XSL: Characteristics, Status and Potentials for the Humanities

Wendell Piez
Mulberry Technologies, Inc.
17 West Jefferson Street, Suite 207
Rockville, MD 20850
Phone: 301/315-9635
Fax: 301/315-8285
wapiez@mulberrytech.com
`http://www.mulberrytech.com`

July 23, 2000

Mulberry
Technologies, Inc.

XSL: Characteristics, Status and Potentials for the Humanities

Exhibits

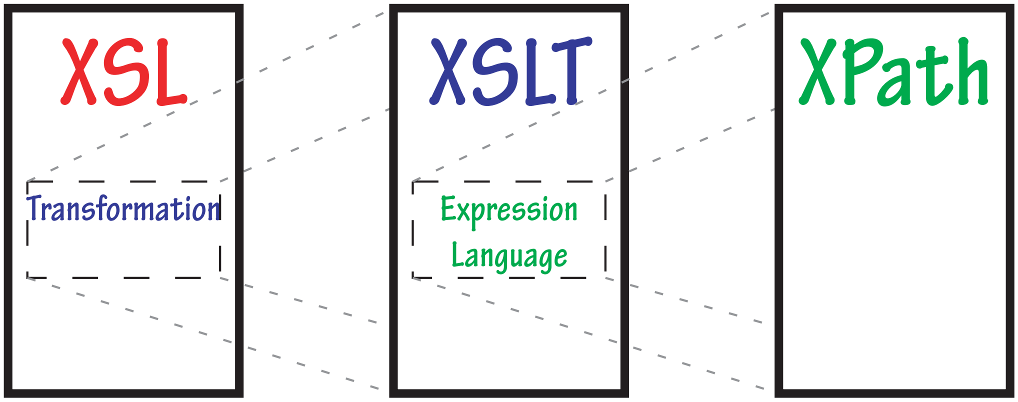
What is XSL

- Extensible Stylesheet Language
- A companion initiative to XML from W3C (Worldwide Web Consortium)
- Targets *presentation* of XML documents
    - On web
    - In print
    - Through other media

XSL Specification in three documents

- **XSL Working Draft**
    - Includes specification of *formatting objects* (XSL FO)
    - Complex and versatile presentation of structured data
        - print, hypertext, other media
    - Not finished yet (July 2000)
- **XSLT (XSL Transformations) Recommendation**
    - Language for transformation of markup structures
    - Main/intended application: arbitrary XML to XSL FO conversion
    - Other applications: XML to XML, HTML, plain text conversion
    - Completed November 1999
- **XPath Recommendation**
    - Expression language for handling components of an XML document
    - Intended for linking, querying, but also XSLT
    - Completed November 1999

XSL, XSLT, and XPath



XSL

Transformation

XSLT

Expression Language

XPath

Transformation + Formatting Objects (FO)

Transforming XML into XML

Addressing parts of XML document

What is XSLT

- Recommendation of the W3C
  `http://www.w3.org/TR/xslt`
- Language to specify transformations from one (XML) structured data object to another
- A *declarative* (“functional”?) language
- Requires XSLT processor (“engine”) to run
  - Several freely available

XSLT processor

- Executable program
- May be on any platform
- Input:
  - source document (any XML)
  - stylesheet (transformation specification) in XSLT
- Output
  - data object(s) for application (e.g. browser), *or*
  - data file: may be
    - XML
    - HTML
    - Plain text formats
    - others (with extensions)

Features of XSLT

- Portable, reusable
- Works on any well-formed XML
  - Does not require a DTD
- Takes advantage of structured document markup
  - Good for “down-” and “cross-translations” (XML to XML; XML to text)
  - Less good for “up-translations” (text to XML)

Applying XSL to TEI

- Create HTML out of TEI
- Other output formats will include XSL Formatting Objects (for print)
- See an example stylesheet in Exhibit 1
- See example output in **`frankenstein.html`**
- Also see:
  - Sebastian Rahtz at Oxford (**`http://users.ox.ac.uk/~rahtz/tei/`**)
  - Nakhimovsky, et al. at Colgate (**`http://csproj.colgate.edu/TextTools.htm`**)

