Applying XML/XSLT Technology: A Case Study[1]

Krishnaprasad Thirunarayan
Department of Computer Science and Engineering, Wright State University
3640, Col. Glenn Hwy, Dayton, Ohio-45435, USA.
Email: tkprasad@cs.wright.edu
URL: http://www.cs.wright.edu/~tkprasad
Tel/Fax: (937)-775-5109 (5133)

Abstract

Typically, paper-based semi-structured documents are used to communicate technical requirements in the industry. One approach to improving the efficiency and flexibility of searching and manipulating these documents is to obtain them in electronic form, and then make explicit the technical content in a form amenable to pattern matching and symbol manipulation. This paper explores the role of XML Technology for these purposes: the use of XML language to make explicit the content of a document in a standard fashion, and the use of XSLT to manipulate the content and mediate between applications that need this content. In order to recognize and appreciate the practical difficulties, a problem in the realm of Content Extraction from Materials and Processing Specification is studied and solved using the available tools supporting the XML Technology. The Case Study provides a realistic assessment of the benefits and the limitations of XML Technology for document handling from the perspective of a developer. It also illustrates non-trivial uses of XSLT constructs.

Keywords: Frameworks for IT; B2B Information Systems, Mining, and Applications

1. INTRODUCTION

Paper-based semi-structured documents written in Natural Language (such as English) are commonly and extensively used to communicate technical requirements in the industry --- among businesses (B2B) and between businesses and the customers (B2C). One approach to improving the efficiency and flexibility of searching and transforming these documents is to obtain them in electronic form, and then recognize and make explicit the technical content in a form amenable to pattern matching and symbol manipulation. This paper explores the role of XML Technology for these purposes: the use of XML language to make explicit the content of a document in a standard form, and the use of XSLT to manipulate the content and mediate between applications that need this content. In order to recognize and appreciate the practical difficulties, a simple problem in the realm of Content Extraction from Materials and Processing Specification is studied and solved using the available tools supporting the XML Technology. The Case Study provides a realistic assessment of the benefits and the limitations of XML Technology for document handling from the perspective of a developer. It also provides illustrative examples of the use of XSLT constructs that are useful to a conventional programmer and that augments the ones in the published literature, which seem to operate on structured information [5,6,7,8].

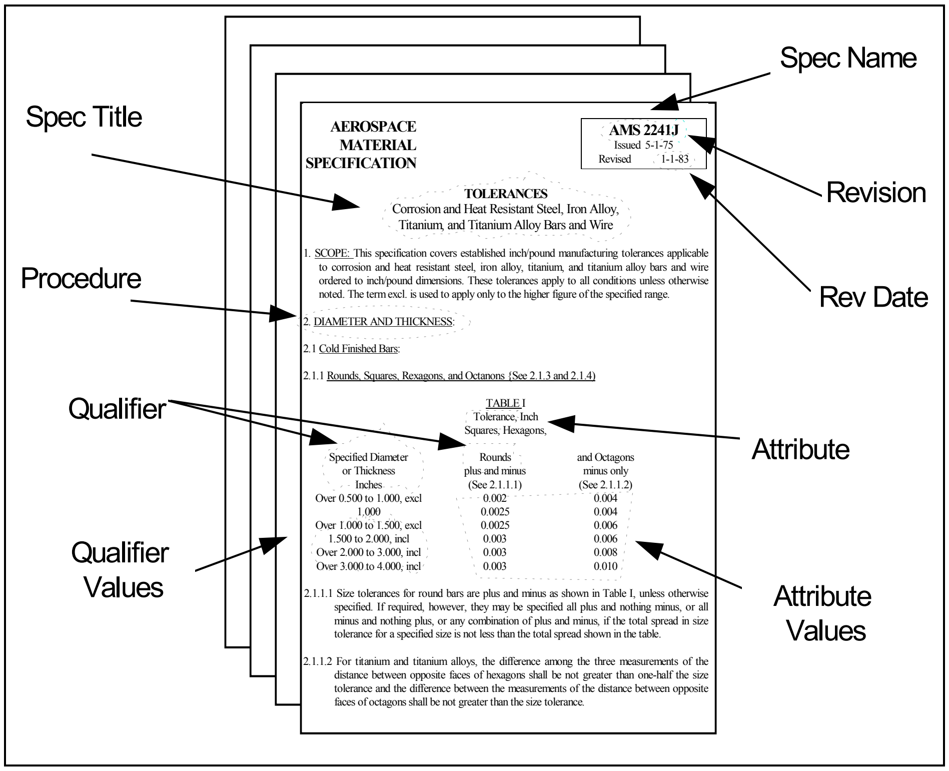
[1] This work was supported in part by NSF SBIR Grant# DMI-0078525 (1999-2002). It does not necessarily reflect the opinions of NSF.

