>
  <pubDate>Tue, 15 May 2012 13:52:38 GMT</pubDate>
  ...
</item>
```

RSS Example Fragment (third item)

```
<item>
  <title>EU forces attack Somalia pirates</title>
  <description>EU naval forces conduct their first raid
    on pirate bases on the Somali mainland, saying they
    have destroyed several boats.</description>
  <link>http://www.bbc.co.uk/news/world-africa-...</link>
  <pubDate>Tue, 15 May 2012 13:19:51 GMT</pubDate>
  ...
</item>
```

Simplified DTD for RSS

```
<!ELEMENT rss             (channel)>
<!ELEMENT channel         (title, link, description,
                           lastBuildDate?, ttl?, item+)>
<!ELEMENT item            (title, description, link?, pubDate?)>
<!ELEMENT title           (#PCDATA)>
<!ELEMENT link            (#PCDATA)>
<!ELEMENT description     (#PCDATA)>
<!ELEMENT lastBuildDate   (#PCDATA)>
<!ELEMENT ttl             (#PCDATA)>
<!ELEMENT pubDate         (#PCDATA)>
```

Validation of XML Documents

- Recall that an XML document is called *valid* if it is well-formed and has been validated against a DTD
- Validation is essentially checking that the XML document, viewed as a tree, is a *parse tree* in the language specified by the DTD
- We can use the W3C validator service
- Each of the following files has the same DTD specified (as on the previous slide):
  - `rss-invalid.xml` giving results
  - `rss-valid.xml` giving results

Referencing a DTD

- The DTD to be used to validate a document can be specified
  - internally (as part of the document)
  - externally (in another file)
- done using a *document type declaration*
- *declare* document to be of type given in DTD
- e.g., `<!DOCTYPE rss ... >`

Declaring an Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE rss [
    <!-- all declarations for rss DTD go here -->
    ...
    <!ELEMENT rss ... >
    ...
]>
<rss>
   <!-- This is an instance of a document of type rss -->
   ...
</rss>
```

- element `rss` must be defined in the DTD
- name after `DOCTYPE` (i.e., `rss`) must match root element of document

Declaring an External DTD (1)

```
<?xml version="1.0"?>
<!DOCTYPE rss SYSTEM "rss.dtd">
<rss>
   <!-- This is an instance of a document of type rss -->
   ...
</rss>
```

- what follows `SYSTEM` is a *URI*
- `rss.dtd` is a relative URI, assumed to be in same directory as source document

Declaring an External DTD (2)

```
<?xml version="1.0"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
     "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">
<math>
   <!-- This is an instance of a mathML document type -->
   ...
</math>
```

- `PUBLIC` means what follows is a *formal public identifier* with 4 fields:
  1. `ISO` for ISO standard, `+` for approval by other standards body, and `-` for everything else
  2. *owner* of the DTD: e.g., `W3C`
  3. *title* of the DTD: e.g., `DTD MathML 2.0`
  4. *language* abbreviation: e.g., `EN`
- URI gives location of DTD

More on RSS

- The RSS 2.0 specification actually states that, for each `item`, *at least one of* `title` or `description` must be present
- How can we modify our previous DTD to specify this?

More on RSS

- The RSS 2.0 specification actually states that, for each `item`, *at least one of* `title` or `description` must be present
- How can we modify our previous DTD to specify this?
- The allowed sequences are:
  1. `title`
  2. `title description`
  3. `description`

More on RSS

- The RSS 2.0 specification actually states that, for each `item`, *at least one of* `title` or `description` must be present
- How can we modify our previous DTD to specify this?
- The allowed sequences are:
  1. `title`
  2. `title description`
  3. `description`
- So what about the following regular expression?

```
title | (title, description) | description
```

Non-Deterministic Regular Expressions

- The regular expression

  `title | (title, description) | description`

  is non-deterministic
- This means that a parser must read ahead to find out which part of the regular expression to match
- e.g., given a `title` element in the input, should a parser try to match
  - `title` or
  - `title description`

Non-Deterministic Regular Expressions

- The regular expression

  `title | (title, description) | description`

  is non-deterministic
- This means that a parser must read ahead to find out which part of the regular expression to match
- e.g., given a `title` element in the input, should a parser try to match
  - `title` or
  - `title description`
- It needs to read the next element to check whether or not it is `description`

Non-Deterministic vs Deterministic Regular Expressions

- Non-deterministic regular expressions are *forbidden* by DTDs and XSDL
- They are allowed by RelaxNG
- A non-deterministic regular expression can always be rewritten to be deterministic

