Authoring Presentation for OpenMath

Shahid Manzoor, Paul Libbrecht, Carsten Ullrich, and Erica Melis

Deutsches Forschungszentrum für Künstliche Intelligenz
{manzoor, paul, cullrich, melis}@activemath.org

Abstract. Some mathematical objects can have more than one notation. When a system compiles mathematical material from multiple sources, a management effort to maintain uniform and appropriate notations becomes necessary. Additionally, the need arises to facilitate the notations editing of the mathematical objects with authoring tools. In this paper, we present our work towards those needs. We have designed a framework that defines an authoring cycle supported by series of tools, which eases the creation of notations for the symbols in the process of publishing mathematics for the web.

1 Introduction

ActiveMath [MAB+01] is an adaptive and interactive web-based learning environment for mathematics. It dynamically generates content adapted to the students profile i.e. goals, preferences (e.g. personalized presentation and field of interest etc), capabilities, and knowledge. The mathematical content in ActiveMath is represented in `OMDoc` [Koh04] and are stored in a knowledge base. The motivation for our work comes from a number of problems we experienced with authors when they write mathematical course material in the ActiveMath environment. For instance, in different languages, different mathematical notations for a symbol are used: the slope symbol in English is written as $slope(F, p)$, or $steigung(F, p)$ in German. Moreover, different authors want to use different notations for the same mathematical objects such as $\frac{1}{2}$ or $1 : 2$, $a * b$ or ab, $\frac{d}{dx}f$ or $f'x$.

However, authors are challenged when writing the presentation of the mathematical expression. The current approaches provide a *meta stylesheet* (an XML encoding to represent the notations for the symbols) as an authoring support, which is converted into XSLT-templates. That is, the authors lack authoring tools which ease their notations editing and tools to ease the publishing tasks.

We propose a framework that defines an authoring cycle for the editing of symbols' notations which involves a series of tools that support the process from editing to previewing the notations. This framework simplifies the symbols' notation authoring process.

We start with a description of the current approaches for authoring OpenMath symbols' notations. Thereafter, we describe problems in these authoring processes. In Sec. 4, we explain the XML encoding for the notation and annotations. We discuss our authoring tools and environment for the notations editing.

Finally we describe how the notations are processed in the presentation architecture of ActiveMath.

2 Previously Existing Approaches to Presentation Generation for OpenMath

MathML is a W3C standard for representing 2-dimensional mathematical formulæ on the web. It has been embedded inside xhtml. It comes with two languages, Presentation MathML (PMML) and ContentMathML-content. The PMML concentrates on the presentation of the expression. On the other hand, MathML-content organizes mathematical formulæ in trees of operators, variables, and numbers with well defined semantics. Its set of possible symbols is, however, fixed.

OpenMath's primary goal is to serve as a communication of mathematical objects between applications. Similarly to MathML-content, it organizes formulæ in trees of mathematical symbols. Contrary to MathML-content, OpenMath has a well defined extensibility mechanism: one can write content-dictionaries (CD) to provide a description of new symbols.

Generally, xslt is used to transform the OpenMath objects into the output formats, such as html or TeX. In [Car00] and [Koh04], an xslt based algorithm is described to generate the presentation. There, an template rule is required for each symbol. Each template generates the notation in output format recursively. The match rule for each template is built with `name` and `cd` attributes. Below is an example of an xslt template for the `divide` symbol.

```
<xsl:template match="om:OMA[om:OMS[position()=1 and
@name='divide'\break and @cd='arith1']]">
  <mfrac>
    <xsl:apply−templates select="∗[2]" />
    <xsl:apply−templates select="∗[3]" />
  </mfrac>
</xsl:template>
```

`OMDoc` [Koh04] provides a `<presentation>` element to write notations aiming to facilitate the authoring support, as hand written xslt templates are tedious and error-prone. The `<presentation>` element points to the symbol for which the presentation is being written. It uses the `<style>`, `<xslt>` and `<use>` elements to generate a particular presentation of a symbol. For complex notations such authoring requires is defining the body of an xslt template. For simple notation binary operators, subscripts, list, and fraction, the easy syntax of the `<use>` element is sufficient. This approach supports the xslt templates generation for multiple output formats. But, authors have to specify the notation for each output format.

