XSLT like transformations in Erlang

User Guide

Mikael Karlsson

Revision History
Revision 1.0 2002-10-25
First Draft
Revision P1.1 2003-02-03
Moved module xserl to xmerl application, renamed to xmerl_xs

Erlang has similarities to XSLT since both languages have a functional programming approach. Using the xpath implementation in the existing xmerl application it is possible to write XSLT like transforms in Erlang. One can also combine the transformations with the erlang scripting possibility in the yaws webserver to implement "on the fly" html conversions of xml documents.

Table of Contents

Terminology

XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language: Transformations

Introduction

XSLT stylesheets are often used when transforming XML documents, to other XML documents or (X)HTML for presentation. There are a number of brick-sized books written on the topic. XSLT contains quite many functions and learning them all may take some effort, which could be a reason why the author only has reached a basic level of understanding. This document assumes a basic level of understanding of XSLT.

Since XSLT is based on a functional programming approach with pattern matching and recursion it is possible to write similar style sheets in Erlang. At least for basic transforms. XPath which is used in XSLT is also already implemented in the xmerl application written i Erlang. This document describes how to use the XPath implementation together with Erlangs pattern matching and a couple of functions to write XSLT like transforms.

This approach is probably easier for an Erlanger but if you need to use real XSLT stylesheets in order to "comply to the standard" there is an adapter available to the Sablotron XSLT package which is written i C++.

This document is written in the Simplified Docbook DTD which is a subset of the complete one and converted to xhtml using a stylesheet written in Erlang.

Tools

xmerl

xmerl [http://sowap.sourceforge.net/] is a xml parser written in Erlang

xmerl_xpath

XPath is in important part of XSLT and is implemented in xmerl

xmerl_xs

xmerl_xs [xmerl_xs.yaws] is a very small module acting as "syntactic sugar" for the XSLT lookalike transforms. It uses xmerl_xpath.

yaws

Yaws [http://yaws.hyber.org/], Yet Another Webserver, is a web server written in Erlang that support dynamic content generation using embedded scripts, also written in Erlang.

Figure 1. The Yaws logo

Yaws is not needed to make the XSLT like transformations, but combining yaws and xmerl it is possible to do transformations of XML documents to HTML in realtime, when clients requests a web page. As an example I am able to edit this document using emacs with psgml tools, save the document and just do a reload in my browser to see the result. The parse/transform time is not visually different compared to loading any other document in the browser.

Transformations

When xmerl_scan parses an xml string/file it returns a record of:

```
-record(xmlElement, {
    name,
    parents = [],
    pos,
    attributes = [],
    content = [],
    language = [],
    expanded_name = [],
    nsinfo = [],% {Prefix, Local} | []
    namespace = #xmlNamespace{}
   }).
```

Were content is a mixed list of yet other xmlElement records and/or xmlText (or other node types).

xmerl_xs functions

Functions used:

xslapply/2	function to make things look similar to xsl:apply-templates.
value_of/1	Conatenates all text nodes within a tree.
select/2	select(Str, E) extracts nodes from the XML tree using xmerl_xpath.
built_in_rules/2	The default fallback behaviour, template funs should end with: `template(E)->built_in_rules(fun template/1, E).`

Note

Text is escaped using xmerl_lib:export_text/1 for "<", ">" and other relevant xml characters when exported. So the value_of/1 and built_in_rules/2 functions should be replaced when not exporting to xml or html.

Examples

Example 1. Using xslapply

original XSLT:

```
<xsl:template match="doc/title">
    <h1>
      <xsl:apply-templates/>
    </h1>
</xsl:template>
```

becomes in Erlang:

```
template(E = #xmlElement{ parents=[{'doc',_}|_], name='title'}) ->
    ["<h1>",
         xslapply(fun template/1, E),
     "</h1>"];
```

Example 2. Using value_of and select

```
  <xsl:template match="title">
    <div align="center"><h1><xsl:value-of select="." /></h1></div>
  </xsl:template>
```

becomes:

```
template(E = #xmlElement{name='title'}) ->
    ["<div align=\"center\"><h1>", value_of(select(".", E)), "</h1></div>"];
```

Example 3. Simple xsl stylesheet

A complete example with the XSLT sheet in the xmerl distribution.

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/xhtml1/strict">

  <xsl:strip-space elements="doc chapter section"/>
  <xsl:output
  method="xml"
  indent="yes"
  encoding="iso-8859-1"
  />

  <xsl:template match="doc">
    <html>
      <head>
        <title>
          <xsl:value-of select="title"/>
        </title>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="doc/title">
    <h1>
      <xsl:apply-templates/>
    </h1>
  </xsl:template>

  <xsl:template match="chapter/title">
    <h2>
      <xsl:apply-templates/>
    </h2>
  </xsl:template>
```

```
  <xsl:template match="section/title">
    <h3>
      <xsl:apply-templates/>
    </h3>
  </xsl:template>

  <xsl:template match="para">
    <p>
      <xsl:apply-templates/>
    </p>
  </xsl:template>

  <xsl:template match="note">
    <p class="note">
      <b>NOTE: </b>
      <xsl:apply-templates/>
    </p>
  </xsl:template>

  <xsl:template match="emph">
    <em>
      <xsl:apply-templates/>
    </em>
  </xsl:template>

</xsl:stylesheet>
```

Example 4. Erlang version

Erlang transformation of previous example:

```
-include("xmerl.hrl").

