

XSLT and Document Transformation
7th CEEnet workshop on networks technology
Budapest, Hungary

Sébastien 'sbi' BLONDEEL (<sbi@april.org>)

August 2001

Abstract

We have studied how to produce XML documents, edit them with different tools, and make sure they match a given grammar, or Document Type Definition. We will now study how to produce and use stylesheets to XML data or documents into different kinds of outputs.

Contents

1	Follow-up on yesterday's lecture	2
2	Editing XML according to a DTD: PSGML	3
3	Installing XML tools	4
4	Stylesheet languages	5
5	Some practical examples of XSLT	7
6	A simple example of usage of XSLT: example 01	7
7	Comments on example 01	8
8	Matching the root element: example 02	9
9	Comments on example 02	10
10	Using siblings: example 03	10
11	Comments on example 03	11
12	Sorting: example 04	11

13 Comments on example 04	12
14 Creating sublists: example 05	12
15 Comments on example 05	13
16 Creating a table of contents: example 06	14
17 Comments on example 06	15
18 To know more about XSLT...	15
19 XPath	16
20 Example: the transformation of this document	17
21 What we have not studied...	18
22 References	18
23 Lab on XSLT and Document Transformation	18

1 Follow-up on yesterday's lecture

Due to technical problems (hopefully solved today!), I was unable yesterday to easily show you the following things:

- Example of usage of XML for data storage of a sketch program (taken from Dia).

```

<?xml version="1.0"?>
<dia:diagram
xmlns:dia="http://www.lysator.liu.se/~alla/dia/">
  <dia:diagramdata>
    <dia:attribute name="background">
      <dia:color val="#ffffff"/>
    </dia:attribute>
    <dia:attribute name="paper">
[... ]
      <dia:attribute name="elem_height">
        <dia:real val="4.8"/>
      </dia:attribute>
    </dia:object>
  </dia:layer>
</dia:diagram>

```

- Example of usage of XML for writing a single source format for a written document, and use it in several output formats (taken from Linux Device Drivers, 2nd

Edition), using the DocBook DTD (See the printed copy you were handed out; there was a bit too much to put on a slide).

```
<sect1><title>Sources of Further Information</title>
<para>
<indexterm><primary>web sites related to Linux
kernels</primary></indexterm>
<indexterm><primary>kernels</primary><secondary>web sites
about</secondary></indexterm>
<indexterm><primary>Internet sites about Linux
kernels</primary></indexterm>
Most of the information we provide in this book is extracted
directly from the kernel sources and related documentation.
In particular, pay attention to the <filename>Documentation</filename>
directory that is found in the kernel source tree. There is a
wealth of useful information there, including
documentation of an increasing part of the kernel API
(in the <filename>DocBook</filename> subdirectory).
</para>
```

- The last two backends I have programmed for this presentation: HTML with or without chunks. All versions of all formats should now be available on my website.

If we sum up, I have written 4 stylesheets, one (simple, for demonstration and teaching) DTD, and 3 to 4 lecture documents since last year, for a total of 12 to 16 usable backend documents. I am therefore starting to really benefit, this year, from last year's work.

2 Editing XML according to a DTD: PSGML

(This slide should have been included in yesterday's presentation.)

PSGML¹ is an Emacs mode to assist the editing of an XML or SGML file according to a given DTD (specified with a system identifier or a public identifier).

It is triggered by the extension of the file being edited (`xml` or `sgml`) and these are the important keystrokes:

- C-c C-p loads (and checks) the DTD as specified in the header
- C-c C-e insert an element where the cursor is, asking you which one if the DTD gives a choice. When an element is chosen and it has implied attributes, PSGML will prompt you for them. It will also recursively fill in all it is sure about for the descendants of that element
- C-c C-v validates the document against the DTD (using `nsgmls`).

¹<http://www.lysator.liu.se/projects/>

I am not sure something similar exists for vi-clones or other editors users; as for me, I know the DTDs I use, or check the documentation.

