Presentation of XML content using WYSIWYG templates

by

Arif-ur-Rahman Joarder

Submitted to the graduate faculty of the Electrical Engineering and Computer Science department and the Graduate Faculty of the University of Kansas School of Engineering in partial fulfillment of the requirements for the degree of Master of Science.

Thesis Committee:

Dr. Jim Miller (Chair)

Dr. Hossein Saiedian

Dr. Perry Alexander

Date Defended: December 19, 2008

The Thesis Committee for Arif-ur-Rahman Joarder certifies
that this is the approved version of the following thesis:

Presentation of XML content using WYSIWYG templates

Thesis Committee:

Chairperson

Date Approved

Abstract

XML has gained worldwide popularity for its ability to represent very general structured content. It is a platform independent format that has been successfully used for both media and more traditional textual data. However, the presentation of XML content has been an area of research ever since its introduction. We discuss the state of the art in XML content presentation, and then describe a new method for user designed transformation using WYSIWYG templates in HTML, that will allow users to design their own XML presentation format.

Acknowledgements

I would like to thank my research advisor, Dr. Jim Miller for his continuous supervision and assistance to help me complete this thesis. It was a pleasant experience working with him. I would also like to thank my family, especially my parents and wife for their support and encouragement.

Table of Contents

List of Figures

Chapter 1

Introduction

In recent years XML has been gaining in popularity as a method for delivery of data and content. One of the primary advantages of XML has been separation of content from how that content is rendered. Nevertheless, rendering the content is important, and it has received considerable attention. Since XML documents do not carry information about how to display the data, some form of transformation needs to be done to present it. XSL is the technology of choice for most generic XML document transformations.

The **eXtensible Stylesheet Language** (**XSL**) is a family of transformation languages which allows one to describe how data encoded in XML should be formatted or transformed. XSL is designed to be data driven and is a language for expressing style sheets. An XSL style sheet is a file that describes how to display an XML document of a given type.

XSL also includes a transformation language for XML documents called **XSLT**. Originally intended to perform complex styling operations like the generation of tables of contents and indexes, it is now used as a general purpose XML processing language. XSLT is thus widely used for generating HTML web pages from XML data. Figure 1 shows an example of XML data and XSL style sheet description for rendering the data for presentation.

In order to view XML data we need an XML transformation mechanism to convert the XML data into a readable format. XSLT has been popular for XML transformations and content generation. However, the challenge is to provide authoring tools to help authors design adaptable documents by authoring transformation sheets easily without knowing any technical details.

```
<instruction>

  <title>The Dot Product</title>

<description>
The dot product is the product of
the length of the two vectors and
the cosine of the angle between
them. Note that the dot product can
be positive or negative.
</description>

</instruction>
```

A XML Example

```
...
<xsl:template match="title">
  <fo:block font-weight="bold">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="description">
  <fo:block background-
color="blue">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
...
```

A partial XSL style sheet for the XML on the left

Figure 1: XSL style sheet example

Various tools have been developed to generate XSLT style sheets by providing visual design interfaces. However, these tools still require too much technical knowledge for an average user in order to support the complex capabilities of the XSLT language.

This thesis describes the design and development of a new method for user-centric design for document assembly that will allow users to design presentation of XML data for delivery on a web page. First a database was designed to store the content presentation information as WYSIWYG templates in HTML. Second, a user interface was developed to design the HTML templates and store the templates in the database. Finally, a document assembler was developed to combine the presentation information in the template with XML content.

Chapter 2

Related Work

XML data transformation and document assembly have been the focus of several research efforts. Most of the work has been on enhancing the use of existing technologies such as XSLT style sheets by automating style sheet generation. Style sheets contain transformation and presentation information for the XML content. Other efforts have focused on dynamic user interfaces similar to this work. We discuss here the major results from this prior work and indicate how our work compares.

XSLT is a functional language especially designed for XML document transformation. As indicated in figure 2, XSLT program or transformation sheet consists of transformation rules (templates) associated with patterns. When a rule pattern matches the source document being processed, the corresponding rule is instantiated to transform that portion of the source document.

Most of the currently available authoring systems provide ways to edit XML documents through lower-level text representations or at most through an enhanced representation such as a graphical tree. Some authoring tools give the user the ability to attach style to XML elements. This association simplifies the authoring of a document by making the

XML content more accessible to the user through the graphical interface. Even with these aids however, style sheets are of very limited help when attempting to design complex presentations.

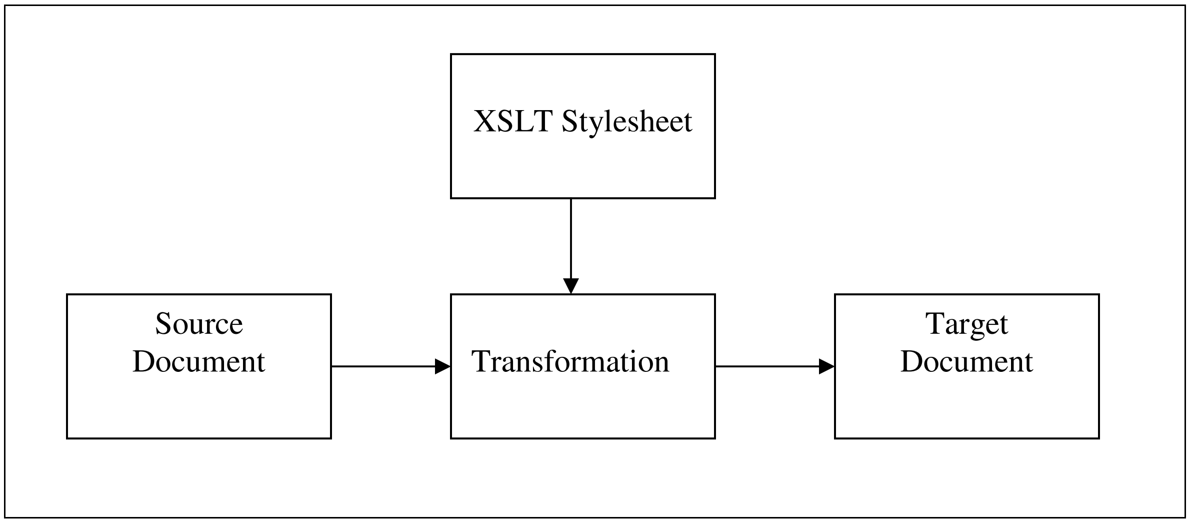


Figure 2: XSLT Transformation

Incremental Transformation with *incXSLT*

Designing XML content and transformation sheets is an inefficient, tedious and error prone process. Villard and Layaïda (2001) suggest an incremental transformation processor *incXSLT* as a better alternative to help in the design of both the content and the transformation sheets. Figure 3 shows the incremental transformation process. The source document is broken down into components with separate transformation rules for each component. If there is any change in the source document, only the rules corresponding to the affected components of the document will need to be executed.

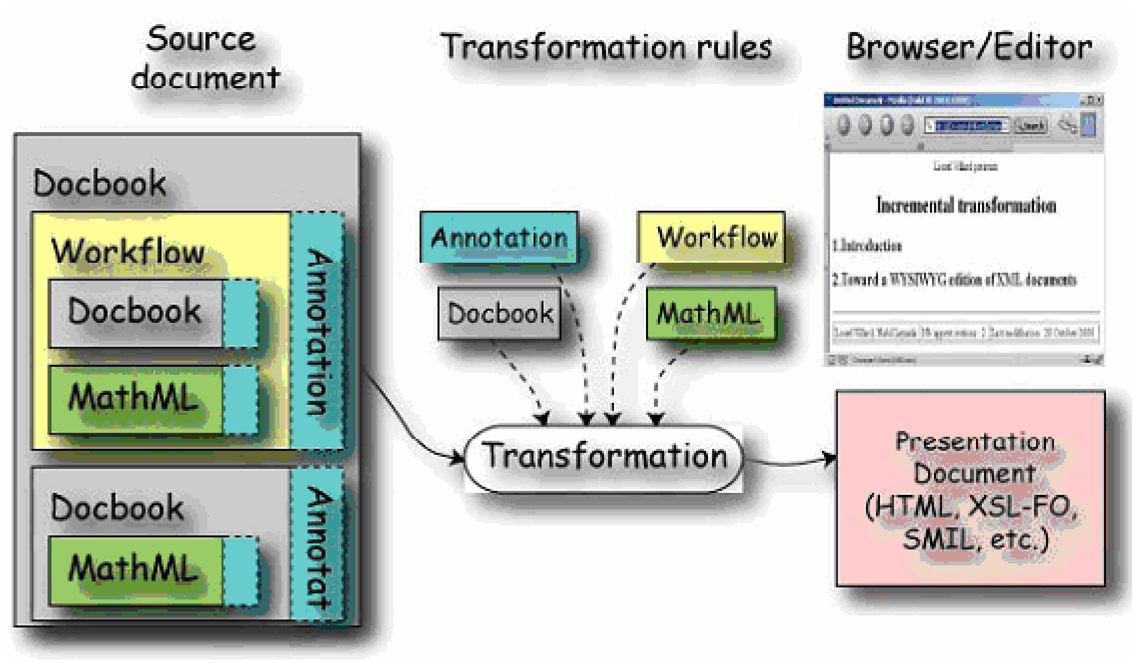


Figure 3: Incremental Transformation (Villard and Layaïda 2001)

In *incXSLT*, Villard and Layaïda (2001) proposed some modification to standard XSL transformations by incrementally transforming only the changed parts of XML documents to display HTML. Unfortunately this only represents an enhancement to an existing transformation process by XSL style sheets and so does not eliminate the need for a transformation description file.

