A Little More XML

INFO 202 - 15 September 2008

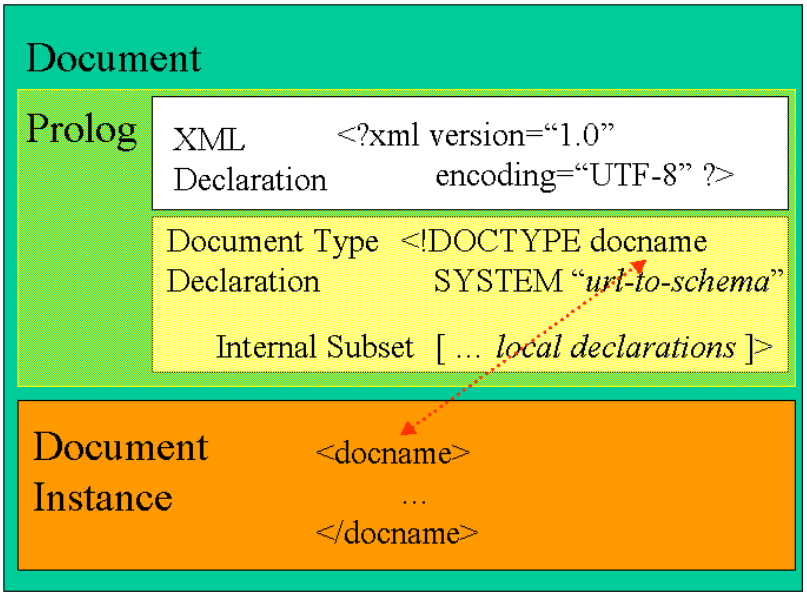
Bob Glushko

Using XML to Encode Document Type Models

Encoding a conceptual information model in XML means choosing elements or attributes as the containers for information, adding information about data types, applying naming rules, creating structures to organize repeated content components, and so on

If you've done a careful document analysis and design, the encoding stage is relatively simple and straight-forward and can even be automated in some cases

Structure of an XML Document



The Document Instance

Begins with the *root* element

Can contain text and binary data

Text consists of markup and character data

- Markup always starts with < or &
- So you can never use these symbols in character data and instead you use: < and &

Elements {and,or,vs} Attributes

Whether to use elements or attributes to contain information is often debated

Elements and attributes differ in what they can contain and this often guides which you should use

Elements can contain other elements while attributes can contain only strings or lists of strings

So elements must be used for any complex components but can be used for simple or primitive ones as well; attributes can only be used for "atomic" items of data

Elements carry the content that would generally appear in any presentation or rendering of the XML instance; attributes carry "strong" metadata or information that is useful in interpreting or presenting the element content

Elements {and,or,vs} Attributes [2]

Attributes are the only way to specify default values and can be constrained to a predefined set of enumerated values

Attributes are also the most sensible way to encode Boolean values

Attributes are inconvenient for long text, large values, or binary entities

If information is primarily encoded as attributes the XML instance can be significantly smaller

"Best practice" is contentious but many people use almost all elements and very few attributes, leaving the latter for just the "purest" metadata

Elements {and,or,vs} Attributes [3]

First way is preferred, second legal but not useful, and third is illegal

```
<Book>
  <Title>Moby Dick</Title>
  <Author>Melville</Author>
  <PublicationYear>1851</PublicationYear>
   <Categories>
      <Category>Fiction<Category>
      <Category>Whales<Category>
  </Categories>
</Book>
```

```
<Book title="Moby Dick"
       author="Melville"
       publicationYear="1851"
       category="Fiction  Whales"  />
```

```
<Book title="Moby Dick"
       author="Melville"
       publicationYear="1851"
       category="Fiction"
       category="Whales" />
```

Names

Names of elements and attributes must begin with a letter or _

After the first character, names can contain letters, digits, ., -,_

Names can't contain whitespace

Don't use : because it has an important role in namespaces (we'll get there in a minute)

Don't start a name with "xml" because it is reserved

Character Entities

Used for encoding "difficult" characters

Five are built-in:

& (&)
' (')
> (>)
< (<)
" (")

XML processors are required to support Unicode (65,536 possible characters) so you can encode Unicode characters (since most aren't on the keyboard) as *numeric character entities* like ç for ç

HTML has about 400 of these characters built in

CDATA (How Did The Previous Slide Work?)