If that package is not included by default in your GNU/Linux distribution, then you need to get the source code and install it, compiling and putting the `elc` files in the right place. There is a French article² detailing the steps of this installation process. It focuses on the (deprecated) LinuxDoc DTD but can be used with other DTDs. But this is less and less likely to be the case as major documentation projects switch to XML and as XML support grows in most systems.

3 Installing XML tools

Most tools (above all XSLT tools) are written in JAVA so as to ensure the maximum portability. A JAVA package is a `.jar` file, actually in the ZIP format, holding a whole tree structure with classes, that must be present in the `CLASSPATH` environment variable to be found. This is the case for XT, that needs XP, both can be downloaded from James Clark's website³.

Python and Perl support for XML is still being worked on and are slowly getting usable and fast enough. The important Python XML suite of the moment is 4Suite⁴.

The environment variable specifying the location of catalog files must be properly set if you wish to use public identifiers. Your lab document will give you the details on how to do that.

If you wish to use \TeX ⁵ as an intermediate language to produce printable versions of your document, you need to install a complete \TeX distribution, which is not an easy thing to do. I suggest you use the `tetex` distribution, that comes at least with GNU/Linux distributions.

One needs to install a recent enough version of Java (for example 1.2.2) and have the "java" executable in his "PATH". One needs then to download and install XP⁶ and XT⁷. It is a good thing to be able to observe something like:

```
$ java -fullversion
java full version "Linux_JDK_1.2.2_RC4"
$ echo $CLASSPATH
/home/sbi/java/sax.jar:/home/sbi/java/xp.jar:/home/sbi/java/xt.jar
```

...where of course the `CLASSPATH` environment variable points to the installed ".jar" files.

Then I suggest you add the following function in your shell startup files in order to be able to just type "xt" to proceed with some XSLT transformation:

²<http://www.linux-france.org/article/appli/emacs/>

³<http://www.jclark.com/>

⁴<http://fourthought.com/4Suite/>

⁵<http://www.tug.org/>

⁶<http://www.jclark.com/xml/xp/index.html>

⁷<http://www.jclark.com/xml/xt.html>

```
function xt() {
  echo "XT $*" >&2
  java com.jclark.xml.sax.Driver $*
}
```

4 Stylesheet languages

A stylesheet transforms an XML document according to rules and templates applied recursively, starting at the root element. The document needs to be well-formed, not necessarily valid.

Of course, it is probably a good idea to include some kind of validation step somewhere in the chain (if not in tests, at least in production) in order to make sure that no funny thing can happen.

DSSSL⁸ (*Document Style Semantics and Specification Language*). is a stylesheet language similar to Scheme, embedded in XML documents.

A full DSSSL stylesheet with block elements, inline elements, and inheriting parameters (courtesy of DocBook: the Definitive Guide⁹).

```
<!DOCTYPE style-sheet PUBLIC "-//James Clark//DTD DSSSL Style Sheet//EN">

<style-sheet>
<style-specification>
<style-specification-body>

(element chapter
  (make simple-page-sequence
    top-margin: 1in
    bottom-margin: 1in
    left-margin: 1in
    right-margin: 1in
    font-size: 12pt
    line-spacing: 14pt
    min-leading: 0pt
    (process-children)))

(element title
  (make paragraph
    font-weight: 'bold
    font-size: 18pt
    (process-children)))

(element para
```

⁸<http://www.jclark.com/dsssl/>

⁹<http://www.docbook.org/>

```

(make paragraph
  space-before: 8pt
  (process-children))

(element emphasis
  (if (equal? (attribute-string "role") "strong")
    (make sequence
      font-weight: 'bold
      (process-children))
    (make sequence
      font-posture: 'italic
      (process-children))))

(element (emphasis emphasis)
  (make sequence
    font-posture: 'upright
    (process-children)))

(define (super-sub-script plus-or-minus
  #!optional (sosofo (process-children)))
  (make sequence
    font-size: (* (inherited-font-size) 0.8)
    position-point-shift: (plus-or-minus (* (inherited-font-size) 0.4))
    sosofo))

(element superscript (super-sub-script +))
(element subscript (super-sub-script -))

</style-specification-body>
</style-specification>
</style-sheet>

```

You can see that this format mixes Scheme and Lisp-like code with a minimal XML-wrapping, which does not make it very easy to use with normal XML tools.