Exhibit 1

A simple TEI to HTML stylesheet

This example of an XSLT stylesheet creates HTML and will serve for a range of simpler TEI documents.

```
<!DOCTYPE xsl:stylesheet [
<!ENTITY ldquo "&#8220;" >
<!-- "&#8220;" =double quotation mark, left-->
<!ENTITY rdquo "&#8221;" >
<!-- "&#8221;" =double quotation mark, right-->
]>

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

<xsl:output method="html"/>

<!-- this template is for diagnostics .. delete when done! -->
<xsl:template match="*">
  <FONT SIZE="-1" COLOR="green">[<xsl:value-of select=
  "local-name()"/>]</FONT>
  <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template match="/">
  <HTML>
    <HEAD>
      <TITLE>
        <xsl:value-of select=
        "TEI.2/teiHeader/fileDesc/titleStmt/title"/>
      </TITLE>
    </HEAD>
    <BODY>
      <xsl:apply-templates/>
    </BODY>
  </HTML>
</xsl:template>

<xsl:template match="TEI.2">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="teiHeader">
  <!-- Hide the header for now...
  <xsl:apply-templates/> -->
</xsl:template>

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

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

<xsl:template match="titlePage">
  <DIV align="center">
    <xsl:apply-templates/>
  </DIV><HR/>
</xsl:template>

<xsl:template match="docTitle">
  <xsl:call-template name="marktarget"/>
</xsl:template>
```

```
<xsl:template match="titlePart">
  <H1><xsl:apply-templates/>
  <xsl:if test="following-sibling::titlePart[@type='subtitle']">
    <xsl:text>; </xsl:text>
  </xsl:if>
  </H1>
</xsl:template>

<xsl:template match="titlePart[@type='subtitle']">
  <H2>
  <xsl:apply-templates/>
  </H2>
</xsl:template>

<xsl:template match="docAuthor|docDate">
  <H4><xsl:apply-templates/></H4>
</xsl:template>

<xsl:template match="div">
  <xsl:apply-templates/><HR/>
</xsl:template>

<xsl:template name="marktarget">
  <A>
    <xsl:attribute name="NAME">
      <xsl:value-of select="local-name()"/>
      <xsl:number level="multiple" format="1.1.1.1.1" count=
      "div|head|p|l|biblopener|salute|signed|trailer|docTitle"/>
    </xsl:attribute>
    <xsl:apply-templates/>
  </A>
</xsl:template>

<xsl:template match="div/head" priority="1">
  <BR/>
  <H2><xsl:call-template name="marktarget"/></H2>
</xsl:template>

<xsl:template match="div/div/head" priority="2">
  <H3><xsl:call-template name="marktarget"/></H3>
</xsl:template>

<xsl:template match="div/div/div/head" priority="3">
  <H4><I><xsl:call-template name="marktarget"/></I></H4>
</xsl:template>
```

```
<xsl:template match="div/div/div/div/head" priority="4">
  <H4><xsl:call-template name="marktarget"/></H4>
</xsl:template>

<xsl:template match="div/div/div/div/div/head" priority="5">
  <xsl:message>[Warning: all heads deeper than 4 divs down are H5]
  </xsl:message>
  <H5><xsl:call-template name="marktarget"/></H5>
</xsl:template>

<!-- Text content elements -->

<xsl:template match="address">
  <DL>
    <xsl:apply-templates/>
  </DL>
</xsl:template>

<xsl:template match="p">
  <P>
    <xsl:call-template name="marktarget"/>
  </P>
</xsl:template>

<xsl:template match="lb">
  <BR/>
</xsl:template>

<xsl:template match="epigraph|cit[@rend='block']">
  <TABLE WIDTH="80%" BORDER="0" CELLPADDING="0" CELLSPACING="10">
    <xsl:apply-templates/>
  </TABLE>
</xsl:template>

<xsl:template match="epigraph/lg|cit[@rend='block']/q/lg">
  <TR><TD ALIGN="LEFT"><FONT SIZE="-1">
    <xsl:apply-templates/>
  </FONT></TD></TR>
</xsl:template>

<xsl:template match="epigraph/bibl|cit[@rend='block']/bibl">
  <TR><TD ALIGN="RIGHT"><FONT SIZE="-1">
    <xsl:apply-templates/>
  </FONT></TD></TR>
</xsl:template>
```

```
<xsl:template match="l">
  <DD><xsl:call-template name="marktarget"/>
  </DD>
</xsl:template>

<xsl:template match="lg">
  <DL><xsl:apply-templates/>
  </DL>
</xsl:template>

<xsl:template match="eg">
  <PRE>
    <xsl:apply-templates/>
  </PRE>
</xsl:template>

<xsl:template match="item">
  <LI>
    <xsl:apply-templates/>
  </LI>
</xsl:template>

<xsl:template match="addrLine|trailer|salute|dateline|signed">
  <DT>
    <xsl:apply-templates/>
  </DT>
</xsl:template>

<!-- Inline elements  -->

<xsl:template match="title|emph|hi[@rend='i']">
  <I><xsl:apply-templates/></I>
</xsl:template>

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

<xsl:template match="mentioned|soCalled|q[@direct='y']">
  <xsl:text>&ldquo;</xsl:text>
    <xsl:apply-templates/>
  <xsl:text>&rdquo;</xsl:text>
</xsl:template>
```

```
<xsl:template match="code|ident|*[@rend='ident']">
  <TT><xsl:apply-templates/></TT>
</xsl:template>

<xsl:template match="pb">
  <FONT COLOR="#6300AA" SIZE="-1" FACE="sans-serif"><B>
    <xsl:text>[</xsl:text>
    <xsl:value-of select="@ed"/>
    <xsl:text> ed. </xsl:text>
    <xsl:value-of select="@n"/>
    <xsl:text>]</xsl:text>
  </B></FONT>
</xsl:template>

</xsl:stylesheet>
```

