Journal of the Text Encoding Initiative

Issue 4 | March 2013
Selected Papers from the 2011 TEI Conference

Logging the Abbot: Reflection-Oriented XSLT Programming for Corpora Conversion and Verification

Brian L. Pytlik Zillig

Electronic version
URL: http://journals.openedition.org/jtei/722
DOI: 10.4000/jtei.722
ISSN: 2162-5603

Publisher
TEI Consortium

Electronic reference
Brian L. Pytlik Zillig, « Logging the Abbot: Reflection-Oriented XSLT Programming for Corpora Conversion and Verification », *Journal of the Text Encoding Initiative* [Online], Issue 4 | March 2013, Online since 27 February 2013, connection on 22 April 2019. URL : http://journals.openedition.org/jtei/722 ; DOI : 10.4000/jtei.722

Logging the Abbot: Reflection-Oriented XSLT Programming for Corpora Conversion and Verification

Brian L. Pytlik Zillig

1. Curation and Conversion

1 The growing number of electronic texts available to scholars affords us the opportunity to think about combining heretofore separate collections for analytical purposes. Distinct XML collections sometimes require conversion into a common format such as that developed by the Text Encoding Initiative (TEI) Consortium. TEI is, for many purposes, a satisfactory format for text corpus aggregation, though not always without attendant difficulties. As John Unsworth writes, "The 'I' in TEI sometimes stands for interchange, but it never stands for interoperability.... (I)f there's a single interoperable format ... it has to be a common or baseline representation that is technically valid and intellectually acceptable in multiple systems" (Unsworth 2011). While a precise definition of such a format may still be evolving, it is clear that an interoperable text markup format should probably, on the most fundamental level, permit, require, and exclude features. TEI is, in the abstract, able to accommodate each of these conditions, and it therefore represents considerable progress toward interoperability. Yet because TEI permits local customizations it is no longer a representation that is fully shared. To arrive at a condition of interoperability, a reliable and verifiable conversion process is crucial. While it can be relatively easy to verify that no words are inadvertently lost or rendered out of sequence for a small text collection, it becomes progressively more difficult with more texts. Curation and verification routines that rely on individual human scrutiny will not operate at a large scale or in a reasonable amount of time.

2 In early 2007, the MONK (Metadata Offer New Knowledge) Project began to develop a procedure for batch converting varying collections of XML-encoded texts into a

specialized application of TEI P5 that we called TEI-Analytics (TEI-A). That effort produced a command-line application, Abbot, which works by analyzing the XML schema that describes the document structure to which the target collection should be converted. Developed by the author, Stephen Ramsay, and Martin Mueller, Abbot uses that analysis—an enumeration of allowable elements and their associated attributes—to programmatically generate a very large XSLT stylesheet that is used for the conversion. Abbot, at last, makes it possible to eliminate customizations or other differences between markup systems, either for the short or long term.

3 While Abbot has already been described in detail elsewhere (see Pytlik Zillig 2009), it might be helpful to briefly explain how it moves from an analysis of the desired output schema to generating a stylesheet which does the conversion. Figure 1 illustrates the Abbot workflow.