XSL¹⁰ (*eXtensible Style Language*) is a recommendation of the W3C. The stylesheets are XML documents. This is the stylesheet language used for the various versions of this presentation.

The language uses two *namespaces*: the `<xsl:` and the `<fo:`.

The first one, *XSL Transformations* or XSLT, is specialized in the recursive transformation of XML trees into usually other trees (XML or XHTML: because the HTML tags appear in the stylesheet between XSL-T tags, all must be closed and properly nested or else the XSLT would not be a well-formed XML document. Therefore the HTML produced must be XHTML).

The second part, *Formatting Objects* or FO, is an XML vocabulary for specifying formatting semantics for printed backends. Unfortunately, this directly produces

¹⁰<http://www.w3.org/Style/XSL/>

PDF in a way that can hardly be better than using directly plain \TeX or \LaTeX2e , and last time I checked nearly no tool works properly and recognizes the whole current state of the draft. They all are incompatible, limited to small documents when they use some flavour of \TeX because they pile up useless nested environments quickly filling up the memory, and can hardly work with their own examples, and not with the examples shipped with other alpha- or beta- applications.

Example of an XSL stylesheet (using FO):

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:fo="http://www.w3.org/1999/XSL/Format">

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

  <xsl:template match="emphasis">
    <fo:sequence font-style="italic">
      <xsl:apply-templates/>
    </fo:sequence>
  </xsl:template>

  <xsl:template match="emphasis/emphasis">
    <fo:sequence font-style="upright">
      <xsl:apply-templates/>
    </fo:sequence>
  </xsl:template>

</xsl:stylesheet>
```

We will explain more in detail the workings of XSLT and how to use it to select some information from the XML file or display it in a certain way.

5 Some practical examples of XSLT

We will now study and comment some practical examples of XSLT. All these examples are available on the website. For each one of them, we will study the three files involved: XML, XSL, output HTML.

6 A simple example of usage of XSLT: example 01

XML:

```
<a>foo</a>
```

XSL:

```
<?xml version="1.0"?>
<!DOCTYPE xsl:stylesheet>
<xsl:stylesheet
  method="html"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
<xsl:output method="html" />

<xsl:template match="*">
  <xsl:value-of select="." />
</xsl:template>

</xsl:stylesheet>
```

Output:

foo

7 Comments on example 01

- No need for a DTD or to be DTD-compliant with something.

But of course it is better to have that kind of safe-check!

- XSL works slightly differently in different output methods; we will just focus on the "html" output method here.

The rationale is as follows: it is difficult to control the flow of characters in XML since the blank characters are not really significant in between elements. Better produce HTML and use special software to produce plain text from the HTML if one needs plain text.

- We need a header and a footer in the stylesheet, which will not get repeated from here on.

They will of course be used in the different examples, but will not be repeated in these slides any longer to help you focus on the things illustrated.

- XSLT works with templates called when their matching condition is met, and recursively from there.

One can call templates from inside a template, with a number of different rules (selecting things about elements or calling templates by names or in some special modes).

- the `xsl:value-of` element returns the value corresponding to the XPath expression that follows

In this particular case, "." means the expanded character content of the current element (that is to say the concatenation of all its internal text fields, which is usually interesting only if that element is a simple text field).

8 Matching the root element: example 02

XML:

```
<root>
  <son>First son</son>
  <son>Second son</son>
  <son>Third son</son>
</root>
```

XSL:

```
<xsl:template match="/">
  <html>
    <head><title>XSL output</title></head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="son">
  <p>Found a son of content ``<xsl:value-of select="."/>''</p>
</xsl:template>
```

Output (HTML):

