HtmlToXsltExperiments

Converting HTML to XSLT for easier templating

After all the recent discussions on easier templating, here's an experiment based on Conal Tuohy's html-to-xslt transform (mucho thanks Conal), enhanced with *template rules* which make it easy to process elements without knowing where they appear in the input (like xsl:templates, which is what they are converted to).

The use case is the conversion of one of our xdoc files, I tried with our userdocs/forms/binding.xml document as an example input (find it here) and the results are pretty decent already, although the HTML template is very easy to understand.

This would be applicable to any output format, not only HTML.

I'm putting the source files inline instead of attaching them, in case people want to improve them we can take advantage of the wiki versioning to keep track of changes.

See Also

- The XSLTAL block in Cocoon, live samples at http://cocoon.zones.apache.org/demos/release/samples/blocks/xsltal/welcome
- discussion of this on cocoon-dev
- Templates - more info about templating, recent discussions, etc
- post on bitflux blog, where Christian experiments with the TAL syntax and tal:match templates

The HTML template

Here's my example template, it converts to above binding.xml xdoc document to HTML when used with the XSLT transform and sitemap shown below.

By converting to XSLT instead of implementing our own thing, we keep all the power of XSLT, but this is much easier to write and explain than an actual XSLT transform.

```
{{{<!--
xhtml template meant to be interpreted by html-to-xslt-v2.xsl

This is a "normal" HTML page using attribute-based templating instructions,
which the html-to-xslt XSLT transform converts to another XSLT transform
which in turn generates an HTML page with this layout (am I being clear here?)

The <div id="atl-templates"> element contains additional "element templates" which can
be used to easily reformat certain elements of the input. Only div elements
under this one are processed, the rest is ignored.
-->

<html>
<head>
<title>{document/header/title}</title>
<style>
body { font-family: Georgia,sans-serif; }
h1 { border-bottom: solid red 1px; text-align: center; }
.source { font-family: courier, monospace; white-space: pre; background-color: #FFFFCC; font-size: 90%; padding: 0.5em;}
.note { font-size: 80%; font-style: italic; }
</style>
</head>
<body>
<h1>{document/header/title}</h1>
<p class="note">The layout of this HTML page is defined doc2html-template.xhtml</p>

<div class="content" for-each="//document/body/s1">
<div class="s1">
<h2>{@title}</h2>
<div apply-templates="*"/>
</div>
</div>

<!-- Template rules definition section -->
<div id="atl-templates">
<h1>HTML templates</h1>
<p class="note">Do not edit below this line in a visual editor, work in HTML source code mode!</p>
(only div elements are considered when generating the XSLT transform, the rest is ignored)

This copies elements which are already XHTML in the input:
<div match="p|tr|td">
<div copy-with-templates="."/>
</div>
```

Convert source element into a div with a class:

Add a border to tables:

<table border="1">

</table>

</body>
</html>
}}}
Simple, not? That's all one must write to convert the XML document to HTML.

The XSLT transform

This is based on Conal Tuhoy's transform, it transforms the above HTML template into an XSLT transform.

I have enhanced it to generate addition xsl:template constructs for individual elements based on the section of the HTML template.

The this-xsl namespace is necessary to differentiate between actual XSLT transform instructions and XSLT instructions that must be generated in the output.

{{{<this-xsl:stylesheet version="1.0"
xmlns:this-xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsl="xsl-alias"
xmlns:nzetc="http://www.nzetc.org/template"
exclude-result-prefixes="nzetc">
<this-xsl:namespace-alias stylesheet-prefix="xsl"
result-prefix="this-xsl"/>

<this-xsl:template match="/">
<xsl:stylesheet version="1.0">
<this-xsl:comment>This stylesheet is generated from an HTML page - don't edit the stylesheet directly</this-xsl:comment>
<xsl:param name="id"/>
<xsl:param name="random">9</xsl:param>
<xsl:template match="/">
<this-xsl:apply-templates/>
<this-xsl:comment>Finally copy all processing instructions from the source doc</this-xsl:comment>
<xsl:copy-of select="//processing-instruction()"/>
</xsl:template>

<this-xsl:comment>Collect templates defined in input</this-xsl:comment>
<this-xsl:apply-templates select="//div[@id='atl-templates']/div" mode="atl"/>
</xsl:stylesheet>

