AUTHORING XML DOCUMENTS WITH XHTML AND MATHML SUPPORT

A Project Report
Presented to
The Faculty of the Department of Computer Science
San Jose State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by Xiaoheng Wu
December 2006

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Cay Horstmann

Dr. Jon Pearce

Dr. Michael Beeson

APPROVED FOR THE UNIVERSITY

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Dr. Cay Horstmann

Dr. Jon Pearce

Dr. Michael Beeson

APPROVED FOR THE UNIVERSITY

Dr. Cay Horstmann

Dr. Jon Pearce

Dr. Michael Beeson

ABSTRACT

AUTHORING XML DOCUMENTS WITH
XHTML AND MATHML SUPPORT
by Xiaoheng Wu

Since the late 1970s, a large number of scientific documents have been authored in TeX or its derivations such as LaTeX. These typesetting systems allow anybody to write high-quality books and articles. But the TeX syntax is not compatible with HTML or XML. So the WWW consortium's answer is MathML. The primary goal of MathML is to enable mathematical documents to be communicated, exchanged, and processed on the Web. Therefore, MathML documents are usually embedded with XHTML documents. Currently, there are several XHTML+MathML editors. The most popular editors use two common approaches. The first approach offers a What-You-See-Is-What-You-Get (WYSIWYG) interface. But experts often find it is difficult to have precise control. For example, font attribute is determined by the direction of the mouse movement during the event of insertion. The second approach uses a text-based form. The entire document is presented as a tree-like structure. The tree-like structure is unintuitive and extremely inefficient to comprehend, particularly for two-dimensional structures such as tables or equations. Here, I present a What-You-See-Is-What-You-Need (WYSIWYN) editing interface that satisfies the needs of experts who have knowledge of XHML+MathML. The WYSIWYN interface is presented in a form that simultaneously makes editing operations unambiguous and that looks recognizable. It avoids unexpected errors by showing enough structure, but still maintain enough visual presentation to avoid confusion.

This report presents a test bench, an XHTML+MathML editor with a new navigation model that demonstrates the WYSIWYN user interface. Similar to a WYSIWYG editor, XHTML+MathML documents can be visualized during editing, and users can check the current XPath position by viewing the status bar. In contrast to the WYSIWYG editor, the new approach offers users the ability to view local structure of the current element with a selected style. In this way, users can magnify any ambiguous positions and still be able to visualize mathematical documents. In addition, the test bench offers multiple WYSIWYN modes with different levels of magnification.

ACKNOWLEDGMENTS

I would like to express my appreciation to Dr. Cay Horstmann who provided the motivation, resources, and invaluable insights without which this project could never have been accomplished.

I would also like to express my gratitude to my committee members Dr. Beeson and Dr. Pearce for their useful comments on my work. I am also grateful for my friends and the students who assisted me for usability study.

Finally, my thanks to my parents for their encouragement and financial support.

Table of Contents

Index of Tables

Illustration Index

1. INTRODUCTION

As XML has become more popular, XML document writers demand an authoring tool with efficiency and precise control that would alleviate the problems that they face. Like other Word documents, XML documents can be stored as regular word processor files, such as Word or OpenOffice documents. But these documents often have their own document structures. Thus they always carry portability issues and have no validation mechanisms (DTD or Schema) for desired XML structure. Obviously, it is important to develop a tool that can ease the conversion between the Word document structure and the desired XML document structure and enforce DTD or Schema constraints. In this way, we can guarantee each building block of the XML document is legal. This is a requirement when independent groups of people want to interchange document data. An editor without DTD validation can be error-prone for XML specific languages. In addition, as a What-You-See-Is-What-You-Get (WYSIWYG) editor, a word processor often has some side effects. For example, the font attribute is determined by the direction of the cursor movement during the event of insertion. Because of this uncertainty, users are often surprised that the character is bold when they are expecting italic.

To ease the conversion between the Word document structure and the desired XML document structure, developers created XML specific editors. Targeted at users unfamiliar with structured contents [3], the most popular XML editors use two common approaches. The first approach tries to separate the presentation view from its logic or the structured view. This approach offers an easy editing environment for beginners. But experts often find it is difficult to have precise control. For example, Amaya uses the What-You-See-Is-What-You-Get (WYSIWYG) interface (Illustration 1.1).

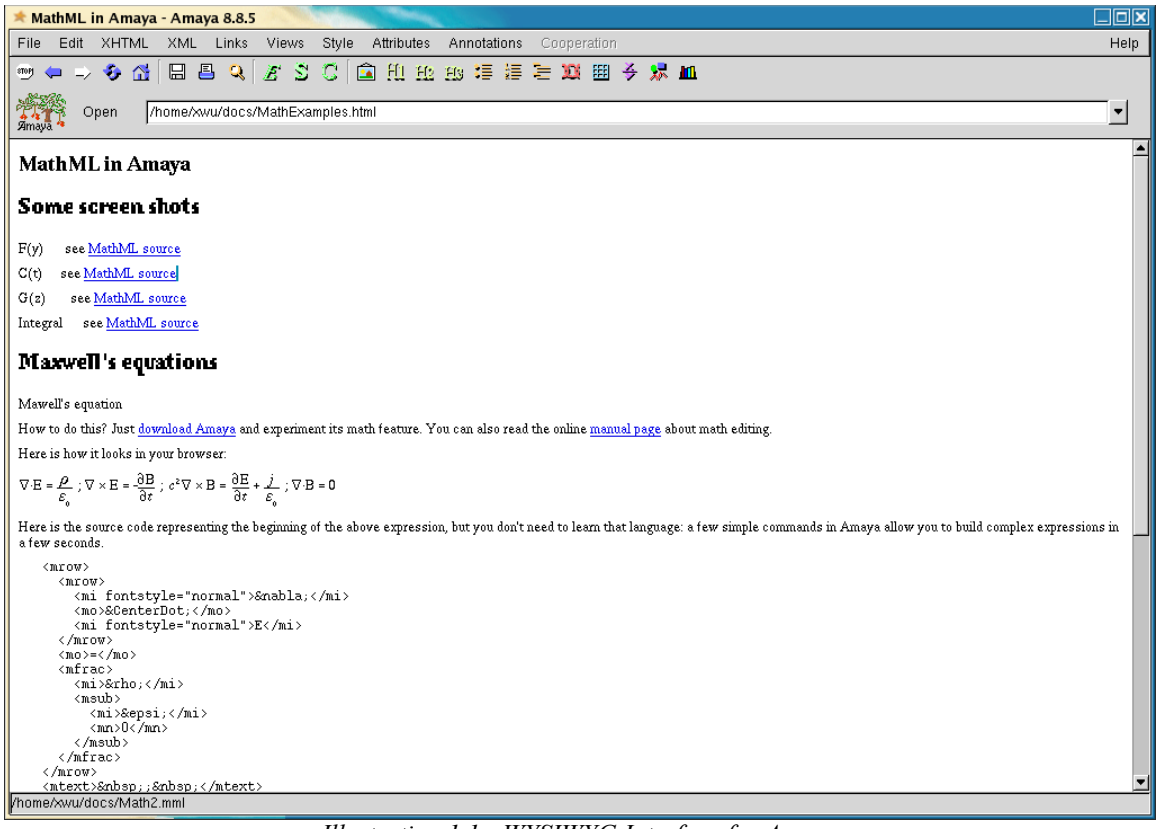


Illustration 1.1.: WYSIWYG Interface for Amaya

It is difficult to control the next insertion point when users try to make space adjustment between paragraph elements. Let's demonstrate this problem that every computer scientist can relate to. Consider a variable x, it is difficult to tell where the cursor position is, just from the WYSIWYG interface (Illustration 1.2).

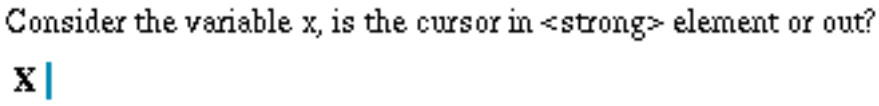


Illustration 1.2.: WYSIWYG Issue

The second approach uses the text-based form. The entire document is treated like a free-flowing string with a tree-like structure. (Illustration 1.3)

```
<p><math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mrow>
      <mi fontstyle="normal">&#x2207;</mi>
      <mo>·</mo>
      <mi fontstyle="normal">E</mi>
    </mrow>
    <mo>=</mo>
    <mfrac>
      <mi>&#x3c1;</mi>
      <msub>
        <mi>&#x3b5;</mi>
        <mn>0</mn>
      </msub>
    </mfrac>
  </mrow>
  <mtext>~;~</mtext>
```

Illustration 1.3.: Tree-like structure

While the tree-like structure improves the readability of the document, the free-flowing string is not constructive. Users can only achieve editing tasks at text level instead of element level. For example, to insert an element of six characters, users will need to type six characters plus starting symbol < and ending symbol >, and not even mention the end tag (assuming the end tag is auto-completed). Furthermore, the cursor navigation is not efficient without a mouse because it has to cross all letters between the starting point and the ending point.

In order to stay away from all the drawbacks of existing XML editors, this project will eradicate unexpected errors by showing enough structure, but still maintain enough WYSIWYG to avoid confusion. In addition, this project supports navigation by XML element structures, and it makes a mouse optional. The ultimate goal is to create a "partial WYSIWYG" editor or a hybrid-structured editor. A hybrid-structured editor was an idea pioneered during the 1980s to early 1990s [1,2,4,5,7], and it is now being revitalized [6].

Within the scope of the hybrid-structured editor, two XML specific languages, namely, MathML+XHTML, will be supported. XHTML uses XML markup and syntax to represent HTML in XML terms. MathML aims to represent mathematical symbols and formulae.

Mathematicians who are aware of T_EX book should recognize 'x' is a text in T_EX which is a typesetting system for creating high quality scientific documents. In contrast to this simple text, a T_EX text can become very complicated and unrecognizable for a large formula such as '$$a_0+{1\over\displaystyle a_1+{\strut 1\over\displaystyle a_2+{\strut 1\over\displaystyle a_3+{\struct a\over a_4}}}}$$'. Just from the semantics of the string, it is difficult to understand its meaning.

MathML documents can offer a better readability and support the hybrid-structure editor. To motivate XHTML+MathML, this project will wire up both XHTML and MathML support with an XML text editing framework developed by several graduate students who were under the direction of Dr. Cay Horstmann.

The XMLEd project was first established by two CS298 students, Nupura Pradhan and Swati Pathak , and then improved by another CS298 student, Tong Ho. The concept of editing XML documents in a hybrid-structure was demonstrated in previous CS298 projects: [8], [9], and [3]. The first two have identified and implemented the essential XML structural operations [3], and the most recent project modified the document model to support the hybrid-structure, and two new features were added: the configurable CSS views and the script driven editing system. Here, I present a What-You-See-Is-What-You-Need (WYSIWYN) editing interface that satisfies the needs of experts who have knowledge of XHML+MathML. The WYSIWYN interface is presented in a form that simultaneously makes editing operations unambiguous and that looks recognizable. It avoids unexpected errors by showing enough structure, but still maintain enough visual presentation to avoid confusion. The new approach offers users the ability to view local structure of the current element with a selected style. In this way, users can magnify any ambiguous position and still be able to visualize mathematical documents. In addition,

the project work offers multiple WYSIWYN modes with different level of magnification. All of the projects have proved that the Swing text editing framework can provide useful building blocks for the creation of the hybrid-structured editor and its additional supports. While the previous works elaborately fabricated the foundation of the hybrid-structured editor, the developed framework still has some deficiencies and pitfalls which are discussed in section 8.

This report is organized as follows. Section 2 gives some basic knowledge of XML languages: XHTML+MathML. It is important to understand the fundamental elements of these languages and their associated attributes. Commonly known elements for XHTML will be ignored here. This section will focus on mostly MathML elements because these elements are math-specific and the interpretation is not quite easy. Section 3 contracts Lisp-like system, T_EX, and MathML. Section 4 evaluates popular tools that currently exist for editing XHTML and MathML. This section will explain the drawbacks of current authoring tools and clarify the need of XMLEd. Section 5 assesses the existing frameworks developed by previous students. The evaluation will be divided into document model and views. For views, the view tree hierarchy, caret implementation, and CSS specific styles will be discussed here. Section 6 describes the accomplishments that were achieved in this project.

- A list of deliverables for MathML.
- The accomplishment of XHTML views.
- The accomplishment of MathML views.
- CSS extensions. Because the CSS specification defined by World Wide Web Consortium (W3C) does not support MathML and other XHTML tags like `<img>`, a CSS extension is added as a supplement of this project.
- A new feature for XPath.
- The WYSIWYN interface.

Section 7 explains usability study and the usability evaluation for the current implementation of WYSIWYN user interface. The last section covers the conclusion and ends up with feature improvements.

2. XHTML AND MATHML

Before the introduction of project work for this semester, it is important to study some fundamental concepts and the basic structure of XHTML+MathML. XHTML is an extended version of HTML reformulated in XML. In other words, it is the successor of HTML. In XHTML, there are three variants: Strict, Traditional, and Frameset. Each variant has its own Document Type Definition (DTD). XMLEd will only care about the first variant – Strict.

MathML is another public XML language that enables mathematical documents to be processed by software such as Web browsers, so mathematicians can share their mathematical notations on-line.