In Fig. 1, the example of `<presentation>` element represents the notation for the `divide` OpenMath symbol in three output formats. Each `<use>` element represents the notation, specified as character data for html (/) and LaTeX (`\frac`) or element attribute for MathML (mfrac).

```
<presentation role="applied" for="divide" theory="arith2">
  <use format="html">/</use>
  <use format="mathml" element="mfrac" />
  <use format="latex">\frac</use>
</presentation>
```

Fig. 1. Example of presentation tag in `OMDoc`

Another approach is discussed in [NW01], in which Naylor and Watt have introduced the concept *meta stylehseet* that generates the XSLT stylesheet automatically for PMML and MathML-content formats. They proposed the *meta stylesheet* as an extension to Content Dictionaries, in which the notation of the symbol in PMML and TEX along with the OpenMath expression is represented together. An example for the `divide` symbol is shown below. The notation is represented in `<version>` element and the OpenMath expression in `<semantic_template>`.

```
<Notation>
  <version precedence="200" style="1" >
    <math>
      <mfrac>
        <mi xref="arg1">a</mi>
        <mi xref="arg2">b</mi>
      </mfrac>
    </math>
  </version>
  <semantic_template>
  <OMOBJ>
    <OMA>
      <OMS cd="arith1" name="divide" />
      <OMV id="arg1" name="a" />
      <OMV id="arg2" name="b" />
    </OMA>
  </OMOBJ>
  </semantic_template>
</Notation>
```

Fig. 2. An example of the notation of the *divide* symbol following [NW01]

In this approach, the XPath of the arguments in the generated XSLT templates is determined by linking the symbols and the notations with their attributes as in Fig. 2: the elements with `id` attribute represent arguments in the OpenMath expression) and `xref` indicates the argument in PMML.

3 Symbols Presentation Authoring Problems

Although *meta stylesheet* approaches discussed in the previous section provide a basic authoring support, authors may experience difficulty in writing notations

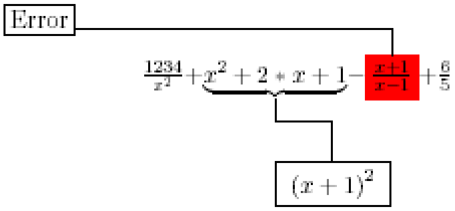


Fig. 3. An example of a mathematical expression with annotations

for the OpenMath symbols. The problems are as follows: first, some mathematical symbols can be presented in many ways. The automatic adaptation of a notation (from many notations for a symbol) in different contexts such as class, book and student is an issue, especially when different materials are presented or merged. For instance, in ActiveMath, materials may be compiled by merging the contents from different authors or collections. Secondly, authors, who have little or no background in programming, are required to write presentation in complex XML or XSLT structures. And last, but not least, authors may wish to highlight mathematical expressions by (a) making a box around the contents, (b) changing the background color, or, (c) labeling etc (See Fig. 3).

Most authors need an easy to use authoring environment, where they can edit the notations for the OpenMath symbols as well as preview it in the target presentation.

4 Solving Authors' Problems

Generally, authors use specialized editors for content authoring, which provide facilities that are best suited to their task. The editor selection depends on the content type or format. For instance, for XML documents editing there are specialized editors that understand the structural behavior and grammar of the documents in general. If an editor is aware of the usage of the XML document, for example when editing OMDoc documents for ActiveMath, it is possible to make the editing easier and more productive.

jEditOQMath [Lib04] is such an authoring environment: it is a package distributed to the authors of content for the ActiveMath learning environment. It is based on the open-source text-editor jEdit and its rich support for the editing of XML documents allows the easy creation and maintenance of valid XML documents. It adds several search facilities specialized for OMDoc and is integrated with the ActiveMath server of the author: it hosts publishing routines which, report consistency errors; it provides quick-open and quick-link facilities using the drag-and-drop paradigm. jEditOQMath is freely available.[1]

Editing mathematical formulæ is done in the OQMath language[2] which is converted to OpenMath formulae. It converts a highly readable, linear, syntax

[1] Please see `http://www.activemath.org/projects/jEditOQMath/`.
[2] Please see `http://www.activemath.org/projects/OQMath/`.

to OpenMath expressions (thanks to the usage of Unicode characters such as ∃ or Ω) in a way that authors can extend with new (input-level) notations. The experience with jEditOQMath thus far has proven that the OQMath verbosity is acceptable for mathematical documents. It has also proven that error feedback is important when using an environment where such a rich presentation process is applied.

Our solution provides a framework that defines an authoring process for notation editing to the publishing in ActiveMath. The framework consists of 2 phases. The Phase I deals with OpenMath and notation editing. The notations are edited by our tool called OpenMath Presentation Editor (OMPE). OMPE is run as a plug-in of jEditOQMath providing the notation authoring facility within the same environment where the authors edit their mathematical documents. Phase II is initiated passively, when the contents are previewed in ActiveMath. In this phase the notations can be adapted according to contextual information such as language, class/group, student and book. The whole process utilizes a `<symbolpresentation>` element, which we discuss in the next section.