Stylesheets to convert to non-XML formats

- One result format of XSLT processor is plain text
- Output in plain text formats, including flat formats, can be controlled from markup in the source
- For example, COCOA suitable for loading into the TACT analytical engine
    - Sample of output in Exhibit 2
    - Stylesheet in Exhibit 3

Exhibit 2

Sample of COCOA Output

```
{[N Frankenstein]}  {[C 1]} CHAPTER 1
  I am by birth a Genevese, and my family is one of the most
distinguished of that republic. My ancestors had been for many years
counsellors and syndics, and my father had filled several public
situations with honour and reputation. He was respected by all who
knew him for his integrity and indefatigable attention to public
business. He passed his younger days perpetually occupied by the
affairs of his country; a variety of circumstances had prevented his
marrying early, nor was it until the decline of life that he became
a husband and the father of a family.
  As the circumstances of his marriage illustrate his character, I
cannot refrain from relating them. One of his most intimate friends
was a merchant who, from a flourishing state, fell, through numerous
mischances, into poverty. This man, whose name was Beaufort, was of
a proud and unbending disposition and could not bear to live in
poverty and oblivion in the same country where he had formerly been
distinguished for his rank and magnificence. {[1831p 19]} Having
paid his debts, therefore, in the most honourable manner, he
retreated with his daughter to the town of Lucerne, where he lived
unknown and in wretchedness. My father loved Beaufort with the
truest friendship and was deeply grieved by his retreat in these
unfortunate circumstances....
```