```
<html>
```

```

<head>
<title>XSL output</title>
</head>
<body>
  <p>Found a son of content ``First son''</p>
  <p>Found a son of content ``Second son''</p>
  <p>Found a son of content ``Third son''</p>
</body>
</html>

```

9 Comments on example 02

- The matching of the root element of the XML document and the production of the html header and footer will not be repeated either in the following examples.
- This example shows the recursivity of the way XSL works.
- This example creates HTML. Because the XSL document must be well-formed XML, then the code it produces is itself well-formed XML, therefore the HTML produced must be XHTML.

This rule is sometimes annoying and it is difficult to find a way around it. In the examples we will deal with in the lab session, we should not be impaired by that. But the functional nature of XSLT programming makes one think in new and unexpected ways.

10 Using siblings: example 03

XML:

```

<root>
  <mousquetaire>Atos</mousquetaire>
  <mousquetaire>Portos</mousquetaire>
  <mousquetaire>Aramis</mousquetaire>
</root>

```

XSL:

```

  <p>The Mousquetaires are:
    <xsl:apply-templates select="*/mousquetaire"/>.</p>
[... ]
<xsl:template match="mousquetaire">

```

```

<xsl:if test="preceding-sibling::mousquetaire">
  <xsl:choose>
    <xsl:when test="not(following-sibling::mousquetaire)">
      <xsl:text> and </xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>, </xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:if>
<xsl:value-of select="."/>
</xsl:template>

```

Output (HTML):

```

<p>The Mousquetaires are:
  Atos, Portos and Aramis.</p>

```

11 Comments on example 03

- We introduced the `xsl:if` element
- It work with a test condition that depends on the siblings of the current node of the XML tree.
- We can notice as well the syntax to negate a test condition
- The IF-THEN-ELSE statement in XSLT exists and you can see how it works and how it is called: `xsl:choose` with embedded `xsl:when` then `xsl:otherwise` sub-elements.

12 Sorting: example 04

XML:

```

<root>
  <mousquetaire>Atos</mousquetaire>
  <mousquetaire>Portos</mousquetaire>
  <mousquetaire>Aramis</mousquetaire>
</root>

```

XSL:

<p>The Mousquetaires are, in alphabetical and descending order:

```
<xsl:for-each select="*/mousquetaire">
  <xsl:sort select="." data-type="text" order="descending"/>
  <xsl:value-of select="."/><xsl:text> </xsl:text>
</xsl:for-each>
</p>
```

Output (HTML):

```
<p>The Mousquetaires are, in alphabetical and descending order:
  Portos Atos Aramis </p>
```

13 Comments on example 04

- Introducing the xsl:for-each element, with a select attribute accepting an XPath expression.
- That element can have one or several children xsl:sort
- These children take a number of attributes making it possible to customize the way the sorting works
- Using several sorting elements makes the system use several sorting key, the first one being the most significative one.

14 Creating sublists: example 05

XML:

```
<root>
  <mousquetaire>
    <name>Atos</name>
    <age>30</age>
    <weight>60</weight>
  </mousquetaire>
  <mousquetaire>
    <name>Portos</name>
    <age>40</age>
    <weight>100</weight>
  </mousquetaire>
```

```
<mousquetaire>
  <name>Aramis</name>
  <age>35</age>
  <weight>55</weight>
</mousquetaire>
</root>
```

XSL:

```
<xsl:for-each select="*/mousquetaire">
  <li>Name: <xsl:value-of select="name"/></li>
  <ul>
    <li>Age: <xsl:value-of select="age"/></li>
    <li>Weight: <xsl:value-of select="weight"/></li>
  </ul>
</xsl:for-each>
```

Output (HTML):

