Using XML, XSLT, and CSS in a Digital Library

Timothy W. Cole, William H. Mischo, Robert Ferrer, and Thomas G. Habing

Grainger Engineering Library Information Center
University of Illinois at Urbana-Champaign

Abstract

The functionality of formats available for the online representation of text continues to evolve. ASCII and bit-mapped image representations of text objects (e.g., journal articles) have been superceded by more functional representations such as application-specific word processing formats and proprietary and non-proprietary information interchange formats (e.g., Adobe PDF, Microsoft RTF, TeX, SGML). Standards like SGML and now XML, which support the representation of a text as an “ordered hierarchy of content objects,” are arguably the best and most sophisticated models available for representing text objects.[1] Ratified as an international standard in 1989, SGML has become a well-established approach for encoding text. However SGML’s complexity and requirements for specialized and expensive tools to implement SGML-based systems have limited its scope as an information interchange standard, particularly in today’s Web-dominated environment. XML, established as a W3 Consortium Recommendation in February 1998, strives to make the best features of SGML more accessible to Web authors and publishers, but the XML specification itself doesn't explicitly deal with the presentation of content, nor does it address document object transformation. These issues must be addressed through the use of CSS and XSLT (both more recent W3 Consortium Recommendations). It is necessary to use these three technologies in concert to create powerful and robust text applications on the Web.

This paper describes how XML, XSLT, and CSS can be used in a digital library application. A digital library Testbed at the University of Illinois at Urbana-Champaign, originally established in 1994 and consisting of over 50,000 SGML-formatted articles from more than 44 sci-tech journal titles, was converted to XML. Associated item-level metadata also was translated into XML, utilizing RDF and Dublin Core syntax and semantics extended with project-specific XML tagging. A comprehensive index of article full-text and metadata allows full-text searching across the entire repository. XSLT and CSS stylesheets are used to present metadata and the articles themselves to end-users accessing the Testbed using either Netscape or Internet Explorer. We focus here in this presentation on the techniques used to transform our SGML collection into well-formed XML, the XML metadata structures adopted for our project, and the XSLT and CSS features employed in our Testbed. Special attention is paid to techniques for rendering mathematics and for transforming real-time between XML and HTML formats.

INTRODUCTION

The University of Illinois at Urbana-Champaign was one of six sites awarded a four year federally funded grant in 1994 under the first phase of the Digital Library Initiative (DLI-I). The principal interest of the Illinois DLI-I project was on developing techniques for the representation and delivery of full-text engineering and physics journal articles in an Internet environment. The Illinois project included research, Testbed, and evaluation components and is described in more detail elsewhere. [2]

The Illinois Testbed was constructed from source text journal articles in SGML (Standard Generalized Markup Language) format contributed by five professional society publishers. The full-text articles for the Testbed have been contributed by: the American Institute of Physics (AIP), the American Physical Society (APS), the American Society of Civil Engineers (ASCE), the Institution of Electrical Engineers (IEE), and, initially, the Institute of Electrical and Electronics Engineers Computer Society (IEEE CS).

The Testbed is presently comprised of some 55,000 articles from 44 journals in a base XML (Extensible Markup Language[3]) format along with accompanying images of figures and illustrations. The Testbed Team has implemented a Web-based retrieval and rendering system, called DeLIver (Desktop Link to Virtual Engineering Resources) which provides broad access to the full-text material. The Testbed is maintained and administered in the Grainger Engineering Library Information Center[4], a $22 million facility that opened in 1994 and is dedicated to the exploration of emerging information technologies.

In 1998, the Testbed Team received a follow-on three-year grant from the Corporation for National research Initiatives (CNRI) to expand and enhance the Illinois DLI-I Testbed. The CNRI development work is being carried out as part of their D-Lib Test Suite program.[5] Additional support for the Illinois Testbed has come from the establishment of a Collaborating Partners program that provides monetary and in-kind support for the Testbed technologies.

TESTBED GOALS AND ISSUES

When work on the UIUC DLI project began in 1994, the World Wide Web (WWW) was in a nascent stage. At that time, NCSA's Mosaic 2.0 beta was the browser of choice, the HTML 2.0 standard was still under development, Netscape had yet to release its first web browser, and Microsoft Windows 3.1 was the standard PC operating system.

The initial task of the Testbed project team was to identify technologies that were both of sufficient maturity to be usable at once and of sufficient potential to evolve over the life of the project. As the Project evolved, two clear trends emerged: one, the WWW has become the standard distributed information system used for text retrieval and display, and as a direct corollary, publishers have taken advantage of emerging Web technologies to establish their own full-text repositories.

The cornerstones of the Testbed, in terms of its retrieval capabilities, are the effective utilization of the exposed article content and structure revealed by XML and the associated article-level metadata, which serves to normalize the heterogeneous XML and provide short-entry display capability. The metadata also contains links to internal and external data, such as forward and backward links to other Testbed articles and links to Abstracting & Indexing Service databases and other full-text repositories, such as the American Institute of Physics and the American Physical Society sites. An important feature of the Testbed design is the separation of the metadata/index files from the full-text. This allows the metadata/index--containing pointers to the full-text--to be logically and physically separated from the full-text records.

The focus of the Illinois Testbed Team has been on the design, development, and evaluation of mechanisms that provide effective access to full-text engineering, physics, and computer science journal articles within a Web-based environment. The primary goals of the Illinois Testbed are:

- the construction and testing of a large-scale, distributed, multi-publisher markup-based full-text Testbed of Sci-Tech journal literature;
- the development and deployment of flexible Testbed search and rendering capabilities offering extensive links to local and remote information resources;
- the integration of the Testbed (and other full-text resources) into the continuum of information resources offered to end-users by both Library and commercial providers;
- examining the efficacy of full-text article searching vis-à-vis document surrogate searching, and exploring end-user full-text searching behavior in an attempt to identify user-searching needs; and
- developing models for the effective publishing and retrieval of full-text articles within an Internet environment and employing these models in the Testbed design and development.

On an overarching level, the Illinois Testbed project has addressed the issues connected with the migration from a print-based journal environment to an Internet-based model, with particular focus on defining retrieval and rendering mechanisms that can optimize user access to full-text journals.

This paper will describe some of the processing and rendering tools and techniques developed and utilized by the Testbed Team during the course of the Testbed implementation. In particular, the use of fine-granularity markup languages, the development of dynamic, value-added metadata structures, and the role of recently introduced information technologies such as XML, CSS (Cascading Style Sheets[6]), and XSLT (Extensible Stylesheet Language Transformations[7]) within the Testbed will be described.