CDATA is character data passed through to an application without being parsed

Useful for including XML and code samples in documents that contain markup characters that parser wants to treat as special like < > [

CDATA syntax:

```
<![CDATA[
                your unparsed XML example goes here
]]>
```

This strange syntax was chosen on the assumption that <![and]]> are unlikely to occur so they can be used as delimiters

The Previous (Previous) Slide

```
<list type="simple">
    <item>
        <p><![CDATA[    &amp; ]]>  (  &amp;  )</p>
    </item>
    <item>
        <p><![CDATA[    &apos; ]]> ( &apos;  )</p>
    </item>
    <item>
        <p><![CDATA[    &gt;]]> ( &gt;  ) </p>
    </item>
    <item>
        <p><![CDATA[    &lt;]]> ( &lt; )   </p>
    </item>
    <item>
        <p><![CDATA[    &quot;]]> ( &quot;  )  </p>
    </item>
</list>
```

The Namespace Problem

Thousands of XML *vocabularies*
– sets of elements and attributes – have been or are being developed by different companies, industries, vendors,...

We'd like to reuse these, since any well-designed vocabulary represents a lot of hard work that we'd rather not re-invent

But if some of the same vocabulary terms occur in more than one vocabulary, and they mean different things in each, how can we use them together?

Motivating Namespaces

```
<EventCalendar>
  <EventCategory>
     <Title>Literary and Cultural Events </Title>
     <Event>
         <Name>Oliver Sachs Book Signing</Name>
         <Location>Cody's Bookstore</Location>
         <Time>June 25, 2003 8pm</Time>
         <Book>
            <Author>
                 <Title> Dr. </Title>
                 <Name> Oliver Sachs </Name>
            </Author>
            <Title> The Island of the Color-Blind </Title>
            <Price> $12.95 </Price>
         </Book>
     </Event>
  </EventCategory>
</EventCalendar>
```

The Namespace Solution [1]

We can avoid name "collisions" where the same name means more than one thing by associating a vocabulary with a *Namespace*

The Namespace consists of a *prefix*
followed by a ":" and a URI (Universal Resource Identifier). This is not used to indicate a "place" or "file" but only to ensure uniqueness

The prefix, a short name that substitutes for the namespace, only has to be unique within the more tightly scoped context of the document instance

When namespaces are associated with elements, the *prefix* and the *local name* are together called the *qualified name* or QName

The Namespace Solution [2]

Let's define namespaces for three specialized XML vocabularies:

- For calendars and events – prefix="ev" and URI="urn:sims:doc-eng-lab:babl:events:0.01"
- For books – prefix "bk" and URI="http://www.mybookstore.com"
- For honorifics and insignias – prefix "hon" and URI="http://www.softwaremarketingresource.com/internationaladdressing.html"

Using Namespaces in the Calendar Example

In the root element of your instance you declare what namespaces it will contain, which allows the use of the 3 <Title> and 2 <Name> tags that mean different things

```
<ev:EventCalendar
  xmlns:ev="urn:sims:doc-eng-lab:babl:events:0.01"
  xmlns:hon="http://www.softwaremarketingresource.com/internationaladdressing.html"
  xmlns:bk="http://www.mybookstore.com">
```

```
  <ev:EventCategory>
     <ev:Title>Literary and Cultural Events </ev:Title>
     <ev:Event>
         <ev:Name>Oliver Sachs Book Signing</ev:Name>
         <ev:Location>Cody's Bookstore</ev:Location>
         <ev:Time>June 25, 2003 8pm</ev:Time>
         <bk:Book>
            <bk:Author>
                 <hon:Title> Dr. </hon:Title>
                 <bk:Name> Oliver Sachs </bk:Name>
            </bk:Author>
            <bk:Title> The Island of the Color-Blind </bk:Title>
            <bk:Price> $12.95 </bk:Price>
         </bk:Book>
     </ev:Event>
  </ev:EventCategory>
</ev:EventCalendar>
```

Whitespace

You can liberally use white space (space characters, newlines, tabs) to make your XML documents more readable

That's because when an XML processor (parser, etc) passes XML on to an application it ignores the white space (it hands off the logical tree structure of the document instead)

You can force white space to be preserved using an xml:space attribute added to any element

One little catch - the XML declaration must be at the exact start of the document, with no whitespace before it, or bad things happen

Comments

You can put comments almost anywhere in an XML document

Comments begin with <!--

Comments end with -->

Comments can't contain --

You should use them liberally... or better yet, you can use an element like <documentation> or <annotation> to convey the information and not use the comment syntax