Section 1.1 introduces Content Extraction, and Section 1.2 recapitulates the evolution of XML Technology providing pointers to Literate Programming that indirectly served to motivate our approach. Section 2 explains the application of XML Technology to Content Extraction, while Section 3 discusses the details using an illustrative example. Section 4 summarizes our conclusions.

1.1 CONTENT EXTRACTION

Material and process specifications are used extensively in organizations involved in complex manufacturing in industries such as aerospace, automotive, and materials. These specs present quantitative information about characteristics such as the material properties, the testing parameters, etc which need to be operated upon as intended by the meaning embedded in the text. Cohesia Corporation defined the Specification Definition Representation (SDR) as an ontology to articulate the semantic view of the components that comprise a spec, and capture the user's interpretation as shown in Figure 1 [1].SDR introduced constructs such as *Procedures* to indicate boundaries for standards requirements such as chemical composition, heat treatment, tensile test, etc. Procedures are composed of elemental *Characteristics* that describe the requirements that are essential for performing the associated process (e.g., minimum temperature, yield strength, percentage composition, etc).

Figure 1. Semantic Markup of a Spec with SDR ontology



AEROSPACE MATERIAL SPECIFICATION

AMS 2241J
Issued 5-1-75
Revised 1-1-83

TOLERANCES
Corrosion and Heat Resistant Steel, Iron Alloy, Titanium, and Titanium Alloy Bars and Wire

1. SCOPE: This specification covers established inch/pound manufacturing tolerances applicable to corrosion and heat resistant steel, iron alloy, titanium, and titanium alloy bars and wire ordered to inch/pound dimensions. These tolerances apply to all conditions unless otherwise noted. The term excl. is used to apply only to the higher figure of the specified range.

2. DIAMETER AND THICKNESS:

2.1 Cold Finished Bars:

2.1.1 Rounds, Squares, Rexagons, and Octanons {See 2.1.3 and 2.1.4)

TABLE I
Tolerance, Inch
Squares, Hexagons,

Specified Diameter or Thickness Inches	Rounds plus and minus (See 2.1.1.1)	and Octagons minus only (See 2.1.1.2)
Over 0.500 to 1.000, excl	0.002	0.004
1.000	0.0025	0.004
Over 1.000 to 1.500, excl	0.0025	0.006
1.500 to 2.000, incl	0.003	0.006
Over 2.000 to 3.000, incl	0.003	0.008
Over 3.000 to 4.000, incl	0.003	0.010

2.1.1.1 Size tolerances for round bars are plus and minus as shown in Table I, unless otherwise specified. If required, however, they may be specified all plus and nothing minus, or all minus and nothing plus, or any combination of plus and minus, if the total spread in size tolerance for a specified size is not less than the total spread shown in the table.

2.1.1.2 For titanium and titanium alloys, the difference among the three measurements of the distance between opposite faces of hexagons shall be not greater than one-half the size tolerance and the difference between the measurements of the distance between opposite faces of octagons shall be not greater than the size tolerance.

The SDR technology has been incorporated into a commercial software system called MASS® (Management & Application of Specifications & Standards) which is integrated with Enterprise Resource Planning (ERP) packages from SAP, Oracle, etc. Cohesia's MASS product processes multiple specs expressed in SDR to produce a combined Master Spec. The conversion into SDR form is done manually by reviewing and interpreting the requirements given in the paper-based spec. SDR is a simple low-level tree language that is efficient for symbolic manipulation, but is very removed from the spec text. In order to raise the level of abstraction *Specification Definition Language* (SDL) was designed and implemented. The *Specification Studio* is an Integrated Development Environment (IDE) for creating and editing specifications in SDL, and compiling them into SDR. It also provides a convenient interface to search and use the *Domain Library of controlled terminology*.