Transforming SGML into well-formed XML

We continue to receive source materials in SGML from our publisher partners. They have a great deal invested in their SGML authoring and publishing tools, and equivalent XML publishing tools lag in terms of functionality – in part because XML DTDs (Document Type Definitions) cannot be as rigorous as SGML DTDs (see further discussion below). Transformation of Testbed articles from validated SGML to well-formed XML is an ongoing process. Fortunately, the SGML feature set used by our publishers when creating their SGML is relatively congruent with the feature set presently available using XML. In creating the algorithms and tools we use to transform from SGML to XML, we addressed several issues.

Simultaneously XML and SGML Compliant

The first issue involved standards compliance. Our goal was as much as possible for the transformed material to be syntactically compliant both as well-formed XML and valid SGML. Also as SGML, the material needed to be renderable by a commercially available web browser plug-in (e.g., Interleaf's, formerly SoftQuad's, Panorama viewer).

For example, tags defined in a DTD as having an "EMPTY" content model (i.e., having no child nodes and no content) are represented in SGML as a single open tag without content and without a corresponding closing tag, e.g.:

```
<Empty>
```

In XML nodes defined with an EMPTY content model are encoded as:

```
<Empty />
```

However, both SGML and XML allow elements defined with parsed character data content models (i.e., #PCDATA content models) to contain no data between start and end tags. Thus the following syntax is allowed in both SGML and XML:

```
<Empty></Empty>
```

By converting from EMPTY to #PCDATA content models in our SGML DTDs and then adding the necessary close tags in the document instances, we were able to make the document instances in our collection at once SGML and XML compliant in this regard.

Unparsed character data (CDATA) is also handled differently in XML and SGML. XML requires that CDATA content be represented as:

```
<Tag><![CDATA[…unparsed character data …]]><Tag>
```

SGML identifies CDATA tags in the DTD without requiring any special markup in the document instance. Currently, CDATA tags are left in their SGML form. Properly identifying and processing CDATA content are achieved during dynamic processing at the time of rendering.

SGML has a looser requirement in regard to processing instruction encoding. Processing instructions may be represented in SGML as:

```
<? … Processing Instruction … >.
```

XML is stricter and requires that processing instructions include a closing question mark character, e.g., XML processing instructions are encoded as:

```
<? … Processing Instruction … ?>
```

Because this format of processing instruction is more strict in XML than SGML, but is allowed in SGML, we were able to simply add the closing question mark to processing instructions in all document instances to make them XML compliant without invalidating them as SGML documents. Of course other non-SGML dependencies should also be checked before implementing this approach.

External Links & Character Entities

It was not possible in all cases to keep document instances simultaneously compliant with both SGML applications (e.g., the Panorama SGML viewer plug-in) and XML applications (e.g., Microsoft's Internet Explorer). For instance, Panorama requires special attribute types and external entity declarations in the SGML DTD and document instance header in order to support linking to external figures and tables. Declarations of external file entities and character entities in the header of a well-formed XML document instance is not supported by Internet Explorer. With proper character entity declarations in the DTD or document instance header and an associated SDATA.MAP file, Panorama supports named entities and numbered character entities up to 255 (i.e., extended ASCII), but Panorama does not support numbered character entities above 255 (i.e., Unicode). Except for a few basic character entities (e.g., &lt; &gt;, &amp;) named entities are not allowed in a well-formed XML document (i.e., in an XML document without an associated DTD), but Unicode numbered character entities are supported.

In order to allow hyperlinking to external documents from a well-formed XML document instance without requiring an XML DTD, the XML Linking Language (XLink) has been proposed[8]. Although browsers do not currently support XLink, it was still useful to convert SGML elements that referred to external files into preliminary XLink semantics. Therefore, nodes such as:

```
  <FIG NAME="F1">
```

were converted into:

```
  <fig xml:link="simple" name="F1" href="fig1.jpg" show="new" actuate="user">
```

At this time, however, Internet Explorer requires the use of HTML namespace elements (e.g., anchor, img) to support linking to external figures and tables. XML document instances are stored on our system with XLink style figure and table anchors embedded. With appropriate modification of our SGML DTDs, these can be simultaneously SGML and XML compliant. Then, when a document is requested, external links are converted to appropriate HTML namespace nodes.

To handle character entities, SGML named character entities in the document instance are converted into empty tags with attributes that specify the named entity, the equivalent ASCII numeric and unicode code point representations, and the font required to render the character using the ASCII numeric value. This is shown as:

```
  <uichar class="Symbol" entity="alpha" isocode="&#97;" unicode="&#x03B1;">
</uichar>
```

Dynamic processing at the time of document is delivered parses the **uichar** node, and determines which combination of attribute values to use to create the appropriate character entity. We need to consider several alternatives in representing various character glyphs because of the current non-standard and incomplete implementation of the coding schemes.

Named character entities present in the original SGML document instance as attribute values are converted into their equivalent ASCII numeric representation. This is possible as long as the named entity refers to basic characters such as parentheses, square barackets, etc. E.g., the following element:

```
<fence lpost=”&lsqb;”…>
```

becomes:

```
<fence lpost=”&#091;”…>
```

and is both SGML and XML compliant.

Well-Formed vs. Valid XML

Creating an XML DTD that could be invoked from the document instances in our collection would simplify some of the work described above. However, there are several functional limitations that impede converting SGML DTDs into XML DTDs. For example, the XML DTD does not permit inclusions and exclusions. This complicates the inclusion of common elements that can appear anywhere in a document such as figures and formulas. Floating elements can be represented in SGML as:

```
<!ELEMENT Article - - (front, body) +(%i.float;)>
```

Front must precede body. Floating elements may occur anywhere within front and body. An XML DTD would require specifying float elements within the content model of each sub-element within front and body. This can be a tedious and complicated process. An option would be to use a less restrictive content model where front no longer must precede body, as shown by:

```
<!ELEMENT Article (front | body | %i.float;)*>
```

However, this approach reduces the usefulness of the DTD as a quality and consistency control mechanism during the authoring and publishing process. Order of elements is no longer enforced, nor is it even required by the DTD that both child nodes appear.

Conversion of an SGML DTD into an XML DTD is further made more complicated by the fact that ‘&’ connectors are not permitted in content models. The ‘&’ connector requires all elements to occur, but in any order. Instead, the content model must be expanded to take into account all possible sequence orders. Another functional limitation is that mixed content models do not permit constraining order or number of occurrences of individual elements. The following content model is not allowed in an XML DTD:

```
<!ELEMENT Other ((author, journal) | (#PCDATA))>
```

The result of such limitations is that it is difficult to make XML DTDs as restrictive as SGML DTDs. Furthermore, SGML authoring tools are still much more mature than XML authoring tools. The course of action we’ve suggested to our publishers is to continue to create document instances in SGML. However, our research has shown that it is possible to modify SGML DTDs and current authoring methodology in small ways in order to generate document instances that are both valid SGML and well-formed XML in all or nearly all respects. It is hoped that the proposed XML schema standards[9] will overcome the current limitations of the XML DTD.