Transformation by Direct Manipulation

Villard (2001) improved upon his incremental transformation model by demonstrating how incremental transformation can be achieved through direct manipulation using a photo album example; an XML description of which is shown in Figure 4. This model represents a set of photos gathered in albums. Each album is composed of meta-data gathered inside the header element and a list of photos. Meta-data are the title, the date of creation and the author of the album. A photo is characterized by the "src" attribute that references a file name containing photo data.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<album>
        <header>
                <title>Vacancy in Corsica</title>
                <date>June 2000</date>
                <author>Lionel Villard</author>
        </header>
        <photo src="Corse/p1.gif" speed="11" aperture="3">
                <author>Lionel Villard</author>
                <location>In the ferry boat</location>
                <comment>
                <para>First contact with the corse isle</para>
                </comment>
        </photo>
        <photo src="Corse/p2.gif" speed="auto" aperture="auto">
                <comment src="Corse/p2_comment.mp3"/>
        </photo>
</album>
```

Figure 4: An instance of photo album model (Villard 2001)

The authoring of transformation sheets can be interactively achieved only if results of transformation modifications are visualized instantaneously. Batch transformation is resource intensive and therefore is not suitable for interactive authoring. Villard (2001) suggests updating transformation results by using an incremental version of the transformation process. The core of this processor is the internal representation of the transformation execution: the execution flow tree. The execution flow tree structure is composed of execution nodes.

Execution nodes of type flow (apply-templates, for-each, if, etc.) contain the value (as hierarchical links) of their associated expression. For instance, apply-templates execution nodes have template nodes as direct children. Template nodes, and only these nodes, have links to the source nodes. Therefore, from apply templates nodes we can retrieve source nodes that compose the context node list. Producer nodes (value-of instruction and literal elements such as the Region element) contain data related to the target tree. This data will be used to restore the target context.

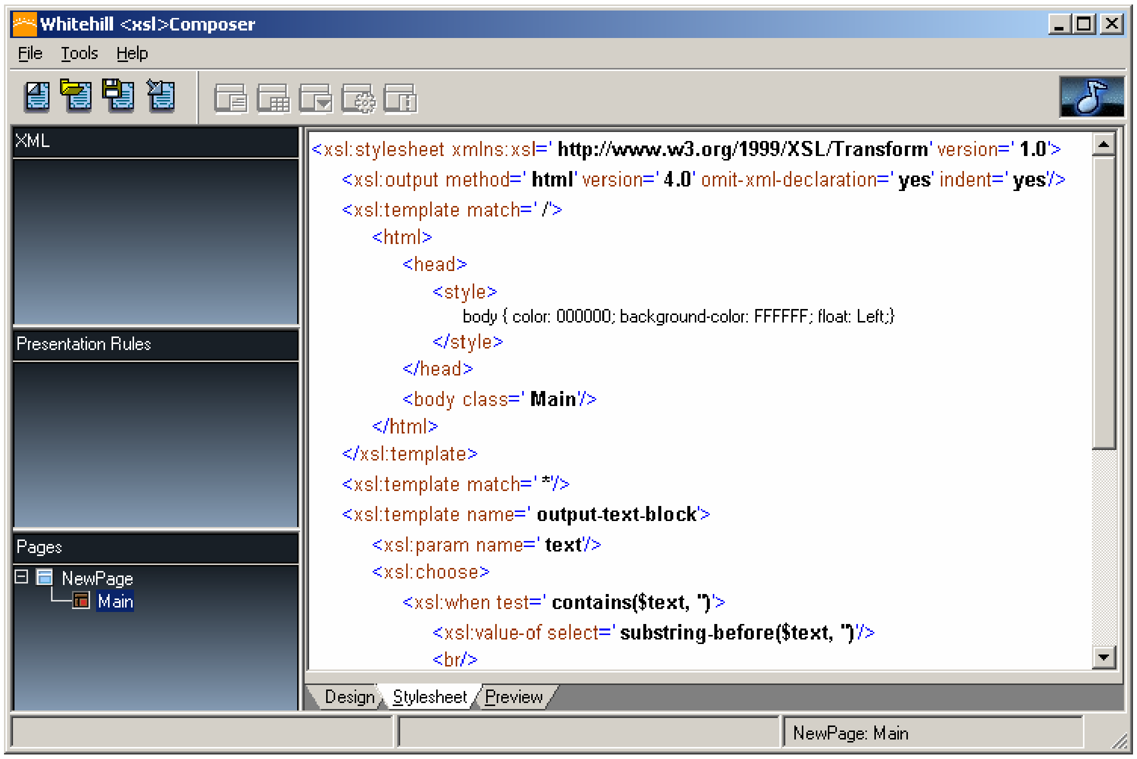
As illustrated in the use case study, the author can edit transformations directly in a target view. The authoring is achieved using classical multimedia editing operations. For example, the author can add a video object, modify the content of a text item, change the style of an object, remove a temporal relation, etc. The result of these authoring operations is a set of transformation rules.

Whitehill XSL Composer

The **Whitehill XSL Composer** allows authoring of XSLT transformations by direct manipulation using a WYSIWIG graphical user interface. However, this tool has many shortcomings: in particular the transformation process is executed from scratch after each modification of the transformation sheet. Thus, it is not as efficient as Villard's incremental transformation method *incXSLT*.

Figure 5 shows a snapshot of the Whitehill XSL Composer. The main window is composed of four parts. The first part shows hierarchically the opened XML document. The second part shows the list of templates. The third part shows applied templates and the last part shows the final presentation and the XSLT code. Dragging and dropping XML elements in the final presentation achieves the creation of new templates.

This is essentially a "MS FrontPage" or "Macromedia Dreamweaver" for XSL style sheets. The editor has three views - Design, Style sheet and Preview. The Design pane allows user to visually enter style sheet rules in a WYSIWYG editor. The Style sheet pane shows the XSL code generated by the program. And the Preview pane shows the final result of transformation.



