Use of the Extensible Stylesheet Language (XSL) for Medical Data Transformation

Yoon-Ho Seol, MS*, Stephen B. Johnson, PhD* and Justin Starren, MD, PhD*†

Departments of Medical Informatics* and Radiology†,
Columbia University College of Physicians and Surgeons, New York, New York

Recently, the Extensible Markup Language (XML) has received growing attention as a simple but flexible mechanism to represent medical data. As XML-based markups become more common there will be an increasing need to transform data stored in one XML markup into another markup. The Extensible Stylesheet Language (XSL) is a stylesheet language for XML. Development of a new mammography reporting system created a need to convert XML output from the MEDLee natural language processing system into a format suitable for cross-patient reporting. This paper examines the capability of XSL as a rule specification language that supports the medical XML data transformation. A set of nine relevant transformations was identified: Filtering, Substitution, Specification, Aggregation, Merging, Splitting, Transposition, Push-down and Pull-up. XSL-based methods for implementing these transformations are presented. The strengths and limitations of XSL are discussed in the context of XML medical data transformation.

INTRODUCTION

Recently, the Extensible Markup Language (XML)[1] has received growing attention as a simple but flexible mechanism to represent medical data that facilitates data exchange.[2,3,4,5] XML is a simplified version of the standard general markup language (SGML). XML allows the user to define tags for data description while maintaining computational simplicity by restricting many of complexities in SGML. The combination of these merits is being welcomed as a data representation and exchange format in health care environment.

It is critical to remember that XML is not, itself, a health data markup language. XML is a standard[1] that allows users to create new markup languages for specific uses.[2-5] Just as different database schemas are designed to meet different needs,[6,7] different XML-based markups will be optimal for different uses. As XML-based markups become more common there will be an increasing need to transform data stored in one XML markup into another markup.

The Extensible Stylesheet Language (XSL)[8] is a stylesheet language for XML. It is a simplified variant of the document style semantics and specification language (DSSSL), the primary stylesheet language for SGML. Although the most common use of XSL is to transform data from a XML markup into HTML for display by a browser, it is also capable of transforming data among various XML markups.

We have examined the usefulness of XSL for the transformation of healthcare data. A number of XML data transformations have been identified and the XSL implementation of each transformation has been studied. The experimental context is the transformation of raw Natural Language Processing (NLP) data into a form suitable for cross patient reporting.

CONTEXT

As part of a new mammography reporting system,[9] there was a need to convert XML output from the MEDLee natural language processing system[10] into a format suitable for cross-patient reporting. The XML output of MEDLee is deeply nested and reflects the source document syntax. In its raw form, MEDLee output is difficult to store in a conventional relational database. While a generic data modeling approach could have been used,[6] querying such a repository requires a non-trivial effort to insulate complexity of data organization from the user.[7,11] Instead, we chose to translate the MEDLee output in a conventional relational schema before storage in the database. To separate the functions of data transformation from data storage, a two step process was chosen. First, data would be converted from the MEDLee XML format into a new XML format that represented the relational schema. Second, the data in the new XML format would be stored in the database. In this way, all data transformations could be performed in the XML domain utilizing XSL. This paper examines capabilities of XSL in the context of an automated mammography data transformation.

Figure 1 shows a simplified example of XML output generated by MEDLee. For the sake of brevity, tags

1091—8280/99/$5.00

have been shortened to single characters and some tags are omitted. Data transformations in XML are often easier to visualize when the document is represented as a graph. The document in Figure 1 can also be represented by the tree data structure in Figure 2.

Input:
"suspicious microcalcification in the left breast for which biopsy is recommended."

Output:

```
<S form = "xml">
    <P  v = "suspicious microcalcification">
        <B  v = "breast">
            <R  v = "left"/>
        </B>
        <Pm  v ="mode1"/>
    </P>
    <Rx  v = "biopsy">
</S>
```

S: structured P: problem
B: body location R: region
Pm: parse mode Rx: recommendation

Figure 1. Simplified example of MEDLee XML output.

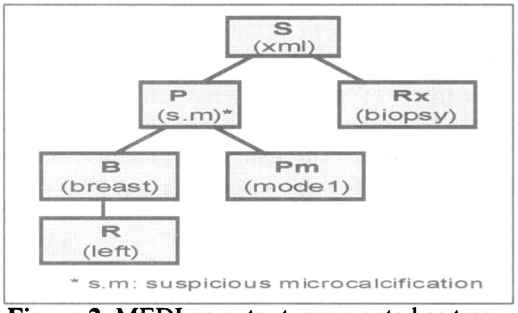

Figure 2. MEDLee output represented as tree diagram.

