Constraining XML

lesson #xml02

James L. Parry
B.C. Institute of Technology

XML Documents...

XML Documents contain a single root element, with child elements.

Each element in an XML document can have a value, attributes, and child elements.

An XML document can be validated by checking it against a constraining specification.

Agenda

DOCUMENT TYPES

- A Document Type Definition (DTD) describes the intended structure and rules for an XML document.
- An XML document is "bound" to a DTD through a `<!DOCTYPE ...>` directive
- A document that claims to be of a certain type can be checked against the DTD.
- This is a *simple* way of constraining an XML document, and not the only way.

Document Type Declaration

- A DTD defines the tags allowed inside an XML document, their order and nesting, and the attributes allowed for each.
- A DTD defines a markup language!
- A DTD can be inferred from an XML document, but not the other way around!

Well-Formed vs Valid XML

- A document that follows the rules of the XML grammar is well formed.
- A document that conforms to the document type definition is valid.
- The two concepts are related but different... an XML document has to be well-formed in order to work with it, while validity improves confidence in it.
- A document can be well formed without being valid - but not the converse.
- A document with no DTD has no concept of validity.

Example: XML With Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE label[
  <!ELEMENT label (name, street, city,
    state, country, code)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT state (#PCDATA)>
  <!ELEMENT country (#PCDATA)>
  <!ELEMENT code (#PCDATA)>
]>
```

```
<label>
  <name>Paul Allen</name>
  <street>Alaskan Way</street>
  <city>Seattle</city>
  <state>WA</state>
  <country>USA</country>
  <code>53251-0054</code>
</label>
```

Example: XML With External DTD

garage.dtd:

```
<!ELEMENT GARAGESALE (DATE, TIME, PLACE,
NOTES)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT PLACE (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>
```

big_weekend.xml:

```
<?xml version="1.0"?>
<!DOCTYPE GARAGESALE SYSTEM "garage.dtd">
<GARAGESALE>
  <DATE>March 6, 2002</DATE>
  <TIME>10:00 AM</TIME>
  <PLACE>GM Place</PLACE>
  <NOTES>Lots of old hockey sticks</NOTES>
</GARAGESALE>
```

DTD MECHANICS

- A DTD contains directives for elements and attribute lists.
- Collectively, these provide rules for the order, multiplicity & content of components of an XML document.

Element Declarations

- Elements are declared using an ELEMENT directive:
  `<!ELEMENT name (content)>`
- "`name`" is the tag name of an element in the XML document. It follows XML naming conventions.
- "`content`" is basicaly a list of child elements for this one.

Order Constraints

- Child elements expected in a specific order are comma-separated:
  `<!ELEMENT order (drink, food, dessert)>`
- Child elements whose order does not matter are vertical-bar-separated, using parentheses for precedence:
  `<!ELEMENT order (drink, (food | dessert))>`

Multiplicity Constraints

- Child element names without modifiers are required children:
  `<!ELEMENT fullanme (firstname, initials, surname)>`
- Child elements that are optional have a question mark modifier:
  `<!ELEMENT fullanme (firstname, initials?, surname)>`
- "Zero or more" child elements are denoted by an asterisk modifier:
  `<!ELEMENT wishlist (gift*)>`
- "One or more" child elements are denoted by a plus sign modifier:
  `<!ELEMENT thought (idea+)>`
- These can be used together:
  `<!ELEMENT thought (idea|inkling)*>`

Content Constraints

- An element without children or text value is denoted EMPTY:
  `<!ELEMENT brain EMPTY>`
- An element containg a value but no children is designated #PCDATA (for parsed character data):
  `<!ELEMENT title (#PCDATA)>`
- The "ANY" designation is for an element that can have any number and order of children, so long as their names are defined in the DTD.:
  `<!ELEMENT wishlist (ANY)>`
- An element which can contain text as well as nested child elements is called "mixed":
  `<!ELEMENT story (#PCDATA|para)*>`

Attribute Declarations

- Attributes are declared using an ATTLIST directive:
  `<!ATTLIST element-name [name, type, modifier]+>`
- Sample attribute declarations:
  ```
  <!ATTLIST category code CDATA #REQUIRED>
  <!ATTLIST quarter number (1|2|3|4) #REQUIRED year CDATA #REQUIRED >
  <!ATTLIST quarter
    number (1|2|3|4) #REQUIRED
    year CDATA #REQUIRED
  >
  ```
- Attributes can be defined & specified in XML document in any order.
- First use in XML sticks, others ignored

Attribute Types

- An attribute whose value is a text string is denoted as CDATA:
  `<!ATTLIST name title CDATA #REQUIRED>`
- An attribute can have an enumeration of allowed values:
  `<!ATTLIST name title (mr|mrs|dr) #REQUIRED>`
  These are separated by vertical bars, and cannot contain spaces.

Attribute Modifiers

- An attribute which is required for an element in the XML document has a modifier of "#REQUIRED":
  `<!ATTLIST name title CDATA #REQUIRED>`
- An attribute which is optional has a modifier of "#IMPLIED":
  `<!ATTLIST name title CDATA #IMPLIED>`
- An attribute with a default value has that default value inside quotes:
  `<!ATTLIST name title CDATA "Mr">`

Attribute ID Notes

There are some additional attribute types, though seldom used because there are other, better tec address their intended purpose.

- A `NMTOKEN` attribute has a value which contains only letters, digits, hyphen, point, underscore, and colon; the value has to start with a letter
- An `ID` attribute is one which is a `NMTOKEN` and whose value has to be unique amongst all `ID` attributes in the document
- An `IDREF` attribute is one whose value is meant to reference an `ID` somewhere in the document

XML VALIDATION

- A DTD can be validated inside NetBeans, by right-clicking inside its editor panel and choosing "Check DTD".
- An XML document with a document type can be validated inside NetBeans, by right-clicking inside its editor panel and choosing "Validate XML".
- An XML document without a document type cannot be validated, but its well-formedness can be checked inside NetBeans, by right-clicking inside its editor panel and choosing "Check XML".

Server-Side Validation (Short Story)

An XML document can also be validated server-side, programmatically. The folllowing shows the general idea.

```
$doc = new DOMDocument();
$doc->validateOnParse = true;
$doc->load('abc.xml');
```

Returns TRUE if valid, FALSE otherwise

Server-Side Validation (Long Story)

More proper server-side validation, with error-checking:

```
libxml_use_internal_errors(true);
if ($doc->validate())
  return 'DTD validated ok';
else {
  $result = "<b>Oh nooooo...</b><br/>";
  foreach (libxml_get_errors() as $error) {
    $result .= $error->message . '<br/>';
  }
  libxml_clear_errors(); // clear the error message buffer
  return $result;
}
```

OTHER TECHNIQUES

- DTDs are simple, widespread, and limited.
- They are generally considered obsolete, being replaced by schemas.
- But they are so simple, while schemas have a much steeper learning curve.

Sample Schema Exceprt

Here is one part of a schema, only defining a country data type, which could be an element or an a

```
<!-- There are three countries in our data set.
It makes sense to restrict a country type to one
of that list. We can add to the list later if North America grows.
Note that we will have to capitalize these later for reporting. -->
<xs:simpleType name="Tcountry">
  <xs:restriction base="xs:string">
    <xs:enumeration value="canada"/>
    <xs:enumeration value="mexico"/>
    <xs:enumeration value="usa"/>
  </xs:restriction>
</xs:simpleType>
```

Sample Relax NG Exceprt

Here is one part of a Relax NG schema....

```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">
  <oneOrMore>
    <element name="card">
      <element name="name">
        <text/>
      </element>
      <element name="email">
        <text/>
      </element>
    </element>
  </oneOrMore>
</element>
```

Congratulations!

You have completed lesson #xml02: Constraining XML

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: xml XML Examples

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.