

Mini-Project II

XML-Processing, Spring 2004

By
Tor Jarle Sagen

All project files are available at:

<http://www.itu.dk/people/tor/XMR/>

Contents

Part 1	3
How the XSLT transformation works	3
Code details	3
The XSLT transformation compared to DOM or SAX API in java.....	4
Source files.....	5
ItuBlogSort.xsl.....	5
myblogsorted.xml	6
Part 2.....	7
How the XSLT transformation works	7
Code details.....	8
The XSLT transformation compared to the SAX API in java	8
Source files.....	9
ItuBlogXHTML.xsl	9
myblog.html	11
myblog.xml transformed in Internet Explorer.....	12
Part 3.....	13
How the XSLT transformation works	13
Code details.....	14
One or two stylesheets for sorting/transforming	14
Problem with many entries on the same date.....	14
Solution	14
Source files.....	16
ItuBlogWML.xsl.....	16
myblog.wml.....	18
myblog.wml tested in the Wapalizer	19

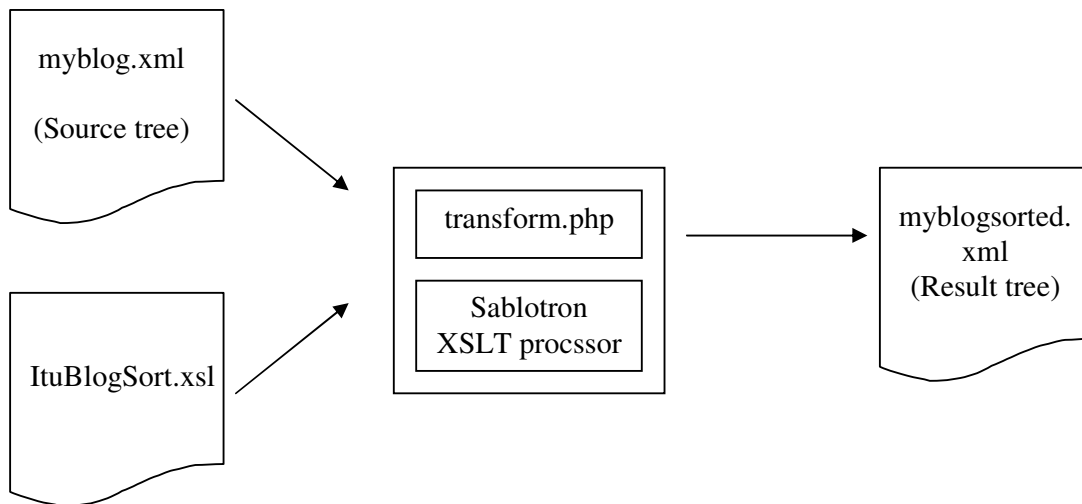
Part 1

Explain in your own words how your XSLT transformation works, and compare it to performing this kind of transformation using the DOM or SAX API in Java.

How the XSLT transformation works

To transform myblog.xml into a new XML-document where the entries are sorted, we are dependent on that three parts are in place. We need a well-formed XML-document (myblog.xml), an XSLT-stylesheet (ItuBlogSort.xsl) and a XSLT-processor (Sablotron).

The figure below shows how the transformation works.



The transformation can be tested by requesting the following URL:

```
http://www.itu.dk/people/tor/XMR/transform.php?xmlin=myblog.xml&xsl=ItuBlogSort.xsl&xmlout=myblogsorted.xml
```

Code details

All the coding done in part 1 is the 20 lines of code that make up ItuBlogSort.xsl. ItuBlogSort.xsl consists of two templates. The templates state what to look for in the source tree (myblog.xml), and what to put in the result tree (myblogsorted.xml). The first template is instanced when the root node of the document is reached. This template just puts the root element in the result tree and then calls another template.

The second template matches the <blog:blog> element in the source tree. The entries in the source document are sorted by year, month and day in a descending order. I then use the <xsl:copy-of> element, which makes it possible to copy parts of the source tree to result tree.

The XSLT transformation compared to DOM or SAX API in java

The sorting of the entries in myblog.xml could have been carried out using DOM together with a programming language like for instance Java. This process would have been much more complex than using XSLT. Using an imperative programming language, like java, you must tell the computer exactly what to do, and how to do it. This would have lead to a much more extensive program. For instance we would have had to know about sorting algorithms to get the sorting done. The reading and writing process would also have required some coding, that we avoid using XSLT.