All MathML elements fall into one of three categories: presentation elements, content elements and interface elements. This project will focus on presentation elements. Presentation elements describe mathematical notation's visually oriented two-dimensional structure [13]. Here is a summary of presentation elements from w3.org (Table 2.1,2.2,2.3,2.4,2.5) and all delivered elements are in bold:

Token Elements

Element	**Description**
mi	identifier
mn	number
mo	operator, fence, or separator
mtext	text
mspace	space
ms	string literal
mglyph	accessing glyphs for characters from MathML

(Source [13])

Table 2.1.: Token Elements

General Layout Schemata

Element	Description
mrow	group any number of sub-expressions horizontally
mfrac	form a fraction from two sub-expression
msqrt	form a square root sign (radical without an index)
mroot	form a radical with specified index
mstyle	style change
merror	enclose a syntax error message from a preprocessor
mpadded	adjust space around content
mphantom	make content invisible but preserve its size
mfenced	surround content with a pair of fences
menclose	enclose content with a stretching symbol such as a long division sign.

(Source [13])

Table 2.2.: General Layout Schemata

Script and Limit Schemata

Element	Description
msub	attach a subscript to a base
msup	attach a superscript to a base
msubsup	attach a subscript-superscript pair to a base
munder	attach an underscript to a base
mover	attach an overscript to a base
munderover	attach an underscript-overscript pair to a base
mmultiscripts	attach prescripts and tensor indices to a base

(Source [13])

Table 2.3.: Script and Limit Schemata

Tables and Matrices

Element	Description
mtable	table or matrix
mtr	row in a table or matrix
mtd	one entry in a table or matrix
maligngroup and malignmark	alignment markers

(Source [13])

Table 2.4.: Tables and Matrices

Enlivening Expressions

Element	Description
maction	bind actions to a sub-expression

(Source [13])

Table 2.5.: Enlivening Expressions

MathML inherits all syntax and grammar rules from XML. Also, there are two additional MathML specific grammar and syntax rules. First, it has criteria on attribute values. For example, it is not possible for pure XML documents to require that an attribute value be a positive integer [13]. Second, some child elements require more restrictions. For example, the mfrac Element requires the first child to be its numerator and the second child to be its denominator. The order matters! Because of this reason, in MathML, in order to distinguish the differences, the children like these in the mfrac element are referred as arguments instead of just children.

Here is a table (Table 2.6) of the argument requirements. 1* indicates an inferred `mrow` element. For example, the MathML for the expression $\sqrt{2}$ is `<msqrt><mrow><mn>2</mn></mrow></msqrt>`, and this MathML equivalent can also be rewritten as `<msqrt><mn>2</mn></msqrt>`.

Element	**Required argument count**	**Argument roles**(when differ by position)
mrow	0 or more	
mfrac	2	numerator denominator
msqrt	1*	
mroot	2	base index
mstyle	1*	
merror	1*	
mpadded	1*	
mphantom	1*	
mfenced	0 or more	
msub	2	base subscript
msup	2	base superscript
msubsup	3	base subscript superscript
munder	2	base underscript
mover	2	base overscript
munderover	3	base underscript overscript
mmultiscripts	1 or more	base (subscript superscript)* [<mprescripts/> (presubscript presuperscript)*]
mtable	0 or more rows	0 or more mtr elements
mtr	0 or more table elements	0 or more mtd elements
mtd	1*	
maction	1 or more	depend on actiontype attribute

(Source 13)

Table 2.6.: Argument Count and Roles

Possible units in MathML are em, ex, px, in , cm, pt, pc, and %. The first two are commonly used for horizontal and vertical units because they are font-relative units.

The whitespace characters in the input stream must be ignored. Here, whitespace characters includes: blanks, tabs, newlines, or carriage returns. A simple example, `<mo> ( </mo>` is equivalent to `<mo>(</mo>`.

3. LISP-LIKE SYSTEMS, TEX, AND MATHML

The earliest approaches to mathematical formula entry involved the use of specialized equation description languages [11]. Using a LISP-like syntax, an entire parse tree for a formula can be expressed in a linear, text-based form [10]. For example, Skribe, a functional programming language for authoring technical documents, can parse the text [text goodies: , (**bold** "bold") and , (**it** "italic") .] as (list "text goodies:" (bold "bold") "and" (it "italic") ".") [18]. The entire text between the brackets is parsed into a list of character strings. The advantage of these linearized system is that users can master the syntax and keywords and produce most quality equations in their pretty formats. But such a system has some the drawbacks: 1) The learning curve is quite steep. 2) Instead of the inherent two dimensional structure of a mathematic formula, now users have to mentally translate the formula to the linear, text-based form. 3) It is difficult, if possible at all, to visualize the semantics of the LISP-like language when a formula becomes large and complex. Because of these drawbacks, users often switched to a better tool.

During the 1980s, Donald Knuth created the first version of TEX. TEX was intended to typeset mathematic formulas for the creation of beautiful books. Many experienced TEX users were satisfied with its powerful capability. But some newcomers found it is a non-trivial task to master these useful tricks in TEX, and other users found it is not compatible with XML syntax which can be processed on the Web. In both cases, they often turned to use other modern authoring tools with the WYSIWYG interface and MathML support.

4. EVALUATION OF EXISTING TOOLS FOR XHTML AND MATHML

Recently, there is only a few XTHML+MathML editing tools available for downloading. Most of them support only a WYSIWYG editing interface. None of them provides the advantages of the WYSIWYN interface. XMLSpy is a generic XML document editor that can be used for authoring XTHML+MathML, but it is only a XML editor with no particular supports for visualizing MathML documents. Amaya is a similar product for the open source community. Experts often find themselves suffering from the precise control in its WYSIWYG interface. While Amaya is designed for the combination of XHTML+MathML, XMLmind XML Editor (XXE for short) is a structured editor that is

particularly designed for XHTML documents, not MathML documents. MathType, a product from Design Science, is platform dependent. SciWriter is a similar commercial product for writing scientific documents quickly and efficiently. Both of them create different document structures and require a converter to transform its own document structure to the desired XHTML+MathML document structure. Most importantly, it is not open source.

More recent systems allow visual feedback and other structured editing features. Even though the modern systems have eased the authoring task, they have also introduced some new drawbacks.

XMLSpy can aid and expedite the development of XML projects, but it is a commercial product that does not support the visualization of MathML documents (Illustration 4.1). It is not adequate to use this generic XML editor for XML specific languages, such as XHTML+MathML. Mathematicians may not want to know the structure of MathML because they are only interested in mathematical expressions. For example, they are more interested in the visualization of a matrix instead of `mtable` itself. XHTML users may want to visualize the document in the WYSIWYG view and still maintain precise control.

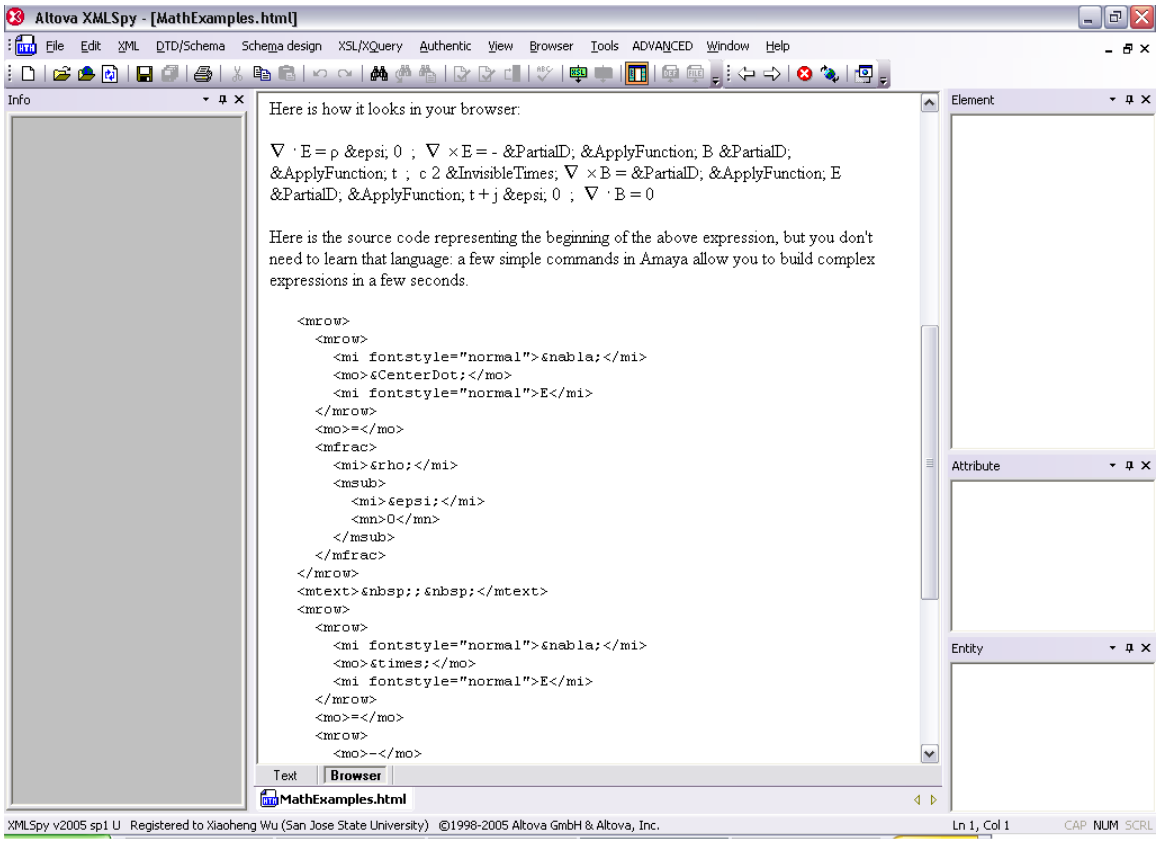


Illustration 4.1.: MathML with XMLSpy

XMLmind is a capable structured editor. It provides a configurable Word processor-like view using cascading style sheets (CSS2). However, it is difficult to track the element position without the status bar of XPath. Because of this limitation, users often have to check XPath on the status bar for the editing position. For example, to insert a sibling element of its parent element, users need to press CTRL + ↑ to move the position to the parent, then insert the new element. To complete this task, users need to verify the current position from the status bar. The frequent verifications of the element position hinders editing speed. Moreover, it only supports XHTML, not MathML. The Illustration 4.2 shows the error message when a user tries to load a MathML document.

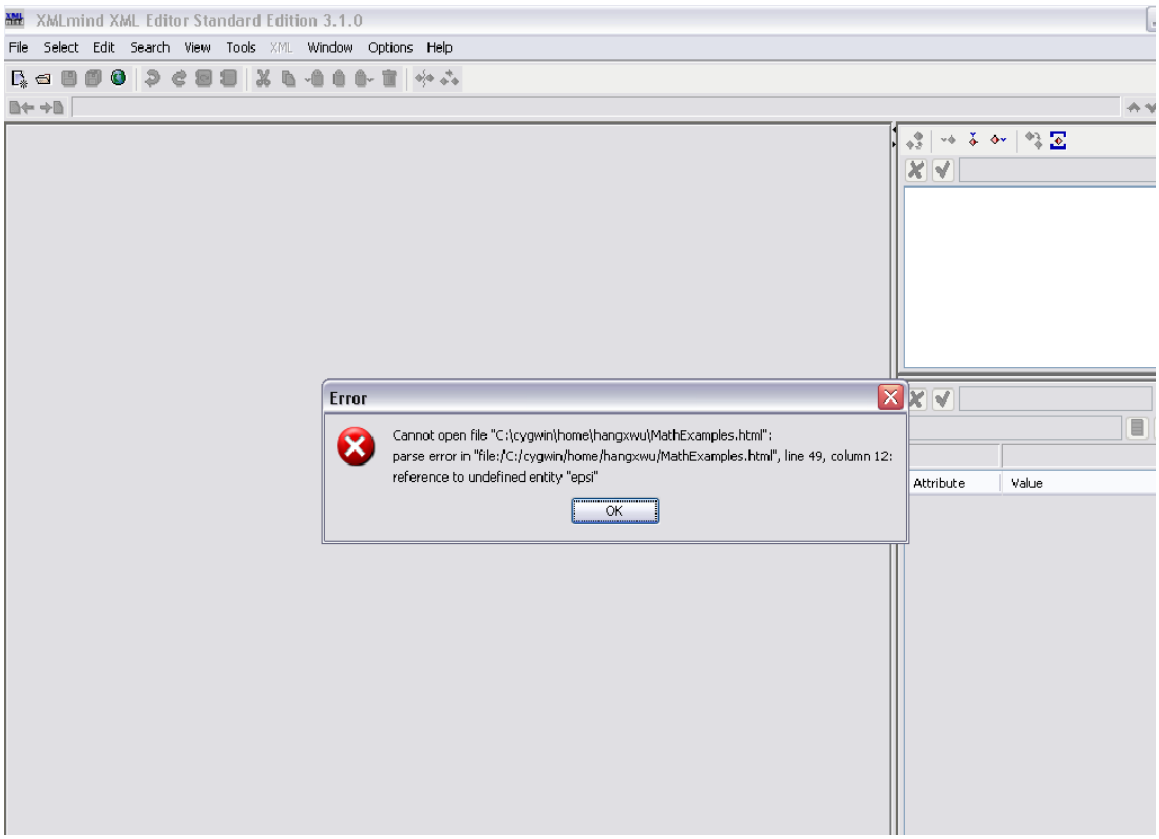


Illustration 4.2.: MathML Error with XMLmind

SciWriter is an equation editor that completely integrates mathematical expressions and text in the same environment. It uses the common visual feedback (WYSIWYG) technique and creates its own document structure. Users can create pretty mathematic documents quickly and efficiently by using SciWriter. However, in order to create a MathML document, it requires a converter to translate its own document structure into the desired MathML document structure. Also, this tool is not platform-independent. It is only for Windows. Most importantly, the tool is licensed as shareware, not free. Other commercial products similar to SciWriter, such as MathType from Design Science carries the same types of problem. Illustration 4.3 is the translator for MathType and SciWriter has the similar feature.