Content extraction involves recognition of phrases in spec that are associated with requirements, and subsequent synthesis of SDL fragments. Conceptually, this can be carried out in two steps: (a) automatically recognize domain library terms present in spec and mark them up appropriately, and (b) manually organize this information to be able to generate SDL from it. Semi-automatic approach attempts to improve quality and efficiency of extractions by minimizing transcription errors, and by automating some of the routine mechanical tasks.

1.2 LITERATE PROGRAMMING AND XML TECHONOLOGY

Every programming language provides syntax to embed documentation in the code. Typically, a comment is either a single-line, or a delimited piece of contiguous text. For instance, languages such as Ada, UNIX Shells, etc provide only single-line comments, while languages such as Pascal, C, etc provide bracketing constructs to delimit comments. Languages such as C++, Java, etc support both forms. In programming language Orwell, there is an inversion of sorts in the way the code and the comments are expressed in a program. Specifically, an Orwell program is a piece of text documenting the program design interspersed with cleanly delimited code, which yields the executable instructions. This approach has also been popularized by Donald E. Knuth under the "Literate Programming" banner. To quote Donald E. Knuth [3]:

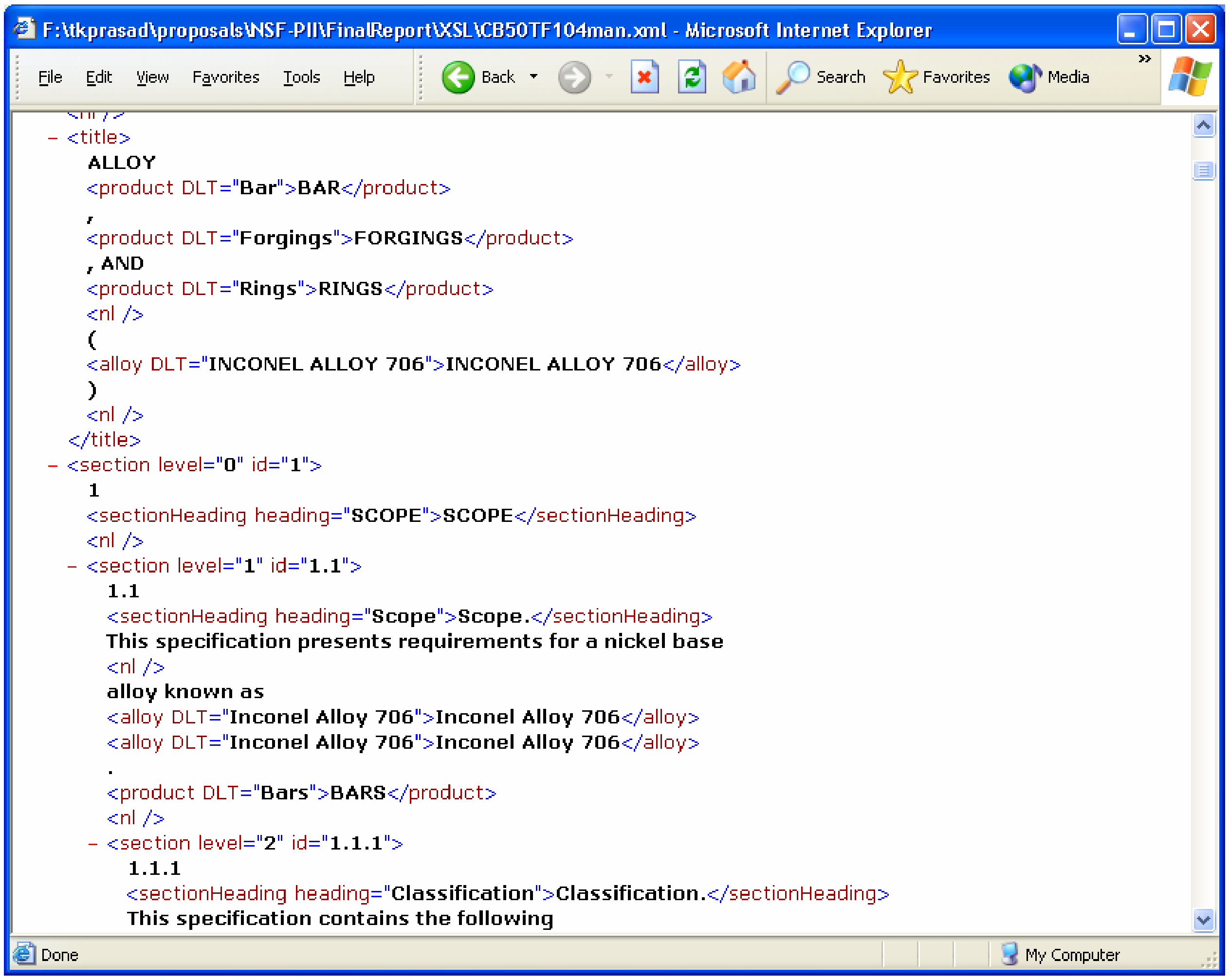
> *Literate programming* is a methodology that combines a programming language with a documentation language, thereby making programs more robust, more portable, more easily maintained, and arguably more fun to write than programs that are written only in a high-level language. The main idea is to treat a program as a piece of literature, addressed to human beings rather than to a computer. The program is also viewed as a hypertext document, rather like the World Wide Web. (Indeed, I used the word WEB for this purpose long before CERN grabbed it!)