</this-xsl:template>

<this-xsl:template match="div[@id='atl-templates']"/>

```
<this-xsl:template match="*">
<!-- handle an element which may have some XSL-style attributes -->
<this-xsl:choose>
<this-xsl:when test="@for-each">
<xsl:for-each select="{@for-each}">
<this-xsl:if test="@sort">
<xsl:sort select="{@sort}"/>
</this-xsl:if>
<this-xsl:call-template name="copy-and-apply-templates-to-children"/>
</xsl:for-each>
</this-xsl:when>
<this-xsl:when test="@if">
<xsl:if test="{@if}">
<this-xsl:call-template name="copy-and-apply-templates-to-children"/>
</xsl:if>
</this-xsl:when>
<this-xsl:when test="@apply-templates">
<xsl:apply-templates select="{@apply-templates}"/>
</this-xsl:when>
<this-xsl:when test="@copy-of">
<xsl:copy-of select="{@copy-of}"/>
</this-xsl:when>
<this-xsl:when test="@copy-with-templates">
<xsl:copy>
<xsl:copy-of select="@*"/>
<xsl:apply-templates/>
</xsl:copy>
</this-xsl:when>
<this-xsl:otherwise>
<this-xsl:call-template name="copy-and-apply-templates-to-children"/>
</this-xsl:otherwise>
</this-xsl:choose>
</this-xsl:template>

<this-xsl:template match="text()">
<this-xsl:call-template name="expand-variables">
<this-xsl:with-param name="value" select="."/>
</this-xsl:call-template>
</this-xsl:template>

<this-xsl:template match="@*">
<xsl:attribute name="{name()}">
<this-xsl:call-template name="expand-variables">
<this-xsl:with-param name="value" select="."/>
</this-xsl:call-template>
</xsl:attribute>
</this-xsl:template>

<this-xsl:template match="style">
<this-xsl:copy-of select="."/>
</this-xsl:template>

<this-xsl:template name="expand-variables">
<this-xsl:param name="value"/>
<this-xsl:choose>
<this-xsl:when test="contains($value,'{')">
<this-xsl:value-of select="substring-before($value,'{')"/>
<xsl:value-of>
<this-xsl:attribute name="select"><this-xsl:value-of select="substring-before(substring-after($value,'{'),'}')"/></this-xsl:attribute>
</xsl:value-of>
<this-xsl:call-template name="expand-variables">
<this-xsl:with-param name="value" select="substring-after(substring-after($value,'{'),'}')"/>
</this-xsl:call-template>
</this-xsl:when>
<this-xsl:otherwise>
<this-xsl:value-of select="$value"/>
</this-xsl:otherwise>
</this-xsl:choose>
</this-xsl:template>

<this-xsl:template name="copy-and-apply-templates-to-children">
<this-xsl:copy>
<this-xsl:copy-of select="@*[name()Unable to render embedded object: File (='for-each' and name()) not found.='if' and name()Unable to render embedded object: File (='apply-templates' and name()) not found.='copy-of' and name()!='sort']"/>
<this-xsl:apply-templates/>
</this-xsl:copy>
</this-xsl:template>
```

```
<this-xsl:template match="div" mode="atl">
<xsl:template match="{@match}">
<this-xsl:apply-templates/>
</xsl:template>
</this-xsl:template>

<this-xsl:template match="*[local-name()='meta'][@name='generator' and @content='HTML Tidy, see www.w3.org']"/>
<this-xsl:template match="*[local-name()='meta' or local-name()='META'][@http-equiv='Content-Type']"/>
</this-xsl:stylesheet>
```

The sitemap

In case you want to try this at home, here's the (dead simple) sitemap.

```
<?xml version="1.0"?>

<!-- atl-proto samples sitemap -->

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

<map:pipelines>

<map:pipeline type="caching">
<map:match pattern="">
<map:redirect-to uri="bindings.html"/>
</map:match>

<map:match pattern="*.html">
<map:generate src="{1}.xml"/>
<map:transform src="cocoon:/xslgen/doc2html.xsl"/>
<map:serialize type="html"/>
</map:match>

<map:match pattern="xslgen/*.xsl">
<map:generate src="{1}-template.html"/>
<map:transform src="html-to-xslt-v2.xsl"/>
<map:serialize type="xml"/>
</map:match>
</map:pipeline>

</map:pipelines>
</map:sitemap>
```