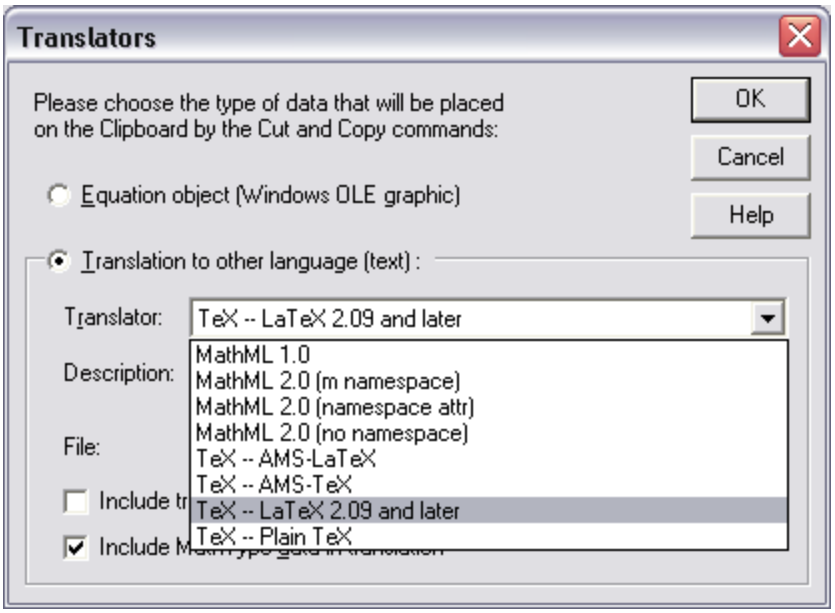


Illustration 4.3.: MathType Translator

Amaya is intended to be a comprehensive Web editor with XHTML+MathML support. The current version has three major views: structured view, source view, and WYSIWYG view. In the pure WYSIWYG view, it is difficult, if possible at all, to avoid errors introduced by ambiguity nature of the WYSIWYG interface. In the source view, Amaya offers no special supports. It is just like a normal text editor, such as Notepad. In the structured view, users can view the exact position of the cursor and do efficient navigation, but are unable to visualize the current mathematical expression (see Illustration 4.4).

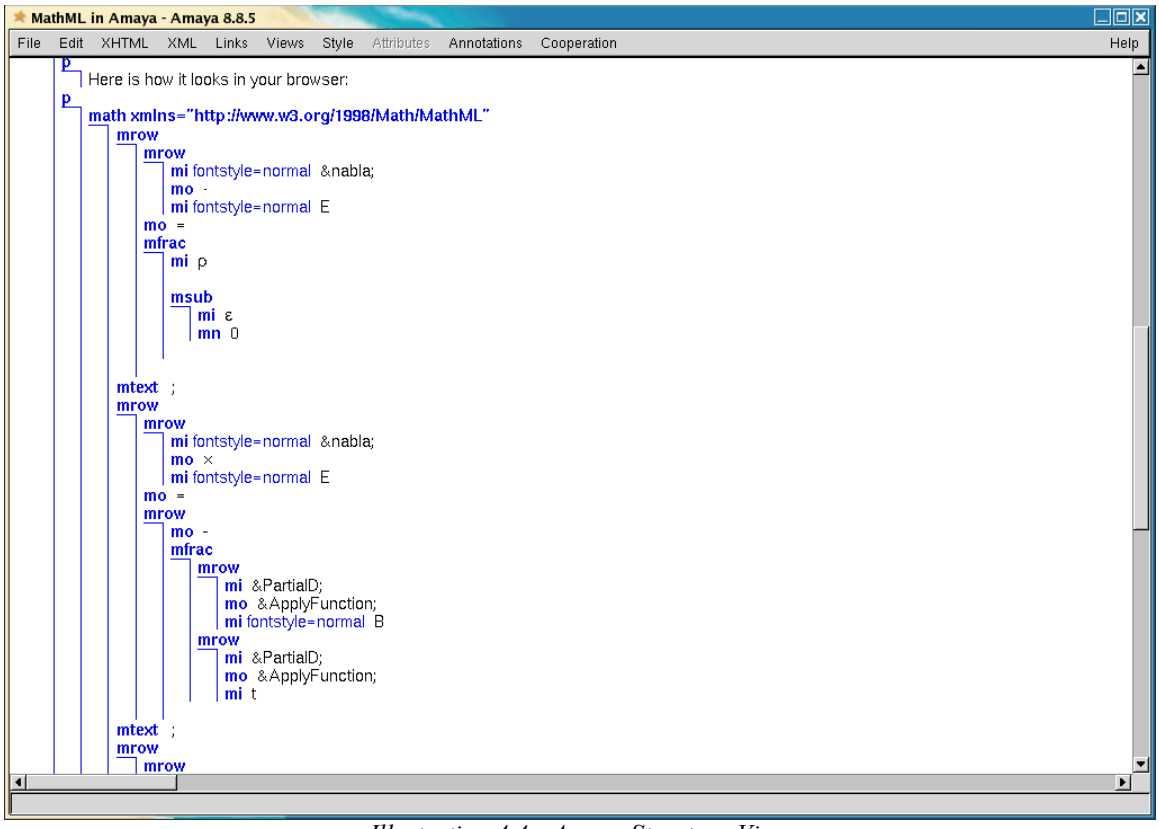


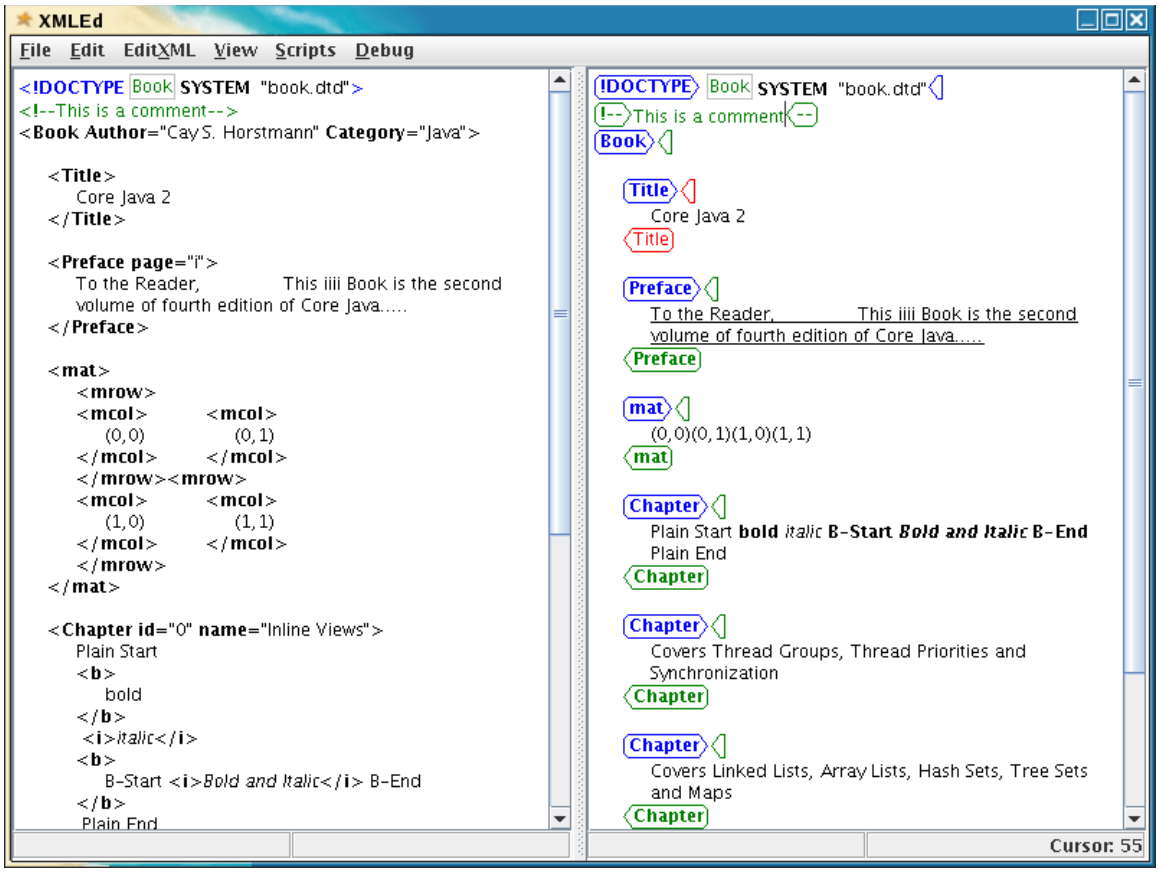
Illustration 4.4.: Amaya Structure View

5. IMPLEMENTATION OF XML VERSION OF XMLEd EDITOR

The XMLEd editor is a XML editor based on the Java Swing text editing framework and developed by several graduate students who were under the direction of Dr. Cay Horstmann.

5.1.Demonstration

Illustration 5.1 shows a screen shot of XMLEd with a CSS style sheet in action. The view on the right panel is called a GUI view which can be configured by a CSS style sheet. As the picture shows, it supports coloring, underline, bold, italic, and more. The next section describes the functionality of the XMLEd editor in more detail.



Source [3]

Illustration 5.1. XMLEd Evaluation Demo

5.2.Functionality of the XMLEd Editor

The XMLEd editor supports following features:

- Common editor requirements. For example, open, save, undo, and redo.
- Debug feature. Users or developers can use the debug menu to view different hierarchical structures, such as view element and model element (see section 4 and 5 in [3] for view element and model element).
- Efficient Caret Navigation. For example, the cursor can skip a start tag name and directly jump to attribute position when a user presses the right arrow key.
- Structural Editing. It supports copy, paste, merge, and split editing operations
- CSS configurable views. The view on the right panel can be configured by CSS rules.

- Scripting. It has a script editing feature.

The previous reports [3,8,9] have more detailed explanations and examples.

5.3.Implementation of the Document Model

The XMLEd editor maintained the Model-View-Controller (MVC) design pattern of Swing text-editing framework. All the Swing text components are derived from JTextComponent (see Illustration 5.2). A text component contain the following major pieces: Document, Highlighter, Caret, Editor Kit, and Views. Furthermore, Swing has a built-in CSS support for stylization of views. However, this support is very restrictive and limited to only HTML semantics. For an XML editor that demands multiple customizable views, a former student, To Hong, improved the XMLEd framework to overcome the limitations of the Swing framework. For more detail, see section 5.4.2 in [3].

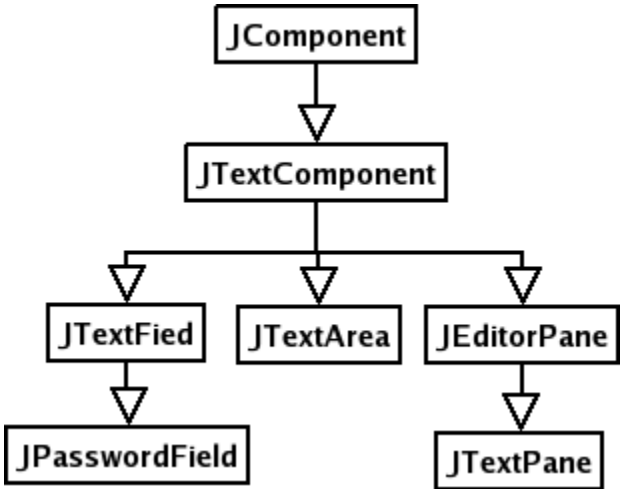
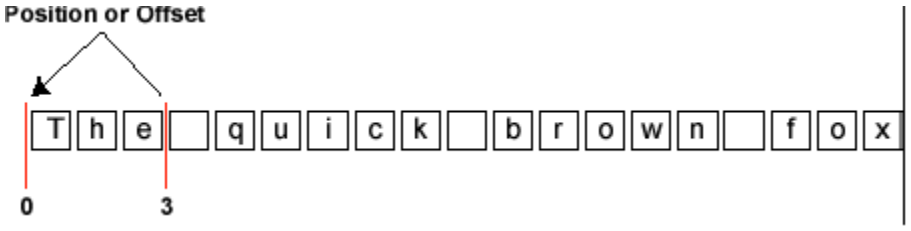


Illustration 5.2.: Swing Text Components

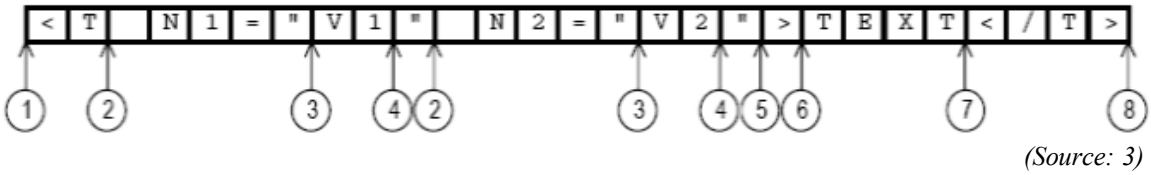
Conceptually speaking, both frameworks follow the same design pattern. Therefore, both frameworks have the similar features and architecture. For example, both frameworks support undo/redo utilities. While both frameworks support the basic editing features, they have some significant technical differences. As I stated in previous paragraph, using the Swing text-editing framework for editing XML document creates complexities that need to be resolved [3]. The XMLEd framework successfully resolved these complexities

and formulated a brand new model with an elegant coordinate system and the `Element` views associated with it. However, the biggest drawback of the current framework is the duplication introduced by tackling the restriction of package visibility from Swing text-editing framework.