A "literate" programmer composes a document containing both the source code and the explanatory text intertwined. The code is extracted and compiled for machine consumption. The text is extracted, formatted, and printed for human consumption. For instance, our VHDL Parser/Pretty-printer System containing both the source code and the design document has been distributed in two forms: (1) a TeX-document with embedded Prolog code, filtered using a C utility, and (2) an HTML-document with embedded Prolog code, filtered using a Java utility [4].

The automatic processing of text documents poses a complimentary problem due to the absence of a rigid syntax and semantics. The manipulation of text documents electronically requires their logical structure, content, and intended meaning to be made "sufficiently" explicit. SGML (Standard Generalized Markup Language) is a standard meta-language for marked-up electronic text. It enables definition of a language of tags using DTD (Document Type Definitions), similarly to context-free grammars that define programming language syntax. SGML effort was very ambitious which resulted in a rich meta-language that was difficult to implement in all its generality.

HTML (HyperText Markup Language) is a non-proprietary format developed as a specific application of SGML. In particular, it is a specific set of tags understood by all Web Browsers. Originally, HTML was meant to describe only the structure of a document, leaving out the presentation details to the Web Browsers. (This is in contrast with the proprietary formats such as Postscript (ps), Portable Document Format (pdf), Rich Text Format (rtf), etc that inextricably mingle content and presentation details.) However, in spite of its purist goals, HTML evolved into a markup language with presentation tags, to incorporate users preferences.

Figure 2. An XML document


F:\tkprasad\proposals\NSF-PII\FinalReport\XSL\CB50TF104man.xml - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Media
Done
My Computer

```
- <title>
    ALLOY
    <product DLT="Bar">BAR</product>
    ,
    <product DLT="Forgings">FORGINGS</product>
    , AND
    <product DLT="Rings">RINGS</product>
    <nl />
    (
    <alloy DLT="INCONEL ALLOY 706">INCONEL ALLOY 706</alloy>
    )
    <nl />
  </title>
- <section level="0" id="1">
    1
    <sectionHeading heading="SCOPE">SCOPE</sectionHeading>
    <nl />
  - <section level="1" id="1.1">
      1.1
      <sectionHeading heading="Scope">Scope.</sectionHeading>
      This specification presents requirements for a nickel base
      <nl />
      alloy known as
      <alloy DLT="Inconel Alloy 706">Inconel Alloy 706</alloy>
      <alloy DLT="Inconel Alloy 706">Inconel Alloy 706</alloy>
      .
      <product DLT="Bars">BARS</product>
      <nl />
    - <section level="2" id="1.1.1">
        1.1.1
        <sectionHeading heading="Classification">Classification.</sectionHeading>
        This specification contains the following
```

XML (eXtensible Markup Language) is a meta-language for designing customized markup languages for different types of documents. XML can be viewed as a restricted form of SGML that is both convenient to use and implement. XML can be used for formalizing HTML.