Non-Deterministic vs Deterministic Regular Expressions

- Non-deterministic regular expressions are *forbidden* by DTDs and XSDL
- They are allowed by RelaxNG
- A non-deterministic regular expression can always be rewritten to be deterministic
- e.g.,

  ```
  title | (title, description) | description
  ```

  can be rewritten as

  ```
  (title, description?)  | description
  ```

Non-Deterministic vs Deterministic Regular Expressions

- Non-deterministic regular expressions are *forbidden* by DTDs and XSDL
- They are allowed by RelaxNG
- A non-deterministic regular expression can always be rewritten to be deterministic
- e.g.,

  ```
  title | (title, description) | description
  ```

  can be rewritten as

  ```
  (title, description?)  | description
  ```

- The rewriting may cause an exponential increase in size

Attributes

- Recall that attribute name-value pairs are allowed in start tags, e.g., `version="2.0"` in the `rss` start tag
- Allowed attributes for an element are defined in an *attribute list declaration*: e.g., for `rss` and `guid` elements

```
<!ATTLIST rss
  version CDATA #FIXED "2.0" >
<!ATTLIST guid
  isPermaLink (true|false) "true" >
```

- attribute definition comprises
  - *attribute name*, e.g., `version`
  - *type*, e.g., `CDATA`
  - *default*, e.g., `"true"`

Some Attribute Types

- `CDATA`: any valid character data
- `ID`: an identifier unique within the document
- `IDREF`: a reference to a unique identifier
- `IDREFS`: a reference to several unique identifiers (separated by white-space)
- `(a|b|c)`, e.g.: (*enumerated attribute type*) possible values are one of `a`, `b` or `c`
- ...

Attribute Defaults

- `#IMPLIED`: attribute may be omitted (optional)
- `#REQUIRED`: attribute must be present
- `#FIXED "x"`, e.g.: attribute optional; if present, value must be `x`
- `"x"`, e.g.: value will be `x` if attribute is omitted

Mixed Content

- In `rss`, the content of each element comprised either only other elements or only text
- In HTML, on the other hand, paragraph elements allow text interleaved with various in-line elements, such as `em`, `img`, `b`, etc.
- Such elements are said to have *mixed content*
- How do we define mixed content models in a DTD?

Mixed Content Models

- Say we want to mix text with elements `em`, `img` and `b` as the allowed contents of a `p` element
- The DTD content model would be as follows:

```
<!ELEMENT p (#PCDATA | em | img | b)* >
```

  - `#PCDATA` must be first (in the definition)
  - It must be followed by the other elements separated by `|`
  - The subexpression must have `*` applied to it
- These restrictions limit our ability to constrain the content model (see XSDL later)

Entities

- An *entity* is a physical unit such as a character, string or file
- Entities allow
  - references to non-keyboard characters
  - abbreviations for frequently used strings
  - documents to be broken up into multiple parts
- An *entity declaration* in a DTD associates a name with an entity, e.g.,

```
<!ENTITY BBK "Birkbeck, University of London">
```

- An *entity reference*, e.g., `&BBK;` substitutes value of entity for its name in document
- An entity must be declared before it is referenced

General Entities

- `BBK` is an example of a *general entity*
- In XML, only 5 general entity declarations are built-in
  - `&` (&), `<` (<), `>` (>), `&quot;` ("), `&apos;` ('),
- All other entities must be declared in a DTD
- The values of *internal* entities are defined in the same document as references to them
- The values of *external* entities are defined elsewhere, e.g.,
  `<!ENTITY HTML-chapter SYSTEM "html.xml" >`
  - then `&HTML-chapter;` includes the contents of file `html.xml` at the point of reference
  - `standalone="no"` must be included in the XML declaration

Parameter Entities

- *Parameter entities* are
  - used only within XML markup declarations
  - declared by inserting % between `ENTITY` and name, e.g.,

```
<!ENTITY % list     "OL | UL" >
<!ENTITY % heading  "H1 | H2 | H3 | H4 | H5 | H6" >
```

  - referenced using % and ; delimiters, e.g.,

```
<!ENTITY % block  "P | %list; | %heading; | ..." >
```

- As an example. see the HTML 4.01 DTD

Limitations of DTDs

- There is no data typing, especially for element content
- They are only marginally compatible with namespaces
- We cannot use mixed content *and* enforce the order and number of child elements
- It is clumsy to enforce the presence of child elements without also enforcing an order for them (i.e. no `&` operator from SGML)
- Element names in a DTD are *global* (see later)
- They use non-XML syntax
- The XML Schema Definition Language, e.g., addresses these limitations