```
<xsl:stylesheet xmlns:xsl=' http://www.w3.org/1999/XSL/Transform' version=' 1.0'>
    <xsl:output method=' html' version=' 4.0' omit-xml-declaration=' yes' indent=' yes'/>
    <xsl:template match=' /'>
        <html>
            <head>
                <style>
                    body { color: 000000; background-color: FFFFFF; float: Left;}
                </style>
            </head>
            <body class=' Main'/>
        </html>
    </xsl:template>
    <xsl:template match=' *'/>
    <xsl:template name=' output-text-block'>
        <xsl:param name=' text'/>
        <xsl:choose>
            <xsl:when test=' contains($text, '')'>
                <xsl:value-of select=' substring-before($text, '')'/>
                <br/>
```

Figure 5: Whitehill XSLComposer

The authoring of transformation sheets is similar to the editing of presentation models and the connection with a database. Microsoft Office allows the creation of presentation models based on style sheets. The content of the model can be filled manually by the author or generated automatically from a relational database. The selection of data is made through several forms.

This tool does not allow nesting of templates. Templates are applied in sequence only. Therefore, it is not possible to create presentations for tables or lists.

Transformation by demonstration

Koyanagi et. al. (2000) presented a method for generating XSLT transformation sheets by demonstration. The source document and the target document are both HTML. The method relies on recording a user's interactions when he/she edits an HTML document. At the end of recording, transformation rules are generated from the history of the user's interactions. This method has the following drawbacks. First, the source document and the target document belong to the same document class. Second, as this method relies on demonstrational interfaces, it inherits its drawbacks. In particular, demonstrational interfaces do not provide a static representation of the program. Therefore, reuse, modification and revision of the program are not possible.

XML Active Transformation (eXAcT): Transforming Documents within Interactive Systems

Olivier Beaudoux (2005) proposed *XML active transformation* (eXAcT) for multi-source and multi-target transformation, source-to-target linking, and bidirectional linking. An *XML active transformation* defines how source DOM documents can be transformed into target DOM documents (Beaudoux 2005). The eXAcT approach differs from incremental versions of XSLT in three ways. First, the transformation process is based on observing the source and updating the target when needed by invoking *rule* methods. In the case of a bidirectional transformation, the process also observes the target document and updates the source by invoking *inverse rule* methods. Second, eXAcT specification is a DOM extension (it uses the IDL specification language) which can be implemented by using various object programming languages. Finally, fragments instantiated by transformations are created by using external tools adapted to their format (Beaudoux 2005). eXAcT updates the target document only when the source document changes. (See figure 6)

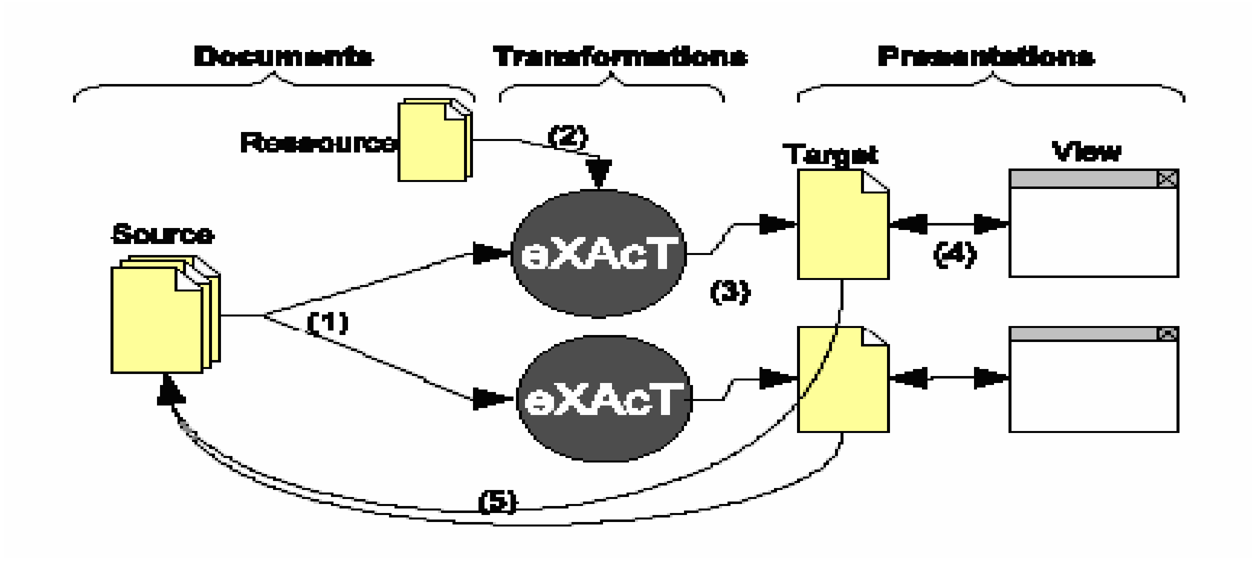


Figure 6: eXAcT Transformation model

A Visual Approach to Define XML to Formatting Objects (FO) Transformations

Canfora and Cerulo (2002) presented Content Container Style Language (CCSL) to define formatting and styling information for XSLT. XSLT is a formal language. It is hard for an inexperienced user to express in a linear textual form like programming language. Canfora and Cerulo (2002) defined a “formatting style oriented” language, named CCSL in which formatting and styling aspects are clearly identified. They also introduced a visual oriented language, named Visual Content Container Style Language (VCCSL) that eases the definition of the format and visualization aspects of XML documents by means of a graphical content/container metaphor.

The CCSL is a textual language based on XSLT, XML Path Language (XPath), and XSL Formatting Objects (FO). XPath is a query language for selecting nodes from an XML document. XSL FO is an XML-based markup language that lets you specify in great detail the pagination, layout, and styling information that will be applied to your content. The XSL FO markup is quite complex. It is also verbose; virtually the only practical way to produce an XSL FO file is to use XSLT to produce a source document. With CCSL, selected features of these different languages are combined to highlight the content and container metaphor. Figure 7 shows how CCSL relates to the other XML components and how the flow of information proceeds along the various steps of the elaboration that transforms an XML document into XSL FO.

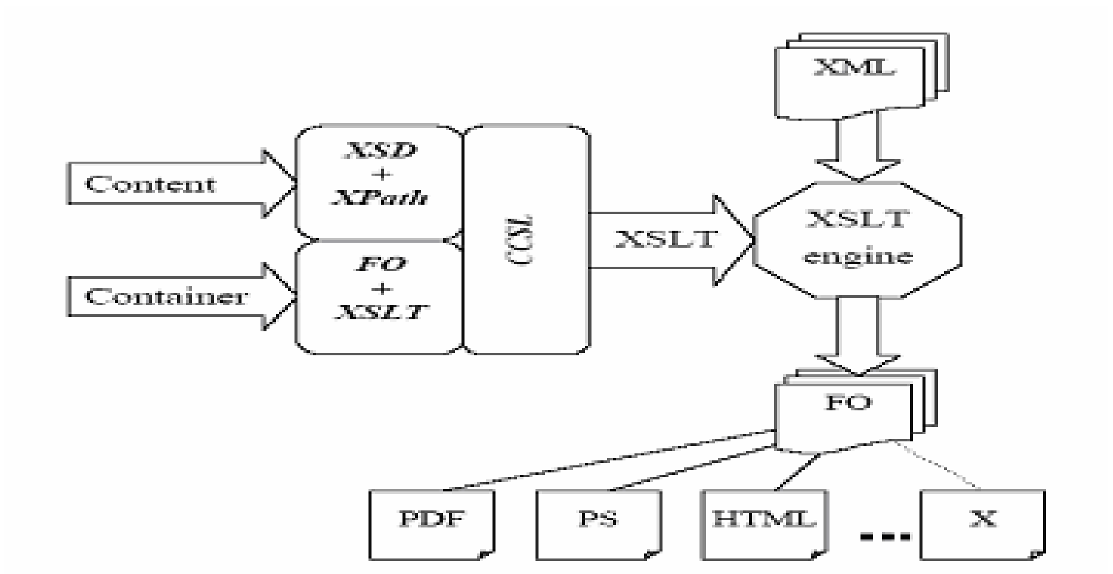


Figure 7: CCSL Transformation (Canfora and Cerulo 2002)

VCCSL is a visual language constructed on top of CCSL. CCSL is hard to use even though it introduces a new paradigm specifically designed for formatting purposes. VCCSL is the start point for a user that aims to define the formatting style of a class of XML documents. As opposed to XSLT, in which the formatting parameters and the logic for extracting information are expressed in separate statements, VCCSL uses the multidimensionality of visual languages to merge these two aspects in a single and homogeneous view (Canfora and Cerulo 2002). It uses a form-like approach based on nested rectangles (container) with attached labels (content). This notation is related to the tree structure of containers and is very intuitive from the user point of view. A similar notation has been successfully used in the context of visual query languages for XML documents. Figure 8 shows the mapping between CCSL and VCCSL notations.

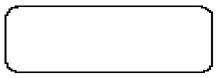
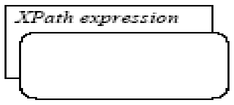
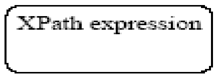
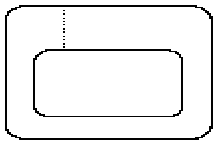
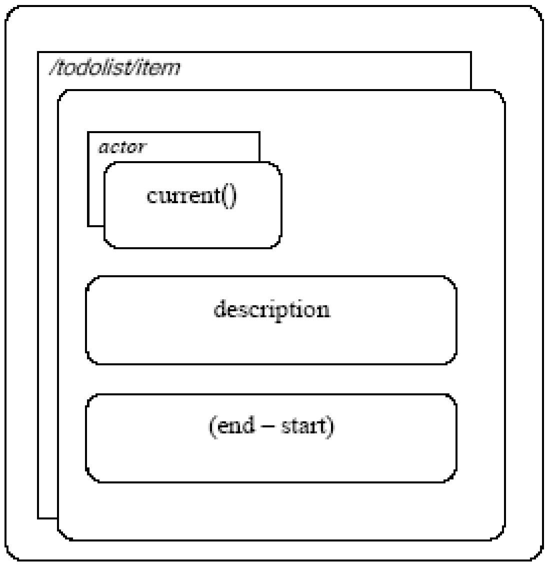
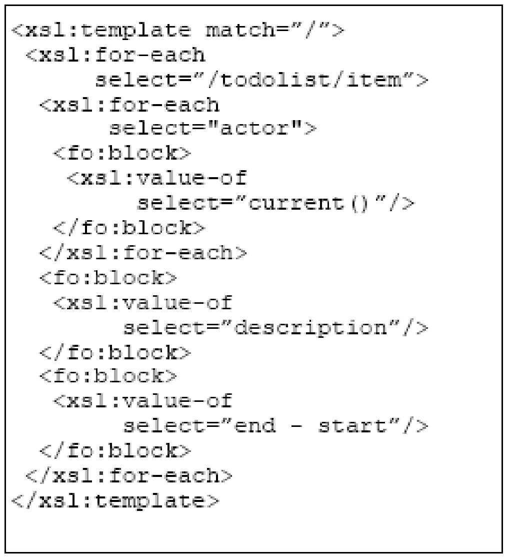
CCSL notation	VCCSL notation
container k	
content c	XPath expression
iteration relation $\mathcal{I}$	
nesting relation $\mathcal{N}$ $x \in \mathcal{C}$	
nesting relation $\mathcal{N}$ $x \in \mathcal{K}$	

Figure 8: Mapping between CCSL and VCCSL notations

A container is graphically represented as a rounded box. Content is represented by a textual label in which is inserted the XPath expression of the content. The relations between content and containers are graphically represented by the relative position of each graphical object. The iteration relation, for example, is graphically represented by placing on the container box header the content label that takes part in the relation. By contrast, a set like diagram representation has been used for the nesting relation. If a container is related to a piece of content (i.e. it contains the content) then the content graphical representation (text label) is placed within the container graphical representation (rounded box).

Recursively, if a container contains another container, then the container graphical representation (rounded box) is placed within a further container graphical representation (rounded box). VCCSL notation can be combined together to form a complete XML → FO transformation definition. Figure 9 shows its CCSL counterpart.