Metadata Schema

Testbed item-level metadata records are used to facilitate searching by normalizing key fields from different publisher DTDs, provide common and easily displayable intermediate search results, and offer value-added information in the form of links to the cited or citing article. Links included in the metadata records point directly to the full-text version of the article, to separate article components (e.g., figures and tables), to related articles in the Testbed, and to related article references in external Abstracting & Indexing database services.

Use of Metadata to Aid in Item Discovery

Experience with documents from various publishers has shown that there are many variations in the way documents are tagged, even for those that claim to use a DTD of a similar type. For example, one DTD points to the author of the article with the tag '**<author>**'. Another publisher points to equivalent information using the tag **'<auth>**'. Sometimes relevant information must be culled from multiple nodes. In other cases where the granularity of markup is large, metadata information must be inferred.

Metadata records are also used to pre-coordinate information that exists as complex constructs. For example, authors are associated to their institutional affiliations by indirection. An internal reference id next to the author's name is resolved at the end of the article, where all affiliations are listed by reference number. The search algorithms provided by the indexing system cannot easily associate the author to the appropriate affiliation. The metadata present the association between author and affiliation in a simpler and easily indexable form.

There are features inherent in SGML/XML that can inhibit successful searches. In addition to tags that comprise the document's regions or access points, a document is also punctuated with a variety of character entities and special purpose tags such as processing instructions. Since these special elements are part of the text of a document they can complicate a search. For example, assume a user is searching for the author O'Shay or Félix. They may be represented in the document as O&rsquo;Shay and F&eacute;lix. Users would generally enter the names as O'Shay or Oshay, and Felix. No hits will be returned. To better normalize content for indexing, metadata content is stripped of all tags and character entities that could interfere with searching.

Other Descriptive Metadata

Metadata records also are used as part of the Testbed's link management scheme. URLs to document full-text and associated external figures and tables are stored in metadata records. For most of the Testbed collection, document full-text is available in PDF as well as in SGML, XML, and HTML. Also, by keeping separate URLs for each figure, users can retrieve figures independently of document full text. In addition to providing document and figure linkages, the metadata also provide linkages to external related resources. These include other articles cited in the document that might be electronically available. Links to the local OPAC can provide location and availability information to printed materials. The metadata used in the Testbed provide links to relevant INSPEC and Compendex records, as well as bibliographic records maintained by various publishers. As new items are added to the Testbed each document is checked for links to articles already in the collection. When such links are found, the metadata record for the earlier item is also updated to show the relationship to the newer document. (E.g., addition of an errata article results in a link be added to the original article's metadata showing linkage to the newly added errata.)

Finally we rely on metadata to serve as document surrogates in the display of intermediate results (as discussed further below). While the full-text of metadata records and journal articles are indexed together, metadata records are maintained separate from the full-text document files themselves. The metadata records are dynamic (e.g., links can be added) while the document instances themselves are static. This helps protect the integrity of the original source documents. This arrangement also facilitates incremental indexing.

Use of Resource Description Framework and Dublin Core

We currently maintain our metadata as XML files using the Resource Description Framework (RDF) syntax[10]. RDF syntax provides a standard way for using XML to represent metadata. The semantics of the metadata conform to the Dublin Core (DC) model,[11] with a further level of granularity offered by the inclusion of an extensive set of project-specific, customized elements. Search clients familiar with the corpus of idli tags can achieve results with much greater precision, while index and search systems only familiar with DC semantics can still be used. For example, several idli tags partition the content within the dc:Source tag:

```
    <dc:Source>
            <idli:publication type="journal article">
                    <idli:journal_title>Applied Physics Letters</idli: journal_title >
                    <idli:journal_title_abbreviation>Appl. Phys. Lett.</idli:
journal_title_abbreviation >
                    <idli:volume>70</idli:volume>
                    <idli:issue>11</idli:issue>
                    <idli:first_page>1372</idli:first_page>
                    <idli:pagination>1372 - 1374</idli:pagination>
            </idli:publication>
    </dc:Source>
```

Our metadata schema is verbose in that DC elements are repeated for each occurrence of an intellectual element (e.g., author information for each article author is contained within a separate DC Creator tag). RDF containers (e.g., **seq**, **alt**, & **bag**) are used to describe relations between repeated DC elements. Thus the two authors of an article could be represented in the metadata record for that article as shown below:

```
<rdf:seq>
  <rdf:li>
    <dc:Creator>
      <idli:author_info>
        <idli:author_name>Giust, G. K.</idli:author_name>
        <idli:organization_name>Department of Electrical Engineering, Arizona
State University, Tempe, Arizona, 85287-5706</idli:organization_name>
      </idli:author_info>
    </dc:Creator>
  </rdf:li>
  <rdf:li>
    <dc:Creator>
      <idli:author_info>
         <idli:author_name>Sigmon, T. W.</idli:author_name>
         <idli:organization_name>Department of Electrical Engineering, Arizona
State University, Tempe, Arizona, 85287-5706</idli:organization_name>
      </idli:author_info>
    </dc:Creator>
  </rdf:li>
</rdf:seq>
```