(Source: 12)

Illustration 5.3.: Swing Document Model Coordinate System



(Source: 3)

Illustration 5.4.: XMLEd Document Model Coordinate System

Both frameworks implement the `AbstractDocument` class as their basic models. However, there are two aspects that fundamentally distinguish the common Swing text-editing model and the current XML editing model. The first model (Illustration 5.3) holds a linear sequence of characters and manipulates the sequence through an integer coordinate system. Also, this model structure uses a hierarchical `Element` tree that is only suitable for linear text editing. In contrast, the second model (Illustration 5.4) uses a location based caret coordinate system that avoids the insertion-ambiguity pitfalls [8]. Most importantly, this new model maintains XML document-specific `Elements` that are used by `View` objects as a good conduit for retrieving the required text and converting the view coordinate and model coordinate. The document-specific `Elements` contain no style information, only pure contents. The design of document `Element` tree provides a convenient way to visualize the document model through multiple Cascaded Style Sheet (CSS) controlled `views`.

5.4.Implementation of Views

The visualization of XML document model supports multiple views by using the Cascaded Style Sheet (CSS) technology. The style sheets perform like drivers that control `View` objects rendering the XML document model. For the rendering components, luckily, Swing editing framework provides some vital functionalities that can be inherited by the current `View` frameworks, such as paragraph breaking and text highlighter. While the current `View` implementation takes advantages of the Swing editing framework, it does not address XML specific languages, such as MathML and XHTML. The rendering for these specific documents requires special handling. For example, `img` is an element of an XHTML document. Its pure responsibility is to display an image with an alternative text for the missing image. In order to make this happen, two Views must be created, one for the image and another for the alternative text. The current framework does not provide a shortcut for the text retrieving and image rendering. In other words, the current framework is a generic XML editing framework. For document-specific languages, a set of views must be obtained either through modification of existing views or by reconstruction.

5.4.1.Caret Implementation

The current caret implementation is based on the model's coordinate system. Unlike caret navigation in a text-based XML editor, where the caret traverses every character in the XML document, caret navigation in XMLEd is efficient. For example, users can move the cursor from beginning of an element name to its attribute value without going through each character in the name of the element. In other words, the caret can skip tags and attribute names. This structural navigation increases the editing speed and avoids ambiguity reported in [8]. However, caret implementation can be improved by providing the element structure information on the status bar. Like other editor implementations, showing the XPath on the status bar can dramatically increase the readability and productivity, and they provide additional knowledge about the accurate location of the current cursor for WYSIWYN presentation.

5.4.2.View Tree

The current `View` tree hierarchy is controlled by an additional layer, namely, style-specific `Element` structure which is determined by CSS style sheets. Unlike the traditional Swing editing `Document` element tree, the style-specific `Element` tree is mutable. That means the tree can reshape itself according to CSS style sheets. Because of the ability of the tree mutation, the support for multiple views by installing different CSS style sheets becomes realistic. For example, for plain text and style text, a bold element should be rendered either bold or **bold.** Without the modification based on the Swing Document/View architecture, this cannot be done.

Whenever the requests for modifications in the actual document content occur, they are passed to the immutable `Document` element tree through an internal reference in the style-specific `Element` tree. This clever design decoupled the style attributes from `Document` elements, so each element can be rendered in multiple `Views`. If its content is changed in one of these `Views`, this change will be updated for all `Views`. In addition, the current `View` supports pseudo CSS-tags, such as :commenttag and :attrname. This feature allows users to change the styles for these abstract terms that are not directly represented by the semantics of the document contents. The following diagram (Illustration 5.5) shows how a CSS style sheet shapes the view `Element` Tree, hence, the `View` tree it maps to.

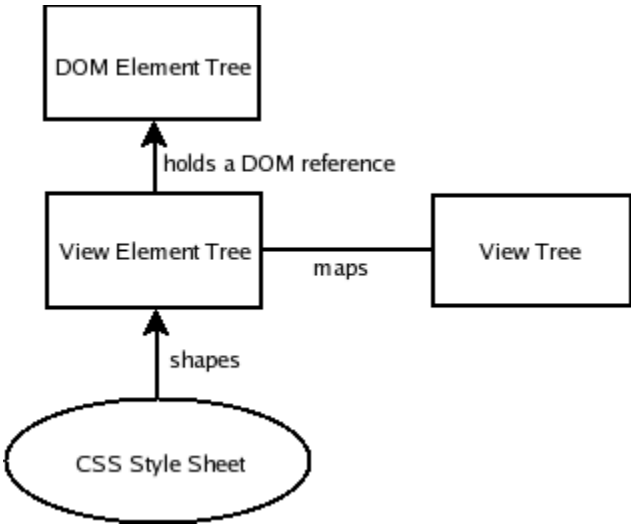


Illustration 5.5.: XMLEd Rendering Technique

5.4.3.CSS Styles

In order to visualize XML documents in a "partial WYSIWYG" view explained in [3], XMLEd implements the standard CSS technology. XMLEd supports a subset of CSS 2.1 and provides a modern feature for authoring XML documents. The W3C specification for CSS style rules has the following form (Illustration 5.6) commented in CSS parser code:

- (statement)*
- statement (@rule | ruleset | block)*
- @rule (block | identifier)*;
- block matching [ ] ( ) { }
- identifier "*" | '*' anything but a [ ] ( ) { }
- ruleset selector declarationBlock
- selector (identifier | (block, except block '{ }'))*
- declarationBlock declaration* block*
- declaration (identifier* stopping when identifier ends with : or ;)

Illustration 5.6.: CSS Form

For example, the CSS rule for rendering a <img> tag may be "img `{ display :img; }`". This simple statement specifies that the `<img>` tag will be displayed as an image.

In order to process CSS rules and generate the CSS customized `Views`, the implementation needs to go through three phases. The first phase is the parsing phase. All CSS style rules must obey the above form. This challenging task is done by the implementation of the standard `StyleSheet` without changes [3]. The significant advantage of the standard implementation is the way it handles non-standard values such as image or sqrt. This provides a convenient way to implement CSS extensions which I will discuss in section 5. The second phase is the middle tier for the cached values in a class called `CSSViewStyles`. This class serves as the broker between a client *`View`* and the `RootStyleSheet` [3]. After the parsing phase, a `Style` object is created by `RootStyleSheet`. The major task for this middle tier class is to translate the queries from the customized `Views`. These Views are typically named as `StyledXXXView` and contain a reference to a sub-class of `CSSViewStyles`. The third phase is the rendering phase that is determined by the customized `Views`. According to the result returned by the queries, `DomElementView` which is a sub-class of `StyledBoxView` will then determine whether or not it should shape or reshape the `View` tree. If it determines to reshape the `View` tree, it will first allocate a new area for the new view `Element` by using the functionalities provided by `ViewElementTree` class that can wrap an existing document tree and group a given sub-tree to a single node. Here is a class diagram (Illustration 5.7) that shows how CSS integrated with customized Views.

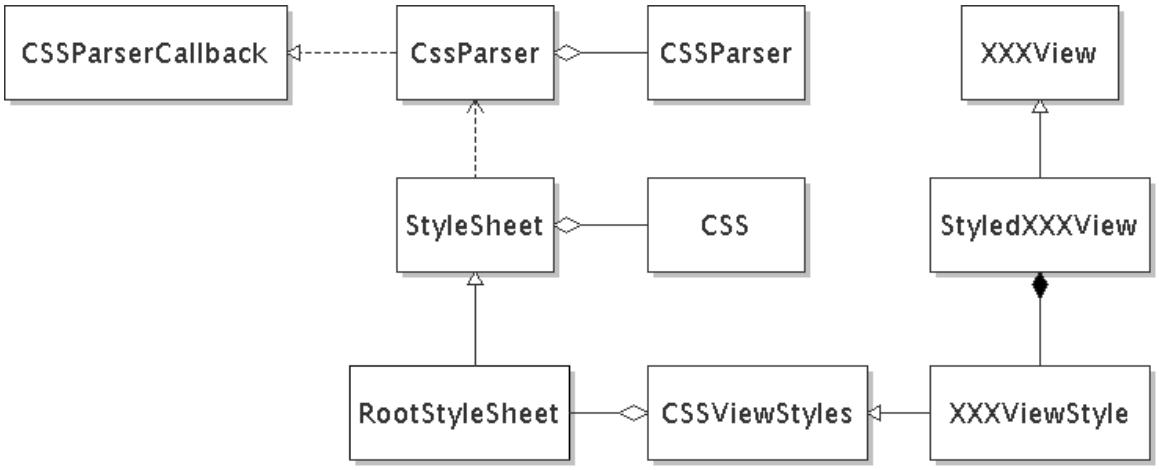


Illustration 5.7.: CSS Integration With Customized Views

As the above example mentioned in previous paragraph, XMLEd supports a subset of CSS specification. However, in order to use the CSS configurable features for MathML and XHTML in XMLEd, a CSS extension must be supported. For example, in the CSS specification, there are seventeen values for property "display". None of them can be used to display the `msqrt` element. In MathML, the `msqrt` element should be displayed as a mathematical expression of a square root. The corresponding CSS rule should be "`msqrt {display: sqrt;}`". However, the value of that display property is not in the CSS specification. Fortunately, the standard implementation of `StyleSheet` can support CSS extensions. The example demonstrates the flexibility and the extensibility of the current framework. I will discuss the CSS extension mechanism in section 6.4.

6. PROJECT WORK

The main focus of this project is to create a test bench that can present a WYSIWYN interface base on the visualization of XHTML+MathML. This project is inherited from the existing framework that was developed by several previous students who contributed enormous effort. At this time, a subset of XHTML+MathML elements has been rendered. The approach demonstrates the current Java Swing editing framework is capable of accomplishing the further implementation.

6.1. Accomplishments

The primary accomplishment is the design and implementation of the editing modes, see section 6.6. All MathML elements implemented here are explained in the following sub-sections in detail. For this test bench, I implemented the most commonly used MathML elements, see table 6.1.

6.2. Visualization of XHTML

The default XMLEd implementation does not support the basic XHTML elements, such as the `img` element. My first attempt was to implement a `StyledImageView` that will display the image specified in `<img>` tag's attribute and the alternative text attribute if there is no image. The `StyledImageView` can handle any image format and refresh itself when the author changes the value of the `src` attribute. The default Swing editing framework is capable of rendering the `<img>` tag. However, the default implementation is too HTML specific. It is not possible to use the default implementation because of the type casting for `HTMLDocument` in the `View` implementation. In order to resolve the conflict, the `StyledImageView` extends the basic `View` class and inherits the robust image loading mechanism of `ImageView`. In addition, the new `View` supports `XMLTextDocument` model and the rendering architecture of XMLEd discussed in section 5.4.2. Here is an example of image view (Illustration 6.1)

Illustration 6.1.: Styled Image

6.3. Visualization of MathML

The rendering of the MathML elements for this project strictly follows the specification defined on the Website of W3C. Many XML and MathML documents are parsed according to their defined DTD and then constructed into an `Element` tree. In [3], the detailed tree construction is well explained. This paper will focus on `View` interpretations for MathML.

Here is the mapping between the implemented MathML elements and their `Views` (Table 6.1):

Elements	Views
mi	StyledMIdentifierView
mn	StyledNumberView
mo	StyledOperatorView
none	StyledNoneView
mprescripts	StyledMPrescriptsView
mspace	StyledMSpaceView
mtext	StyledMTextView
msqrt	StyledSqrtView
math	StyledMathView
mrow	StyledMROWView
mfenced	StyledMFencedView
mfrac	StyledMFractionView
mmultiscripts	StyledMMultiScriptsView
mover	StyledMOverView
munder	StyledMUnderView
munderover	StyledMUnderOverView
msub	StyledMSubView
msup	StyledMSupView
msubsup	StyledMSubSupView
mtable	StyledMTableView
mtr	StyledMTRView
mtd	StyledMTDView

Table 6.1.: Current MathML Elements and Views

The top level of MathML document is the `math` element and its corresponding view is `StyledMathView`. This view is responsible for laying out all its children views both horizontally and vertically and for maintaining a style reference of `CSSViewStyles` to insure the inheritance property of CSS specification. The `StyledMathView` is

constructed with the x axis as its major axis to reduce the implementation complexity. In general, most MathML elements are rendered horizontally in the order in which these elements occur. The Java Swing Framework provides two methods that control the major axis and minor axis layouts. The major axis refers to the axis in which the children are titled and its layout can be modified in `layoutMajorAxis` method. The minor axis refers to the orthogonal axis of the major axis and its layout can be modified in `layoutMinorAxis` method. All layout strategies for MathML elements are performed in these two methods with some extra helper methods. Helper methods are generally used to determine the horizontal base for a mathematical expression. For example, for the following MathML document, the equal operator needs to find its horizontal base and it has to be aligned with the base.

```
<math>
   <mrow>
      <mi>C</mi>
      <mo>=</mo>
      <mrow>
         <mfrac>
            <mn>1</mn>
            <mrow>
               <mi>x</mi>
               <mo>+</mo>
               <mfrac>
                  <mn>1</mn>
                  <mi>y</mi>
               </mfrac>
            </mrow>
         </mfrac>
      </mrow>
   </mrow>
</math>
```

Here is the visual representation of MathML document above, Illustration 6.2

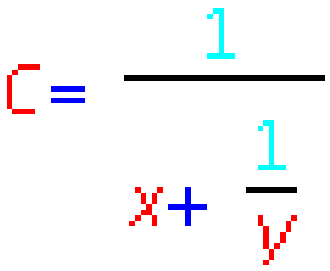


Illustration 6.2.: Base Layout

`Views` for token elements are inherited from `DomTextView` and implements `MathBasicView` interface. The primary purpose of the token `Views` is to display the contents in the `Elements`.