I am not quite sure if it would be practical to use SAX for this job. Since SAX don't read the whole XML-document into the computers memory at once, like DOM does, but react to events, I can imagine that it would be more difficult to get the sorting done.

SAX is a read-only API, so it wouldn't be possible to write the result of the sorting to the XML-document myblogsorted.xml.

Using XSLT makes the task much easier. The XSLT-processor hides much of the complexity from us. We just tell the XSLT-processor to perform the sorting, and the processor takes care of the task, without bothering us about the details.

Source files

ItuBlogSort.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:blog="http://www.itu.dk/blog">
  <xsl:output method="xml" encoding="UTF-8"/>

  <xsl:template match="/">
    <blog:blog xmlns:blog="http://www.itu.dk/blog">
      <xsl:apply-templates select="blog:blog"/>
    </blog:blog>
  </xsl:template>

  <xsl:template match="blog:blog">
    <xsl:copy-of select="blog:author"/>
    <xsl:for-each select="blog:entry">
      <xsl:sort select="blog:date/@year" data-type="number" order="descending"/>
      <xsl:sort select="blog:date/@month" data-type="number" order="descending"/>
      <xsl:sort select="blog:date/@day" data-type="number" order="descending"/>
      <xsl:copy-of select="."/>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

myblogsorted.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<blog:blog xmlns:blog="http://www.itu.dk/blog">
  <blog:author>
    <blog:name>Michael Schwartzbach</blog:name>
    <blog:email>mis@brics.dk</blog:email>
  </blog:author>
  <blog:entry>
    <blog:date day="23" month="02" year="2004"/>
    <blog:title>XQuery Lecture at ITU</blog:title>
    <blog:text>
      I was very cold today.
    </blog:text>
    <blog:reply>
      <blog:author>Thomas Hildebrandt</blog:author>
      <blog:date day="24" month="02" year="2004"/>
      <blog:text>
        Today the weather is nice. Great day to look at houses!
      </blog:text>
      </blog:reply>
      <blog:reply>
        <blog:author>Student</blog:author>
        <blog:date day="25" month="02" year="2004"/>
        <blog:text>
          Why do you keep talking about houses??
        </blog:text>
      </blog:reply>
    </blog:entry>
  <blog:entry>
    <blog:date day="16" month="02" year="2004"/>
    <blog:title>XPath Lecture at ITU</blog:title>
    <blog:text>
      <blog:paragraph>
        Today I lectured on
        <blog:link
          url="http://www.brics.dk/~amoeller/XML/linking/index.html">XPath</blog:link>.
          which is a <blog:bold>very</blog:bold> useful topic.
        </blog:paragraph>
        <blog:paragraph>
          Look at the child axis here:
          <blog:image url="http://www.brics.dk/~amoeller/XML/linking/child.jpg"/>
        </blog:paragraph>
      </blog:text>
      <blog:reply>
        <blog:author>Thomas Hildebrandt</blog:author>
        <blog:date day="17" month="02" year="2004"/>
        <blog:text>
          Wow, what a nice picture! It looks a little bit like a house.
        </blog:text>
      </blog:reply>
    </blog:entry>
</blog:blog>
```

Part 2

Explain in your own words how your XSLT transformation works, and compare it to performing the transformation using the SAX API in Java.

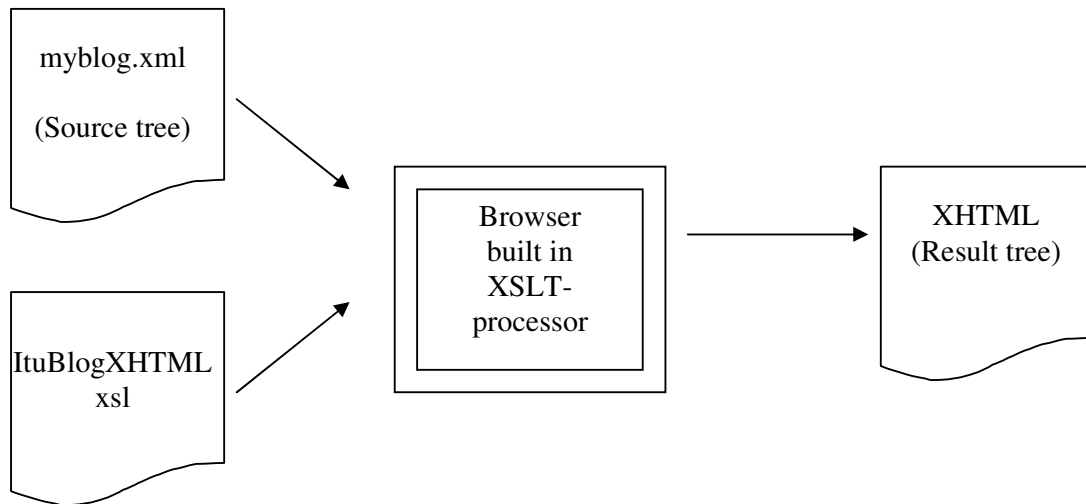
How the XSLT transformation works

In part 1 the XSLT- transformation was done on the server, using a php-script (transform.php) together with the Sablotron XSLT-processor. In part 2 the XSLT-transformation is done on the client using the built in XSLT-processor in a web browser (Explorer or Mozilla).

This is done by attaching processing instructions to the XML-document. The processing instruction attached to myblog.xml looks like this:

```
<?xml-stylesheet type="text/xsl" href="ItuBlogXHTML.xsl"?>
```

When the browser reads the XML-code, the built in XSLT-processor will carry out the transformation on-the-fly and show the result of the transformation in the user's browser window.



The transformation can be tested by requesting the following URL:

<http://www.itu.dk/people/tor/XMR/myblog.xml>

A pre-processed version of the XHTML version of myblog.xml is tested at <http://validator.w3.org/> and found to be valid XHTML 1.0 Transitional. The source for this file is to be found under “Source files” and online at: <http://www.itu.dk/people/tor/XMR/myblog.html>

Code details

The purpose of `ItuBlogXHTML.xsl` is to get the content from `myblog.xml` and transform it into valid XHTML. That means that a lot of the code that makes up this stylesheet is just ordinary html tags.

I have chosen to use an `<for-each>` element to go through each of the entries in the blog. Then named templates are called to put the content into appropriate html-tags.

The only problem that I encountered in the development of this stylesheet was when text was placed among other elements, like it is in the first entry in the blog. My solution was to use the built-in templates to handle ordinary text, and then apply templates for elements like `<blog:bold>` `<blog:link>` etc.

The XSLT transformation compared to the SAX API in java

It is my opinion that using an XSLT-stylesheet makes the programming task much easier, rather than using SAX/Java, when transforming an XML-document into XHTML. I guess it is just a question of choosing the right tool for the right job. XSLT is designed for transforming XML-documents into other formats, so it is no doubt that it is the right tool for this job.

In Mini-Project I, one of the sub-parts (`blog2XHTML.java`) was to transform the `Itublog` into XHTML using SAX/Java. It is no doubt that this was a much more complex task, and in my opinion much less appropriate way of getting the job done.

To some extent we can say that XSLT works like SAX. An XSLT-processor compares the nodes in the source tree (e.g. `myblog.xml`) to the templates in the stylesheet (e.g. `ItuBlogXHTML.xsl`). When a match is found, the template is instanced, and the result is added to the result tree.

In a somewhat similar way SAX reacts to events, which invokes a corresponding callback method that the programmer writes. We can say that XSLT and SAX is a push model, rather than a pull model like DOM.

Source files

ItuBlogXHTML.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:blog="http://www.itu.dk/blog" xmlns="http://www.w3.org/1999/xhtml" exclude-
result-prefixes="blog">
  <xsl:output method="xml" encoding="UTF-8" doctype-public="-//W3C//DTD XHTML
1.0 Transitional//EN" doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"/>
  <xsl:template match="/">
    <html>
      <head>
        <title>ITU WebBlog</title>
      </head>
      <body>
        <div align="center">
          <h1>ITU WebBlog</h1>
          by <xsl:value-of
select="blog:blog/blog:author/blog:name"/>
          <br/>
          Email: <a
href="mailto:{blog:blog/blog:author/blog:email}">
            <xsl:value-of
select="blog:blog/blog:author/blog:email"/>
          </a>
          <br/>
          <br/>
        </div>
        <xsl:apply-templates select="blog:blog"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="blog:blog">
    <xsl:for-each select="blog:entry">
      <table width="80%" border="0" align="center">
        <tr>
          <td bgcolor="#CCFF66">
            <xsl:call-template name="date"/>
          </td>
        </tr>
        <tr>
          <td>
            <xsl:call-template name="title"/>
          </td>
        </tr>
        <tr>
          <td>
            <xsl:call-template name="text"/>
          </td>
        </tr>
        <tr>
          <td>
            <xsl:call-template name="reply"/>
          </td>
        </tr>
      </table>
    </xsl:for-each>
  </xsl:template>
  <xsl:template name="date">
```

```

        <b>
            <xsl:value-of select="blog:date/@day"/>.<xsl:value-of
select="blog:date/@month"/>.<xsl:value-of select="blog:date/@year"/>
        </b>
    </xsl:template>
    <xsl:template name="title">
        <h2>
            <xsl:value-of select="blog:title"/>
        </h2>
    </xsl:template>
    <xsl:template name="text">
        <xsl:apply-templates select="blog:text"/>
    </xsl:template>
    <xsl:template match="blog:text">
        <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="blog:paragraph">
        <p>
            <xsl:apply-templates/>
        </p>
    </xsl:template>
    <xsl:template match="blog:bold">
        <b>
            <xsl:value-of select="text()"/>
        </b>
    </xsl:template>
    <xsl:template match="blog:image">
        <br/>
        
        <br/>
        <br/>
    </xsl:template>
    <xsl:template match="blog:link">
        <a href="{@url}">
            <xsl:value-of select="text()"/>
        </a>
    </xsl:template>
    <xsl:template name="reply">
        <xsl:if test="blog:reply">
            <h4>Reply to entry: <xsl:value-of select="blog:title"/>
            </h4>
            <xsl:apply-templates select="blog:reply"/>
        </xsl:if>
    </xsl:template>
    <xsl:template match="blog:reply">
        <p>
            <xsl:value-of select="blog:text/text()"/>
        <br/>

        Posted by <xsl:value-of select="blog:author"/>, <xsl:value-of
select="blog:date/@day"/>.<xsl:value-of select="blog:date/@month"/>.<xsl:value-of
select="blog:date/@year"/>
        </p>
    </xsl:template>
</xsl:stylesheet>