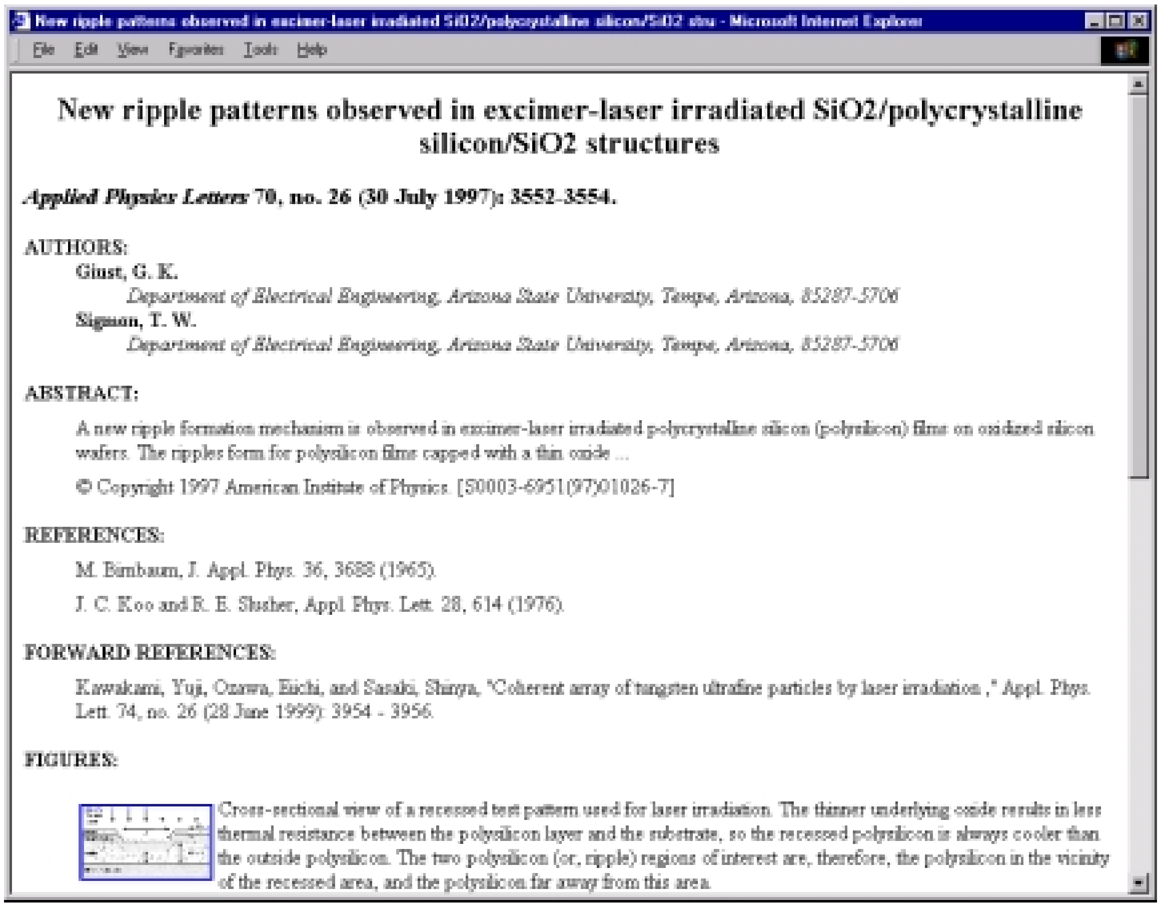
Creating the Metadata

Metadata are extracted from the XML files. The parser uses the document structure to extract and normalize key elements. Currently, the parser treats the document as a text stream and loads the parsed information into data structures. Value-added content is incorporated into the metadata. Most of the added data concerns links and relations to other articles inside and outside the testbed. Additional information (e.g., publisher name, copyright statement) not included in the document instance as originally generated for internal use by the publishers is also added as necessary. Calls made to various database repositories determine which links to include in the metadata.

A version of the parser used to generate the metadata is being developed that relies on the Document Object Model (DOM) and the transformative component of XSL (XSLT) to extract the metadata information. XSL stylesheets can be used to selectively expose segments of the original document. This should insure a greater degree of parser flexibility and portability.

Using XSLT to Generate Metadata Views

As described above, the item-level metadata in our digital library Testbed is designed both to assist in item discovery and to describe the items in the Testbed. Enough descriptive information about each item is included in its metadata record to support both summary and extended citation views. Metadata records include descriptive information about the text (e.g., author, title, abstract), about other discrete article components (e.g., figures and tables stored as separate files), and about the document object's relationships with other document objects. Figure 1 illustrates an extended citation view as generated from the XML RDF metadata record for a representative article in the Testbed.


New ripple patterns observed in excimer-laser irradiated SiO2/polycrystalline silicon/SiO2 stru - Microsoft Internet Explorer
File Edit View Favorites Tools Help

New ripple patterns observed in excimer-laser irradiated SiO2/polycrystalline silicon/SiO2 structures

***Applied Physics Letters* 70, no. 26 (30 July 1997): 3552-3554.**

AUTHORS:

Giust, G. K.
Department of Electrical Engineering, Arizona State University, Tempe, Arizona, 85287-5706

Sigmon, T. W.
Department of Electrical Engineering, Arizona State University, Tempe, Arizona, 85287-5706

ABSTRACT:

A new ripple formation mechanism is observed in excimer-laser irradiated polycrystalline silicon (polysilicon) films on oxidized silicon wafers. The ripples form for polysilicon films capped with a thin oxide ...

© Copyright 1997 American Institute of Physics. [S0003-6951(97)01026-7]

REFERENCES:

M. Birnbaum, J. Appl. Phys. 36, 3688 (1965).

J. C. Koo and R. E. Slusher, Appl. Phys. Lett. 28, 614 (1976).

FORWARD REFERENCES:

Kawakami, Yuji, Ozawa, Eiichi, and Sasaki, Shinya, "Coherent array of tungsten ultrafine particles by laser irradiation ," Appl. Phys. Lett. 74, no. 26 (28 June 1999): 3954 - 3956.

FIGURES:

Cross-sectional view of a recessed test pattern used for laser irradiation. The thinner underlying oxide results in less thermal resistance between the polysilicon layer and the substrate, so the recessed polysilicon is always cooler than the outside polysilicon. The two polysilicon (or, ripple) regions of interest are, therefore, the polysilicon in the vicinity of the recessed area, and the polysilicon far away from this area.

Figure 1 – Extended citation view (some text deleted for brevity)

In our project we use XSLT template structures within XSL stylesheets to generate both brief and extended citation views real-time. The XSLT templates are used to selectively present, reorganize, and transform content and markup contained in our metadata records. To fine-tune presentation, CSS instructions are then applied to the transformed output document generated by the XSLT processor. Typically it is easiest to construct the XSLT templates by working backwards from a representative example of the desired result. This was the approach taken to create the initial drafts of the XSLT templates we now use.

Creating the Model of the XSLT Output

We began by inspecting a single, representative XML RDF metadata record. Using content and markup from this XML file, we then manually encoded in HTML the brief and extended citation views desired. Because these HTML documents were to form the basis for our XSLT templates, we made these views of the metadata XHTML-compliant. It was also desirable that the XHTML encoding of these views be uncluttered and human-readable. Tools that add verbose and unnecessary markup are less useful for this purpose. We actually resorted to Microsoft Notepad and Microsoft XML Notepad when creating our XHTML views in order to give us maximum control over the encoding process.

At this stage we created CSS stylesheets to augment default XHTML display behaviors as necessary to create the precise views desired. We also chose to use CSS stylesheets in order to facilitate fine-tuning of metadata views later. It's important to remember that CSS stylesheets created to work in concert with XSL stylesheets work on the output generated by applying the XSLT stylesheet, not on the original XML document instance. To facilitate construction of our CSS stylesheets we added class and id attributes to several of the XHTML elements used. For our metadata views our CSS stylesheets were brief. Browser-specific versions were not required (but see below the further discussion of more advanced CSS techniques).

Creating the XSLT Templates

The next step was to convert the XHTML citation views of our representative metadata record into XSLT templates that could then be implemented in an XSLT stylesheet and applied across our collection of metadata records. Conceptually, this is a four-part process.

First, we identified the XML nodes from our metadata records that were to be transformed (i.e., which XML nodes contained the object-specific content and markup we had included in our manually generated XHTML views). In an XSLT template, content and optionally mark-up from the source XML file is incorporated into the transformed output document by inserting XSLT **value-of** and XSLT **copy** elements (**<xsl:value-of …>** and **<xsl:copy …>**) at the locations in the template where the content and markup should appear. Thus it was at this stage we replaced any content and markup manually copied from the XML metadata record into our XHMTL citation views with one of these elements.