The `StyledMIdentifierView` and `StyledNumberView` are two views for mathematical identifiers and numbers, respectively. They are simply inherited from `DomTextView` to take advantage of `StyledGlyghView` which extends Java Swing Framework's `GlyghView` and has the responsibility to draw the contents.

The `StyledOperatorView` is responsible to render the `mo` element according to the MathML specification of W3C. A simple example is the `StyledOperatorView which` can decide whether or not to grow according to its default value or dictionary. If the operator is a plus sign (+), by default, it will not grow. In contrast, if the operator is a bracket, set by dictionary, it grows. Here is a simple example that will not stretch. A stretchable illustration is in the description of matrix table (`mtable`).

```
<mrow>
    <mo>(</mo>
    <mrow>
      <mn>0</mn>
      <mo>,</mo>
      <mn>1</mn>
    </mrow>
    <mo>)</mo>
</mrow>
```

The above MathML is rendered as the following (Illustration 6.3):

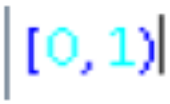


Illustration 6.3.: None Stretchable Operator

The `StyledMTextView` and `StyledMSpaceView` are the views representing arbitrary text itself and a blank space of any size, respectively. They offer users a way to mix text and mathematics with space oriented display. In general, both views are used for commentary text.

The `StyledMRowView` is used to group any number of children into one single element `mrow`. This is particularly useful when multiple operators with their operands have to be treated as a single horizontal row. For example, suppose we have a MathML document with a fraction that contains multiple mathematical operators, identifiers, and numbers for its numerator.

```
<mfrac>
      <mrow>
            <mn>1</mn>
            <mo>+</mo>
            <msqrt>
                  <mn>2</mn>
            </msqrt>
      </mrow>
      <mn>2</mn>
</mfrac>
```

The above example is rendered as Illustration 6.4:

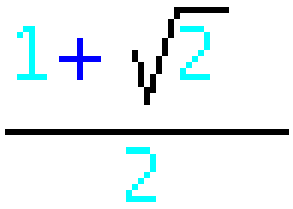


Illustration 6.4.: Grouping Elements

In this case, since mfrac only takes two arguments, namely, numerator and denominator, mrow must be used to group the sub-elements as its numerator. By the specification, mrow should layout the sub-elements horizontally in the order in which they appear. The class StyledMROWView directly inherits from Java Swing Framework's BoxView class instead of StyledParagraphView class to avoid unnecessary modification that is only suitable for plain paragraphs. The default construction for StyledParagraphView uses the y axis as its major axis. It is inadequate for StyledMROWView which lays out its arguments horizontally in general. The rendering of the mfenced element is handled by the StyledMFencedView class which allows arbitrary number of sub-elements. It is similar with the StyledOperatorView class that is responsible for mathematical operators, such as square brackets. In contrast to the StyledOperatorView class, the StyledMFencedView class is a container instead of a basic element and only responsible for braces, brackets, and parentheses, and possibly commas between arguments. Here is a MathML document using mfenced for the similar mathematical expression demonstrated as Illustration 6.3.

```
<mrow>
   <mfenced>
   <mn>0</mn>
   <mn>1</mn>
   </mfenced>
</mrow>
```

The StyledSqrtView class is for the msqrt element. This view will draw the lines that form a square root and grows when its children are inserted. It is a container view inherited from StyledParagraphView which is implemented for a generic XML document. Consider the following part of a MathML document.

```
<msqrt>
   <mn>1</mn>
   <mo>+</mo>
   <msqrt>
      <mn>1</mn>
      <mo>+</mo>
```

```
        <mi>x</mi>
    </msqrt>
</msqrt>
```

The above MathML is rendered as the follow (Illustration 6.5):

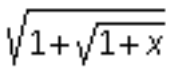


Illustration 6.5.: Square Root

The `StyledMFractionView` is used to form a fraction from its numerator and denominator. This view takes advantage of the `BoxView` class that has the y axis as its major axis and tiles its children along with the major axis. However, for a correct display, two modifications are required. First, the horizontal layout for two subexpressions are not appropriate. The two subexpressions must be horizontally centered. Second, the `paint` method must draw a horizontal line in the middle of the tiled children. The first task is solved by calculating the horizontal start offset. The code below is for this calculation. By default, the alignment value is 0.5 which means to center it. Here is the code that centers the numerator and denominator.

```
int pre = (int)v.getPreferredSpan(axis);
float align = v.getAlignment(axis);
offsets[i] = (int) ((targetSpan - pre) * align);
```

To draw a line between its numerator and denominator is a simple task which can be done by adding the preferred height span for its numerator from y position that is allocated by its parent view. The result is a correct y position for the starting point. The x position can be determined by the x value of the `Rectangle` object allocated for this view. The width of the line is the addition of the width of the `Rectangle` object and both left inset and right inset of the view.

Here is a classic example that demonstrates the use of `StyledMFractionView`. The following MathML document is for quadratic equation and illustration 6.6 shows the rendering result.

```
<math>
   <mfrac>
      <mrow>
         <mrow>
            <mo>-</mo>
            <mi>b</mi>
            <mo>&PlusMinus;</mo>
         </mrow>
         <msqrt>
            <msup>
               <mi>b</mi>
               <mn>2</mn>
            </msup>
            <mo>-</mo>
            <mrow>
            <mn>4</mn>
            <mi>a</mi>
            <mi>c</mi>
            </mrow>
         </msqrt>
      </mrow>
      <mrow>
         <mn>2</mn>
         <mi>a</mi>
      </mrow>
   </mfrac>
</math>
```

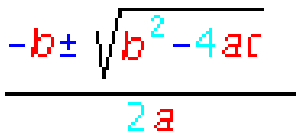


Illustration 6.6.: Quadratic Equation

The `StyledMTableView` is inherited from `StyledTableView`. Therefore, it has the basic rendering features, such as the layout of the columns (`mtd`) elements and the

improved row (mtr) alignment. However, the StyledMTableView implements a fixed matrix view that will not grow when the window is resized. In contrast to StyledTableView, each row view StyledMTRView, is created from the ViewFactory instead of directly created by its parent view StyledMTableView. In this way, StyledMTRView can be fully controlled by CSS style sheet. If the row views are directly created by its parent view, then CSS style rules can not be applied due to the missing style reference. In addition, the StyledMTableView turns the grid lines off, which is not necessary for a mathematical matrix. Here is a sample of MathML document for a matrix and the illustration 6.7 shows the visual representation. Notice that, the surrounding parentheses are also stretched to the height of the matrix.

```
<math>
   <mrow>
      <mi>A</mi>
      <mo>=</mo>
      <mo>(</mo>
      <mtable>
         <mtr>
            <mtd>
               <mn>0</mn>
            </mtd>
            <mtd>
               <mn>1</mn>
               <mo></mo>
               <mi>&lambda;</mi>
            </mtd>
         </mtr>
         <mtr>
            <mtd>
               <mn>1</mn>
            </mtd>
            <mtd>
               <mn>0</mn>
            </mtd>
```

```
            </mtr>
            <mtr>
               <mtd>
                  <mn>1</mn>
               </mtd>
               <mtd>
                  <mn>1</mn>
               </mtd>
            </mtr>
         </mtable>
         <mo>)</mo>
      </mrow>
      <mspace>   </mspace>
      <mi>&lambda;</mi>
      <mo>&ne;</mo>
      <mn>0</mn>
</math>
```

The above MathML document is rendered as:

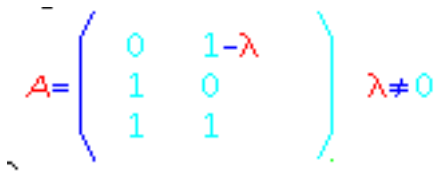


Illustration 6.7.: A Matrix

MathML provides seven specialized scripting elements:

- `msup` for superscript (Illustration 6.8)
- `msub` for subscript (Illustration 6.8)
- `msubsup` for sub and sup script (Illustration 6.9)
- `mover` for over script (Illustration 6.10)
- `munder` for under script (Illustration 6.10)
- `munderover` for under and over script (Illustration 6.10)
- `multiscripts` for multi-scripts (Illustration 6.11)

Some examples for the scripting elements above are presented here.

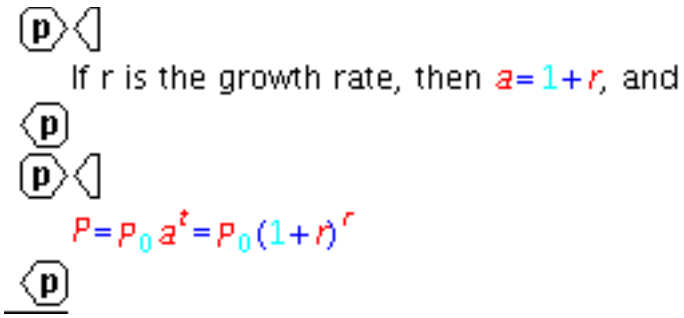


Illustration 6.8.: Subscript and Superscript

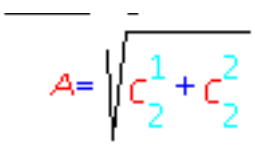


Illustration 6.9.: SubscriptSuperscript

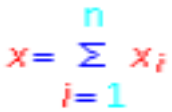


Illustration 6.10.: Over Under

The semantic meaning for first three scripts are well known in a common editor such as Word. The next three scripts are used to place embellishments above or below the base. The last one allows pairs of subscript or superscript vertically aligned around one base expression. By convention, postscripts are placed to the right of the base in visual representation and prescripts are placed to the left of the base in visual representation. Because prescripts are rarely used, they are always placed after postscripts whenever necessary. The empty element for pairs of scripts are represented by `none` element. Here is a sample document with `multiscripts` and illustration 6.11 shows the visual representation.

```
<math>
    <mrow>
        <mi>H</mi>
        <mo>=</mo>
        <mrow>
```

```
            <mmultiscripts>
               <mi>F</mi>
               <mi>i</mi>
               <mi>j</mi>
               <mprescripts/>
               <mi>k</mi>
               <none/>
            </mmultiscripts>
         </mrow>
      </mrow>
</math>
```

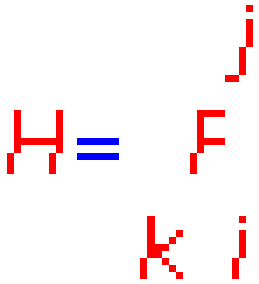


Illustration 6.11.: Multipscript

The layout algorithms for these scripts are done in the `layoutMinorAxis` and `layoutMajorAxis` methods. Because `mmultiscripts` accepts any number of pairs of sub/superscripts with optional `mprescripts` element, it requires additional computations to handle the layout position according to the existence of `mprescripts` element. If `mprescripts` is absent, then the indexes with odd number must be subscripts and the indexes with even number must be superscripts. Otherwise, the same rule only applies these pairs of sub/superscripts before `mprescripts`. For these pairs of sub/superscripts after `mprescripts`, the indexes with even number must be subscripts and the indexes with odd number must be superscripts.

6.4. CSS Extensions

The CSS specification from W3C does not support the rendering of MathML values of the display property. Therefore, it is necessary to create a CSS extension to handle the XHTML+MathML elements if the default specification does not support the language

specific elements. For example, in section 3.2.3, we see the CSS style rule: `msqrt {display: sqrt;}`. Because the rule of "`display:sqrt`" is not supported by the CSS specification, it is implemented in this project to map the `StyledSqrtView` for the `msqrt` element. The new implementation cannot be achieved without the concrete classes of `CSSViewStyles`, which acts as a broker between the client views and the style sheet. The default CSS parser forwards the values not defined by the specification to a generic CSS value. Therefore, `CSSViewStyles` wires up the new style rule with its associated client view. First, the `DomElementView` class asks a `CSSViewStyles` object for its cached value by calling a boolean function, for example, asking for `cachedline` by calling `isInline()` method. According to the returned value, the `DomElementView` class decides whether or not to reshape the view `Element` tree. In the `CSSViewStyles` class, it connects to the CSS style rules by refreshing the cached values. For example, here is the code to refresh the `cachedSqrt` in `refreshCachedProperties` method:

```
if(booleanValue(CSS.Attribute.DISPLAY,"inline-sqrt"))
{
   cachedInlined = true;
   cachedSqrt = true;
}
else
{
   cachedInlined = booleanValue(CSS.Attribute.DISPLAY,"inline");
   cachedSqrt = booleanValue(CSS.Attribute.DISPLAY,"sqrt");
}
```

The code above will refresh the values of `cachedInlined` and `cachedSqrt` and return the result to the client views that request the values. This similar technique applies to all customized property values to map their corresponding `View` objects.

