Morf: a more flexible markup language

Alexander Powell
17 July 2012

Defining an XML markup scheme that allows us to capture all the structural, stylistic and semantic distinctions we deem to be important in our documents is sometimes easier said than done. If only we could make simultaneous use of several distinct markup schemes within a document, or employ staggered elements, say. In the belief that simply lowering the validation bar by demanding that documents just be well-formed isn't necessarily the answer when greater flexibility is required, I've been musing about the possibilities for more flexible markup languages. One result is Morf ('**mor**e **f**lexible markup language').

Morf markup is very like XML markup, but some of XML's constraints are dropped and new ones added in order to expand the space of permissible document structures. In particular the requirement that tags be nested is made optional. The hash character (#) can be used to append a numerical index to the element name, allowing even staggered pairs of the same element to be disambiguated:

<a> ... ... :	OK, disambiguation of different element close tags not a problem when the elements differ ...
<a> ... <a> ... ... :	... but now which open tags do the close tags correspond to?
<a#1>...<a#2>... </a#1> ... </a#2> :	Aha, now it's clear!

The possibility of employing multiple markup schemes in the same document is Morf's other major feature. To specify in a tag an element from a specific scheme, the element name is preceded by the scheme name (itself prefaced with a dollar character, $). The scheme name and the element name are separated by a period character ('.').

I should stress that this is all highly experimental at the moment; taking the idea of a new markup language seriously would of course require new tools for authoring and processing, and not just new validation mechanisms. Talking of which, I propose that Morf document classes be defined not via DTDs or schemas but by way of what I shall call Morf document definition (MDD) files. I'm thinking that these should be expressed in JSON format, and I'm currently playing with ways of enabling multiple markup schemes (set of elements and attributes) to be readily expressed, along with a range of properties against which Morf document instances could be validated. My intention is that Morf validation will be in some ways looser than validation of an XML file against a DTD or schema, but will still provide for a degree of type checking.

One markup scheme will be declared in the MDD file to be the primary scheme, and if in a tag the element name is not preceded by a scheme name then the element is presumed to belong to this primary scheme. The tag nesting requirement familiar from XML can be switched on or off for each scheme, although there is no global requirement that markup in one scheme must nest nicely with markup from another scheme. Nesting could perhaps also be switched on or off for each element within a scheme, relative to the other elements in the scheme.

Examples

The following are all valid Morf documents:

(1) Single markup scheme; nesting ‘on’:

```
<doc>
<title>Sample document 1</title>
<para>This is the first paragraph of the sample document.</para>
<para>This is the second paragraph of the sample document.</para>
</doc>
```

(2) Two markup schemes; nesting ‘on’ for both:

```
<doc><$B.demo1>
<title>Sample document 1</title>
<para>This is the <$B.demo2>first paragraph of the sample document.</para></$B.demo2>
<para>This is the second paragraph</$B.demo1> of the sample document.</para>
</doc>
```

(3) Two markup schemes; nesting ‘off’ for scheme B (bold shows content overlap between two demo1 elements):

```
<doc><$B.demo1#1>
<title>Sample document 1</title>
<para>This is the <$B.demo1#2>first paragraph of the sample document.</para></$B.demo1#1>
<para>This is the second paragraph</$B.demo1#2> of the sample document.</para>
</doc>
```

In document instance (2) note the use of a second markup scheme (markup scheme B), and the use of ‘.’ as a separator between the scheme name and element name. Two elements from scheme B are used, demo1 and demo2, and they form a nested pair, although their placement is unconstrained with respect to elements in the primary scheme. Because nesting has been switched off for scheme B in instance (3), the two demo1 elements are numbered to allow the disambiguation of closing tags using the ‘#’ character before an index number.

I am currently working on validation mechanisms. My aim is to keep validation basic and easy to implement using a simple LIFO stack-based parser. My intention is that the MDD file format should provide for (per scheme, assuming nesting is switched on):

1. Specification of permissible following sibling elements
2. Specification of permissible preceding sibling elements
3. Specification of an element’s allowable parent elements
4. Specification of an element’s permissible overall context elements (whether or not a parent)
5. Declaration of ‘free-range’ elements (allowed anywhere within a specified context element)
6. Specification of the number of times an element may occur in a document (Min <= N <= Max)
7. Definition of abstract named groups of element, which function as proxies for elements in specifications

(If nesting is not required in relation to a scheme then perhaps one could do little more than provide for specification of allowable contexts for open and close tags.)

Examples and details of the MDD format will be provided in due course.

(v1.1)