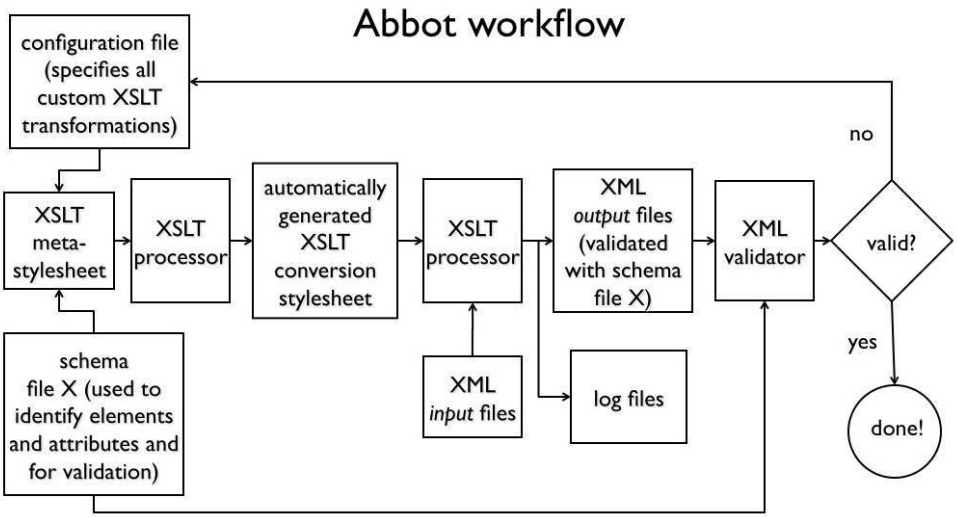

Figure 1: Abbot workflow

4 When the program is launched, a meta-stylesheet reads a schema file for the desired output and a configuration file that details what custom transformations, if any, are needed. The schema contains information about elements and their allowed attributes. For example, given an element <p> in the input, if an element with the same name is specified in the output schema, Abbot will retain the <p> tags and any attributes that are associated with the element in the schema. Abbot assumes that an input element resembles the desired output element, as is often true. But when this assumption isn't true and a user wants to rename elements or perform a complex or conditional mapping of an input element, a custom transformation must be specified in the configuration file as an XSLT template (see fig. 4 below). XML validation reports and Abbot transformation logs help identify the presence of elements and attributes that are not accounted for.

5 All subsequent steps of the Abbot pipeline involve an XSLT processor using a conversion stylesheet to convert one or many input files that are then validated. Output files generate logs of all processed elements and any associated changes. Valid files require only celebration. Invalid files require alteration of the configuration file, and the process is repeated. With Abbot, it is normal to process an input collection several times before all files are valid.

6 A text collection that is valid in structure may still benefit from some additional scrutiny to verify that no words were inadvertently lost or rendered out of sequence—that is, that the conversion was lossless. For MONK and its texts, this scrutiny was undertaken manually and on a selective basis by members of the project team—an approach that worked for a limited project such as this. In the case, though, of the more than 40,000 texts produced by the Text Creation Partnership (a collection more than ten times larger than the MONK corpus), the problem of simultaneously validating markup and verifying textual fidelity becomes clear, and new procedures are needed. While it can be relatively easy to verify losslessness for a small text collection, it becomes progressively more difficult with more texts.

2. Verification at Scale

7 Abbot is a meta-program, in the sense that it is "code that writes code." Abbot observes and modifies its own structure and behavior at runtime, and it performs self-adjustments and dynamically calculates the effects of transformations. Of course, calculation of the results of markup transformations can itself pose technical problems due to scale. Abbot's solution to the problem is inspired by *distant reading*, the sort of reading that one does when there are too many texts to read closely (Moretti 2005). Distant *verification*, if we may call it that, becomes necessary when there are too many text alterations, or too many texts, to verify closely and individually.

8 Since we were unable to find a sufficiently robust XSLT logging framework, we built one from scratch. Several requirements had to be satisfied. First, because Abbot's core transformations are built in XSLT, its desired logging framework had to operate inside XSLT templates. Next, those templates had to be self-identifying, self-describing, and self-differencing. Self-reflective templates begin to fulfill this threefold requirement. Such a template would operate in the following way—rendered below in a first-person conceit.

9 For Abbot, identification looks like this: "I am template d2e1645. I perform a specific functional operation, and I will record my every action and where I perform it." The benefit is that one can account for each operation of a given template. This is a helpful feature when one needs to meet the goal of knowing what changes are derived from what procedures.

10 In Abbot, templates that identify themselves must also *describe* what they do: "I replace markup structure X with structure Y." Here is an example of a given replacement: "Replace `<sup>` with `<hi rend="sup">`." Of course, this example is quite simple, though more complex transformations are possible.

11 Besides identifying and describing what templates do, Abbot's templates perform a *differencing* function. This step uses the Levenshtein edit distance algorithm to calculate the differences between element names, attribute names, and the contents of text nodes (Levenshtein 1966; Bernholz and Pytlik Zillig 2011). Abbot reports all differences, such as: "I changed the `<eebo>` tags to `<TEI>` tags, with a Levenshtein distance of 4 bytes" or "I deleted four children of the current element." Jeni Tennison offers one version of a Levenshtein implementation code in XSLT that served as a partial inspiration for the Abbot log (Tennison 2007).