6.5. XPath

A common feature for an XML editor is to show the current XPath on a small panel to help users keep track of their current editing location. XPath is a way to describe the XML document's logic structure by addressing a path that contains the current node and all its parents. The XMLEd implementation only displays the current cursor position in `Document` model. While the cursor position may be helpful during a debugging phase,

it provides no meaningful information to a user. For a user friendly interface, it is useful to view the current XPath during editing. The challenge of implementing XPath status is to locate the current element where the cursor is in. Once the current element is found, it is simply passed to `getXPath` helper method to find all its parent. By using a recursive call, it is not hard to locate the current element. Here is the `locateCurrentElement` method that does this recursion and illustration gives an example.

```
if (!inFocus)
   return;
int n = elem.length;
for(int i = 0; i < n; i++)
{
   int start = elem[i].getStartOffset();
   int end = elem[i].getEndOffset();
   if(dot >= start && dot <= end)
   {
      if(elem[i].isLeaf())
      {
         current = (XMLTextElement)elem[i];
         return;
      }
      else
         locateCurrentElement(((XMLTextElement)elem[i]).get
         Children());
   }
}
```

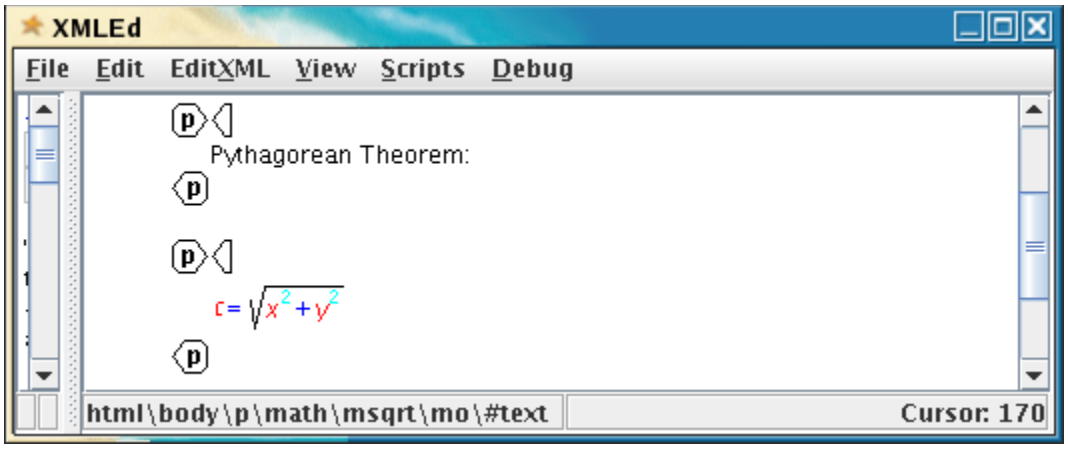


Illustration 6.12.: XPath Example

6.6. What You See Is What You Need (WYSIWYN) Editing Interface

XMLEd has two main views: a simple source view and a GUI view. The simple source view supports basic editing operations. In addition, it provides structural editing and navigation features with CSS control discussed in [3,8,9]. Similar to the plain text view, the styled view supports all basic editing operations, such as copy, paste, and cut. In contrast, the style view is a view that has hidden in-line tags which can maintain enough visual representation during editing. In this view, there are three modes to provide variant degrees of precise control with different levels of visual representation: Full Tag Mode, Dot Mode, and Zoom Mode. In the full tag mode, all hidden tags are shown, optionally all attribute marks (the small triangles next to tag names) can be hided to provide a better representation. The full tag mode might not be useful if there is a large number of tags. Illustration 6.13 shows the mathematical expression sinΘ = y / x in full tag mode.

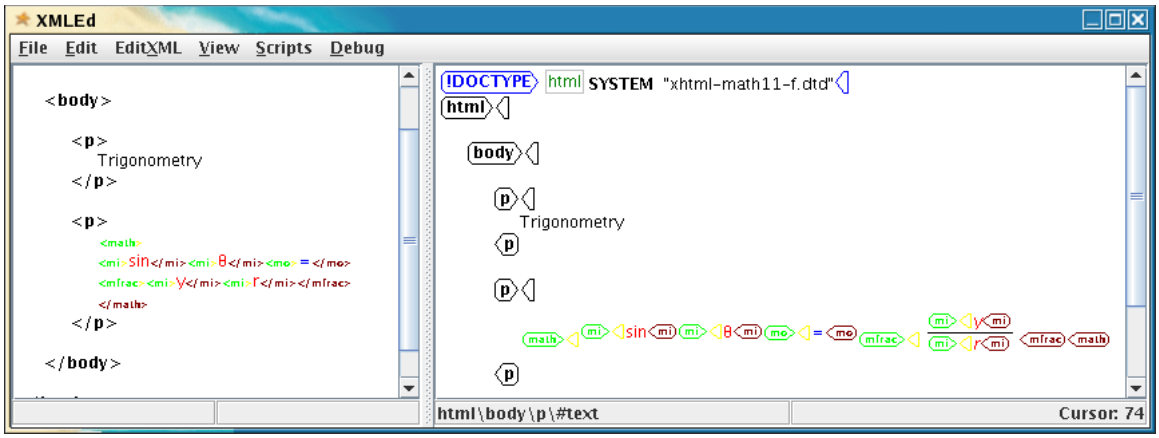


Illustration 6.13.: Trigonometry With Full Tags

In the Dot Mode, all the hidden tags are displayed as tiny dots. By having these dots, experts can avoid surprises by precisely navigating cross each element and view the current position at the status bar and still maintain visual representation. Illustration 6.14 gives an example of mathematical expression in Dot Mode.

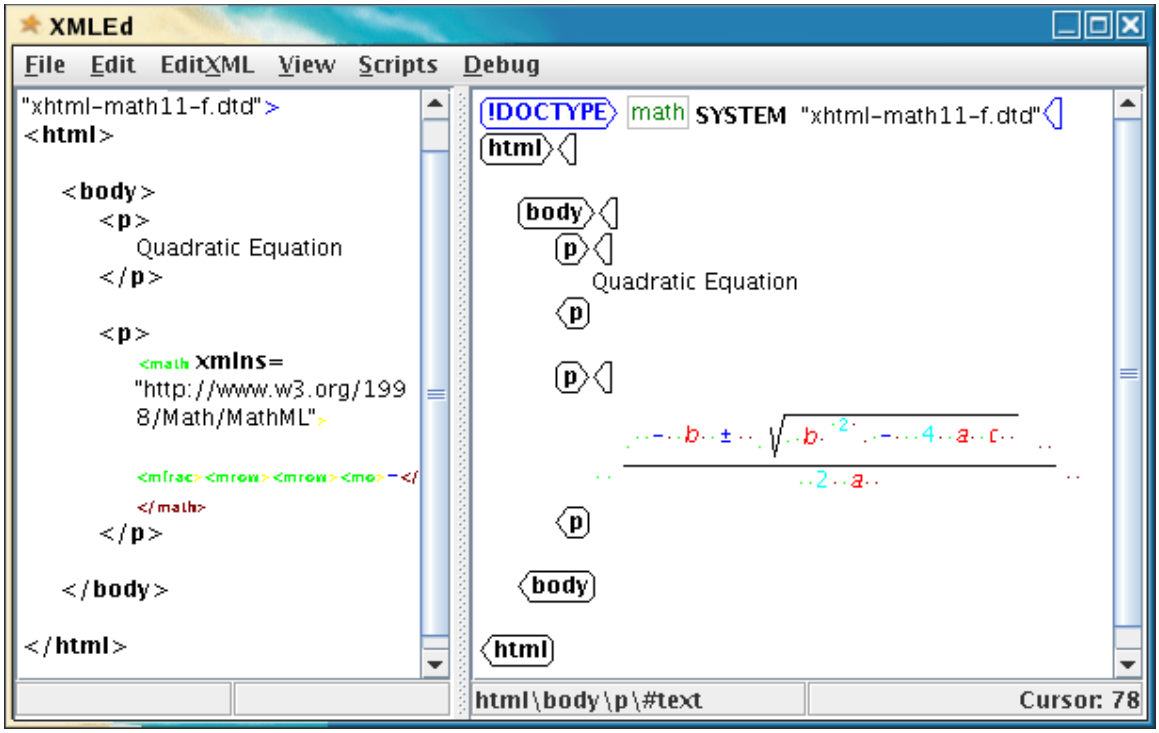


Illustration 6.14.: Quadratic Equation In Dot Mode

It is very important to know where you are during editing. For example, suppose you want to change b^2 to into b^{12}, you want to make sure to add 1 to the exponent, not b. In this case, the dot notation can be very helpful.

For a large mathematical document, if all mathematical expressions used the full tag or the dot notion, the document itself would be difficult to navigate. In this case, Zoom Mode can provide a more user friendly interface that combines precise control and the benefits of a WYSIWYG editing interface. In this mode, all mathematical expressions are rendered in WYSIWYG fashion, with the exception of the structure that is currently being edited. When the cursor position is moved into a MathML element, the surrounding hidden tags for the current position expand to expose its detailed structure. If the current cursor position is moved out, the tags will collapse together to its previous visual representation. Users can control the level of expansion and the tag style (dot/full tag/full tag with a mark). Optionally, users can also deactivate/activate this “zoom mode”. Illustration 6.15 shows the start of Zoom Mode.

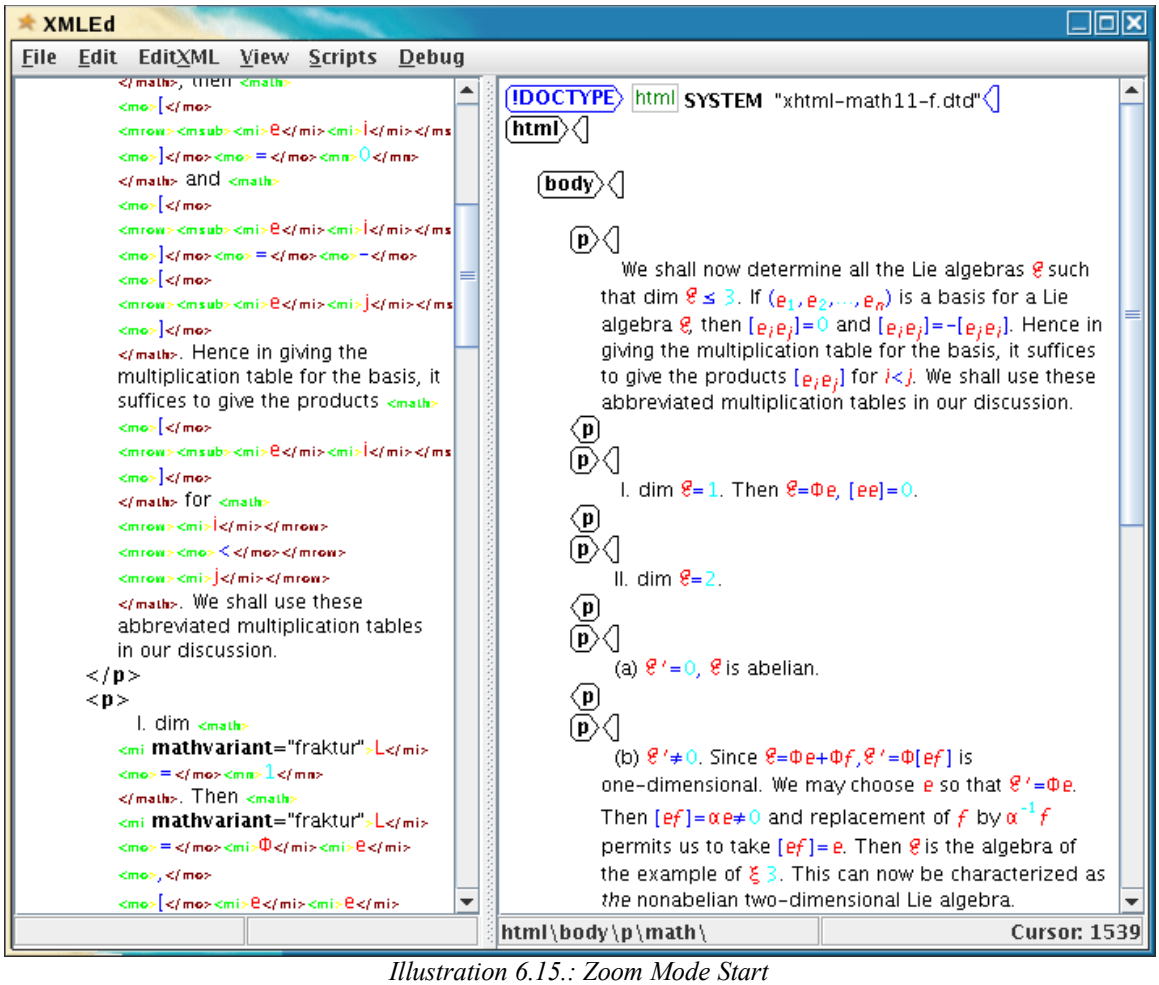


Illustration 6.15.: Zoom Mode Start

Users can zoom in any mathematical expression by moving the cursor position to a MathML element. Illustration 6.16 is an example with the display of the local structure.