Conceptually, an XML document consists of annotated text as shown in Figure 2. The text fragments are recursively delimited using matched *tags*. The tags are used to make explicit the type and the properties of the enclosed text using *attribute- value* pairs. In general, the properties can be syntactic / display oriented (such as COLOR, FONT, TYPE-FACE, etc) or semantic (such as MEANING, TYPE, NORMAL-FORM, etc). The XML tags can also simplify parsing.

To promote flexible manipulation and convenient rendering of XML documents, XSL (eXtensible Stylesheet Language) was developed. It consists of two parts: XSLT for transforming XML documents and XSLFO for formatting XML documents [5,6,7,8]. In the following sections, we study the role and the application of XML technology for transforming spec content and discuss its pros and cons. The examples will also illustrate the practical use of XSLT constructs.

2. MARRYING CONTENT EXTRACTION AND XML TECHNOLOGY

A spec can be viewed as a partially structured text that is organized into sections, paragraphs, tables, etc [9]. Related specs from an organization typically use similar technical vocabulary and format. However, in the absence of a rigid syntax and formal semantics, the automatic synthesis of SDR/SDL extractions is not feasible, in general.

Interestingly, XML Technology can provide assistance in the extraction task as follows:

- Most of the information present in extractions can be shown to originate from the spec (their *raison d'etre*). So, one can attempt to annotate spec text using tags that capture the structure and content using techniques developed for Information Extraction [9,10,11]. The resulting XML document, that has embedded ontological tags in spec text, can also facilitate verification. This approach is a stepping stone towards building a domain-specific *Semantic Web* envisioned by Berners-Lee [13].
- XML can be used to describe the spec format, which can differ from organization to organization (e.g., GEAE, ASTM, SAE, etc), or even within specs from the same organization (GEAE B-family specs, P-family specs, etc). Here XML tags simplify parsing of spec format information using available XML APIs and tools.
- A suitably annotated spec can serve as the XML Master Spec that can be transformed or rendered as desired using the XSL Technology. For instance, the Master can be used to regenerate the original spec, or converted into HTML and displayed as a highlighted spec to make explicit phrases that contain requirements (to aid proofing), or transformed into SDR/SDL form of differing granularity, merely by applying suitable stylesheets using tools such as Apache's Xalan processor.
- XML can also be used to provide a standard format for information exchange, that is, for performing data I/O among various applications. Here XSLT stylesheets bridge potential incompatibilities.

In the context of semi-automatic approach to extraction, productivity gains can be obtained by automating routine chores, while leaving out complex parts in an intelligible fashion for completion by an human extractor. This approach can be realized, if one can maintain a link between fragments of a spec and its translation. XSLT stylesheets can serve to codify such links. Thus, XML Master with semantic/ontological tags focuses on the content, while XSLT stylesheets transform XML documents, and XSL-FO stylesheets supply presentation details.

3. CASE STUDY DETAILS

A spec can be obtained in electronic form as a text file either by saving the available MSWORD-file appropriately or by scanning and OCRing a paper-based hard copy. The text can then be preprocessed and tagged to obtain the XML Master using a string pattern matcher. For concreteness, the lexical analyzer introduces tags delimiting `section, section heading, title, product type, spec class, alloy,` etc.[2] In addition, it also marks the location of a newline, in order to be able to recover the original text with its look preserved. Figure 2 depicts the tagged fragment of GEAE Spec# B50TF104. As seen here, certain tag *elements* have associated *attribute-value* pairs. The tag pair `<product> …</product>` containing the element "`product`" has an associated attribute called "`DLT`" (for Domain Library Term) whose value captures the meaning of the enclosed text by mapping the spec word/phrase to a standardized controlled term in the domain library (a dictionary containing material and processing vocabulary). In the figure, the spec word and the domain library term are the same, but in general the spec phrase may look different from its normalized form in the domain library due to the presence of acronyms, aliases, suffixes, superfluous words, shared words, etc as discussed in [12].

In what follows, we discuss the transformation of the XML Master spec into different forms using XSLT. The emphasis is on illustrating the use of XSLT constructs and primitives in a non-contrived context, rather than automating extraction. These examples underscore the role of XSLT as a declarative language and serve to contrast algorithms coded in mainstream programming languages such as C++ and Java.