```
<ul>
<li>Name: Atos</li>
<ul>
<li>Age: 30</li>
<li>Weight: 60</li>
</ul>
<li>Name: Portos</li>
<ul>
<li>Age: 40</li>
<li>Weight: 100</li>
</ul>
<li>Name: Aramis</li>
<ul>
<li>Age: 35</li>
<li>Weight: 55</li>
</ul>
</ul>
```

15 Comments on example 05

- This is very similar to what we have done earlier, trying to give some kind of structure to the list of participants here

- Once again we can observe the fact that the HTML produced is in fact XHTML
- This is a simple case of recursive inner calls, but I just wanted to make you more comfortable with the language
- We used `xsl:for-each` but we could have used another `xsl` element; they are probably interchangeable

16 Creating a table of contents: example 06

XML:

```
<root>
  <section>
    <title>Section 1</title>
    <content>This is the content of the first section</content>
  </section>
  <section>
    <title>Section 2</title>
    <content>This is the content of the second section</content>
  </section>
  <section>
    <title>Section 3</title>
    <content>This is the content of the third section</content>
  </section>
</root>
```

XSL:

```
<xsl:template match="/">
  <html>
    <head><title>XSL output</title></head>
    <body>
      <h1>Table of contents</h1>
      <ul><xsl:apply-templates mode="toc" select="*/section"/></ul>
      <xsl:apply-templates select="*/section"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="section" mode="toc">
  <li><xsl:value-of select="title"/></li>
</xsl:template>
```

```
<xsl:template match="section">
  <h1><xsl:value-of select="title"/></h1>
  <p><xsl:value-of select="content"/></p>
</xsl:template>
```

Output (HTML):

```
<h1>Table of contents</h1>
<ul>
<li>Section 1</li>
<li>Section 2</li>
<li>Section 3</li>
</ul>
<h1>Section 1</h1>
<p>This is the content of the first section</p>
<h1>Section 2</h1>
<p>This is the content of the second section</p>
<h1>Section 3</h1>
<p>This is the content of the third section</p>
```

17 Comments on example 06

- we need to use "modes" in order to distinguish between the interpretation of a section in the table of contents or as a part of the document
- we browse some parts of the XML tree several times
- of course, the right way to do this is to have each item of the table of contents be a cross-link to the corresponding section; this is actually what happens in the stylesheets for this presentation.

18 To know more about XSLT...

XSLT is a language where it sometimes takes a lot of code to do what other languages could express in just a few characters of line. It is not aimed to be concise, but logical and easy to manage from a software engineering point of view.

Norman Walsh wrote tutorials¹¹ about XSL, and these slides are self-sufficient and rather complete. I would need more time to review all what they mention, we would not have time to do everything again in exercises, and you would probably end up pretty confused.

¹¹<http://www.nwalsh.com/docs/tutorials/index.html>

I will let it to the faster or the most curious among you to explore those issues on their own, they have all the information they need. The rest is only a matter of industrialization, automation and procedures.

19 XPath

XPath¹² is the W3C recommendation for expressing paths and referring to nodes in an XML tree.

It is used in XSLT to select recursively some descendants or brothers of the node currently being worked on. There is a good piece of documentation¹³ on XPath as well as about XSL elements under the XSLT tool SAXON, even if I do not use that tool and I prefer using XT.

The SAXON¹⁴ documentation for XPath explains more clearly (to me) than the normative reference, a number of things:

- Constants, Variable References
 - string constants are written between single or double quotes
 - numeric constants are written using the Java rules
 - there are no boolean constants; one should use the true() and false() function calls instead
 - the value of a variable can be obtained using its name preceded with a dollar sign: \$name; the variable must have been declared first
- Parentheses and operator precedence
 - if an expression is enclosed in parentheses it takes precedence
 - else, operator precedence is as follows: predicate, child or descendant nodes, union, arithmetic multiplicative operations, arithmetic additive operations, comparisons, test of equality, boolean and, boolean or

- String Expressions

There are a number of functions operating on strings, letting one transform an expression in a string, normalize the blanks in a string, translate some characters in other characters, testing whether something is a substring of a given string, and one can even generating an id for an element on the fly, this id being compatible with the order of the elements and unique in the session.

- Numeric Expressions

One can obtain the position of node among a node list (this is how I numbered the slides and the table of contents in the HTML versions of this document), one can transform an expression in a number obeying certain formatting rules,

...

¹²<http://www.w3.org/TR/xpath>

¹³<http://saxon.sourceforge.net/>

¹⁴<http://saxon.sourceforge.net/>

- Boolean Expressions

The number 0, the empty string, the empty node are treated as false, everything else is treated as true (this is not the case in DSSSL, which can lead to confusion). One can perform classical boolean tests and check whether a given node has a language attribute (inherited or specified).

- Nodeset Expressions

One can specify relative or absolute paths, use wildcards, refer the the parent node, the preceding-sibling node, the attributes named this or that in the current node or one of its descendant nodes, select all descendant nodes having such a parent and such an attribute, ...

20 Example: the transformation of this document