Throughout the paper, the XSL transformation examples will be based on the tree in Figure 2 if not specified otherwise. Space does not allow a full discussion of the details of the XSL syntax but the general flow of a XSL processing will be discussed.

DATA TRANSFORMATIONS

We have identified a set of nine transformations for use in a mammography data transformation. These are supported by XSL to varying degrees. Figure 3 summarizes these transformations. Examples of each transformation will be described in more detail, along with XSL implementation issues.

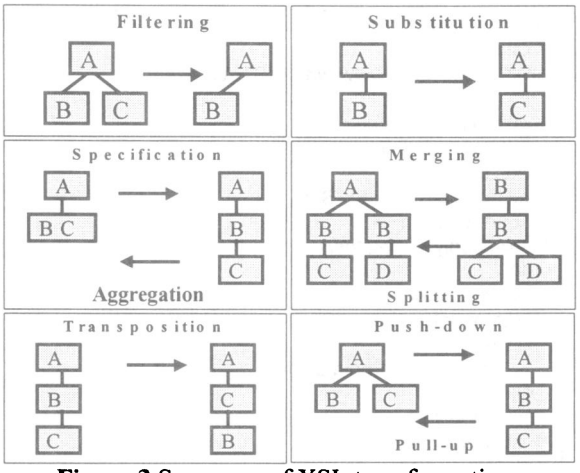

Figure 3 Summary of XSL transformations

Filtering

Filtering is useful when the MEDLee output contains data that does not need to capture in the repository. For example, the parse mode <Pm> is system information for MEDLee that will not be stored in our database. XSL supports this transformation by allowing us to extract only the data of our interest. Figure 4. is a complete XSL stylesheet. Each template rule in the stylesheet corresponds to a node in the source tree and performs a specified action in the rule. In our example, the template rule prints the data in the node. The lack of a <Pm> rule means that parse mode information is ignored.

Substitution

Substitution is necessary to map multiple concepts in source documents into a single canonical concept. For example, multiple clinical problems detected in mammography could be mapped to a single term that represent those concepts collectively. Figure 5 shows a template rule that maps two terms, 'suspicious microcalcification' and 'calcification,' into a canonical term, 'calcification.' Although massive XSL documents could, in theory, be used for large-scale vocabulary mapping, this would create a maintenance nightmare. A more likely scenario is that XSL would be reserved for relatively simple translations and table-driven systems would be used for large-scale applications.

Such mapping may result in some information loss due to different levels of granularity in multiple similar concepts needs to be mapped to a canonical concept. In this example, we lost information such as a grade of the problem 'suspicious' as a result of the canonical mapping. This can be addressed through specification.

```
<?xml version="1.0" ?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">
 <xsl:apply-templates/>
 </xsl:template>
<xsl:template match="S">
  <S v="{@form}">
   <xsl:apply-templates/>
  </S>
</xsl:template>
<xsl:template match="P">
  <P v="{@v}">
      <xsl:apply-templates />
  </P>
</xsl:template>
<xsl:template match="B">
  <B v="{@v}">
      <xsl:apply-templates />
  </B>
</xsl:template>
<xsl:template match="R">
  <R v="{@v}">
    <xsl:apply-templates />
  </R>
</xsl:template>
<xsl:template match="Rx">
 <Rx v="{@Rx}">
    <xsl:apply-templates />
 </Rx>
</xsl:template>
</xsl:stylesheet>
```

```
Output:

<S v="xml">
   <P v="suspicious
        microcalcification">
      <B v="breast"
         <R v="left"/>
      </B>
   </P>
   <Rx v="biopsy"/>
</S>
```

Figure 4 XML Stylesheet showing filtering.

```
<xsl:template match="P[@v='suspicious
          microcalcification' or @v='calcification']">
   <P v="calcification">
      <xsl:apply-templates />
   </P>
</xsl:template>
```

```
Output:
<P v="calcification">
</P>
```

Figure 5 Substitution.

```
<xsl:template match="P[@v='suspicious
     microcalcification']">
  <P v="calcification">
     <G v="suspicious"/>
     <xsl:apply-templates />
  </P>
</xsl:template>
```

```
Output:
<P v="calcification">
   <G v="suspicious" />
</P>
```

Figure 6 Specification.

```
<xsl:template match="B/R">
   <B v="{@v} {../@v}>
      <xsl:apply-templates />
   </B>
</xsl:template>
```

```
Output:

<B v="left breast">
</B>
```

Figure 7 Aggregation

Specification vs. Aggregation

To retain the information that was in the of data substitution example, we need to provide a template rule that captures the severity or grade information of the problem. This type of transformation we have termed data specification. In Figure 6, not only the problem is substituted with a term 'calcification' but also the problem is specified with the tag <G> in order to preserve the grading information of the calcification.