3.1 REGENERATING THE ORIGINAL

Conceptually, the original spec can be retrieved from the XML Master Spec by dropping all tags. From the perspective of manipulation, an XML document can be viewed as a rooted labeled tree containing nodes for *processing instructions, elements, text, comments, attributes, namespaces,* etc as shown in Figure 3 [5]. The XML Master Spec contains a root-element called `document`, other elements such as `section, newline, title, product, alloy,` etc, attributes such as `ID, DLT,` etc, and text nodes containing fragments of spec. An XSLT stylesheet is a set of declarative template rules; each rule contains a pattern to select a tree-node and a template to be filled in and emitted for the matching node. To drop all tags, a "*" can be used to match a top-level node, use default template, and then have this rule applied recursively to all its descendents in the tree. The text nodes and the empty nl-tag (<nl/>) nodes need special treatment for regenerating the original. By default, the contents of a text node are emitted verbatim, which is appropriate here. However, a rule for substituting each `<nl/>` by a newline character[3] needs to be explicitly included as shown in Figure 4.

[2] The motivation for generating such coarse granularity SDR/SDL extractions (called Method C extractions) is to enable automatic processing by MASS to determine applicable fragments of a spec for a sales order, whose details are outside the scope of this paper.

[3] In fact, it turned out to be a non-trivial task to discover the XML code for performing this rather trivial translation. Internet search provided both the answer and the difficulty experienced by programmers due to paradigm shift.

Figure 3. An abstract tree view of an XML document

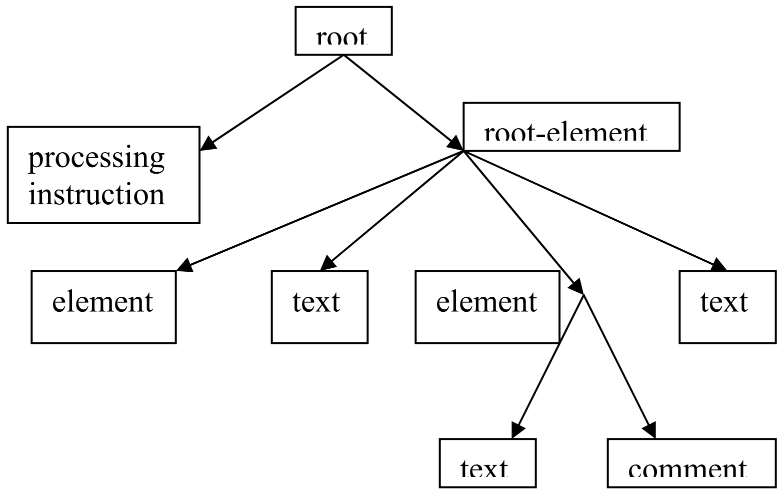


Figure 4. A simple XSLT stylesheet for recovering original

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text" omit-xml-declaration="yes" encoding="ascii"/>
<xsl:template match="*">
<xsl:apply-templates/>
</xsl:template>
<xsl:template match="nl">
<xsl:text>
</xsl:text>
</xsl:template>
</xsl:stylesheet>
```

3.2 GENERATING SPEC IN SDL FORM FROM XML DOCUMENT

Another practically important transformation of the XML Master Spec is obtaining Method C Extraction in SDL. In order to illustrate the use and the power of XSLT constructs that will be beneficial to readers not interested in extraction per se, a number of transformations are explained informally and then expressed formally in XSLT to underscore its role as an "executable documentation".

A spec is identified by its `spec number`, `title`, `organization`, `type`, `revision`, `issue date`, etc. Its scope is indicated in the Applicable Products and Materials (`APM`) portion by enumerating all the products and alloys the spec deals with. This information is collected from the `DLT` attribute bindings for the `Products` identified using the Domain Library in the spec `Title` and the section named `Scope`. The recognition of this information and its rendition in SDL is expressed using the template rule in Figure 5 and the result of its application in Figure 6.

Figure 5. Header Template Rule