```

myblog.html

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>ITU WebBlog</title></head>
<body>
<div align="center">
<h1>ITU WebBlog</h1>
by Michael Schwartzbach<br />
Email: <a href="mailto:mis@brics.dk">mis@brics.dk</a><br /><br />
</div>

<table width="80%" border="0" align="center">
<tr>
<td bgcolor="#CCFF66"><b>16.02.2004</b></td>
</tr>
<tr>
<td><h2>XPath Lecture at ITU</h2></td>
</tr>
<tr>
<td><p>Today I lectured on <a
href="http://www.brics.dk/~amoeller/XML/linking/index.html">XPath</a>.
which is a <b>very</b> useful topic.</p>
<p>Look at the child axis here:
<br /><br /><br />
</p>
</td>
</tr>
<tr>
<td><h4>Reply to entry: XPath Lecture at ITU</h4>
<p>Wow, what a nice picture! It looks a little bit like a house.<br />
Posted by Thomas Hildebrandt, 17.02.2004</p>
</td>
</tr>
</table>
<table width="80%" border="0" align="center">
<tr>
<td bgcolor="#CCFF66"><b>23.02.2004</b></td>
</tr>
<tr>
<td><h2>XQuery Lecture at ITU</h2></td>
</tr>
<tr>
<td>I was very cold today.</td>
</tr>
<tr>
<td><h4>Reply to entry: XQuery Lecture at ITU</h4><p>
Today the weather is nice. Great day to look at houses!<br />
Posted by Thomas Hildebrandt, 24.02.2004</p>
<p>Why do you keep talking about houses??<br />
Posted by Student, 25.02.2004</p>
</td>
</tr>
</table>
</body>
</html>
```

myblog.xml transformed in Internet Explorer

ITU WebBlog

by Michael Schwartzbach
Email: mis@brics.dk

16.02.2004

XPath Lecture at ITU

Today I lectured on [XPath](#), which is a very useful topic.

Look at the child axis here:

Internet

Part 3

Explain in your own words how your transformation works. Try to explain the difference between having

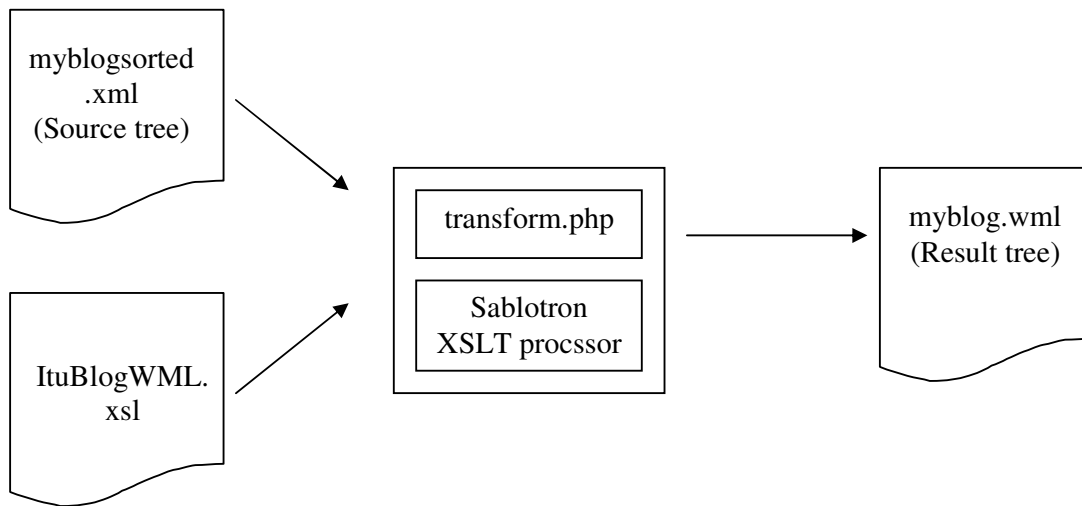
- one stylesheet for sorting and one for transforming to WML, or
- one stylesheet that both sorts and outputs to WML

The WML format above does not show how to handle the situation when there are many entries on the same date. Try to explain what may go wrong, and how it could be handled.

How the XSLT transformation works

The transformation works exactly like the transformation in part 1. In both cases the transformation is done on the server using transform.php/sablotron XSLT-processor. The only difference is the coding in the XSLT-stylesheet.