The XSLT **value-of** element extracts content and only content from the XML metadata record. Any embedded markup present in the original XML file is discarded and content from sub-elements concatenated. XSLT includes a pattern-matching syntax used to identify the XML nodes in the source file that each XSLT element transforms. For the XSLT **value-of** element, the XML element in the original metadata record from which content is to be extracted is determined by the value of the XSLT **value-of** element's select attribute or, if no select attribute is present, by the current context where the **value-of** element appears. The XSLT value-of element has an EMPTY content model (i.e., has no child nodes).

The XSLT **copy** element can be used to copy both markup and content into the XSLT output stream. Markup attributes and child element markup may be included in the copy if desired. In XSLT templates, the XSLT **copy** will have child nodes – at least one of which will be an XSLT **apply-templates** element. The markup and content to be copied is determined by the current context at the point in the template where the copy element is invoked and by the select attribute of the **apply-templates** element.

Note that, as required, elements and attributes may be added to output encoding either explicitly or through the use of the XSLT **element** and **attribute** elements (**<xsl:element>** and **<xsl:attribute>**). The name attribute of these XSLT elements becomes the name of the element or attribute created in the output stream and the content of these elements is the content of the created element or the value of the created attribute. When using the XSLT **attribute** element, the created attribute is added to the current element at the point in the template where the attribute element occurs. The XSLT **attribute** element in particular is useful when adding to the output stream an href or src attribute whose value is to be the content of some XML node in the original XML metadata record. The following XSLT template fragment illustrates the use of this technique to add an href attribute to an HTML anchor tag. In this instance the highlighted text of the resulting anchor in the output document (i.e., the content of the HTML anchor node) is the value of the **idli:link** element's role attribute in the source XML metadata record, while the value of the href attribute itself is the content of the **idli:alternate** node in the source XML metadata record.

```
… <a>
   <xsl:attribute name="href">
      <xsl:value-of select="idli:alternate" />
    </xsl:attribute>
    <xsl:value-of select="idli:link/@role" />
</a> …
```

The second part of the process of turning our XHTML citation views into XSLT templates was to identify which transformed XML node names from our source metadata records might be repeated. The XSLT **for-each** element (**<xsl:for-each>**) can be used to efficiently handle such repeatable content. The output encoding and XSLT elements contained within a for-each element are implemented for each occurrence of the repeatable node in the original XML document. Again current context and the **for-each** element's select attribute are used to identify the node in the source file to which the for-each transformation should apply.

The third part of the process is to address how allowable variations will be handled (e.g., optional elements, punctuation that depends on how many occurrences of a particular node are present in the original XML metadata document, etc.). XSLT **if** and **choose** elements (**<xsl:if>** and **<xsl:choose> … <xsl:when> … <xsl:otherwise>**) can be used to control branching within the XSLT stylesheet. The XSLT **if** element works much the same as an IF statement in most programming languages. The XSLT **choose** element, which contains child **when** and (optionally) **otherwise** elements, is most similar to a programming language CASE statement. The following XSLT template fragment shows how the XSLT **choose** element can be used to control punctuation and formatting of the list of article authors based upon how many author names are present in the metadata record. Note that in XSLT pattern matching the first occurrence of a node is always numbered 0 rather than 1.

```
… <h4>
<xsl:choose>
   <xsl:when test="//rdf:li[dc:Creator][3]">
            <xsl:value-of select="//rdf:li[dc:Creator][0]//idli:author_name" />,
            <xsl:value-of select="//rdf:li[dc:Creator][1]//idli:author_name" />,
            <xsl:value-of select="//rdf:li[dc:Creator][2]//idli:author_name" />, et al.
   </xsl:when>
   <xsl:when test="//rdf:li[dc:Creator][2]">
            <xsl:value-of select="//rdf:li[dc:Creator][0]//idli:author_name" />,
            <xsl:value-of select="//rdf:li[dc:Creator][1]//idli:author_name" />, and
            <xsl:value-of select="//rdf:li[dc:Creator][2]//idli:author_name" />
   </xsl:when>
   <xsl:when test="//rdf:li[dc:Creator][1]">
            <xsl:value-of select="//rdf:li[dc:Creator][0]//idli:author_name" /> and
            <xsl:value-of select="//rdf:li[dc:Creator][1]//idli:author_name" />
   </xsl:when>
   <xsl:when test="//rdf:li[dc:Creator][0]">
            <xsl:value-of select="//rdf:li[dc:Creator][0]//idli:author_name" />
   </xsl:when>
</xsl:choose>
</h4> …
```

Finally, a few transformations required direct manipulation of the content extracted from the original XML metadata records. While the current XSLT Recommendation from the W3 Consortium anticipates the need for extensions to handle such situations, the mechanisms to be used have not yet been finalized. Microsoft's XSLT processor provides two provisional XSLT elements, **<xsl:eval>** and **<xsl:script>** to support XSLT functional extensions pending adoption of a standard. The XSLT **eval** element is used within XSLT template elements to invoke an intrinsic or user-defined function. User-defined functions are described external to the XSLT templates in an XSLT **script** element. The value returned by the function replaces the XSLT **eval** element in the output stream or document. By adding an attribute (expr) to the XSLT **if** and **when** elements, Microsoft has also unilaterally implemented an XSLT extension that allows resolution of conditionals based on true / false returns from user-defined functions.

Implementation Issues

Current W3 Consortium Recommendations and Working Drafts describe how XSLT templates are to be included within XSL stylesheets and how such stylesheets can be attached to XML documents using XML processing instructions. The details are generally straightforward (e.g., XSL stylesheets must be delivered as an appropriate MIME type, appropriate namespace declarations must be used, etc.). However, implementation of these guidelines in production systems is incomplete.[12] Current XSL and XSLT specifications anticipate the need for multiple XSL stylesheets to be attached to a single XML document (e.g., to support multiple views of the data contained within that document), but the mechanisms to determine which of the referenced stylesheets to apply under which circumstances is unclear. At present Microsoft's Internet Explorer web browser always implements the first-named style sheet.

Other proposed mechanisms for combining XSLT templates and XSL stylesheets (e.g., the XSLT **import** and XSLT **include** elements) remain unimplemented by most systems as does the XSLT **output** element, intended to allow explicit setting of document-level characteristics of documents created by XSLT processors. Not all intrinsic functions specified in the current XSLT Recommendation are implemented in all systems, although workarounds generally exist. As a result, some trial and error work is needed when implementing XSLT templates and in some cases workaround approaches must be used to get desired results.