12 Since completing the MONK project, we have extended Abbot to be able to verify the fidelity of all transformations by measuring the inputs and outputs and calculating and

logging every difference. For each XML node, a log entry is made that records any changes to the node, including the node name, the names of child nodes, the attribute names and values, the text nodes that are children of the current node, and counts of each of the above. Abbot stamps the date and time of every change. Moreover, it records the locations of all changes in the file.

13 These alterations are made as part of the Abbot transformation pipeline and logged in a file that is produced in comma-separated values (CSV) format. While a command-line diff operation could potentially be used to perform the task of comparing XML files to their source texts, Abbot adds this comparison functionality as a first-class operation to the processing pipeline. It is now possible to test and quantify possible outcomes of various conversions. The CSV format makes it a trivial task for a spreadsheet program to view the consequence of a single conversion, or all conversions. Moreover, conversion from CSV to XML (if desired) is trivially easy.

14 Every substantive change to the XML structure or to the text content is recorded. Abbot's measurement of nodal difference is not based on simple string comparison, which would report differences such as those between `<foo n="1" id="a"/>` and `<foo id="a" n="1"/>`. In this example, the order of the attributes is reversed, but the two nodes are otherwise the same and the change is non-substantive. While XML differencing applications exist, they are not sufficient for the present purpose because they are unable to refer to the specific code responsible for a given change. The same pipeline that alters the XML input nodes and writes the output nodes must be able—as Abbot now is—to log all differences.

15 In Abbot, templates are created at runtime based on input that is gathered at runtime. They vary depending on the source texts and on the desired output schema. The richness of the Abbot transformation logs may, because of their length, present problems of scale in their own right. For example, with an input file that contains 20,000 XML elements, Abbot makes a corresponding number of log entries. While the log files are eminently readable in theory, it would be helpful if, in future iterations, the software permitted the user to refine the results in some way. For example, users might want to suppress those entries that record non-substantive alterations.

Template ID	Description	Element name - input	Element name - output	Levenshtein edit distance	Count of descendants - input	Count of descendants - output	Difference	Unique list of descendant element names - input	Unique list of descendant element names - output	Unique list of attribute names - input	Unique list of attribute names - output	Count of attributes - input	Count of attributes - output	Difference	String-length of current text node - input	String-length of current text node - output	String-length difference
d2e1585	process the root element	eebo	TEI	4	228	224	-4	teiHeader filedesc titlestmt title author extent publicationstmt publisher pubplace date idno availability p sourcedesc biblfull notesstmt note encodingdesc projectdesc editorialdecl profiledesc langusage language text front div1 pb hi body head lg l trailer	teiHeader fileDesc titleStmt title author extent publicationStmt publisher pubPlace date availability p sourceDesc biblFull notesStmt note encodingDesc projectDesc editorialDecl profileDesc langUsage language revisionDesc change name text front div pb hi body head lg l trailer	id type	xml:id	2	1	1	0	0	0
d2e1811	add 'change' in revisionDesc	teiHeader	teiHeader	0	43	39	-4	filedesc titlestmt title author extent publicationstmt publisher pubplace date idno availability p sourcedesc bibfull notesstmt note encodingdesc projectdesc editorialdecl profiledesc langusage language	fileDesc titleStmt title author extent publicationStmt publisher pubPlace date availability p sourceDesc biblFull notesStmt note encodingDesc projectDesc editorialDecl profileDesc langUsage language revisionDesc change name			0	0	0	0	0	0
d2e1851	delete attribute	title	title	0	0	0	0			type l2		2	0	2	160	160	0
d2e1603	suppress 'idno' element	idno		4	0	-1	-1			type		1	0	1	9	0	9
d1e2742	examine	availability	availability	0	1	1	0	p	p			0	0	0	0	0	0
d2e1639	suppress 'p' tags in several contexts	p	p	0	0	0	0					0	0	0	202	202	0
d2e1834	add attribute	language	language	0	0	0	0				ident	0	1	-1	3	3	0
d2e1912	convert 'text' element of quoted text to 'floatingText'	text	text	0	183	183	0	front div1 p pb hi body head lg l trailer	front div p pb hi body head lg l trailer	lang	xml:lang	1	1	0	0	0	0