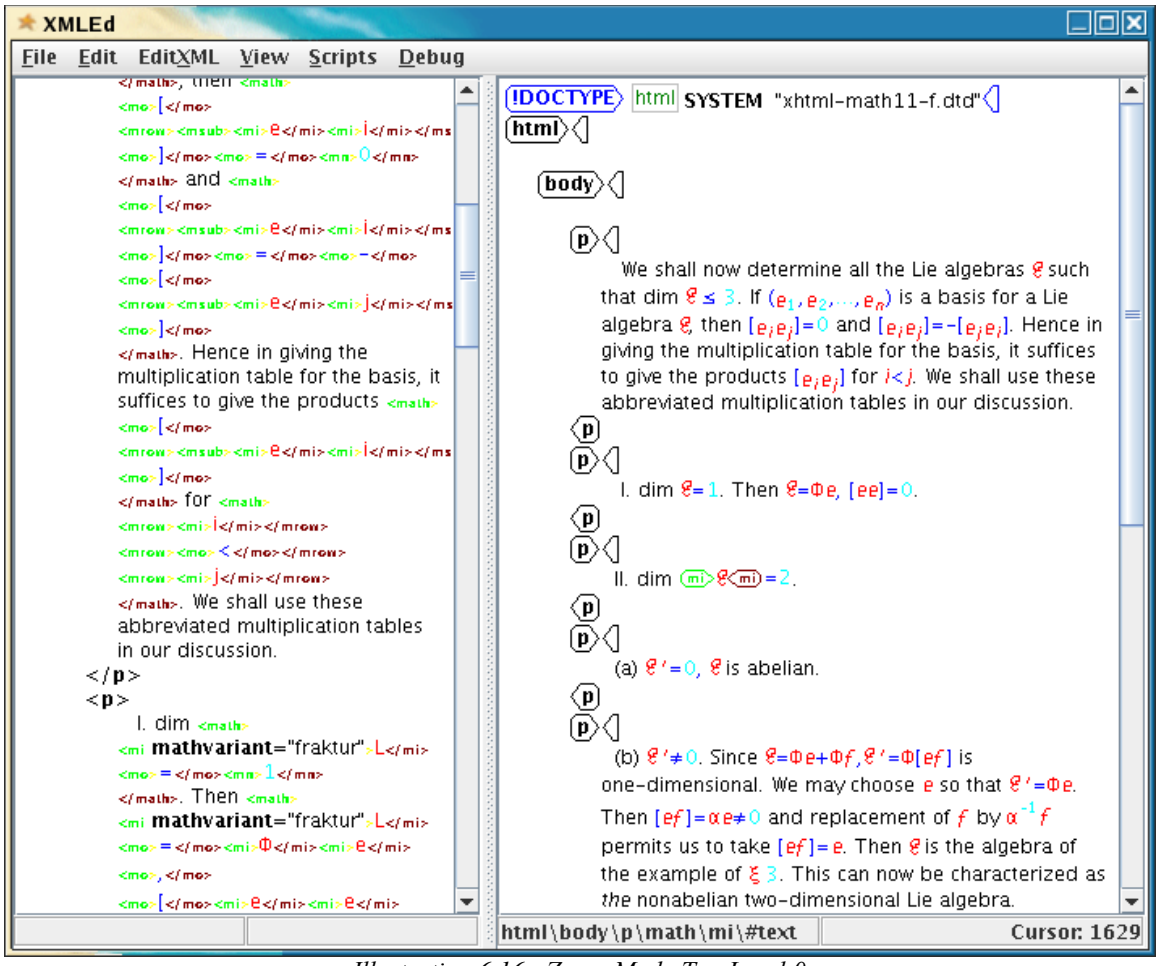


Illustration 6.16.: Zoom Mode Tag Level 0

A MathML structure level describes how deep a element node spans. For example, the following MathML document has a level of two. And all token elements, such as `mi`, `mo`, and `mn`, has a level of zero.

```
<math>
      <msqrt>
            <mi>x</mi>
      </msqrt>
</math>
```

Since the `math` element has a child `msqrt` and the `msqrt` element has a child `mi`, the `math` element has a level of two and the msqrt element has a level of one. The `mi` element is a token element, so the level for the `mi` element is zero.

In XMLEd, users can set the magnification level for ("zoom mode"). If the selected level is greater than the maximum level of the current `math` element, then the selected level is set to its maximum level. Illustration 6.17 shows the expanded tags with its maximum level.

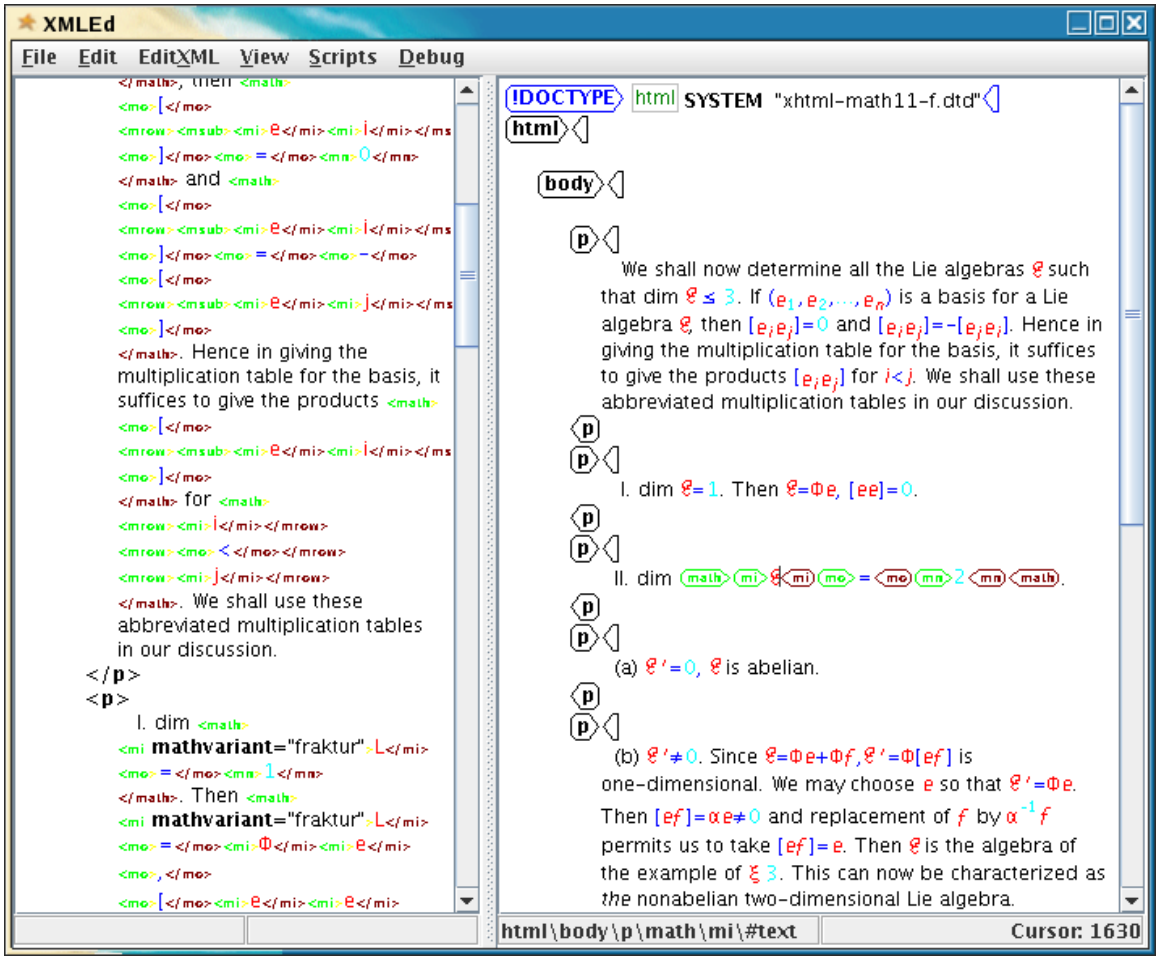


Illustration 6.17.: Zoom Mode Tag Maximum Level

Here is a same document with its local structure displayed in dot style (Illustration 6.18).

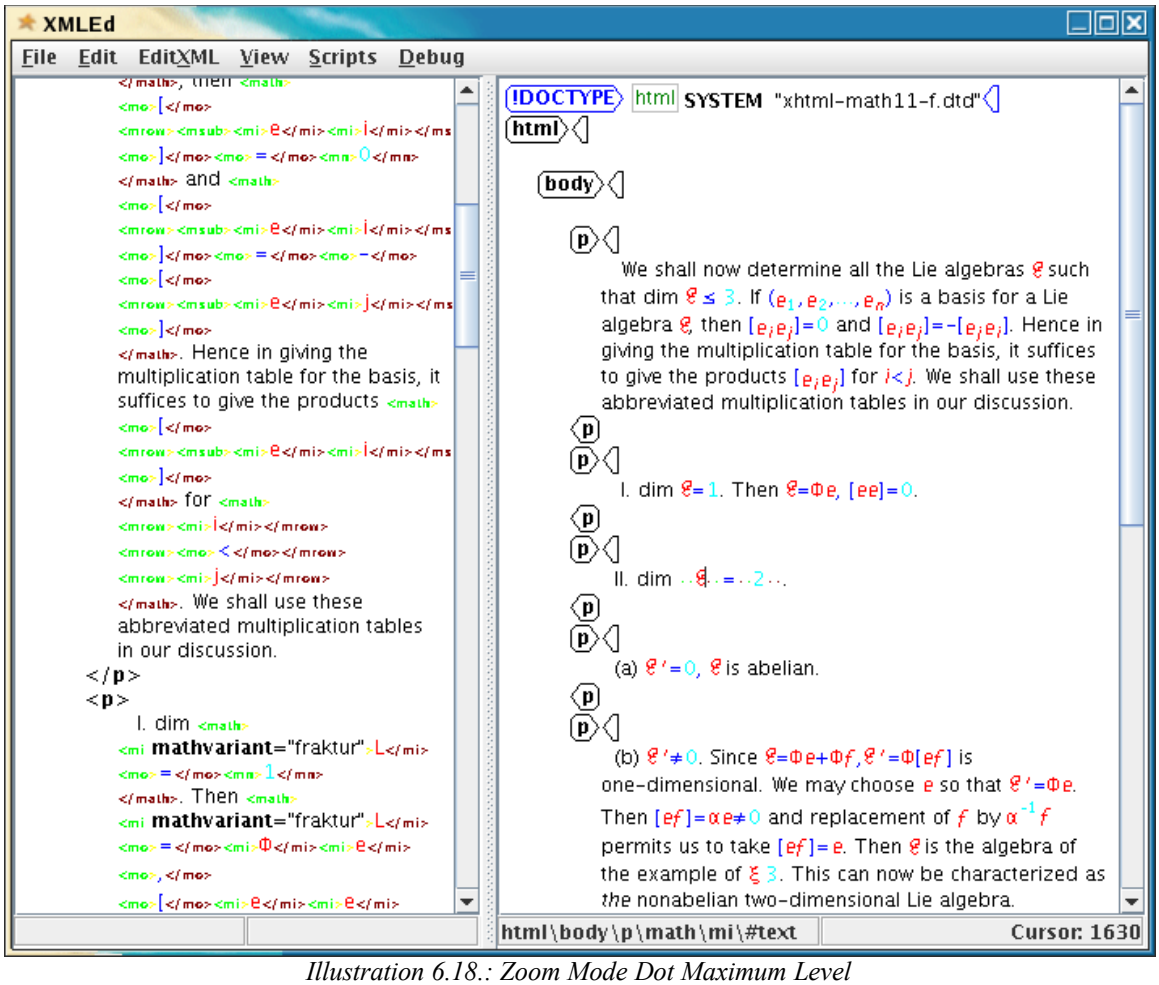


Illustration 6.18.: Zoom Mode Dot Maximum Level

Here, by showing enough structure, now experts can view any ambiguous position and move the cursor into an exact position they need. In other words, experts can avoid unexpected errors and still be able to visualize the mathematical expressions to avoid confusion. WYSIWYN Interface Mode Structure is summarized as below:

- Full Tag
- Dot Mode
- Math
  - Show Local (On/Off)
    - On
      - Local Full Tag Style
        - Show Start Tag Mark

        - Local Icon Style (dot)
        - Set Expansion Level (default is 0)
    - Off

To insert a MathML tag, select one of the MathML elements listed on the menu EditXML or select “Insert Markup Tag” from EditXML menu and use one of the MathML elements from the combo box.

Some useful commands are listed here. Users can also use View menu for the commands below, but a highly skilled expert always performs better with a keyboard-oriented interface:

1. Switching Modes:

Commands	Description
CTRL + M	Zoom Mode
CTRL + D	Dot Mode
CTRL + T	Full Tag Mode

Table 6.2.: Mode Switching

2. Commands for Zoom Mode.

Commands	Description
CTRL+G	Change local tag style to full tag
CTRL+I	Change local tag style to dot
CTRL+L	Freeze the current view or Deactivate/Activate expansion feature
CTRL+K	To show start tag mark (showed with local full tag, otherwise ignored)
ALT+Minus	One more expansion level down
ALT+SHIFT+EQUAL	One more expansion level up
ALT+0	Set expansion level to 0
ALT+1	Set expansion level to 1
ALT+2	Set expansion level to 2
ALT+3	Set expansion level to 3
ALT+4	Set expansion level to 4
ALT+5	Set expansion level to 5
ALT+6	Set expansion level to 6
ALT+7	Set expansion level to 7
ALT+8	Set expansion level to 8
ALT+9	Set expansion level to 9

Table 6.3.: Zoom Mode Commands

3. You can also quickly insert one MathML element by pressing these function keys, for example, press F1 to insert a `math` element, F2 to insert `mi` element. These commands are also available from EditXML menu.

Function Keys	Description
F1	math
F2	mi
F3	mn
F4	mo
F5	msub
F6	msup
F7	mfrac
F8	msqrt
F9	mtable
F10	mtr
F11	mtd
F12	mrow

Table 6.4.: Function Keys For Frequently Used MathML Elements

The implementation of the new approach was achieved by carefully analyzing the previous design and code walk through for hundreds of methods. Here is the logic steps I used to approach the solution. The algorithm contains the following phases:
Phase 1: Find the current view which the cursor is positioned. It requires a recursive call to traverse the `View` hierarchy by comparing the start offset and the end offset.
Phase 2: Locate the start offset of `STagName` view and the end offset of `ETag` for the current view. Each MathML element is mapped to its corresponding *`View`* object which is always constructed with a `STagName` view for a part of the start tag name (≤math), a `STagMark` view for the (&gt); character in its start tag name, and a `ETag` view for the entire end tag (&lef;/math>). Optionally, it has a view between `STagName` and `STagMark` for an attribute table. Since the fat attribute table looks ugly in mathematical expressions, it is suppressed from visual representation. Illustration 6.19 gives an example of these views in its view hierarchy.