Server-Side Implementation of XSLT

One such workaround approach is to implement server-side XSLT processing to handle certain situations. For example, because Netscape web browsers do not yet support XSLT processing themselves, processing XSLT instructions on the webserver and sending only the resultant output to the web browser allows the use of XSLT in Web applications designed to be used by both Netscape users and Microsoft Internet Explorer users. Server-side implementation of XSLT also provides a means to deal with the multiple XSL style sheet issue. System links can be assigned so as to allow the client web browser to handle the primary or default XSL stylesheet, while alternate views are delivered through calls to server-side scripts which apply alternate XSL stylesheets. Finally, server-side XSLT implementations can be used to reserve certain data contained in the full XML source. Since XSLT template transformations selectively include content from the source XML document, content not explicitly extracted or transformed by the XSLT template is never sent to the client when the transformation is done on the server rather than on the client. (Of course in certain situations this may be seen as less desirable.)

While the exact mechanisms employed to implement XSLT on a webserver will vary and detailed description of these mechanisms are beyond the scope of this paper, the implementation does not have to be difficult. Microsoft's Internet Information Server (IIS) provides components with associated methods that make the implementation straightforward. Below is a VBScript fragment designed to be run using the built-in IIS Active Server Platform component. In this illustration, the transformNode method actually applies the XSL stylesheet. Note also that in practice the name of the XML file to transform would typically be provided as a script argument or parameter and that the name of the XSL stylesheet to apply would normally be found by reading the processing instructions contained in the XML file.

```
Set xml = Server.CreateObject("Microsoft.XMLDOM")
Set xsl = Server.CreateObject("Microsoft.XMLDOM")

xml.async = false
xml.load("http://myServer/myXMLDocument.xml")
xsl.async = false
xsl.load("http://myServer/myXSLStylesheet.xsl")
Response.write xml.transformNode(xsl.documentElement)

set xml = Nothing
set xsl = Nothing
response.end
```

Using CSS and Scripting to Render Complex Mathematics

One of the major deterrents to scientific or technical publishing on the web has been the difficulty of rendering mathematics precisely as desired. Until recently most web browsers simply did not support the level of formatting, positioning, and special characters needed for anything beyond the simplest of mathematics. The situation has improved with the implementation of several standard technologies in the latest web browsers (e.g., Unicode, CSS, downloadable fonts, JavaScript, the DOM), though there remain problems. Depending on the effects desired and your expectations of which web browsers and which web browser versions will be used to access your site, it may be necessary to create multiple CSS stylesheets for each view to handle browser variations. While both Netscape Communicator and Microsoft's Internet Explorer support the CSS Version 1 specification well, neither supports all of CSS Version 2. Exactly which CSS Version 2 features are supported is different for each browser, and in several cases there are variations in how the specification has been interpreted and implemented.

We have been most successful in utilizing CSS to improve native rendering of complex mathematics using version 5 of the Microsoft Internet Explorer web browser (IE5). IE5 is currently the only widely available browser that supports all of the above technologies to the extent needed for native rendering of complex mathematics. We have had less success with Netscape web browsers to date, though Netscape browsers are capable of rendering some complex mathematics acceptably.

Other Approaches to Rendering Mathematics

One of the typical approaches to rendering complex mathematics in a web browser has been to convert all mathematics into image formats, either GIF or JPEG, which can then be embedded easily in web pages. There are disadvantages to this approach. For scientific or technical articles, there can be hundreds of mathematics images required for a single web page, making such an approach unwieldy. Tempering this approach by only translating the more difficult mathematics to image format ameliorates this problem somewhat. (E.g., for one segment of the Testbed collection we translate to image format all display mathematics, which is in any event provided to us by the publisher in TeX rather than SGML, but render inline mathematics using CSS and character entities.) An additional drawback of this approach is that the images will not automatically scale with the rest of the text if the user decides to increase the font size for easier viewing. Moreover both GIF and JPEG images suffer in viewability somewhat when rescaled.

Another approach has been to use browser plug-ins that support the viewing of different mathematics formats, such as TeX, MathML, or proprietary formats, such as MathType. Such plug-ins typically support viewing of mathematics embedded in marked up text, though the integration is rarely completely seamless. The major disadvantage of this approach is that it requires all users to take special steps to acquire and install the plug-in. Based on our experience, not all users have the desire or expertise to do this, and are particularly reluctant if the viewer is not available freely.

Using CSS to Render Mathematics

In our approach to rendering mathematics we take mathematics marked up as well-formed XML, perform transformations on the server to improve its renderability, send it to the browser, and allow the browser to render the XML using CSS rules, JavaScript, and downloadable fonts. There are a number of advantages to dealing with mathematics natively in the browser. One is that the mathematics will scale with the rest of the page if the user changes font size. The user can also programmatically search for text or characters that occur in the mathematics. No special plug-ins are required, and the network bandwidth required is generally less than if bitmap versions of the mathematics were downloaded.

There remain difficulties with this approach, most associated with a lack of standardization in the markup used. Considerable effort is required to analyze each markup approach and decide how best to handle it in CSS. Mathematics markup schemas that predate the Web and XML are also poorly optimized for use in a Web environment. Adoption of an appropriate standard schema (e.g., MathML[13]) would reduce these problems. Such increased standardization possibly also would facilitate development of tools and plug-ins that could directly read and use the mathematical equations presented in other applications (e.g.,mathematics processors like Mathematica).

Illustrations

The following is step by step description of how the following simple inline equation was rendered in IE5:

$$\frac{1}{3}\, T_F$$

The original well-formed XML mark-up for this equation is as follows:

```
…<formula><f><fr><nu>1</nu><de>3</de></fr>T<inf>F</inf></f></formula>…
```

The first thing that occurs when the article containing the above math is requested is that a script on our server is invoked which preprocesses the mathematics markup to prepare the article for rendering by the requesting browser. This server-side preprocessing sends the appropriate header text to the browser, specifying the appropriate cascading style sheet file, JavaScript file, and downloadable font files. Which files are sent depends on the markup scheme being used and the client web browser. The XML source file is transformed into HTML (for Netscape) or sent as XML (for IE5). In the latter case the header text also specifies that the ‘html’ namespace will be used within the XML. Following is the header text used for IE5:

```
[1]   <?xml version="1.0" standalone="yes"?>
[2]   <?xml-stylesheet type="text/css" href="aps_ie5_xml.css" ?>
[3]   <xml xmlns:html="uri:html">
[4]   <html:html>
[5]   <html:head>
[6]   <html:title>Theory of Transitions in Gated Semiconductors</html:title>
[7]   <html:base href="http://forseti.grainger.uiuc.edu/~aps/sc.asp" />
[8]   <html:script language="javascript" TYPE="text/javascript" SRC="aps_script.js">
[9]   </html:script>
```

```
[10]  </html:head>
[11]  <html:body>
```

Line 1 is the standard header for all XML files. Line 2 specifies the cascading style sheet to use. Line 3 identifies that the ‘html’ namespace may be used. This means that any tag name preceded by ‘html:’ is treated as a standard HTML element, instead of a generic XML element. Lines 4-11 are standard HTML headers. Line 8 identifies the client-side JavaScript file to be included.