```
<xsl:template match="/presentation">
  <html><head><title>
    <xsl:apply-templates select="title"/>
  </title></head>
  <body>
    <xsl:call-template name="introduction"/>
    <xsl:call-template name="toc"/>
    <xsl:apply-templates select="slide"/>
  </body>
</html>
</xsl:template>
```

One can call a template by name, to make the stylesheet more modular and readable. One can as well apply templates recursively, selecting templates matching an XPath expression or not. XSLT is a functional language designed to be easy to implement, but the drawback is it is very verbose and one cannot reffect variables. This is particularly a problem when one needs to transform XML-defined tables into L^AT_EX₂ε, where the syntax is extremely different (as opposed to what happens with the HTML syntax for tables).

This is an example of an IF-THEN-ELSE statement in XSLT (excerpt from the stylesheets of this presentation):

```
<xsl:variable name="prevslide">
  <xsl:choose>
    <xsl:when test="$slidenumber > 1">
      <xsl:number value="$slidenumber - 1" format="01"/>
      <xsl:text>.html</xsl:text>
    </xsl:when>
    <xsl:otherwise>index.html</xsl:otherwise>
  </xsl:choose>
</xsl:variable>
```

```
</xsl:choose>
</xsl:variable>
```

Imagine what this will start looking like if you have to nest several such tests! I wrote in XSLT a function to return the age of someone knowing today's date and that person's birthday. One first has to compare the year numbers, then if they are equal the months numbers, then the day in the month. It quickly took a whole screen!

Another problem is that whitespace is handled in a mysterious way to many in XSLT. In languages where whitespaces are not very significant, such as HTML, this is not a problem, but in languages designed in a very different way, such as \LaTeX 2 ϵ , this can be a problem. It is very hard to have a properly formatted, easy to read, stylesheet, while still producing a properly formatted, easy to read, and without bugs, generated document.

21 What we have not studied...

XSLT is much richer than what we had the time to mention here. One can pass it command-line options, ask it to output its results in files as opposed to the standard output, react differently according to the context, and more. I will repeat and insist on the fact that the time and resources only let us through a quick overview of this language and methods, and that nothing can replace a personal research of documents available on the web or in books.

22 References

- XML on W3C
- James Clark: XP, XT
- Norman Walsh: website using XML, stylesheets for DocBook, presentations and tutorials for DocBook and XSL
- SAXON: another easy to read documentation about XSL and XPath

23 Lab on XSLT and Document Transformation

In the laboratory session following up this presentation, you will do the following things:

- Install the necessary software to be able to use XT to perform XSLT XML document transformation.
- Retrieve the examples from the website (or from pc01:XML) and uncompress them

- Test the XSLT document transformation on the examples given, checking the results are what is expected, and displaying the resulting HTML in a browser to check that all is right
- Re-use the structured document with the list of people attending (or the list of people in track2) and write some new stylesheets to create different documents from the same data file, using the features of XSLT we have mentioned here, and maybe others you will find browsing the web and the references.