Figure 2: Detail of an Abbot log

16 Figure 2, a detail of a much larger log, illustrates eight nodes as processed and logged by Abbot. Here, eight discrete templates, identified in the left-most column, are responsible for the transformations shown to the right in the corresponding rows. These specific entries show: (1) renaming the root element, (2) adding a change element within `<revisionDesc>`, (3) deleting an attribute, (4) deleting an element, (5) examining a text node for any differences, (6) deleting `<p>` elements in certain conditions, (7) adding an `@ident` attribute to `<language>`, and (8) changing `<text>` to `<floatingText>`. The following example shows a generic reflective XSLT template, somewhat simplified for brevity:

```
<xsl:comment>
    <xsl:value-of select="$templateID"/>
    <xsl:value-of select="$desc"/>
    <xsl:value-of select="name()"/>
    <xsl:value-of select="$thisNodeAfterTransformation/*[name()!
='emptyNode']/name()"/>
    <xsl:copy-of select="d:levenshteintest(string(name()), string
($thisNodeAfterTransformation/*
         [name()!='emptyNode']/name()))"/>
    <xsl:value-of select="count(descendant::*)"/>
    <xsl:value-of select="if (boolean
($thisNodeAfterTransformation)=false) then 0 else
          count($thisNodeAfterTransformation//*[name()!='emptyNode'])-1"/>
    <xsl:value-of select="number(if (boolean
($thisNodeAfterTransformation)=false) then 0 else
         count($thisNodeAfterTransformation//*[name()!='emptyNode'])-1) -
count(descendant::*)"/>
    <xsl:value-of select="distinct-values(descendant::*/name())"/>
    <xsl:value-of select="distinct-values($thisNodeAfterTransformation/
child::*
         [name()!='emptyNode']/descendant::*/name())"/>
    <xsl:value-of select="$listOfAttributes"/>
    <xsl:value-of select="distinct-values($thisNodeAfterTransformation/
child::*
         [name()!='emptyNode']/@*/name())"/>
    <xsl:value-of select="count(@*/name())"/>
    <xsl:value-of select="count($thisNodeAfterTransformation/child::*[name
()!='emptyNode']/@*)"/>
    <xsl:value-of select="number(count(@*/name())) - number(
         count($thisNodeAfterTransformation/child::*[name()!='emptyNode']/
@*))"/>
    <xsl:variable name="allCurrentTextNodesInputFile">
        <xsl:for-each select="current()/text()">
            <xsl:value-of select="."/>
        </xsl:for-each>
    </xsl:variable>
    <xsl:value-of select="string-length($allCurrentTextNodesInputFile)"/>

    <xsl:variable name="allCurrentTextNodesOutputFile">
        <xsl:for-each select="$thisNodeAfterTransformation/child::*
             [name()!='emptyNode']/current()/text()">
            <xsl:value-of select="."/>
        </xsl:for-each>
    </xsl:variable>
    <xsl:value-of select="string-length($allCurrentTextNodesOutputFile)"/>
    <xsl:value-of select="string-length($allCurrentTextNodesInputFile) -
         string-length($allCurrentTextNodesOutputFile)"/>
    <xsl:choose>
        <xsl:when test="$allCurrentTextNodesInputFile =
```

```
$allCurrentTextNodesOutputFile"/>
        <xsl:otherwise>
            <xsl:copy-of select="d:levenshteintest(string
($allCurrentTextNodesInputFile),
                string($allCurrentTextNodesOutputFile))"/>
        </xsl:otherwise>
    </xsl:choose>
    <xsl:value-of select="current-dateTime()"/>
    <xsl:for-each select="ancestor-or-self::*">
        <xsl:variable name="nodeName">
            <xsl:value-of select="name()"/>
        </xsl:variable>
        <xsl:value-of select="name()"/>
        <xsl:for-each select="@*">
            <xsl:text>{@</xsl:text>
            <xsl:value-of select="name()"/>
            <xsl:text>=</xsl:text>
            <xsl:value-of select="."/>
            <xsl:text>}</xsl:text>
        </xsl:for-each>
        <xsl:if test="following-sibling::*[name()=$nodeName]">
            <xsl:text>[</xsl:text>
            <xsl:value-of select="count(preceding-sibling::*[name()=
$nodeName]) + 1"/>
            <xsl:text>]</xsl:text>
        </xsl:if>
        <xsl:text>/</xsl:text>
    </xsl:for-each>
</xsl:comment>
```