In other cases, deeply nested information needs to be aggregated in to a single tag. MEDLee output for location is typically nested. In the example, body location <B> is modified by region <R>. It is possible for regions to be nested several levels deep. This involves aggregating a set of nodes in a linear hierarchy and collecting contents of those nodes. This is similar to the process of concept clustering.[12] In the example in Figure 7, the body location <B> and its region <R> are aggregated in order to produce a single location 'left breast' in the output XML document.

Merging vs. Splitting

Data merging is needed to remove duplicate data structures in the MEDLee output. Because MEDLee output reflects the source document, two sentences that refer to the same structure can appear as two parallel branches of the tree. The sentences "The breast is dense" and "The breast is enlarged" could be represented in a form shown in Figure 8. The 'breast' is duplicated and needs to be merged into a single instance. The template rules that accomplish this transformation are shown in Figure 8.

Data splitting is the reverse of the data merging. A single instance is split, with different children staying with each new parent. The template rules in Figure 9 perform the data splitting on the example in Figure 8.

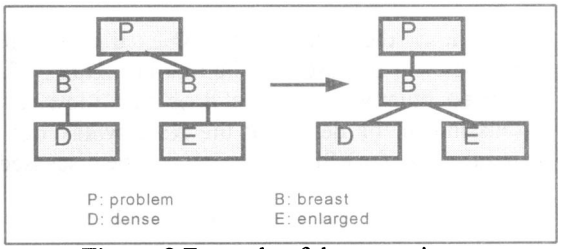

Figure 8 Example of data merging.

Transposition

Another useful transformation is the one which exchanges two adjacent (direct parent and child) nodes in a source tree. For example, MEDLee output is structured so that body locations <B> are children of problems <P>, but for certain uses, it may be preferable to have the problem <P> as a child of location <B>. This could facilitate the graphical

display of diseases or findings anatomically. An example of this is shown in Figure 11.

```
<xsl:template match="P[B]">
 <!-- check if P has child B -->
   <P><B>
     <xsl:apply_templates select="B/*" />
   </B></P>
</xsl:template>
<xsl:template match="E">
   <E> <xsl:apply_templates/> </E>
</xsl:template>
<xsl:template match="C">
   <C> <xsl:apply_templates/></C>
</xsl:template>
```

Figure 9. Merging.

```
<xsl:template match="B/D">
  <B><D>
     <xsl:apply_templates/>
   </D></B>
</xsl:template>
<xsl:template match="B/C">
   <B><C>
     <xsl:apply_templates/>
   </C></B>
</xsl:template>
```

Figure 10. Splitting

```
<xsl:template match="P">
  <!--skip-->
  <xsl:apply_templates/>
</xsl:template>
<xsl:template match="P/B">
   <B v="{@v}">
      <P v="{../@v}">
       <xsl:apply_templates/>
      </P>
   </B>
</xsl:template>
```

Figure 11. Transposition.

Push-down vs. Pull-up

The push-down and pull-up transformations require special handling. Push-down transformation is a process to take one child of the tree and make it a child of another child as in Figure 3. The pull-up is the reverse transformation that allows a child to move up the hierarchy. If we know the structure of the entire source tree in advance, the push-down transformation can be viewed as a combination of the filtering and the specification transformations. The only difference is that the push-down operation queries a sibling for the value to use for the specification operation. The pull-up transformation would involve breaking the parent and child relationship as in Figure 12.

Unfortunately, the optional nature of many XML tags means that the exact structure of most XML documents is not known in advance. For instance, if the structure of the node B in Figure 3 is unknown, we cannot specify <B></B> in the pull-up template rule in Figure 12. A more complex approach for both push-down and pull-up transformations is required. Working Draft 3 of XSL added an xsl:copy-of element which allows for the movement of entire sub-trees from one location to another. Unfortunately, this does not allow for any processing of the moved nodes. If processing is required, the xsl:param-variable element can be used to pass a flag to the template for node <C> (Figure 13). Testing for this flag provides a way to insure that node <C> is only processed once.

```
<xsl:template match="B">
  <B></B>
  <xsl:apply-templates/>
</xsl:template>
</xsl:template match="C">
   <C>
     <xsl:apply-templates/>
   </C>
</xsl:template>
```

Figure 12 Pull-up transformation when document structure is known in advance.

```
<xsl:template match="B">
   <xsl:apply-templates select="../C">
   <xsl:param name="flag"> 1 </xsl:param>
   </xsl:apply-templates>
   <xsl:apply-templates/>
</xsl:template>
<xsl:template match="C">
  <xsl:param-variable name="flag"> 0
  <xsl:param-variable>
  <xsl:if  test="number($flag) = 1">
    <C>
      <xsl:apply-templates/>
    </C>
   </xsl:if>
</xsl:template>
```

Figure 13. Use of parameters and conditional processing to perform Push-down operation when document structure is variable.