Exhibit 3

Stylesheet for converting TEI to COCOA

The output created appears in Exhibit 2

```
<!DOCTYPE xsl:stylesheet [
<!ENTITY ldquo '"' >
<!-- double quotation mark, left would be &#x201C;-->
<!ENTITY rdquo '"' >
<!-- double quotation mark, right would be &#x201D;-->

<!-- Open and close delimiters for the COCOA tags: -->
<!ENTITY co "<xsl:text> {[</xsl:text>" >
<!ENTITY cc "<xsl:text>]} </xsl:text>" >

]>
```

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

<xsl:output method="text" encoding="iso-8859-1"/>

<xsl:strip-space elements="div"/>

<xsl:template match="teiHeader">
  <!-- Hide the header for now...
  <xsl:apply-templates/> -->
</xsl:template>

<xsl:template match="titlePage">
    <!-- Also hide the title page ...
    <xsl:apply-templates/> -->
</xsl:template>

<xsl:template match="text()">
  <!-- overrides the built-in template to normalize
       spacing on output -->
  <xsl:value-of select="normalize-space(.)"/>
</xsl:template>

<xsl:template match="div">
  <xsl:text>&#xA;</xsl:text>
  &co;<xsl:text>N </xsl:text>
  <xsl:call-template name="narrator"/>&cc;
  <xsl:choose>
    <xsl:when test="@type='Letter'">
      &co;<xsl:text>L </xsl:text>
      <xsl:value-of select="@n"/>&cc;
    </xsl:when>
    <xsl:when test="@type='Chapter'">
      &co;<xsl:text>C </xsl:text>
      <xsl:value-of select="@n"/>&cc;
    </xsl:when>
    <xsl:otherwise/>
  </xsl:choose>
  <xsl:apply-templates/>
</xsl:template>
```

```
<xsl:template name="narrator">
  <!-- Call this template any time the 'narrator' parameter
       might be taken to change.
       The narrator is stated on the 'ana' attribute of elements
       in the novel. -->
  <xsl:value-of select="ancestor-or-self::*[@ana][1]/@ana"/>
</xsl:template>

<xsl:template match="pb">
    <!-- COCOA tag for PB... -->
    &co;<xsl:value-of select="@ed"/>
    <xsl:text>p </xsl:text>
    <xsl:value-of select="@n"/>&cc;
</xsl:template>

<!-- COCOA format (designed for FRANKENSTEIN):
     N -  indicates narrator
     L - indicates letter (1-4)
     C - indicates chapter
     p1831 - indicates page number (in 1831 edition)
-->

<xsl:template match="div/head">
  <xsl:value-of select="translate(.,
    'abcdefghijklmnopqrstuvwxyz',
    'ABCDEFGHIJKLMNOPQRSTUVWXYZ')"/>
</xsl:template>

<!-- Text content elements -->

<xsl:template match="list|address|p|l|lg|opener">
  <xsl:text>&#xA;  </xsl:text>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="addrLine|trailer|salute|dateline|signed">
  <xsl:text>&#xA;   </xsl:text>
  <xsl:apply-templates/>
</xsl:template>
```

```
<!-- Inline elements  -->

<xsl:template name="decorate-inline">
  <xsl:param name="front" select="''"/>
  <xsl:param name="back" select="''"/>
  <xsl:if test="preceding-sibling::node()">
    <xsl:text> </xsl:text>
  </xsl:if>
  <xsl:value-of select="$front"/>
  <xsl:apply-templates/>
  <xsl:value-of select="$back"/>
  <xsl:if test="following-sibling::node()">
    <xsl:text> </xsl:text>
  </xsl:if>
</xsl:template>

<xsl:template match="title">
  <xsl:call-template name="decorate-inline">
    <xsl:with-param name="front" select="'_'"/>
    <xsl:with-param name="back" select="'_'"/>
  </xsl:call-template>
</xsl:template>

<xsl:template match="mentioned|soCalled|q[@direct='y']">
  <xsl:call-template name="decorate-inline">
    <xsl:with-param name="front" select="'&ldquo;'"/>
    <xsl:with-param name="back" select="'&rdquo;'"/>
  </xsl:call-template>
</xsl:template>