17 This example illustrates the XSLT used to create individual log entries. Key features include, for each element in the source markup, the following features:

- Line 2: ID number of the template responsible for the given conversion.
- Line 3: Description of the template's transformation
- Line 4: Element name—input
- Line 5: Element name—output
- Lines 6–7: Levenshtein edit distance between the input and output element names
- Line 8: Count of descendants—input
- Lines 9–10: Count of descendants—output
- Lines 11–12: Difference between the input and output descendant counts
- Line 13: Unique list of descendant element names—input
- Lines 14–15: Unique list of descendant element names—output
- Line 16: Unique list of attribute names—input
- Lines 17-18: Unique list of attribute names—output
- Line 19: Count of attributes—input
- Line 20: Count of attributes—output
- Lines 21–22: Difference between the input and output attribute counts
- Line 28: String-length of current text node—input
- Line 35: String-length of current text node—output

- Lines 36–37: Difference between the input and output string lengths
- Lines 41-42: Levenshtein edit distance between the input and output text nodes
- Line 45: Date and time of the transformation
- Lines 46–64: Path to root node from the current node

3. Sample Use

18 In a simple example, suppose that we are attempting to convert a text collection that contains many instances of the following unusual customization signifying a page break:

```
<break n="1" ref="00000001.tif"/>
```

19 Because <break/> is unspecified in the output schema, and because it contains no text node, Abbot will remove this element. While this may sound a bit reckless, it is the job of the Abbot log to report the fact and consequence of this removal. The report that an element called <break/> has been removed signals to the user that it may be desirable to add a custom routine to the configuration file, such as in the following example:

```
<transformation type="xslt" activate="yes">
   <desc>convert 'break' to 'pb' and its @ref attribute to @facs
attribute </desc>
   <xsl:template match="break | BREAK" priority="1">
      <xsl:element name="pb">
         <xsl:for-each select="@*">
            <xsl:choose>
               <xsl:when test="lower-case(name())='ref'">
                  <xsl:attribute name="facs">
                     <xsl:value-of select="."/>
                  </xsl:attribute>
               </xsl:when>
               <xsl:otherwise>
                  <xsl:copy-of select="."/>
               </xsl:otherwise>
            </xsl:choose>
         </xsl:for-each>
         <xsl:apply-templates/>
      </xsl:element>
   </xsl:template>
</transformation>
```

20 With this custom transformation in place, the output node (both its name and its attributes) is immediately recognizable and valid TEI P5:

```
<pb n="1" facs="00000001.tif"/>
```

21 In this example, the <break/> tag has been converted to <pb/>, @n has been preserved, and @ref has been replaced with @facs. The log entry for the <break/> element, shown here in tabular form, confirms these facts:

TemplateID	d2e1749
Description	convert 'break' to 'pb' and its @ref attribute to @facs attribute
Element name - input	break
Element name - output	pb
Levenshtein edit distance	5
Count of descendants - input	0
Count of descendants - output	0
Difference	0
Unique list of attribute names - input	n ref
Unique list of attribute names - output	n facs
Count of attributes - input	2
Count of attributes - output	2
Difference	0

4. Conclusion and Future Direction

22 Simply put, Abbot's aim is to remove markup differences when aggregation, temporary or not, is desired. Abbot makes it possible, on a large scale of thousands or tens of thousands of documents, to identify, quantify and rectify such problems using the log that records every changed character in a document conversion effort.