```
<xsl:template match="/">
 <xsl:for-each
      select="/todolist/item">
  <xsl:for-each
       select="actor">
   <fo:block>
    <xsl:value-of
         select="current()"/>
   </fo:block>
  </xsl:for-each>
  <fo:block>
   <xsl:value-of
        select="description"/>
  </fo:block>
  <fo:block>
   <xsl:value-of
        select="end - start"/>
  </fo:block>
 </xsl:for-each>
</xsl:template>
```

A VCCSL example

The CCSL counterpart of the example on the left

Figure 9: CCSL counterpart of a VCCSL example

XML Spy

XML Spy is a commercial IDE for supporting the editing of XML documents. It automatically recognizes most XML name spaces, including XSL and FO. The tool comes with a plug-in supporting XSLT design. This plug-in, named XSLT Designer, allows a user to drag and drop XML data elements into the main design window through a WYSIWYG text editor interface. The resulting XSLT style sheet is automatically generated and can be previewed using a built-in browser. The main limitation of this tool is that it does not support effectively repeating XML elements. Thus it can not transform XML elements into lists and tables.

Style sheet Design in StyleVision

Altova StyleVision® 2008 utilizes a visual approach to designing style sheets for transforming XML and databases to HTML, PDF, Word/RTF etc. Users select components from a data source, drag them on to the design pane, and specify their presentation and layout properties in helper windows. StyleVision® can integrate data from multiple XML sources, or even combine data from an XML schema and a relational database. It also allows creation of designs to publish XML documents stored inside relational database columns. Each design simultaneously produces standards conformant XSLT 1.0/2.0 and XSL:FO style sheets and the corresponding output in HTML, PDF or Word/RTF. Since these style sheets are stored in a file system, they are more difficult to manage in a multi user preference based system where individual user preferences will be stored in separate style sheet files. Since StyleVision 2008 is not a server based tool, users preference changes will not be immediately reflected in displays.

It is not possible to use StyleVision® 2008 with an instance XML file only. You also need a Schema or DTD upon which this instance file is based. Although it provides a WYSIWYG graphical user interface, users have to check the preview pane regularly to see the actual transformation output. Figure 10 shows a screen shot of StyleVision® 2008 editor.

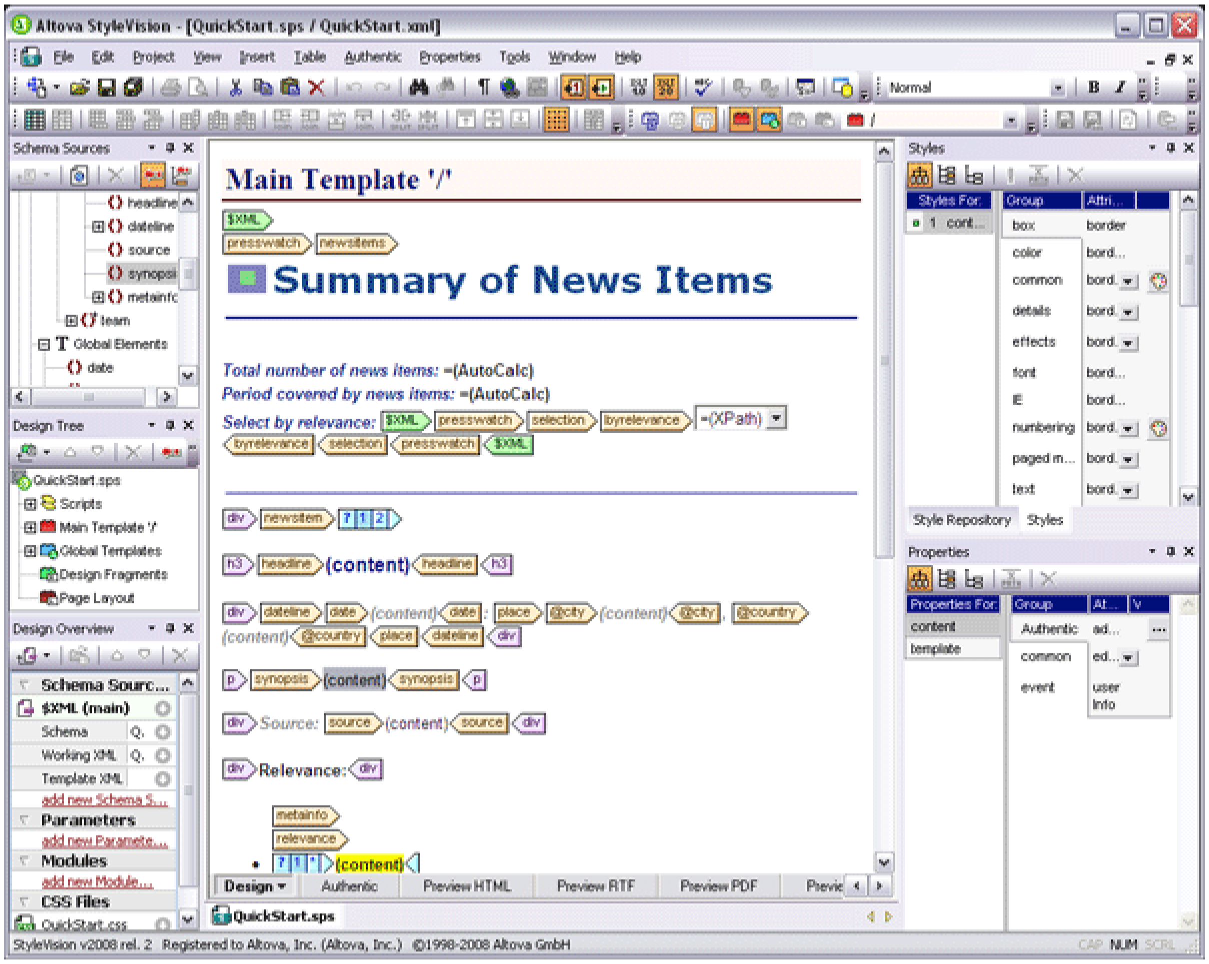

Altova StyleVision - [QuickStart.sps / QuickStart.xml]
File Edit Project View Insert Table Authentic Properties Tools Window Help
Schema Sources
Design Tree
Design Overview
Main Template '/'
Summary of News Items
Total number of news items: =(AutoCalc)
Period covered by news items: =(AutoCalc)
Select by relevance:
Source:
Relevance:
Design Authentic Preview HTML Preview RTF Preview PDF
QuickStart.sps
Styles
Style Repository Styles
Properties
StyleVision v2008 rel. 2 Registered to Altova, Inc. (Altova, Inc.) ©1998-2008 Altova GmbH

Figure 10: Altova StyleVision® 2008

Summary

In this chapter we have reviewed several transformation models for XML data, starting with incremental transformation incXLT by Villard and Layaïda (2001), transformation by demonstration by Koyanagi et. al. (2000), XML active transformation (eXAcT) by Olivier Beaudoux (2005), Content Container Style Language (CCSL) by Canfora and Cerulo (2002), and finally some of the commercially available tools such as Whitehill

XSL Composer, XML Spy and Altova StyleVision® 2008. All these transformation models are useful for certain purposes. However, one of the main limitations of these models is that they all require a solid understanding of the XML language as well as the specific XML data structure that needs to be transformed. Some even require additional technical knowledge that regular non-technical users will not have.

The major goal of this project was to implement a transformation model that anyone could use without any technical knowledge of XML. While most XML transformation models focus on general XML transformations, the novel model developed as a part of this thesis effort was designed specifically for data presentation.

Chapter 3

The Transformation Process

The purpose of this research was to develop a user friendly transformation system for XML data presentation within an HTML browser that does not require any technical knowledge of the XML data structure or any sort of programming such as that required to use XSLT style sheets. Our method does not generate any XSLT style sheet. Instead it generates HTML Templates with placeholders for data. These data placeholders are later described as tags. At the point of viewing, a document assembler will extract the data from the XML data structure and replace the tags with data.

The transformation process has two parts:

1. Generate HTML Templates from a sample XML data file.
2. Generate HTML documents from the HTML templates and a collection of XML documents.

Assumptions

In order to transform data in an XML structure into an HTML page, the following assumptions were made:

1. A complete example of an XML document containing all possible elements has to be available to allow the system to discover all possible tag nestings so that a simple user interface can be generated, allowing the user to create HTML templates for transformation of XML documents.
2. All displayable content is in the terminal nodes of the XML tree.
3. Non-terminal nodes contain only organizational structure.

Generating HTML Templates from XML

Step 1: From a sample XML document, a general tree structure is generated with the root element the root node of the tree.

Step 2: From the tree structure, a set of unique paths from the root node to all non-terminal nodes with at least one terminal node are identified. These unique paths are given textual representative names and called "tags". The tags are generated by concatenating names of the nodes in the path starting with the root node in the format [ELEMENT].[CHILD_ELEMENT].[GRAND_CHILD_ELEMENT].

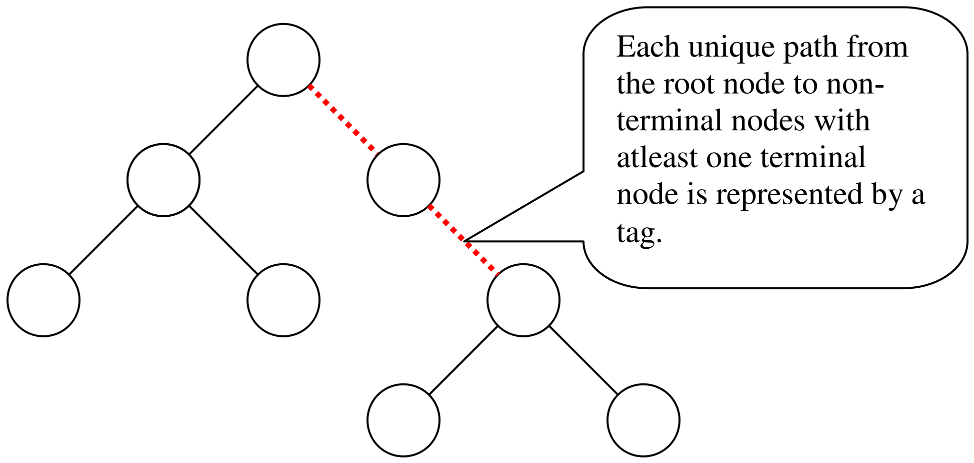

Each unique path from the root node to non-terminal nodes with atleast one terminal node is represented by a tag.

Figure 11: Example of a unique path represented by a tag

Step 3: Using an HTML editor, the tags are placed on a document canvas. Formatting of the document including the tags is done using the Graphical User Interface of the editor. For the purpose of this project a commonly available WYSIWYG HTML editor called TinyMCE was used. TinyMCE is an open source editor freely available on the World Wide Web and works on all major browsers. Figure 12 shows a snapshot of this editor.

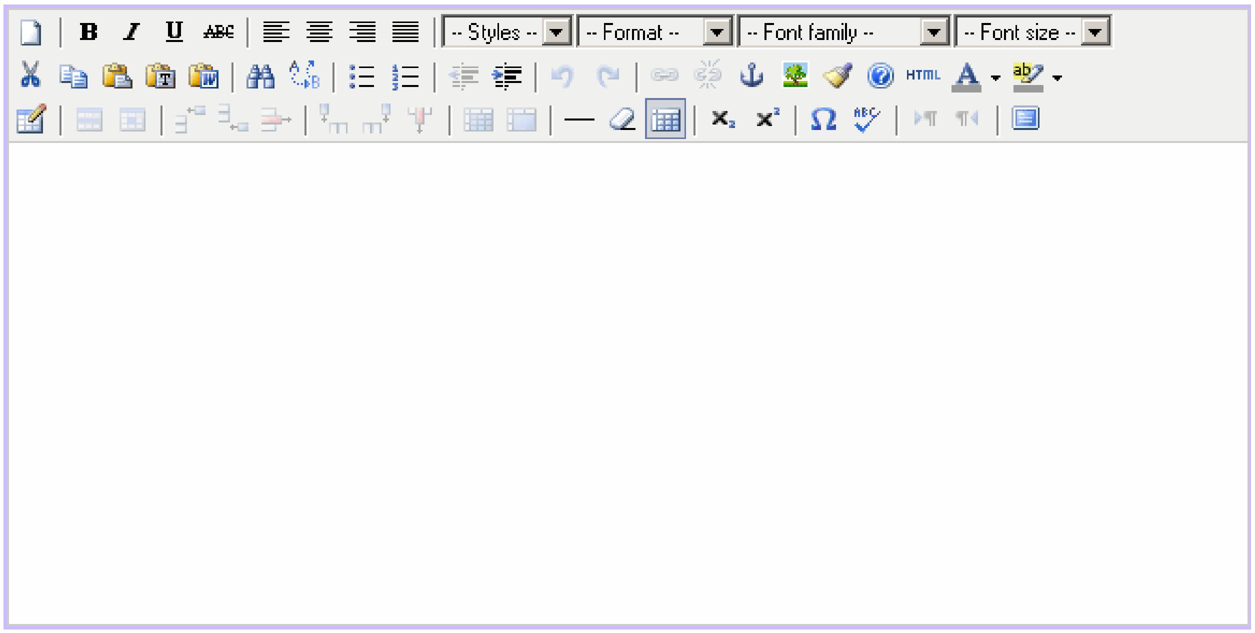
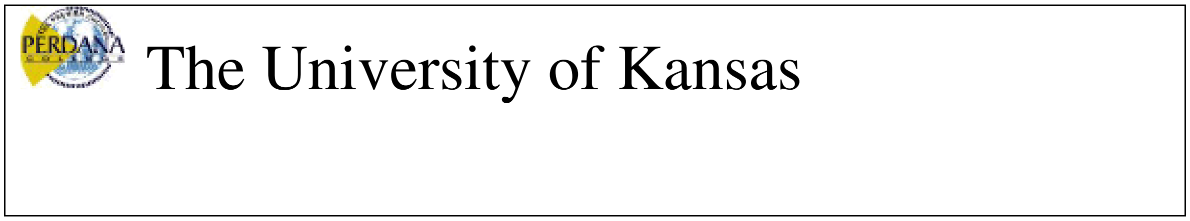


Figure 12: View of the TinyMCE editor

Step 4: The editor saves the HTML code of the edited document template in a database. We call this the "Main Template" for transforming the XML data structure.

Step 5: Actual XML documents are passed through a document assembler which uses the template and replaces tags in the code with corresponding data elements from the XML document.



The University of Kansas

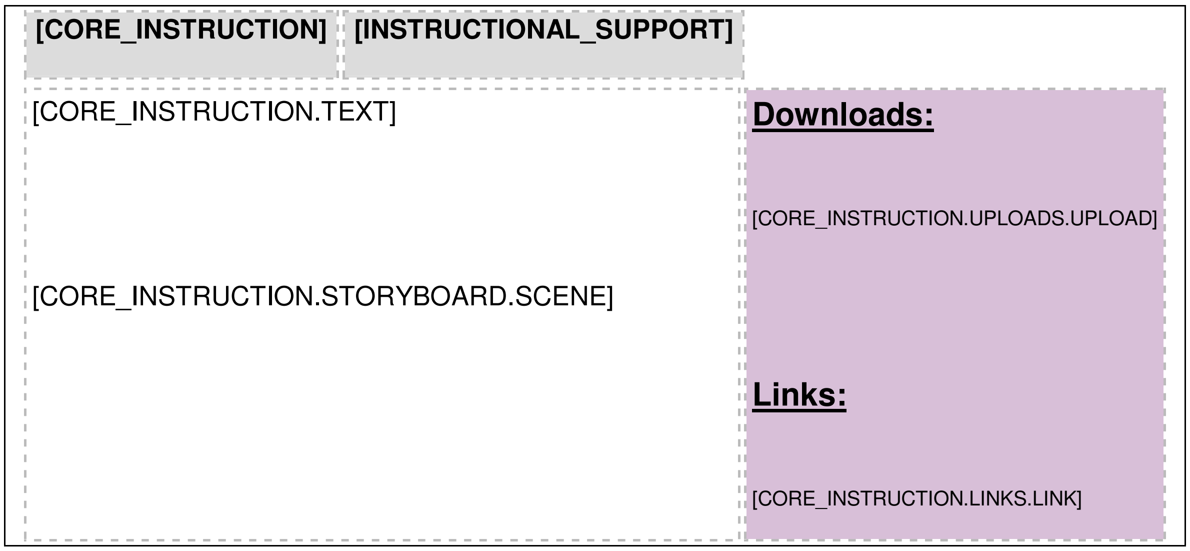

[CORE_INSTRUCTION]
[INSTRUCTIONAL_SUPPORT]
[CORE_INSTRUCTION.TEXT]
[CORE_INSTRUCTION.STORYBOARD.SCENE]
Downloads:
[CORE_INSTRUCTION.UPLOADS.UPLOAD]
Links:
[CORE_INSTRUCTION.LINKS.LINK]

Figure 13: A sample HTML Template showing Tags for data insertion

Step 6: A separate set of optional templates can be used for transforming subtrees under the non-terminal nodes containing at least one terminal node, at the end of the unique paths identified in Step 2. We will call these the “Subtree Templates”. These templates are optional because if they are not rendered on a template, they can still be displayed in a default format. Figure 14 identifies a subtree for our “Subtree Template” on an XML tree structure.

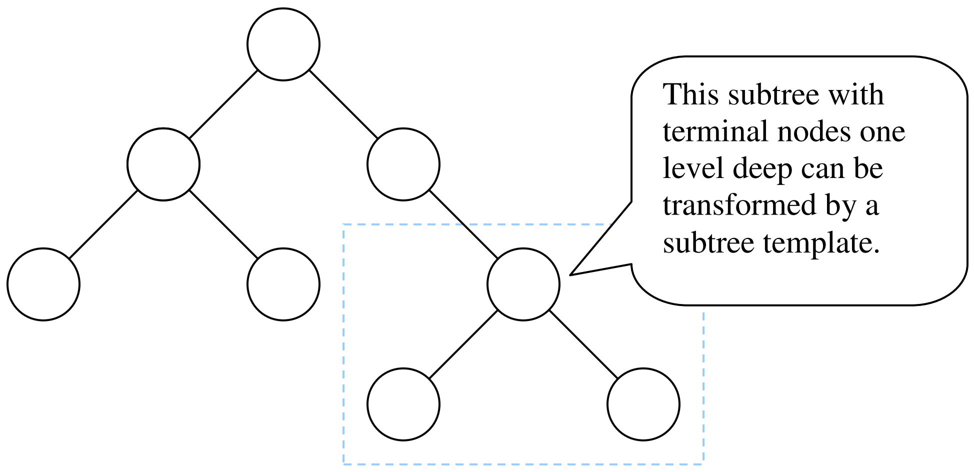

This subtree with terminal nodes one level deep can be transformed by a subtree template.

Figure 14: Subtrees transformed by subtree templates

Step 7: Each of the terminal nodes in the subtrees identified in Step 6 are given names which will be used as tags for the Subtree Templates just as the tags used on the Main Template.

Step 8: Just like the Main Template, the Subtree Templates will be stored in a database. The Subtree Templates can be retrieved back from the database by the tag name used to identify the path from the root node to the subtree represented by the template.

Generating HTML document from HTML templates and XML data

The document assembler introduced in step 5 above merges the XML data with the HTML templates stored in the database to generate viewable content in HTML. This operation proceeds as follows:

Step 1: Read the Main Template from database.

Step 2: Read an XML document to be rendered and generate a general tree structure with the root element as the root node of the tree.

Step 3: From the tree structure, identify the list of unique paths as was done for step 2 on page 26.

Step 4: For each tag in the list identified in Step 3, search for the tag in the main HTML template.

a. If there is a Subtree Template stored in the database by the name of the tag
    i. load the Subtree Template
    ii. identify the corresponding element from the XML data structure
        1. identify child elements
        2. for each child element search and replace corresponding tags in the Subtree Template with element data
    iii. replace the first matching tag in main template by transformed subtree template
b. If no Subtree Template is stored in database by the name of the tag
    i. identify the corresponding element from the XML data structure
    ii. replace the first matching tag in main template by element data

Step 5: Remove any unused tags in the Main Template.

Step 6: Send the Main Template which is now complete HTML code, to the browser for display. Figure 15 shows transformation of an XML document into an HTML document using HTML templates following the above steps.

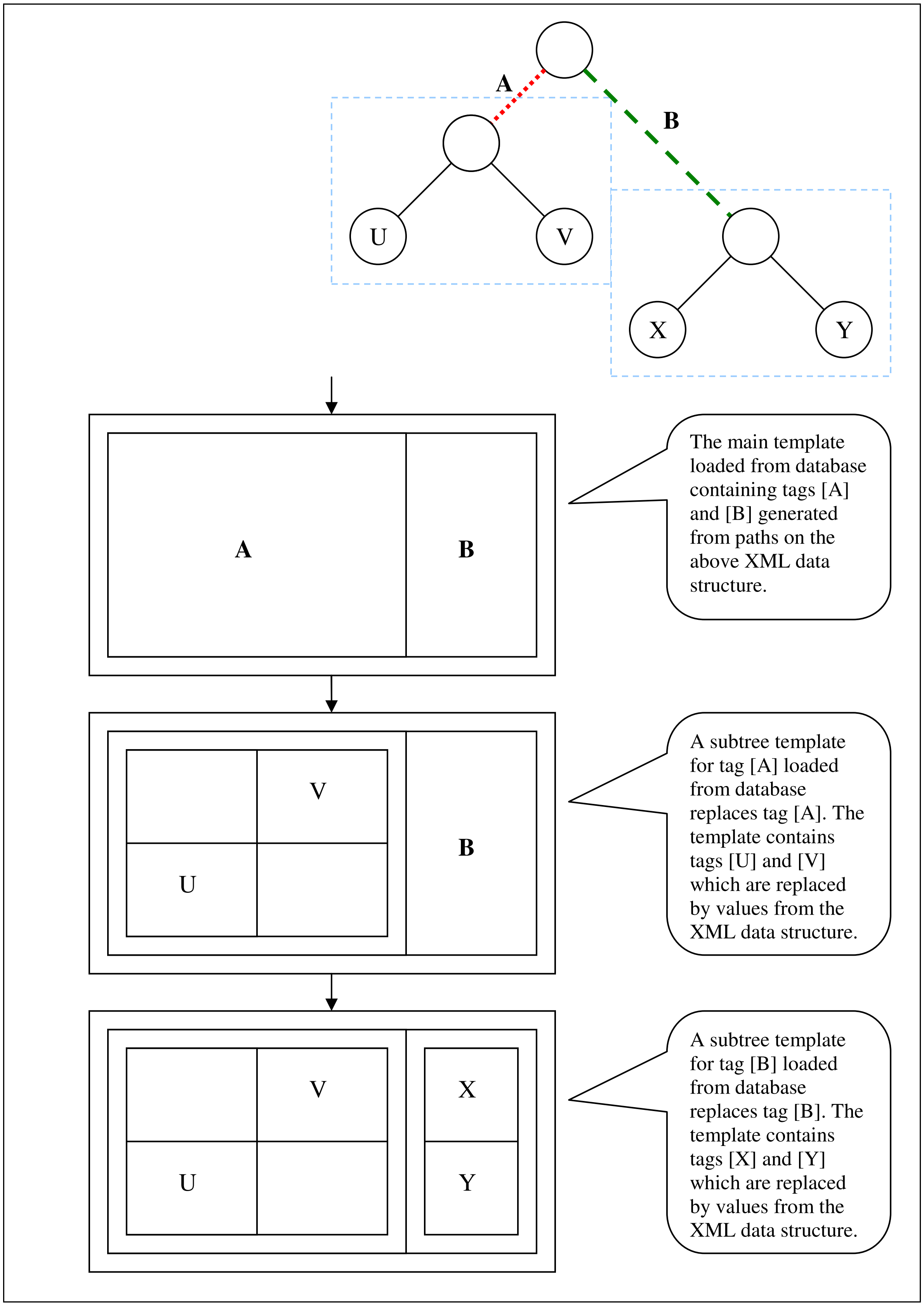

A
B
U
V
X
Y
A
B
The main template loaded from database containing tags [A] and [B] generated from paths on the above XML data structure.
V
U
B
A subtree template for tag [A] loaded from database replaces tag [A]. The template contains tags [U] and [V] which are replaced by values from the XML data structure.
V
U
X
Y
A subtree template for tag [B] loaded from database replaces tag [B]. The template contains tags [X] and [Y] which are replaced by values from the XML data structure.

Figure 15: Contruction of an HTML document from XML data and HTML templates

Unlike XSL style sheets which are stored in files, we store our HTML templates in a database. This allows us to design and make changes to the design on the fly and generate documents dynamically. Dynamic document generation allows us to have multiple presentation designs catered to different kinds of users. Figure 16 below shows an overview of the transformation process.

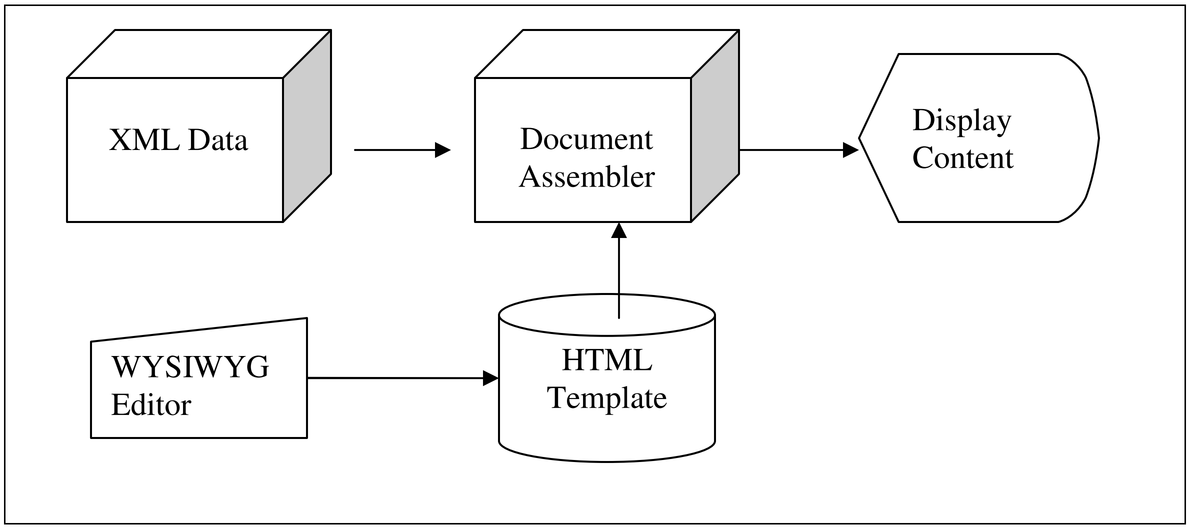


Figure 16: Process flow diagram of the proposed XML data presentation system

Chapter 4

Testing and Evaluation

In order to understand the effectiveness of the proposed model, a project was implemented using the techniques of this model, to deliver content for e-Learning. Meyen and Miller (2007) at the University of Kansas have developed an XML based data model for storing course content that is meant to be re-usable. These re-usable content modules are called **Re-Usable Learning Objects (RLO)**. Many of these RLOs are put together to develop a course. Instructors from different institutions or within the same institution may collaborate and share these RLOs to prepare their online courses. However, since different instructors and institutions have different presentation requirements, we must be able to allow the users to configure the presentation of the RLO contents in XML.

The structure of a sample RLO is shown in Appendix A. An RLO consists of several sections of a lesson such as core instruction, supplementary resources, references etc. The experiment showed that the proposed model can transform an XML structure of an RLO properly in most cases.

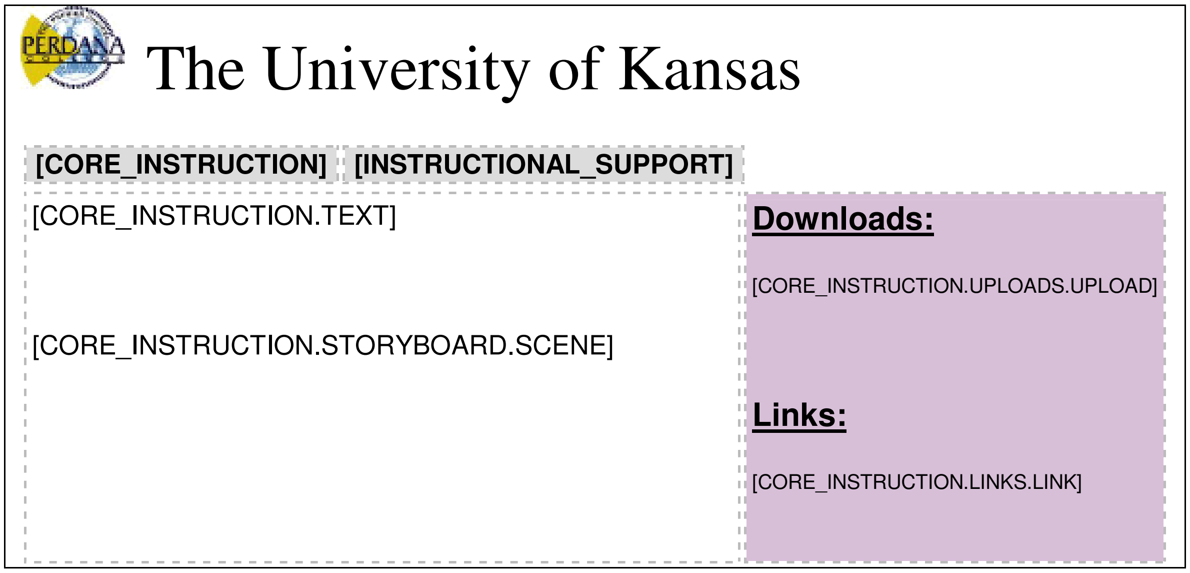
Figure 18 shows an HTML template for an RLO and Figure 19 shows an HTML page generated after transformation of an XML document using the template. In the RLO several tags were used such as [CORE_INSTRUCTION.TEXT], [CORE_INSTRUCTION.STORYBOARD.SCENE], [CORE_INSTRUCTION.LINKS.LINK] and [CORE_INSTRUCTION.UPLOADS.UPLOAD]. These tags have been extracted from the XML data structure. The [CORE_INSTRUCTION.TEXT] tag displays all headings and contents within the <TEXT> elements under <CORE_INSTRUCTION>. The [CORE_INSTRUCTION.STORYBOARD.SCENE] tag displays images, headings and contents within the <SCENE> elements under <STORYBOARD>.

The [CORE_INSTRUCTION.UPLOADS.UPLOAD] tag displays links to files for download, their titles and descriptions within the <UPLOAD> elements. The [CORE_INSTRUCTION.LINKS.LINK] tag displays links to other web pages, their titles and descriptions within the <LINK> elements an XML description of which looks like the following in Figure 17:

```
<LINKS>
<LINK>
<URL>http://people.eecs.ku.edu/~miller/cryph</URL>
<TITLE>The cryph software utilities</TITLE>
<DESCRIPTION>This set of pages reviews all the code available to
support vector operations.
</DESCRIPTION>
</LINK>
<LINK>
<URL>http://people.eecs.ku.edu/~miller/cryph</URL>
<TITLE>The cryph software utilities 2</TITLE>
<DESCRIPTION>This set of pages reviews all the code available to
support vector operations.
</DESCRIPTION>
</LINK>
</LINKS>
```

Figure 17: XML description of <LINKS> element in the RLO of Appendix A

To transform the <LINK> element to an actual HTML URL that a browser can display, a subtree template was used. Using the TinyMCE editors' user interface, a link was added setting the url to [URL] tag and title to [TITLE] tag. The editor generated the following HTML code for the subtree template: `<a href="[URL]">[TITLE]</a>`



The University of Kansas

[CORE_INSTRUCTION] [INSTRUCTIONAL_SUPPORT]

[CORE_INSTRUCTION.TEXT]

[CORE_INSTRUCTION.STORYBOARD.SCENE]

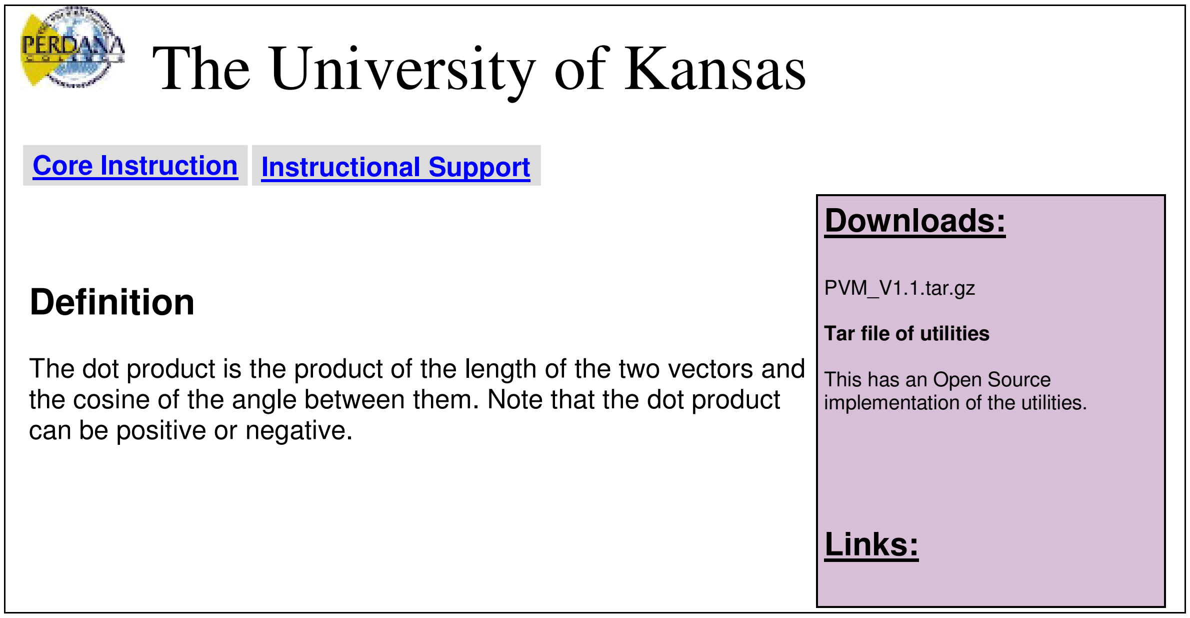
Downloads:

[CORE_INSTRUCTION.UPLOADS.UPLOAD]

Links:

[CORE_INSTRUCTION.LINKS.LINK]

Figure 18: HTML template of Core Instruction section of an RLO



The University of Kansas

Core Instruction Instructional Support

Definition

The dot product is the product of the length of the two vectors and the cosine of the angle between them. Note that the dot product can be positive or negative.

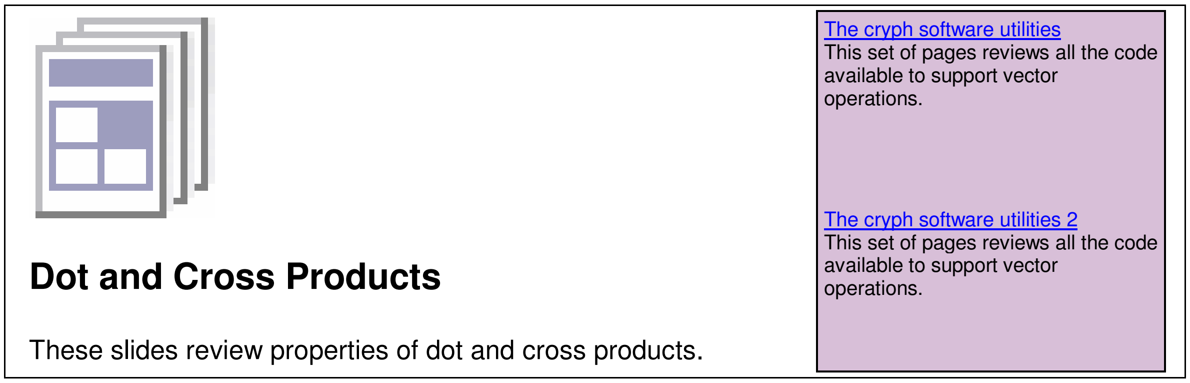
Downloads:

PVM_V1.1.tar.gz

Tar file of utilities

This has an Open Source implementation of the utilities.

Links:



Dot and Cross Products

These slides review properties of dot and cross products.

The cryph software utilities
This set of pages reviews all the code available to support vector operations.

The cryph software utilities 2
This set of pages reviews all the code available to support vector operations.

Figure 19: The Core Instruction page after transformation

Evaluation Results

There are several advantages to this proposed model. This model allows users to design transformation of content without any technical knowledge of XML, XSLT or HTML. Therefore, users will have greater control over their own content presentations. Since it is web-based and database driven, the publisher may also define presentation of certain sections of the data and leave the final presentation design to the user. Several users may also use the same content and render parts of the content differently. Use of a GUI editor that automatically generates the HTML template makes it easier to customize content presentation.

Elements containing data as well as child elements

One of the assumptions of the proposed model is that all displayable content will be in the terminal elements. The example in Figure 20 shows one case, where an element contains content as well as child elements where the content needs to be displayed, and the child elements need to be rendered:

```
<PARAGRAPH>
The dot product is the product of the length of the two
<DEFINITION>vectors</DEFINITION> and the
<DEFINITION>cosine</DEFINITION> of the angle between them.
</PARAGRAPH>
```

Figure 20: An XML element containing displayable content as well as child elements

The <PARAGRAPH> element contains display text with <DEFINITION> elements inserted in the text. The algorithm currently ignores the display text inside the <PARAGRAPH> element. It needs to be modified so that the text before the first <DEFINITION> element is displayed before the content of the first <DEFINITION> element, and similarly the text before the second <DEFINITION> element can displayed before the content of the second <DEFINITION> element. The last piece of display text after the last child element, need to be displayed at the end. This solution is yet supported in our system because the XML parsing library that was used to implement it does not preserve the internal structure of a node such as that illustrated by the <PARAGRAPH> element above.

Chapter 5

Conclusion

The objective of this research was to develop an XML data transformation and presentation system that requires little knowledge of the XML data structure itself, and little or no technical knowledge of HTML, unlike other existing methods such as XSL style sheets. Existing transformation models for XML data, by XSL style sheets require transformation description files that can not be changed on the fly and transformation can not be done dynamically based on user preferences. Multiple customized user preference based transformation descriptions will require multiple description files that are harder to manage compared to the proposed database based solution.

Incremental transformation incXLT by Villard and Layaïda (2001), transformation by demonstration by Koyanagi et. al. (2000), and XML active transformation (eXAcT) by Olivier Beaudoux (2005) have made some improvements on the generation of XSLT transformation description files but they have not managed to get rid of the requirement for the description file itself.

Some of the commercially available tools such as Whitehill XSL Composer, XML Spy and Altova StyleVision® 2008 require knowledge of the XML data structure which make them unusable or unfriendly for average users who just want the data to be rendered a certain way without knowing the actual structure of the data in XML.

In the proposed model designing transformation of an XML document has been made easy not just by presenting the XML structure to the user but by actually hiding the XML structure and providing a visual environment for users to describe rendering of parts of a document using HTML templates which can be edited by WYSIWYG Graphical User Interface based HTML editors.

Two types of HTML templates have been used to describe rendering of XML data structure. The Main Template contains description for overall presentation of the document on a page. The Subtree Templates help render components within the Main Template. Since the transformation templates are stored in a database and not style sheet files, this allows dividing the transformation description among several users, where each user may use different Main Template for presentation of same content to personalize according to his/her needs, and use common Subtree Templates for presenting parts of the document.

Appendix A

XML description of a sample RLO