In addition to providing a standard header and footer, the server-side preprocessing also modifies or adds to the other elements in the original XML so they may be rendered in the browser. This includes converting the XML table mark-up into standard HTML table markup, adding HTML anchor ‘**<html:a href=...>**’ and image ‘**<html:img src=…>**’ tags where appropriate, and other transformations. However, most of the preprocessing modifications that occur are for the mathematics rendering. Following is how the server-side preprocessor modifies the XML for the above simple equation (modifications and additions are shown in italic):

```
[1]   <formula>
[2]   <html:nobr>
[3]   <f>
[4]   <fr>
[5]   &#160;
[6]   <html:span id="dli10" class="fr" >
[7]   <nu>
[8]   <norm>1</norm>
[9]   </nu>
[10]  <de>
[11]  <hidden_sup>|</hidden_sup>
[12]  <norm>3</norm>
[13]  <hidden_inf>|</hidden_inf>
[14]  </de>
[15]  </html:span>
[16]  &#160;
[17]  <html:script language="JavaScript">SetWidth("dli10");</html:script>
[18]  </fr>
[19]  T
[20]  <inf>F</inf>
[21]  </f>
[22]  </html:nobr>
[23]  </formula>
```

The following describes the various transformations that were done on the original XML:

- Lines 2 and 22 enclose the mathematics in an HTML ‘**nobr**’ tag so that inline equations will not have line breaks in them.
- Lines 5 and 16 add some spacing around the fraction primarily to improve its appearance.
- Lines 6 and 15 enclose the fraction inside an HTML ‘**span**’ tag with a ‘class’ attribute of ‘fr’ and a unique ‘id’ attribute. This is required so that we can uniquely identify the fraction, and we can refer to it later. An ‘id’ attribute cannot simply be added to the original XML ‘**fr**’ tag because without a DTD an ‘id’ attribute has no special meaning to XML. However, an ‘id’ attribute always has special meaning to HTML, allowing the element with the unique id to be referenced in script.

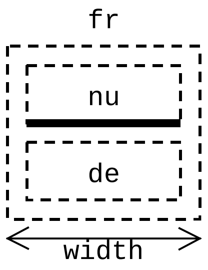
- Lines 8 and 12 enclose numbers inside a '**norm**' tag. This is required so that the numbers can be rendered in a normal, non-italicized font. By default, all text in a formula is rendered in italic, except for some exceptions such as numbers, fences, and some others. The preprocessing script detects these cases as it streams over the XML file, and encloses the appropriate text in '**norm**' tags.
- Lines 11 and 13 add some invisible superscripted and subscripted shims around the fraction denominator. This is required so that multiple fractions inside of one formula line up vertically, even if some of the denominators have sub or superscripts and others do not.
- Line 17 invokes a JavaScript function that sets the appropriate width for the fraction, identified by the previously set 'id' attribute. This function call at the end of the fraction is needed because the way that fractions are styled requires the width of the fraction to be explicitly set according to what it contains in its numerator or denominator (whichever is wider). The JavaScript 'SetWidth' function accepts an id number to uniquely identify the fraction (or radical, overline, or underline). It then uses the XML Document Object Model (DOM) to analyze the fraction and determine how wide to make it. This function essentially counts the characters inside the numerator and denominator. It must also account for arbitrary mathematical constructs contained inside the fraction, including other nested fractions, special characters, and so forth. There is a similar function called 'SetHeight' that is used to determine the height of fences and radicals, based on what they contain.

Following is an excerpt of lines relevant to the above example from the Cascading Style Sheet:

```
[1]   formula {font-style:italic}

[2]   norm {font-style:normal}
[3]   formula html\:span.fr {line-height:normal; font-size:70%; vertical-align:middle}
[4]   formula html\:span.fr nu {display:block; vertical-align:center;
      text-align:center; border-bottom:"thin solid black"}

[5]   formula html\:span.fr de {display:block; vertical-align:center; text-align:center}

[6]   inf {vertical-align:sub; font-size:80%}
[7]   sup {vertical-align:super; font-size:80%}

[8]   hidden_inf {visibility:hidden; vertical-align:sub; font-size:80%}
[9]   hidden_sup {visibility:hidden; vertical-align:super; font-size:80%}
```

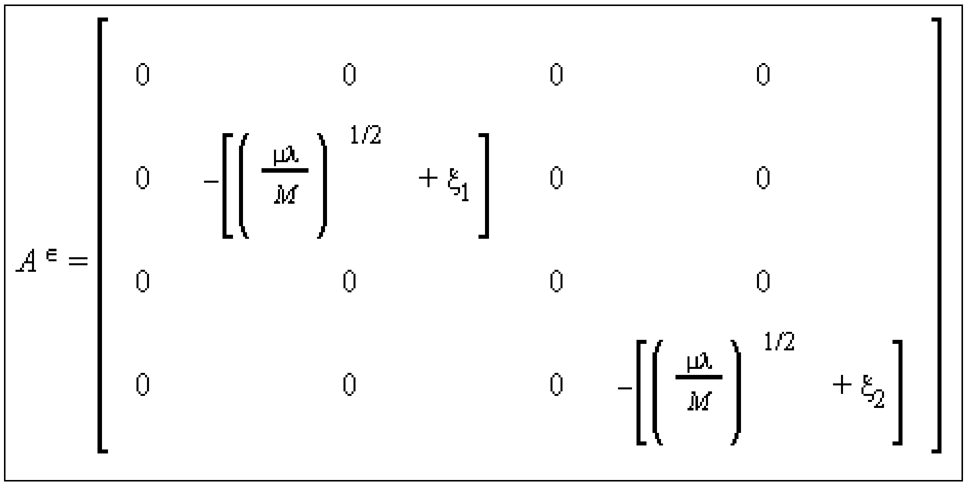
- Line 1 specifies that '**formula**' should use the italic font style, unless overidden for particular text (e.g., by the '**norm**' tag, as in line 2).
- Lines 2 to 4 specify the styling used within **formula** elements for fraction (**span.fr**) elements and for numerator (**nu**) and denominator (**de**) elements inside fraction elements. Because the JavaScript 'SetWidth' function has explicitly set the fraction's width style, its two children nodes, **nu** and **de**, are constrained to flow within that same width inside of the fraction. The numerator and denominator are stacked vertically inside the fraction because they each have a 'display:block' style, meaning that each element is separated from the surrounding text by line breaks. The horizontal bar separating the numerator and denominator is specified by the 'border-

bottom:thin solid black' style which specifies that the numerator should have a thin, solid, black line on its bottom border. See the following diagram:



- The font size of fractions inside of inline formulas has also been reduced to 70% to ensure the fraction will fit in a normal line of text without overlapping the previous or following lines. The 'vertical-align:middle' style ensures that the fraction is vertically aligned with the center of the surrounding text.
- Lines 6 and 7 specify that '**inf**' and '**sup**' tags should be styled as subscripts or superscripts with a smaller font. Lines 8 and 9 are the same as 6 and 7, except that they are styled as hidden. Hidden means that they will take up space on the page as normal, but they will not be visible.