The output of this transformation is a WML-file (Wireless Markup Language), to be used with WAP (Wireless Application Protocol) and mobile phones.



The transformation can be tested by requesting the following URL:

<http://www.itu.dk/people/tor/XMR/transform.php?xmlin=myblogsorted.xml&xsl=ItuBlogWML.xsl&xmlout=myblog.wml>

Code details

The main section of ItuBlogWML.xsl is made up of two named templates. The template named “titles” examines each entry and produces the “titles card” to the result tree (myblog.wml).

The template named “entries” goes through each entry, to make up the cards for each day and replies.

The other templates in this stylesheet are just there to see to that the text comes out right, almost identical to the stylesheet in part 2.

One or two stylesheets for sorting/transforming

This question is a puzzle to me. I can imagine that using one stylesheet to do both tasks would be more complicated.

Using one stylesheet would reduce the traffic to the server. This could be in interest if it is a lot of documents that need sorting/transforming.

Problem with many entries on the same date

If we transform an xml-file with many entries on the same date, we would get a problem in the WML result tree. We would get duplicate anchors in the “titles” card.

```
<a href="#day16">XPath Lecture at ITU</a>
<a href="#day16">XML Lecture at ITU</a>
```

We would also get duplicate id's on different cards.

```
<card id="day16" title="XPath Lecture at ITU">
<card id="day16" title="XML Lecture at ITU">
```

We would get the same problem with the replies.

Solution

This problem can easily be solved by adding a time attribute to the date element under each entry. The date element in the XML-file will then look like this:

```
<blog:date day="23" month="02" year="2004" time="1030"/>
```

This will separate the entries from each other and avoid the problem with duplicate date entries. With a little adjustment in the XSLT-stylesheet, the “titles” card in the output WML-file would look like this for two entries with the same date.

```
<a href="#day16-1030">XPath Lecture at ITU</a>
<a href="#day16-1340">XML Lecture at ITU</a>
```

When then adjust the cards for each day to match the anchor in the “titles” card.