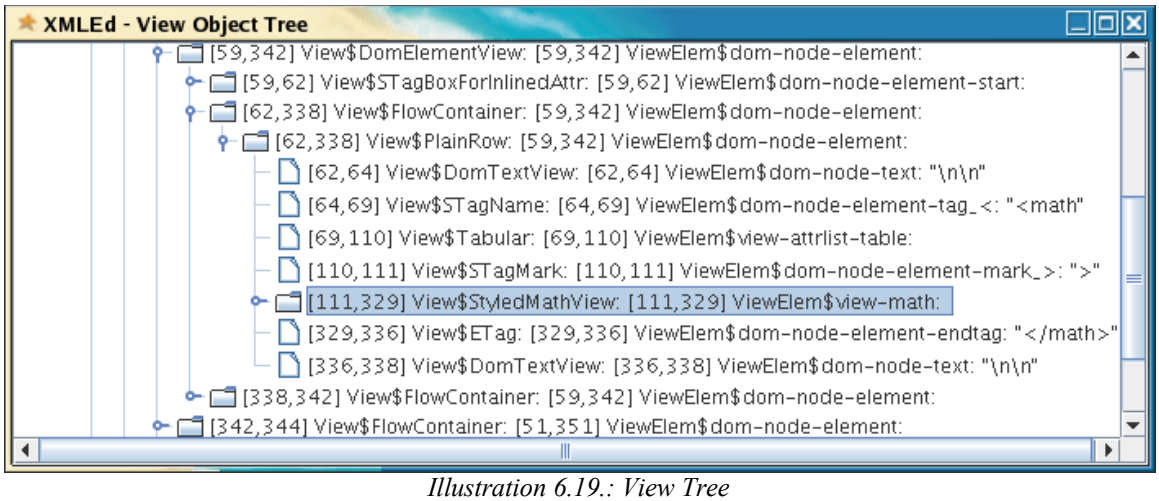


Illustration 6.19.: View Tree

Phase 3: For each type of view, sets its visibility which depends on whether or not the cursor is between the start offset of `STagName` and the end offset of `ETag` or other criteria and revalidates the current view.

Phase 4: Clean up. If the cursor moves horizontally, `PlainRow` view can correctly handle its visibility changes. But if the cursor moves vertically, the tags displayed in the former `PlainRow` need to be explicitly removed.

7. USABILITY STUDY

Technically speaking, usability is not just about the visual part of the user interface components. It is about the interaction part of the system that helps the user to achieve specified goals with efficiency and accuracy. Usually, usability relates how the system interacts with the user and it includes five basic attributes: learnability, efficiency, user retention over time, error rate, and satisfaction [14]. The main reason for studying usability is to increase user productivity and still maintain accuracy. Here, this report presents a usability evaluation performed during the test of one approach to the WYSIWYN editing interface. The goal of this study is to test whether the WYSIWYN interface increases accuracy and speed. The new interface was tested by nine students from Dr. Horstmann's classes. The spirit of this usability test was focused on the modification aspect of a conventional WYSIWYG editing interface. A comparison between the WYSIWYN interface used by XMLEd and the WYSIWYG interface used by Amaya, for instance, is quite sufficient to perform. First, the students can install both

XMLEd and Amaya for free. Second, Amaya has a refined WYSIWYG editing interface that was designed to alleviate the problem of creating and editing complex mathematical documents. However, experts often find it is difficult to have precise control. Last but not least, both interfaces are easy to learn. Therefore, these students can accomplish the considered tasks during a limited time constraint.

The factor of these participants' experience with XML technology and the Amaya editing tool can dramatically change the outcome of the evaluation. For example, a highly skilled typist might perform better with a keyboard-oriented interface, while a low-skilled typist might do better with a GUI [17]. To make a fair comparison, participants were required to study some background materials for some basic concepts and terminologies, such as, MathML and different views (or modes) for both editing interfaces. Also, I designed a questionnaire sheet to collect the participants' profiles for further task analysis. Sometimes, it is called user analysis. Task analysis describes a set of techniques people use to get things done [16]. For example, the participants record their keystrokes. The user analysis is taken as an input for task analysis [17], and both the task analysis and the user analysis are jointly performed. This combined strategy or methodology for user analysis and task analysis is often called contextual inquiry [15].

Here, I designed two tasks to prove or disprove my approach to the WYSIWYN editing interface. The first task focused on modification aspect of both editing interface. The participants need to locate a specific position in a mathematical document and insert a `mi` element between another `mi` element and a `mo` element. This is a common editing operation for any MathML editors. For example, illustration shows the visual representations of the document before and after the changes.

BEFORE:

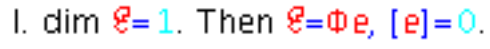


Illustration 7.1.: Task 1 Before

AFTER:

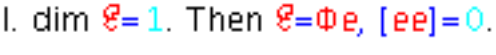


Illustration 7.2.: Task 1 After

The second task focused on the insertion to demonstrate the basic ability for editing operations. This analysis is only interested in the evaluation of the first task because it demonstrates the spirit of the innovative interface which makes editing operations unambiguous and recognizable.
Here is the summary of the testing results for both tasks:

Task 1: XMLEd Group

Name	ID	XMLEd Time	XML Experience	XMLEd Experience	Correctness
HT	1	4	Low	No	Correct
BT	2	5	Low	No	Correct
TM	3	20	Low	No	Correct
SL	4	5	Low	No	Correct

Illustration 7.3.: Task 1 XMLEd Group

Task 1: Amaya Group

Name	ID	Amaya Time	XML Experience	Amaya Experience	Correctness
SP	5	10	Low	No	Not Correct
JM	6	11	Medium	No	Not Correct
TC	7	Incomplete	Low	Yes	Not Correct
DB	8	2.5	Low	Yes	Not Correct
GJ	9	2.5	Medium	Yes	Not Correct

Illustration 7.4.: Task 1 Amaya Group

For the first task, four of them used XMLEd and five of them used Amaya. For the XMLEd group, three students who used XMLEd completed the task in just 4 to 5 minutes with the correct answer and only one took 20 minutes. In another hand, for the Amaya group, two of them claimed they completed in 10 to 11 minutes, but the answer was incorrect. And two of them who had Amaya experience claimed they spent only 2.5 minutes, but the answer was incorrect. One student with Amaya experience produced that was incomplete. The result suggests that the WYSIWYN editing interface can provide efficiency and accuracy over the conventional WYSIWYG editing interface for modification aspect. In another hand, it demonstrated that the conventional WYSIWYG editing interface may give users the illusion that they completed the task which is

incorrect. This situation may be even worse than incompleteness. The errors concealed by this mistake may be a potential defect tomorrow.

Task 2: XMLEd Group

Name	ID	XMLEd Time	XML Experience	XMLEd Experience	Correctness
HT	1	15	Low	No	Correct
BT	2	30	Low	No	Correct
TM	3	20	Low	No	Correct
SL	4	12	Low	No	Correct

Illustration 7.5.: Task 2 XMLEd Group

Task 2: Amaya Group

Name	ID	Amaya Time	XML Experience	Amaya Experience	Correctness
SP	5	30	Low	No	Correct
JM	6	11	Medium	No	Correct
TC	7	3	Low	Yes	Correct
DB	8	10	Low	Yes	Correct
Glenn Jahnke	9	7	Medium	Yes	Correct

Illustration 7.6.: Task 2 Amaya Group

For the second task, the same four students used XMLEd and the same five students used Amaya. This task is just to create a quadratic equation. It aims to test the basic editing operations and to show that XMLEd is capable of creating common mathematical expressions with a reasonable amount of effort. For the XMLEd group, two students completed the task in 20 to 30 minuts and another two students completed the task in 12 to 15 minutes. In another hand, the result from Amaya group is quite interesting. The first student who did not have Amaya experience and had a low XML experience finished the task in 30 minutes. But the second student with the similar condition, but with a medium XML experience, was able to finish the task in just 11 minutes. And the three students who had Amaya experience were all able to complete the task faster. Comparing to the first student, the WYSIWYG editing interface can dramatically reduce the task completion time after users became adapted to it. But for the XMLEd group, there are no experienced users. Therefore, there are no large variations for the task completion time.

8. CONCLUSION

During implementation, I encountered two issues. First, for a generic XML editor, there is always an attribute table next to its corresponding element tag. For MathML, this is not the case because users do not want to see a fat attribute table in mathematical expression. Therefore, I had to do some reverse engineering to hide existing attribute tables and take care of cursor navigation. Second, this project implemented a number of hot keys to facilitate the usage for experts. I successfully mapped each key combination to the corresponding actions, except one, the "+" key. I used VK_MINUS to map "-" key and I easily succeeded. So logically, I thought that I should use VK_PLUS to map SHIFT and "+" combination. Unfortunately, it never worked. Instead, I had to use VK_EQUALS and SHIFT_DOWN_MASK combination to make it work. The API document does not adequately explain this.

At the end, the implementation shows that the current Java Swing editing framework is capable of accomplishing the task of the visualization of XHTML+MathML. Instead of simply implementing the rendering for XHTML+MathML, I have accomplished the following interesting, challenging, and intriguing works for this project.

- I have invented a new approach to the WYSIWYN interface which combines precise control and the benefits of the WYSIWYG interface. Users will be able to avoid unexpected errors by switching to different editing modes and still maintain visual representation. This new navigation model is not available in existing XHTML+MathML authoring tools.
- I have designed and carried out a usability study. This usability study suggests that the WYSIWYN interface can increase accuracy and speed for modification aspect.
- I have rendered the most commonly used MathML elements (see table 6.1).
- I have implemented a number of hot keys for a keyboard-oriented interface (see table 6.2, 6.3, 6.4).
- I have extended the CSS package of XMLEd framework. As a result, XHTML+MathML views are controlled by CSS technology. Without this

enhancement to XMLEd framework, XHTML+MathML support would not be possible.

- I have added a new feature that allows users to view XPath for XML documents. This feature improves readability and helps experts to track cursor position.

The implementation of this project is built on the top of a combination of Java Swing's Model-View-Controller editing framework and Tong Ho's generic XML editor. There are 275 classes to analyze, over 20000 lines of code to understand. Also, the Swing text editing framework lacks documentation because it has found little usage beyond the implementation of the Swing HTML and plain text controls. In summary, the work of this project has demonstrated a non-trivial approach to the ambiguity nature of the conventional WYSIWYG interface.

REFERENCES

[1] Coombs, J. H., Renear, A. H., and DeRose, S.J. Markup systems and the future of scholarly text processing. *Communications of the ACM*, v.30, n.11, p.933-947, November 1987.

[2] D. D. Cowan, E. W. Mackie, G. M. Pianosi, and G. de V. Smit. Rita – an editor and user interface for manipulating structured documents. *Electronic Publishing*, v.4, n.3, p.125-150, September 1991.

[3] Ho, Tong. A Scriptable XML Editor With Multiple CSS Configurable Views. *CS298 Final Report*, May 2005

[4] Furuta, R., Vincent Quint, V., and André, J. Interactively Editing Structured Documents. *Electronic Publishing*, v.1, n.1, p19-44, 1988.

[5] Hammer, M., Ilson, R., Anderson, T., Gilbert, E., Good, M., Niamir, B., Rosentein, L, and Schoichet, S. The implementation of Etude, an integrated and interactive document production system. In *proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, Portland, June 1981.

[6] Quint, V. and Vatton, I. Techniques for authoring complex XML documents. In *Proceedings of the 2004 ACM symposium on Document engineering*, Milwaukee, October 2004.

[7] Strömfors, O. and Jonesjö, L. The implementation and experiences of a structure-oriented text editor. In *Proceedings of the ACM SIGPLAN SIGOA symposium on Text manipulation*, Portland, June 1981.

[8] Pradhan, Nupura. Inline XML authoring and validation. *CS298 Final Report*, May 2004.

[9] Pathak, Swati. XML editor commands with multiple undo/redo. *CS298 Final Report*, May 2004.

[10] William A. Martin. Syntax and display of mathematical expressions. *Technical Report AI Memo 85*, MIT, July 1965.

[11] Steve Smithies, Kevin Novins, and James Arvo. A handwriting-based equation editor, April 2006. http://www.cs.queensu.ca/drl/ffes/papers/smithies_GI99.pdf

[12] Prinzing, T. Using the Swing text package. The Swing Connection, October 2004. http://java.sun.com/products/jfc/tsc/articles/text/overview/index.html

[13] http://www.w3.org/TR/MathML2/, April 27, 2006

[14] Xavier Ferre, Natalia Juristo, Helmut Windl, and Larry Constantine. Usability basics for software developers. *Software IEEE*, v18, issue 1, p22-29, Jan.-Feb. 2001.

[15] H. Beyer and K. Holtzblatt, Contextual Design: A Customer-Centered Approach to Systems Design, Morgan Kaufmann, San Francisco, 1997.

[16] J. Preece et al., Human-Computer Interaction, Addison-Wesley Longman, Reading, Mass., 1994.

[17] Deborah J. Mayhew. The usability engineering lifecycle: a practitioner's handbook for user interface design. Morgan Kaufmann Publishers, Inc. 1999.

[18] http://www-sop.inria.fr/mimosa/Manuel.Serrano/publi/jfp05, January 16, 2007