Similar techniques to those described above are used for the various other mathematics constructs, such as radicals, integrals, arrays, stacks, fences, etc. We have also, with some amount of success, applied these techniques in a general fashion to mathematics of arbitrary complexity. Following are some example screen dumps of some more complex mathematics natively rendered by the IE5 browser using our scripts and stylesheets:



$$A^{\varepsilon} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -\left[\left(\frac{\mu\lambda}{M}\right)^{1/2} + \xi_1\right] & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\left[\left(\frac{\mu\lambda}{M}\right)^{1/2} + \xi_2\right] \end{bmatrix}$$

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{p=1}^{N} \sum_{j=1}^{n} (T_j^p - Y_j^p)^2}{N\text{-}n}}$$

$$p_0 = \frac{\eta}{\mathrm{F}^2} + \frac{2\left[\left(\frac{\partial h}{\partial x}\right)^2 \frac{\partial u}{\partial x} - \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \frac{\partial h}{\partial x} + \frac{\partial v}{\partial y}\right]}{\mathrm{R}\left[1 + \left(\frac{\partial \eta}{\partial x}\right)^2\right]} - \frac{K}{\mathrm{W}}$$

$$\omega_{\mathrm{lim}} = \frac{\epsilon_{M,u} E_{frp}}{f_k} \rho_{v,\mathrm{lim}}$$

$$= \frac{(g+1)}{\left(\frac{\epsilon_{frp,u}}{\epsilon_{M,u}} - g\right)} \left[\frac{0.4(g+1)}{\gamma_M} \frac{1}{\left(1 + \frac{\epsilon_{frp,u}}{\epsilon_{M,u}}\right)} - \frac{N_{Rd}}{\ell t f_k} \right]$$

Remaining Issues

The above approach does have limitations. The quality of the result, while understandable, does not approach the quality of a dedicated mathematics typesetting system and the quality of the printed mathematics. You can see some of these problems from the above examples: e.g., fraction bars that do not align, radical symbols detached from the overbar, etc. Font issues also remain. In some cases public-domain glyphs aren't available. In other cases the downloadable font mechanisms fail when used in conjunction with certain CSS techniques (most notably on Netscape web browsers).

Some of these problems can be overcome using existing browser capabilities, given enough time and energy, but the preferred approach would be to attack the problem from both directions – that is to improve both stylesheet techniques and browser functionality and consistency. That is only likely when and if the community converges on more consistent ways of marking up complex mathematics. That of course is the goal of the MathML effort, and also of efforts such as STIX[14], a project by the Scientific and Technical Information Publisher's Group to formulate (and include within Unicode) a comprehensive collection of characters needed in the course of scientific and technical publishing.

Conclusion

Our experience to date shows the potential of these new technologies. XML is an excellent format for search and retrieval of textual data. It's supports fine granularity and is useful not only for the markup of primary source material, but also for the markup of metadata about other objects. XSLT and CSS enhance the usefulness of XML in an Web environment. Though still relatively new, both are mature enough and robust enough to be used for large text collections immediately. However the current technologies do have limits. Current implementations are not robust enough to adequately render very complex mathematics, and major implementation variations from vendor to vendor remain. The communities involved need to continue the process of developing and implementing standards that improve the reliability and functionality of digital library systems.

---

[1] Steven J. DeRose, David G. Durand, Elli Mylonas, & Allen H. Renear, "What is Text, Really?" *The Journal of Computer Documentation (ACM SIGDOC)* 21, no. 3 (August 1997): 1 -- 25. (As reprinted from *Journal of Computing in Higher Education* 1, no. 2 (Winter1990): 3 - 26.) See also commentaries that follow on pages 26 – 44.

[2] See:
UIUC DLIB TestSuite Project Home Page. Online. URL <http://dli.grainger.uiuc.edu/> [10 January 2000].

Bruce Schatz, William H. Mischo, Timothy W. Cole, Joseph Hardin, Ann Bishop, and Hsinchun Chen, "Federating Diverse Collections of Scientific Literature" IEEE Computer 29: no. 5, 1996, pp 28-37. Also online. URL <http://www.computer.org/pubs/computer/dli/r50028/r50028.htm.> [10 January 2000].

Bruce Schatz, William H. Mischo, Timothy W. Cole, Ann Bishop, Susan Harum, Eric Johnson, Laura Neumann, and Hsinchun Chen, "Federated Search of Scientific Literature: A Retrospective on the Illinois Digital Library Project," IEEE Computer 32: no. 2, February 1999, pp. 51-60.

William H. Mischo and Timothy W. Cole, "Processing and Access Issues for Full-Text Journals," Successes and Failures of the Digital Library Initiative, Proceedings of the 35th Annual Graduate School of Library and Information Studies, University of Illinois at Urbana-Champaign, March 22--24, 1998 (in Press).

[3] W3 Consortium. *Extensible Markup Language*. Online. URL <http://www.w3.org/XML/> [10 January 2000].

[4] *Grainger Engineering Library Information Center Home Page.* Online. URL <http://www.library.uiuc.edu/grainger/> [10 January 2000].

[5] *DLIB TestSuite Project Home Page*. Online. URL <http://www.dlib.org/test-suite/index.html> [10 January 2000].

[6] W3 Consortium. *Cascading Style Sheets*. Online. URL <http://www.w3.org/Style/CSS/> [10 January 2000].

[7] W3 Consortium. *Extensible Stylesheet Language Transformations*. Online. URL <http://www.w3.org/TR/xslt> [10 January 2000].

[8] W3 Consortium. *XLink Working Draft.* Online. URL <http://www.w3.org/TR/1998/WD-xlink> [10 January 2000].

[9]W3 Consortium *XSchema Working Drafts.* Online. URL <http://www.w3.org/TR/xmlschema-1> and <http://www.w3.org/TR/xmlschema-2> [10 January 2000].

[10] W3 Consortium. *Resource Description Framework*. Online. URL <http://www.w3.org/RDF/> [10 January 2000].

[11] *Dublin Core Metadata Initiative.* Online. URL <http://purl.org/DC/> [10 January 2000].

[12] E.g., see Microsoft, *XSL Working Draft Conformance Notes*. Online. URL<http://msdn.microsoft.com/library/psdk/xmlsdk/xslp269f.htm> [10 January 2000].

[13] W3 Consortium. *Math Home Page.* Online. URL <http://www.w3.org/Math/> [10 January 2000].

[14] *STIX Project Home Page.* Online. URL<http://www.ams.org/STIX/> [10 January 2000].