```
<xsl:template match="document">
document [<xsl:value-of select="specNumber"/>] {
 title = "<xsl:value-of select="title"/>";
  org  = "GE Aircraft Engines";
  type = "specification";
 define APM {
   <xsl:for-each select="title/product">
    [Products] is "<xsl:value-of select="@DLT"/>";
   </xsl:for-each>
    <xsl:for-each select=
      "//section[contains(ancestor::section/sectionHeading/@heading,'SCOPE')]">
      <xsl:for-each select="product[@DLT != /title/product/@DLT]">
        [Products] is "<xsl:value-of select="@DLT"/>";
      </xsl:for-each>
    </xsl:for-each>
    <xsl:for-each select="alloy | title/alloy">
      [Alloy] is "<xsl:value-of select="@DLT"/>";
    </xsl:for-each>
    <xsl:for-each select=
    "//section[ancestor::section[contains(sectionHeading/@heading,'SCOPE')]]">
      <xsl:if test="@DLT != preceding::alloy/@DLT">
         [Alloy] is "<xsl:value-of select="@DLT"/>";
      </xsl:if>
    </xsl:for-each>
 }
 using APM;
 revision [<xsl:value-of select="issueNumber"/>] {
   <xsl:apply-templates select="section"/>
}
</xsl:template>
```

The construct "`title/product`" stands for the contents within `product`-tags included in `title`-tags, and "`@DLT`" means the value of the attribute `DLT`. The predicate "`//section[contains(ancestor::section/sectionHeading/@heading,'SCOPE')]`" is true of any nested section contained within a section named `SCOPE`. The predicates "`[@DLT != /title/product/@DLT]`" and "`[@DLT != preceding::alloy/@DLT]`" eliminate duplicates. Observe that, XSLT enables one to think of XML document and its transformation in terms of its tree structure. In contrast, a conventional program needs low-level control structures to orchestrate rule application and pattern matching. Furthermore, the output template is embedded in print-statements, rather than made clearly explicit. The SDL translation can be generated by applying the XSLT stylesheet to the XML document using Apache's Xalan processor, which has been coded in Java.

Figure 6. SDL Translation fragment depicting the header

```
document [B50TF104] {
  title = "ALLOY BAR, FORGINGS, AND RINGS(INCONEL ALLOY 706)";
   org  = "GE Aircraft Engines";
   type = "specification";
  define APM {
       [Products] is "Bar";
       [Products] is "Forgings";
       [Products] is "Rings";
       [Alloy] is "INCONEL ALLOY 706";
  }
  using APM;
revision [S4+AM1] {

         note " =   Chemical Composition Weight Percent."
         "B50TF104-3.1  Chemical Composition Weight Percent. "
         "Material supplied to this ";
      . . .
}
```

Method C views the spec as a sequence of nested sections, with the translation of each section dictated by the section heading and the copyright restrictions in place. Typically, the text in the body of a section is converted into a qualified labeled note, where the qualifiers are the `product type, alloy,` etc. In situations where the copyright restrictions prohibit the reproduction of spec text verbatim, the section number is emitted instead. For GEAE B-family specs, the Method C extraction *eliminates* sections with headings such as `Scope, Applicable Documents, Notes,` etc, *combines* non-reportable nested sections under the heading `Quality Assurance Provisions, Packaging`, etc, and *outputs* the text of other sections as a labeled SDL Notes (comment). A note is a sequence of double-quotes delimited strings separated by whitespaces containing a newline and terminated by a semicolon. If these notes are selectively applicable only to a subset of qualifiers such as `product type, alloy,` etc, an SDL conditional statement with appropriate boolean expression is constructed. The latter requires introducing infix boolean operators such as `and, or,` etc, which requires special XSLT primitives and logic, especially for distinguishing first and last occurrence from others. All this is exemplified in Figure 7.

Figure 7. Template rule for processing sections (outputs only Section ID)