<xsl:template match="emph|hi[@rend='i']">
  <xsl:call-template name="decorate-inline">
    <xsl:with-param name="front" select="'*'"/>
    <xsl:with-param name="back" select="'*'"/>
  </xsl:call-template>
</xsl:template>

</xsl:stylesheet>
```

Stylesheets for analytical processing

- Advantages
    - Requires only a “stylesheet”, not “programming”
    - Stylesheets can be easily reused, refitted
    - System can be web-based
    - Tools are free
    - Skills are transferable
- Disadvantages
    - XSLT not designed for all tasks: sometimes a stretch
    - Performance will suffer on larger texts

Kinds of Analytical Work

- Text processing
    - Indexes
    - Concordances
    - Counting things
- Text representation
    - E.g., “text mapping”
- Validation of markup

Assembling, grouping, sorting

- Most straightforward: stylesheets that take advantage of explicit markup
- Also: an XSLT stylesheet can process more than one XML document at once
- Archival finding aids
  - XML register of original order
  - XSLT generates sorted order(s)
- Bibliographies
  - Different views
  - Indexes
- Web site / document repository
  - XSLT generates links pages

An XSLT “concordancer” for TEI

- XSLT stylesheet can take as a parameter a string to be located
- Could generate a classic concordance or index
- Can generate links to full text
- See `fr-natural.html` for example output

Strengths and weaknesses of the technique

- Strengths
    - Results are web-ready, fairly easy to read and use
    - Web-ready can mean web-based
- Weaknesses
    - XSLT doesn't do string-processing “naturally”: must be forced
    - Can't link from hits broken by markup
    - Can't find “loose” hits such as case-insensitive etc.

XSLT as an analytical tool box

- To infer facts about text, XSLT can exploit either
    - markup, or
    - simple string-handling functions of XPath standard
        - `contains(DIV, "natural")` returns “true” if (the first) child node `DIV` contains string `"natural"`
        - `string-before(`*`string1`*`,`*`string2`*`)`, `string-after(`*`string1`*`,`*`string2`*`)`
        - etc.
- These rudimentary functions can (should) be extended
- Could be provided through an XML browser/XSL editor combination

Text representation or “text mapping”

- XSLT can convert one form of XML to another
- For example, SVG (Scalable Vector Graphics) as a target format
  - combine this with analytical processing...
  - ...and create graphs of features in the source text
- Look at **`natural.svg`** for an example
  - (requires an SVG viewer: try the Java application from IBM Alphaworks or the browser plug-in from Adobe)
    - **`www.alphaworks.ibm.com`**
    - **`http://www.adobe.com/support/downloads/main.html#s`**
- This is even more powerful if processes can be chained

Validation and normalization

- An XSLT transformation can show where a text (or its markup) meets or departs from an expected model
  - Can generate warnings
  - Can generate normalized or corrected form
- E.g. use XSLT to batch-process TEI Headers in a repository
- Work is proceeding on XSLT-based validation
  - Cf. *Schematron* (Rick Jelliffe, Academia Sinica), **`http://www.ascc.net/xml/resource/schematron/schematron.html`**

Near-conclusions

- Production work
  - XSLT for HTML is immediately useful
  - Other formats have great potential
- Analytical work
  - XSLT provides a useful framework
  - Can be extended for better performance, new functionality
    - Pattern recognition
    - Tokenizing
  - Creates output “ready to wear”
    - HTML or other forms

Questions Raised

- XSLT-based processing is limited by
  - markup-centric, tree-structured data model
  - top-down processing
- Is this the best data model?
- Is there a best data model?

New/Old Lessons

- XSLT (and any such declarative approach?) is both
    - Powerful
    - Resistant (cannot go beyond the data model)
- Capitalizes on markup
- Yet demonstrates how encoding only partially represents the phenomenal richness of texts
    - “texts” exist as reflections of the richness of our perceptions of them....