DISCUSSION

Medical data transformation is a complex process. We have presented a set of data transformations supported by XSL. The version of XSL evaluated represents a "working draft." Therefore, it is likely that the syntax of, at least, some of these transformations may change.

A characteristic that will likely not change is that XSL lacks some features usually supported by the programming languages. This relates to the fact that XSL favors computational simplicity over extensive functionality as a stylesheet language. XSL has recently added the ability to call external functions, which may circumvent some of these limitations.

An example of a limitation is the handling of recursively nested structures. In specifying locations in MEDLee, it is possible for a region <R> to contain another region <R>. An example would be "medial portion of the anterior segment of the right middle lobe of the right lung." Nesting a tag within an identical tag can create problems during data aggregation. To handle this situation, two template rules would be needed: one for the (nested) child region and the other one is for the (non-nested) parent region. These rules must be structured so that the nested rule has precedence over the non-nested rule, otherwise the non-nested rule will fire repeatedly at each level of nesting. This is handled in XSL with rule priorities. Unfortunately, even with priorities number of rules increases linearly as the degree of the nesting increases. For instance, if region <R> can be nested up to 5 levels, then we need different template rules to handle each possible depth of nesting. There is no way to specify in a single rule recursive processing of nested tags.

The independent nature of XSL template rules poses other limitations. It is easy to envision cases where there would be the need for the value of a tag in one portion of the tree to govern the processing of another portion of the tree. Alternately, the values obtained from the data aggregation may need to be manipulated and transformed into an another value in a different portion of the tree. Although values in one portion of the tree can be explicitly queried from another portion of the tree, there is currently no simple way to set the value of a global variable from within a deeply nested template.

CONCLUSION

Our experience with XSL suggests that it is a promising technology for medical data transformation. Despite its limitations, XSL supported a variety of data transformations. Our current efforts involve the implementation of a prototype system utilizing these transformations. An open area of research is whether XSL is robust enough to survive in a high-volume production environment.

Acknowledgement

The authors thank Dr. Carol Friedman for access to the MEDLee NLP data. This work was funded by a grant from the New York State Science and Technology Foundation.

References

1. XML specification 1.0 from W3 consortium. URL:http://www.w3.org/TR/1998/REC-xml-19980210
2. Chueh HC, Raila WF, Berkowicz DA. An XML Portable Chart Format. Proc. AMIA Annu Fall Symp. 1998:730-734.
3. Dubey AK, Chueh H. Using the Extensible Markup Language (XML) in Automated Clinical Practice Guidelines. Proc. AMIA Annu Fall Symp. 1998:735-739.
4. Dudeck J. Aspects of implementing and harmonizing healthcare communication standards. Int Journal of Med Inform 48:1998:163-171.
5. Dolin RH, Rishel W, Biron PV. SGML and XML as Interchange Formats for HL7 Messages. Proc. AMIA Annu Fall Symp. 1998:720-724.
6. Friedman C, Hripcsak G, Johnson SB, CiminoJJ, Clayton PD. A generalized relational schema for an integrated clinical patient database. Proc. 14th Annu Symp Comp Appl Med Care. 1990:335-339.
7. Johnson SB, Hripcsak G, Chen J, Clayton P. Accessing The Columbia Clinical Repository. Proc. AMIA Annu Fall Symp. 1994:281-285.
8. XSL working draft from W3 consortium. URL:http://www.w3.org/TR/WD-xslt
9. J. Starren, C. Friedman, and S. B. Johnson. The Columbia Integrated Speech Interpretation System (CISIS). Proceedings - the Annual Symposium on Computer Applications in Medical Care:985-985, 1995.
10 C. Friedman, G. Hripcsak, W. DuMouchel, S. B. Johnson, and P. D. Clayton. Natural Language Processing in an Operational Clinical Information System. Natural Language Engineering 1 (1):83-108, 1995.
11. Nadkarni PM, Brandt C. Data Extraction and Ad Hoc Query of an Entity-Attribute Value Database. J Am Med Inform Assoc. Vol.5 (6) 1998:511-527.
12. Fisher D. Approaches to conceptual clustering. Proc. Int Joint Conf on AI. 1985:691-697.