```
<xsl:template match="section">
   <xsl:choose>
   <xsl:when test="sectionHeading[contains(@heading,'SCOPE') or …]"/>
   <xsl:when test="sectionHeading[contains(@heading,'REQUIREMENTS')]">
      <xsl:apply-templates select="section"/>
   </xsl:when>
   <xsl:when test="sectionHeading[contains(@heading,'PACKAGING') or …]">
         note " = <xsl:value-of select="sectionHeading"/>"
         "Shall be in accordance with paragraph "
         "<xsl:apply-templates select="./section[1]"/>
          <xsl:for-each select="./section[(position() &gt; 1)
                               and (position() != last())]">,
          <xsl:apply-templates select="."/>
           </xsl:for-each>
          <xsl:if test="./section[last() &gt; 1]">
                        and <xsl:apply-templates select="./section[last()]"/>.
         </xsl:if>"
   </xsl:when>
   <xsl:when test=
     "ancestor::section/sectionHeading[contains(@heading,'PACKAGING')]"
         or …>
     <xsl:value-of select="@id"/>
   </xsl:when>
   <xsl:otherwise>
      <xsl:if test="product | specClass | alloy">
      if <xsl:if test="product">
                ( [Product Type] is
                       "<xsl:value-of select="./product[1]/@DLT"/>"
                 <xsl:for-each select="./product[position() &gt; 1]">
                    or [Product Type] is "<xsl:value-of select="@DLT"/>"
                  </xsl:for-each> )
                </xsl:if>
        <xsl:if test="product">
           <xsl:if test="specClass | alloy"> and
           </xsl:if>
        </xsl:if>
          . . .
       then {
       </xsl:if>
          note " = <xsl:value-of select="sectionHeading"/>"
          "Shall be in accordance with paragraph <xsl:value-of select="@id"/>"
         <xsl:apply-templates select="section"/>
      <xsl:if test="product | specClass | alloy"> } </xsl:if>
   </xsl:otherwise>
   </xsl:choose>
</xsl:template>
```

4. CONCLUSIONS

Manual processing of paper-based documents is laborious and error prone. Given that the general extraction problem is insoluble, it is important to propose viable solutions and understand their precise limitations. This work attempted to use the available XML Technology and Tools to develop a reasonable approach to formalizing and manipulating domain-specific text documents. The real-world legacy documents have a number of idiosyncrasies and errors that render obvious solutions fragile, thus providing opportunities for developing robust and flexible techniques. Insofar as an XML Master Spec can be created, XSLT provides a nice framework for generating various demand-driven summarizations of a spec. However, even though XSLT provides a declarative language for XML-transformation that is both readable and maintainable, it does not have the power (data structures and control structures) that a typical programmer is used to, due to the paradigm shift. The work reported here was done in the context of developing a Web-based Domain-specific Information System, and is in line with the longer term vision of building the Semantic Web of Knowledge [13].

ACKNOWLEDGEMENTS

I wish to thank Aaron Berkovich and Dan Sokol for a number of fruitful discussions on Content Extraction and the need for a Master Spec, and Prasanna Soundarapandian for implementing an XML Tagger for Spec text in FLEX.

5. REFERENCES

1. Sokol, D.Z., “Concurrent Engineering Design System for High Technology Material Suppliers”, NSF Phase II Final Report, 1997.

2. Wadler, P. An Introduction to Orwell (draft), University of Glasgow, 1985.

3. Knuth, D. E., *Literate Programming*, CSLI Lecture Notes, No. 27, Center for the Study of Language and Information (Stanford, California), 1992.

4. Thirunarayan, K., Ewing, R. L., and Reintjes, P., VHDL-AMS Parser/Pretty-Printer System, Wright State University, 1995-2002. (Available from: http://www.cs.wright.edu/~tkprasad)

5. Du Charme B., *XSLT Quickly*, Manning Publications Co., 2001.

6. Tidwell D., *XSLT*, O’Reilly, 2001.

7. Harold E. R., *XML Bible*, Hungry Minds Inc., 1999.

8. Dietel H. M., et al, *XML How to Program*, Prentice Hall Inc., 2000.

9. Soderland S. G., “Learning Information Extraction Rules for Semi-structured and Free Text,” Machine Learning, 1999.

10. Proceedings of the Message Understanding Conferences, Morgan Kaufman, 1991, 1992, 1993, 1996, 1997.

11. Grishman R., "Information Extraction: Techniques and Challenges", Information Extraction (International Summer School SCIE-97), ed. Maria Teresa Pazienza, Springer-Verlag, 1997.

12. Thirunarayan K., Berkovich A., and Sokol D. Z., "Semi-automatic Content Extraction". (submitted)

13. Berners-Lee T., Hendler J., and Lassila O., "The Semantic Web", Scientific American, May 2001 Issue. (http://www.sciam.com/2001/0501issue/0501berners-lee.html)