23 Generic XSLT templates such as those described here could be used as a basis for a logging library intended to account for changes in XML documents. With support from the Andrew W. Mellon Foundation, Abbot is being rewritten, in XSLT by the author and in Clojure by Stephen Ramsay, with an emphasis on speed and scalability. We anticipate that

soon Abbot will gain an application programming interface and a graphical user interface. The former will help Abbot to work with other tools in complex pipelines, and the latter will improve general usability.

24 It is a goal of the Abbot project to help keep the "I" in TEI. Anna Gold asserts that a "great challenge of data curation is ensuring that data, once preserved, remains meaningful either within the same research area or ideally across areas or even across domains" (2010). The change-logging extension of Abbot, by making the integrity of texts verifiable across transformations, solves an important obstacle to keeping curated data meaningful. When the happy day arrives, perhaps soon, that we have at our disposal the "million(s of) books" that Gregory Crane (2006) writes about, we will curate them with precision and care and caution and a complete accounting of alterations.

BIBLIOGRAPHY

Bernholz, Charles D., and Brian L. Pytlik Zillig. 2011. "Comparing Nearly Identical Treaty Texts: A Note on the *Treaty of Fort Laramie with Sioux, etc., 1851* and Levenshtein's Edit Distance Metric." *Literary and Linguistic Computing* 26(1): 5–16. doi:10.1093/llc/fqq016.

Crane, Gregory. 2006. "What Do You Do with a Million Books?" *D-Lib Magazine* 12(3). doi:10.1045/march2006-crane.

Gold, Anna. 2010. "Data Curation and Libraries: Short-Term Developments, Long-Term Prospects." Library, California Polytechnic State University, San Luis Obispo. April 4. Accessed May 13, 2011. http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1027&context=lib_dean.

Levenshtein, Vladimir I. 1966. "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals." *Soviet Physics Doklady* 10: 707–710.

Moretti, Franco. 2005. *Graphs, Maps, Trees: Abstract Models for a Literary History*. New York: Verso.

Pytlik Zillig, Brian L. 2009. "TEI Analytics: Converting Documents into a TEI Format for Cross-collection Text Analysis." *Literary and Linguistic Computing* 24(2): 187–92. doi:10.1093/llc/fqp005.

Tennison, Jeni. 2007. "Levenshtein distance on the diagonal." *Jeni's Musings* (blog), May 6. http://www.jenitennison.com/blog/node/12.

Unsworth, John. 2011. "Computational Work with Very Large Text Collections: Interoperability, Sustainability, and the TEI." *Journal of the Text Encoding Initiative* 1.

ABSTRACTS

This article describes an XSLT-based logging framework developed for Abbot, a markup conversion and interoperability tool. Abbot logs structural and textual divergence from an XML source. Logging is a useful component of verification when there are too many text alterations, or too many texts, to verify closely and individually. Abbot's conversion and logging

transformations are built inside XSLT templates which are self-identifying, self-describing, and self-differencing. Abbot's templates *identify* themselves and indicate where in a source text they made any changes. Moreover, they *describe* what they do and perform a *differencing* function to calculate divergences between element names, attribute names, and the contents of text nodes.

INDEX

Keywords: interoperability, XSLT, code generation, Abbot

AUTHOR

BRIAN L. PYTLIK ZILLIG

Brian Pytlik Zillig is Professor and Digital Initiatives Librarian at the Center for Digital Research in the Humanities. Brian has been involved in digital humanities for more than a decade, working on numerous projects, including the Journals of the Lewis and Clark Expedition Online and the Walt Whitman Archive, and many others. He has received grants from the National Endowment for the Humanities and the Andrew W. Mellon Foundation. For the Metadata Offer New Knowledge (MONK) Project, he created (with Stephen Ramsay, Martin Mueller) a command-line file conversion application, Abbot, which transforms variant TEI/XML texts into a common interoperable format called TEI-Analytics. As principal investigator for the Mellon-funded Abbot (2.0) Project, he provided project management and XSLT programming experience, as well as the skills and perspective of a professional academic librarian. His professional activities involve programming, prototype development, Scalable Vector Graphics (SVG), data animation, algorithmic XSLT code generation, and XML transformation.