AttributeTemplates

Attribute Driven Templates

This document is based on the post (RT) Attribute Driven Templates. Please update it and add your comments, corrections, new constructions, better examples and explanations etc.

The idea is to design a template language such that HTML designers can edit it with Dreamweaver.

I'll give some background info and then I'll show how an attribute driven template language for Cocoon could look. The discussion is based on TAL, Tapestry and some of the functionallity in JXTemplateGenerator.

We need two kinds of constructions: expressions and directives.

Expressions

Expressions are the

```
${expr}
```

stuff in e.g. JXTG and should be use able both in text in attributes and element content. Do we need the "$"? Couldn't we just use

```
{expr}
```

as in XSLT?

- _I'm for

  ```
  {expr}
  ```

  if it's not a problem to implement, with an escape mechanism, of course, for when you actually need to write

  ```
  {expr}
  ```

  verbatim in your document. – [BertrandDelacretaz]_

- *No problem to implement at all, it is exactly what is used in attributes in XSLT so the escaping mechanism from XSLT can be used. I happen to have an implementation of it that I used in another project that we can reuse. – [DanielFagerstrom]*

IIUC TAL doesn't use embeded expressions but instead replace element and attribute content with replace directives (in attributes). It might make it even more Dreamweaver friendly as you can look at example content instead of expressions in your design window. But it makes template writing much harder IMO. We could have such replacement directives if people want it but, I would prefer having embeded expresions as well.

Directives

TAL uses one name spaced attribute for each directive, e.g.:

```
<p tal:condition="here/copyright"
   tal:content="here/copyright">(c) 2000</p>
```

Meaning that there will be a paragraph element with the content of `here/copyright` if `here/copyright` has any content otherwise, no output.

Tapestry instead uses only one attribute, `jwcid`, with the directive name as content.

```
<tr jwcid="loop">
   <td><span jwcid="insertFirstName">John</span></td>
   <td><span jwcid="insertLastName">Doe</span></td>
</tr>
```

Having either one attribute for each directive or having one special attribute with directives as content are the two main alternatives for attribute driven templates I would presume.

A problem with "attribute name as directive" is that xml attributes are unordered. So in the case you need e.g. a double loop over an element, test if you want to perform a loop or do a loop and test each element, a mechanism for deciding the order is needed. TAL uses a predetermined order for the directives. That is far to implicit for my taste and it doesn't solve the double loop. I would prefer to do something more explicit like:

```
<p tal:define-1="x /a/long/path/from/the/root"
    tal:condition-2="x"
    tal:content-3="x/txt"
    tal:attributes-4="class x/class">Ex Text</p>
```

The "-number" is my adition, TAL uses the predetermined order: define, condition, content, attributes.

IMO the "attribute name as directive" becomes to complicated so I will focus on the "attribute content as directive" approach. And give a proposal on how we could use that in Cocoon.

NIH

First some motivation about why I don't think that TAL or Tapestry templates are good alternatives for using as is in Cocoon (besides that its much more fun to implement them ourselves 😉). AFAIU both TAL and Tapestries directives are very closely connected to the both frameworks respective component models. We need something that works with our world view. Furthermore TAL is develped by the Zope community so everything is in Phyton style, and at that feels less natural, at least for me.

A Proposal

Based on the previous I propse that the attribute driven template language should be invoked like:

```
<element do="<directive>" attr1="...">
   <content>...</content>
</element>
```

The idea is that the attribute "do" triger the execution of the directive. "do" is just an example we could make the trigger attribute configurable and/or namespaced so that it doesn't colide with your "host" XML language.

When the directive is executed it has read only access to a context object containing the same kind of info as FOM, (and maybe some more stuff, more details have been discused earlier in the template design threads). It also has access the result of executing its containing element. I.e. the directive has access to the XML fragment:

```
<element attr1="...">
   <content>...</content>
</element>
```

Where the attributes are expanded wrt expressions and the content is evaluated.