-import(xmerl_xs,
  [ xslapply/2, value_of/1, select/2, built_in_rules/2 ]).

doctype()->
    "<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 Transitional//EN\"\
 \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd \">".

process_xml(Doc)->
  template(Doc).

template(E = #xmlElement{name='doc'})->
    [ "<\?xml version=\"1.0\" encoding=\"iso-8859-1\"\?>",
      doctype(),
      "<html xmlns=\"http://www.w3.org/1999/xhtml\" >"
      "<head>"
      "<title>", value_of(select("title",E)), "</title>"
      "</head>"
      "<body>",
      xslapply( fun template/1, E),
      "</body>"
      "</html>" ];
```

```
template(E = #xmlElement{ parents=[{'doc',_}|_], name='title'}) ->
    ["<h1>",
     xslapply( fun template/1, E),
     "</h1>"];

template(E = #xmlElement{ parents=[{'chapter',_}|_], name='title'}) ->
    ["<h2>",
     xslapply( fun template/1, E),
     "</h2>"];

template(E = #xmlElement{ parents=[{'section',_}|_], name='title'}) ->
    ["<h3>",
     xslapply( fun template/1, E),
     "</h3>"];

template(E = #xmlElement{ name='para'}) ->
    ["<p>", xslapply( fun template/1, E), "</p>"];

template(E = #xmlElement{ name='note'}) ->
    ["<p class=\"note\">"
     "<b>NOTE: </b>",
     xslapply( fun template/1, E),
     "</p>"];

template(E = #xmlElement{ name='emph'}) ->
    ["<em>", xslapply( fun template/1, E), "</em>"];

template(E)->
    built_in_rules( fun template/1, E).
```

It is important to end with a call to `xmerl_xs:built_in_rules/2` if you want any text to be written in "push" transforms. That are the ones using a lot `xslapply( fun template/1, E )` instead of `value_of(select("xpath",E))`, which is pull...

The largest example is the stylesheet to transform this document from the Simplified Docbook XML format to xhtml. The source file is `sdocbook2xhtml.erl`.

Tips and tricks

for-each

The function for-each is quite common in XSLT stylesheets. It can often be rewritten and replaced by select/1. Since select/1 returns a list of #xmlElements and xslapply/2 traverses them it is more or less the same as to loop over all the elements.

position()

The XSLT position() and #xmlElement.pos are not the same. One has to make an own position in Erlang.

Example 5. Counting positions

```
<xsl:template match="stanza">
  <p><xsl:apply-templates select="line" /></p>
</xsl:template>

<xsl:template match="line">
  <xsl:if test="position() mod 2 = 0">  </xsl:if>
  <xsl:value-of select="." /><br />
</xsl:template>
```

Can be written as

```
template(E = #xmlElement{name='stanza'}) ->
    {Lines,LineNo} = lists:mapfoldl(fun template_pos/2, 1, select("line", E)),
    ["<p>", Lines, "</p>"].

template_pos(E = #xmlElement{name='line'}, P) ->
    {[indent_line(P rem 2), value_of(E#xmlElement.content), "<br />"], P + 1 }.

indent_line(0)->"  ";
indent_line(_)->"".
```

Global tree awareness

In XSLT you have "root" access to the top of the tree with XPath, even though you are somewhere deep in your tree.

The xslapply/2 function only carries back the child part of the tree to the template fun. But it is quite easy to write template funs that handles both the child and top tree.

Example 6. Passing the root tree

The following example piece will prepend the article title to any section title

```
template(E = #xmlElement{name='title'}, ETop ) ->
    ["<h3>", value_of(select("title", ETop))," - ",
     xslapply( fun(A) -> template(A, ETop) end, E),
     "</h3>"];
```

Utility functions

The module xmerl_xs contains the functions `mapxml/2`, `foldxml/3` and `mapfoldxml/3` to traverse `#xmlElement` trees. They can be used in order to build cross-references, see sdocbook2xhtml.erl for instance where `foldxml/3` and `mapfoldxml/3` are used to number chapters, examples and figures and to build the Table of contents for the document.

Future enhancements

More wish- than task-list at the moment.

- More stylesheets
- On the fly exports to PDF for printing and also more "polished" presentations.

References

1. XML source file [sdocbookex1src.yaws] for this document.
2. Erlang style sheet [sdocbookerlsrc.yaws] used for this document. (Simplified Docbook DTD).
3. Open Source Erlang [http://www.erlang.org/]