```
<card id="day16-1030" title="XPath Lecture at ITU">  
<card id="day16-1340" title="XML Lecture at ITU">
```

After doing the same with replies, the problem is solved.

Source files

ItuBlogWML.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:blog="http://www.itu.dk/blog" exclude-result-prefixes="blog">
  <xsl:output method="xml" doctype-public="-//WAPFORUM//DTD WML 1.3//EN"
doctype-system="http://www.wapforum.org/DTD/wml13.dtd"/>
  <xsl:template match="/">
    <wml>
      <xsl:apply-templates select="blog:blog"/>
    </wml>
  </xsl:template>
  <xsl:template match="blog:blog">
    <xsl:call-template name="titles"/>
    <xsl:call-template name="entries"/>
  </xsl:template>
  <xsl:template name="titles">
    <card id="titles" title="mis@brics.dk BLOG">
      <xsl:for-each select="blog:entry">
        <p>
          <b>
            <xsl:value-of select="blog:date/@day"/>-
<xsl:value-of select="blog:date/@month"/>:
          </b>
          <a href="#day{blog:date/@day}">
            <xsl:value-of select="blog:title"/>
          </a>
          <br/>
          <a href="#replies{blog:date/@day}">
            <xsl:variable name="rep">
              <xsl:value-of
select="count(blog:reply)"/>
            </xsl:variable>
            <xsl:choose>
              <xsl:when test="$rep > 1">
                <xsl:copy-of select="$rep"/>
replies</xsl:when>
              <xsl:when test="$rep > 0">
                <xsl:copy-of select="$rep"/>
reply</xsl:when>
              <xsl:otherwise> </xsl:otherwise>
            </xsl:choose>
          </a>
        </p>
      </xsl:for-each>
    </card>
  </xsl:template>
  <xsl:template name="entries">
    <xsl:for-each select="blog:entry">
      <xsl:variable name="cardDate">
        <xsl:value-of select="blog:date/@day"/>
      </xsl:variable>
      <xsl:variable name="cardTitle">
        <xsl:value-of select="blog:title/text()"/>
      </xsl:variable>
    </xsl:for-each>
  </xsl:template>

```

```

<card id="day{$cardDate}" title="{ $cardTitle}">
  <xsl:apply-templates select="blog:text"/>
  <do type="options" label="Replies">
    <go href="#replies{$cardDate}"/>
  </do>
</card>
<card id="replies{$cardDate}" title="{ $cardTitle}">
  <xsl:for-each select="blog:reply">
    <p>
      <b>
        <xsl:value-of select="blog:date/@day"/>-
<xsl:value-of select="blog:date/@month"/> (<xsl:value-of
select="blog:author"/>):</b>
        <xsl:value-of select="blog:text/text()"/>
      </p>
    </xsl:for-each>
  </card>
</xsl:for-each>
</xsl:template>
<xsl:template match="blog:text">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="blog:paragraph">
  <p>
    <xsl:apply-templates/>
  </p>
</xsl:template>
<xsl:template match="blog:bold">
  <b>
    <xsl:value-of select="text()"/>
  </b>
</xsl:template>
<xsl:template match="blog:image">
  (image)
</xsl:template>
</xsl:stylesheet>

```

myblog.wml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.3//EN"
"http://www.wapforum.org/DTD/wml13.dtd">
<wml>
  <card id="titles" title="mis@brics.dk BLOG">
    <p>
      <b>23-02:</b>

      </b>
      <a href="#day23">XQuery Lecture at ITU</a>
      <br/>
      <a href="#replies23">2 replies</a>
    </p>
    <p>
      <b>16-02:</b>

      </b>
      <a href="#day16">XPath Lecture at ITU</a>
      <br/>
      <a href="#replies16">1 reply</a>
    </p>
  </card>
  <card id="day23" title="XQuery Lecture at ITU">
    I was very cold today.
    <do type="options" label="Replies">
      <go href="#replies23"/>
    </do>
  </card>
  <card id="replies23" title="XQuery Lecture at ITU">
    <p>
      <b>24-02 (Thomas Hildebrandt):</b>
      Today the weather is nice. Great day to look at houses!
    </p>
    <p>
      <b>25-02 (Student):</b>
      Why do you keep talking about houses??
    </p>
  </card>
  <card id="day16" title="XPath Lecture at ITU">
    <p>
      Today I lectured on
      XPath.
      which is a <b>very</b> useful topic.
    </p>
    <p>
      Look at the child axis here:

      (image)

      </p>
    <do type="options" label="Replies">
      <go href="#replies16"/>
    </do>
  </card>
  <card id="replies16" title="XPath Lecture at ITU">
    <p>
```

```
<b>17-02 (Thomas Hildebrandt):</b>  
Wow, what a nice picture! It looks a little bit like a house.  
</p>  
</card>  
</wml>
```

myblog.wml tested in the Wapalizer (<http://www.gelon.net/>)