Before describing how to handle the case with several directives for an element I'll make it more concrete by listing some core directives. They are mainly taken from JXTG and TAL.

if(<test>)

example:

```
<div do="if(count(cart/item) == 0)">
   Your cart is empty
<div>
```

- *I'd much prefer and <div tl:unless...> for the opposite case, as Vadim suggests on cocoon-dev, and would to the same for other constructs: tl:whatever syntax instead of tl:do="whatever". It's clearer and probably easier to implement – [BertrandDelacretaz]*

- *Agree with the use of if and unless, but not about the syntax. The syntax parsing might be easier but the need precedence order of attributes complicates and above all make it harder to understand what happens, (see http://marc.theaimsgroup.com/?l=xml-cocoon-dev&m=110243773223364&w=2 for motivation). – [DanielFagerstrom]*

forEach(<sequence>)

example:

```
<table>
   <tr do="forEach(cart/item)">
     <td>${index}</td>
     <td>${description}</td>
     <td>${price}</td>
   </tr>
</table>
```

for(var=<name>,begin=<expr>,end=<expr>,step=<expr>)

example:

```
<table>
   <tr do="for(var=i,begin=1,end=10)">
     <td>${cart[i]/index}</td>
     <td>${cart[i]/description}</td>
     <td>${cart[i]/price}</td>
   </tr>
</table>
```

context=<expr>

Set the current evaluation context for expressions.

example:

```
<table do="context=/shopping/cart">
   ...
</table>
```

let(<name>=<expr>,...,<name>=<expr>)

Define a number of variables for the current context. Observe that this not is a set operation it just defines the content of the variable `$<name>` in the current context, it doesn't intoduce any possibilities to have side effects.

example:

```
<h1 do="let(greeting=concat('Hello', user)>
   The value of greeting is ${$greeting}
<h1>
```

macro(<name>,<param-name>,...,<param-name>)

example:

```
<table do="macro(mytable,list,td-class)">
   <tr do="forEach($list)">
     <td class="${$class}">${item}</td>
   </tr>
</table>
```

We also need an evalBody as in JXTG. And maybe we should have a possibilty to give a name space to the macro name.

eval(<name>,context=<expr>,<param-name>=<expr>,...,<param-name>=<expr>)

Evaluates the named macro in either the curent context or the excplictliy chosen, with a number of forma parameters. The containing element is replaced by the result of the evaluation. And the content of the the containing element is available in the macro through "evalBody".

example:

```
<table do="eval(mytable,list=carts,class='checked-out')"/>
```

We could maybe even have the synatax:

```
<table do="mytable(list=carts,class='checked-out')"/>
```

for user defined macros.

replace(<expr>)

Replaces the containing element with the result of the expression, useful for inserting DOM etc from the flow layer.

example:

```
<div do="replace(.)"/>
```

content(<expr>)

Like replace but replaces only the content.

attributes(<name>=<expr>,...,<name>=<expr>)

Inserts an attribute.

Several directives

So, how do we handle multiple directives for one element? We could handle the TAL example above like:

```
<p do="let(x=/a/long/path/from/the/root);if(x);content(x/txt);attributes(class=x/class)">
   Ex Text
</p>
```

The idea is that when the leftmost directive is executed it has the ability to access the result of executing the directive sequence right of it in its current context. It will be the rightmost directive that has direct access to the containing element with evaluated attributes and body.

The corresponding tag based template expression would be (in pseudo jx):

```
<jx:let name="x" value="/a/long/path/from/the/root">
   <jx:if test="x">
     <jx:content select="x/txt">
       <jx:attribute name="class" value="x/class">
         <p>Ex Text</p>
       </jx:attribute>
     </jx:content>
   </jx:if>
<jx:let>
```

The jx:attribute and jx:content returns the patched variant of its respectively body. Not particulary natural constructions in a tag based template language, but hopefully it explains the direcive combination construction.

Connection to JXTG

The directives are almoust defined in such a way that they could be translated to a tag based templating language like:

```
directive(param_1=value_1,...,param_n=value_n)

<=>

<jx:directive param_1="value_1" ... param_n="value_n"/>
```

Maybe we could find a attribute directive language that allows for complete correspondance. And make tag or directive syntax selectible and in such way make this effort compatble with JXTG?

Externaly defined directives?

If we chose to allow extenally defined directives they probably should be given own namespaces to avoid clashes.

Formating?

IMO most of the basic formating can be done in a special conversion layer as we have discussed in the convertor thread.