```
<?xml version="1.0"?>
<INSTRUCTIONAL_UNIT ID="iu_1_1188954731_1">
  <META>
    <VERSION/>
    <TITLE>The Dot Product</TITLE>
    <DESCRIPTION>This IU talks about the mathematical and computational
definition of the dot product.</DESCRIPTION>
    <KEYWORDS>math, vectors</KEYWORDS>
    <REF_COUNT>1</REF_COUNT>
  </META>
  <CORE_INSTRUCTION>
    <TEXT>
      <HEADING>Definition</HEADING>
      <CONTENT>The dot product is the product of the length of the two
vectors and the cosine of the angle between them. Note that the dot
product can be positive or negative.</CONTENT>
    </TEXT>
    <STORYBOARD>
      <SCENE>
        <IMAGE ALT="" BORDER="" HEIGHT="" TYPE=""
WIDTH="">../../varsity_jsp/images/batch.gif</IMAGE>
        <HEADING>Dot and Cross Products</HEADING>
        <CONTENT>These slides review properties of dot and cross
products.</CONTENT>
      </SCENE>
    </STORYBOARD>
    <LINKS>
      <LINK>
        <URL>http://people.eecs.ku.edu/~miller/cryph</URL>
        <TITLE>The cryph software utilities</TITLE>
        <DESCRIPTION>This set of pages reviews all the code available
to support vector operations.</DESCRIPTION>
      </LINK>
      <LINK>
        <URL>http://people.eecs.ku.edu/~miller/cryph</URL>
        <TITLE>The cryph software utilities 2</TITLE>
        <DESCRIPTION>This set of pages reviews all the code available
to support vector operations.</DESCRIPTION>
      </LINK>
    </LINKS>
    <UPLOADS>
      <UPLOAD ID="1189526309" TYPE="1.tar.gz">
```

```
        <FILENAME>PVM_V1.1.tar.gz</FILENAME>
        <TITLE>Tar file of utilities</TITLE>
        <DESCRIPTION>This has an Open Source implementation of the
utilities.</DESCRIPTION>
      </UPLOAD>
    </UPLOADS>
  </CORE_INSTRUCTION>
  <INSTRUCTIONAL_SUPPORT>
    <HANDOUTS>
      <UPLOAD ID="1189526396" TYPE="pdf">
        <TITLE>Description of Utilities</TITLE>
        <DESCRIPTION>This has the properties of these
operations.</DESCRIPTION>
        <FILENAME>DotCross.pdf</FILENAME>
      </UPLOAD>
      <TEXT>
        <TITLE>Comparison</TITLE>
        <DESCRIPTION>Here is some more information.</DESCRIPTION>
        <CONTENT>Dot products differ from cross products in that they
produce a scalar as a result.</CONTENT>
      </TEXT>
      <LINK>
        <TITLE>Comparison with Algebra</TITLE>
        <DESCRIPTION>Look at this to see why vector approaches are
better.</DESCRIPTION>

<URL>http://people.eecs.ku.edu/~miller/Courses/VectorGeometry/AlgebraVs
Geometry.html</URL>
      </LINK>
      <LINK>
        <TITLE>Comparison with Algebra</TITLE>
        <DESCRIPTION>Look at this to see why vector approaches are
better.</DESCRIPTION>

<URL>http://people.eecs.ku.edu/~miller/Courses/VectorGeometry/AlgebraVs
Geometry.html</URL>
      </LINK>
    </HANDOUTS>
    <CASESTUDIES>
      <CASESTUDY>
        <TITLE>Using the Dot Product</TITLE>
        <CONTENT>We use the dot product for many things. One example is
to decompose vectors.</CONTENT>
      </CASESTUDY>
      <CASESTUDY>
        <TITLE>Another application</TITLE>
        <CONTENT>You can also use them to find angles between
vectors.</CONTENT>
      </CASESTUDY>
    </CASESTUDIES>
    <READINGS>
      <JOURNAL>
```

```
        <TITLE>Vector Geometry for Computer Graphics</TITLE>
        <AUTHORS>James R. Miller</AUTHORS>
        <PUBLICATION>IEEE Computer Graphics and
Applications</PUBLICATION>
        <VOLUME>19</VOLUME>
        <ISSUE>3</ISSUE>
        <DATE>
          <DAY/>
          <MONTH>5</MONTH>
          <YEAR>1999</YEAR>
        </DATE>
        <PAGES>
          <START>66</START>
          <END>73</END>
        </PAGES>
        <ISBN>144435</ISBN>
        <ISSN>6637272</ISSN>
        <CALL_NUMBER>QA 76 .I2</CALL_NUMBER>
        <URL>http://computer.org</URL>
        <ABSTRACT>Vector Geometry Basics</ABSTRACT>
      </JOURNAL>
      <CONFERENCE_PAPER>
        <TITLE>Modeling and Visualizing Uncertainty in a Global Water
Balance Model</TITLE>
        <AUTHORS>J. R. Miller, D. C. Cliburn, J. J. Feddema, and T. A.
Slocum</AUTHORS>
        <CONFERENCE_NAME>ACM Symposium on Applied Computing (SAC
2003)</CONFERENCE_NAME>
        <PLACE>Melbourne, Florida</PLACE>
        <PUBLISHER>ACM</PUBLISHER>
        <VOLUME>1</VOLUME>
        <DATE>
          <DAY/>
          <MONTH>3</MONTH>
          <YEAR>2003</YEAR>
        </DATE>
        <PAGES>
          <START>972</START>
          <END>978</END>
        </PAGES>
        <ISBN>232354</ISBN>
        <ISSN>234553</ISSN>
        <CALL_NUMBER>2533245</CALL_NUMBER>
        <URL>http://www.acm.org</URL>
        <ABSTRACT>Model and visualize uncertainty.</ABSTRACT>
      </CONFERENCE_PAPER>
      <BOOK>
        <TITLE>Procedural Elements for Computer Graphics</TITLE>
        <AUTHORS>David F. Rogers</AUTHORS>
        <EDITION>second</EDITION>
        <PUBLISHER>WCB McGraw-Hill</PUBLISHER>
        <PLACE>Boston, Massachusetts</PLACE>
```

```
    <DATE>
      <DAY/>
      <MONTH>1</MONTH>
      <YEAR>1997</YEAR>
    </DATE>
    <PAGES>
      <START>287</START>
      <END>403</END>
    </PAGES>
    <ISBN>0-07-053548-5</ISBN>
    <ISSN>6565</ISSN>
    <CALL_NUMBER>T385 .R63  1998</CALL_NUMBER>
    <URL>http://www.mhhe.com</URL>
    <ABSTRACT>A book about computer graphics.</ABSTRACT>
  </BOOK>
  <NEWSPAPER>
    <TITLE>All about Life</TITLE>
    <AUTHORS>Sam Spade and Gloria Williams</AUTHORS>
    <PUBLICATION>Lawrence Journal World</PUBLICATION>
    <EDITION>Morning</EDITION>
    <SECTION>B</SECTION>
    <DATE>
      <DAY/>
      <MONTH>2</MONTH>
      <YEAR>2007</YEAR>
    </DATE>
    <PAGES>
      <START>B2</START>
      <END>B5</END>
    </PAGES>
    <ISBN>34953948</ISBN>
    <ISSN>357</ISSN>
    <CALL_NUMBER>357354</CALL_NUMBER>
    <URL>http://www.ljworld.com</URL>
    <ABSTRACT>A story about life's problems.</ABSTRACT>
  </NEWSPAPER>
</READINGS>
<GLOSSARY>
  <ENTRY>
    <TERM>affine space</TERM>
    <DEFINITION>A position in space</DEFINITION>
    <ALT>as</ALT>
    <ALT>a-space</ALT>
    <ALT>position</ALT>
  </ENTRY>
  <ENTRY>
    <TERM>vector space</TERM>
    <DEFINITION>a relative quantity</DEFINITION>
    <ALT>vs</ALT>
    <ALT>v-space</ALT>
    <ALT>direction</ALT>
  </ENTRY>
```

```
    </GLOSSARY>
    <NOTES>
      <NOTE>Don't forget to read this note.</NOTE>
      <NOTE>Here's another one to read.</NOTE>
      <NOTE>Don't forget this.</NOTE>
    </NOTES>
    <ACTIVITIES>
      <TEXT>
        <TITLE>visit a vector</TITLE>
        <DESCRIPTION>Get over your fear of vectors by talking with
one.</DESCRIPTION>
        <CONTENT>Come to the "vectors are everywhere" park.</CONTENT>
      </TEXT>
      <LINK>
        <TITLE>interview a vector</TITLE>
        <DESCRIPTION>Here is the text of an interview.</DESCRIPTION>
        <URL>http://www.sunflower.com</URL>
      </LINK>
      <LINK>
        <TITLE>interview a vector</TITLE>
        <DESCRIPTION>Here is the text of an interview.</DESCRIPTION>
        <URL>http://www.sunflower.com</URL>
      </LINK>
      <UPLOAD ID="1189527521" TYPE="jpeg">
        <TITLE>Do you want to see a file</TITLE>
        <DESCRIPTION>Whether you want it or not, here it
is.</DESCRIPTION>
        <FILENAME>Spaces.jpeg</FILENAME>
      </UPLOAD>
    </ACTIVITIES>
    <ASSESSMENT>
      <MULTIPLE_CHOICE>
        <QUESTION>What is a vector?</QUESTION>
        <OPTION ISCORRECT="true">
          <TEXT>A relative quantity</TEXT>
          <REMEDIATION>Good answer!</REMEDIATION>
        </OPTION>
        <OPTION ISCORRECT="">
          <TEXT>An absolute quantity</TEXT>
          <REMEDIATION>You need to review the lesson</REMEDIATION>
        </OPTION>
        <OPTION ISCORRECT="">
          <TEXT>An unknown quantity</TEXT>
          <REMEDIATION>Very metaphysical of you</REMEDIATION>
        </OPTION>
        <OPTION ISCORRECT="">
          <TEXT>A quality</TEXT>
          <REMEDIATION>Better yet...</REMEDIATION>
        </OPTION>
      </MULTIPLE_CHOICE>
      <MULTIPLE_CHOICE>
        <QUESTION>Why are vectors better than oreo cookies?</QUESTION>
        <OPTION ISCORRECT="true">
          <TEXT>They have fewer calories</TEXT>
          <REMEDIATION>Uh - yes</REMEDIATION>
        </OPTION>
        <OPTION ISCORRECT="">
          <TEXT>They can point to sandwiches.</TEXT>
          <REMEDIATION>Try again.</REMEDIATION>
        </OPTION>
        <OPTION ISCORRECT="">
          <TEXT>Because points are not.</TEXT>
          <REMEDIATION>Actually, points are, too.</REMEDIATION>
        </OPTION>
        <OPTION ISCORRECT="true">
          <TEXT>Because they can be computed.</TEXT>
          <REMEDIATION>I guess that's OK.</REMEDIATION>
        </OPTION>
        <OPTION ISCORRECT="">
          <TEXT>Because they never get lost.</TEXT>
          <REMEDIATION>That doesn't make any sense.</REMEDIATION>
        </OPTION>
      </MULTIPLE_CHOICE>
    </ASSESSMENT>
    <DIRECTED_QUESTIONS>
      <DIRECTED_QUESTION>
        <QUESTION>What is the relationship between points and
vectors?</QUESTION>
        <ANSWER>The difference between two points is a vector. Also,
points belong to an affine space. Associated with an affine space is a
vector space containing vectors.</ANSWER>
      </DIRECTED_QUESTION>
      <DIRECTED_QUESTION>
        <QUESTION>Do you like points?</QUESTION>
        <ANSWER>Come on -- what's not to like?</ANSWER>
      </DIRECTED_QUESTION>
      <DIRECTED_QUESTION>
        <QUESTION>Have you had enough of this lesson?</QUESTION>
        <ANSWER>Absolutely!</ANSWER>
      </DIRECTED_QUESTION>
    </DIRECTED_QUESTIONS>
  </INSTRUCTIONAL_SUPPORT>
</INSTRUCTIONAL_UNIT>
```

References

“What is XSL?” http://www.w3.org/TR/REC-xml

Lionel Villard and Nabil Layaïda, "*iXSLT: An Incremental XSLT Transformation Processor for XML Document Manipulation*", submitted to World Wide Web Journal, Kluwer publisher, 2001, pp. 474-485.

Lionel Villard*, "Authoring Transformations by Direct Manipulation for Adaptable Multimedia Presentations"*, ACM Symposium on Document engineering, 2001, pp 125-134.

Whitehill <XSL>Composer product information.
http://www.simtel.net/product.php%5Bid%5D56871%5BSiteID%5Dsimtel.net

Teruo Koyanagi, Kouichi Oni, and Masashiro Hori*, "Demonstrational Interface for XSLT Style sheet Generation"*, Graphic Communications Association, Extreme Markup Languages Conference, 17 August 2000.

Bray, T.; Paoli, J.; and Sperberg-McQueen, C. M. {editors} (1998). Extensible Markup Language (XML) 1.0. (W3C Recommendation). Website: http://www.w3.org/TR/REC-xml

Clark, J. {editor} (1999). XSL Transformations (XSLT) Version 1.0 (W3C Recommendation). Website: http://www.w3.org/TR/xslt

Miro Lehtonen, Renaud Petit, Oskari Heinonen and Greger Lind´en, "*A Dynamic User Interface for Document Assembly*", Proceedings of the 2002 ACM symposium on Document engineering, 2002

Olivier Beaudoux, "*XML Active Transformation (eXAcT): Transforming Documents within Interactive Systems*", Proceedings of the 2005 ACM symposium on Document engineering, 2005, pp. 146-148.

Gerardo Canfora and Luigi Cerulo, "*A Visual Approach to Define XML to FO Transformations*", Proceedings of the 14th international conference on Software engineering and knowledge engineering, 2002, pp 563-570.

Altova XMLSpy XML Editor. http:// www.altova.com/products/xmlspy/xml_editor.html

Altova StyleVision® 2008 Style sheet Designer.
http://www.altova.com/products/stylevision/xslt_stylesheet_designer.html

Nathalie Colineau, Julien Phalip, Andrew Lampert, "*The Delivery of Multimedia Presentations in a Graphical User Interface Environment*", Proceedings of the 11th international conference on Intelligent user interfaces, 2006

E. L. Meyen and J. R. Miller, A System for Creating and Managing Reusable Learning Objects, Proceedings of the Sixth IASTED International Conference on Web-Based Education, March 14-16, 2007, Chamonix, France, pp. 353-358.