4.1 Knowledge Representation of the Symbols Notations

In order to define different notations of a symbol, we have introduced the element `<symbolpresentation>`. It encapsulates the data to generate the xslt templates for multiple output formats, currently html, MathML and LaTeX. The grammar of the `<symbolpresentation>` element is shown in the table below.

tag	attributes	children
symbolpresentation	xref, id	(notation)+
notation	notation, precedence, format, language, lbrack, rbrack, style	(math,OMOBJ)

symbolpresentation: represents notations for a symbol. Its `xref` attribute points to the symbol for which notation is being written. It can contain multiple notations of a symbol, and each notation is represented by the `<notation>` element.

notation: Each notation element represents one notation of the symbol. For this purpose, it contains two structures, ie. PMML and OpenMath. Its format attribute contains a list of output formats for the symbol. Three attributes, `precedence`, `lbrack` and `rbrack`, control the bracket printing around the symbol. The left bracket is only printed, if the symbol `precedence` is less than the parent symbol. The same algorithm works with the right bracket with the corresponding attributes. Its `style` and `language` attributes are used to define selectors for a symbol that will be used at rendering time to choose the appropriate notation.

OMOBJ: The OMOBJ element represents the OpenMath expression containing the prototype of the symbol for which a presentation is being written. The OpenMath symbols in this expression are used in three possible ways: 1) as function, applied to its arguments represented inside the

OMA element, 2) standalone, or 3) having an attribute inside the attribution object (OMATTR). This expression allows the production of XSLT-templates applied to symbols or more complex expressions. The arguments are represented by `<OMV>` elements.

math: This element contains Presentation MathML (PMML) expression representing the notation for a symbol. The arguments in the notation (PMML) is represented by an `<mi>` element. The arguments in both PMML and OpenMath expression are mapped by pairing the `name` attribute of the `<OMV>` element and content of the `<mi>`. The XPath expression in the XSLT template is calculated by matching the arguments in both the expressions.

An example of the `plus` notation in `<symbolpresentation>` element is shown in Fig. 4.

```
<symbolpresentation xmlns="http://www.activemath.org/namespaces/am_content"
    id="arith1plus_77_24" xref="mbase://openmath-cds/arith1/plus">
  <notation format="html |pmathml |TeX" precedence="110" lbrack="("rbrack=")">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mrow>
        <mi>a</mi>
        <mo>+</mo>
        <mi>b</mi>
      </mrow>
    </math>
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMA>
        <OMS cd="arith1" name="plus" />
        <OMV name="a" />
        <OMV name="b" />
      </OMA>
    </OMOBJ>
  </notation>
</symbolpresentation>
```

Fig. 4. Example of `<symbolpresentation>` element for the `plus` symbol

4.2 Dealing with Different Notations for the Same Symbol

Our structure allows the authors to define different notations for a symbol. For each notation, a new definition of `<notation>` element is required. Below are contexts we have identified to influence the adaptation in symbol notation.

Language: This deals with internationalization for the symbols. Some symbols have different notations depending on the languages, e.g. the greatest common divisor symbol is written $\gcd(a, b)$ in English but is written $\mathrm{kgV}(a, b)$ in German. Using our notation infrastructure allows to solve the internationalized symbol presentation in LeActiveMath as discussed in the report [LW05]. To associate a notation with a language, we are using the `language` attribute of the `<notation>` element.

Different patterns of the arguments: One of the variations in the symbol rendering occurs when it has a different organization or number of the arguments. Consider the two notations, $\sum_{x=1}^{n} x$ and $\sum_{x \in m} x$, of the sum symbol. In the example, both sum symbols differ in their first arguments, i.e. be it the interval 1 to n or the set m. For each case like this, an author has to define a `<notation>` element.

Authors Styles: There are situations in which the symbol notation differs depending on the authors' styles even though the symbols are alike in all respects. Example of such presentations are: $\frac{1}{2}$ or $1:2$, $a*b$ or $a{\cdot}b$ or ab, $\frac{df}{dx}$ or $f'x$. For this, we provide a style attribute of the notation which allows the authors to define their own specific style notation for a symbol.

This multiple styles notations raise the question of the selectivity of a particular notation especially, when a material is compiled by merging the contents from different sources. We defined the following priorization for a consistent material presentation in ActiveMath from least to highest priority:

- **System Defaults** has the least priority in the system. All collections XSLT templates are merged into one, the import order of XSLT rules is used to manage this priority.
- **Author/Collection** This is the authors' default notation defined for a particular collection. It is automatically selected, if no other priority level is defined.
- **Book** A book can be generated by selecting a list of learning concepts from different content collections. At this priority level and other following levels, priority should be assigned by the notation selector tool which should store the priority information in the book configuration.
- **Group** The group represents a group of learners, for example a class, studying a common course. We expect, for example, teachers responsible of a course to adapt notations for their classes.
- **Individual** The students himself may be able to assign the priority to a notation. His choices, which we expect to be rare, have the highest priority level.

Within the Same Collection: In this context, the authors want to define different notations of the same symbol within the same collection. Take, for example, the associativity law of the `plus` symbol, i.e $a+(b+c)=(a+b)+c$. The default notation for the `plus` symbol will not print the bracket, but explicit brackets are required to explain the law. This kind of cases is handled with OpenMath attribution object. We have introduced a `type` attribute for the application OpenMath object to define for multiple notations of the same symbol. The `type` attribute abstractly defines a class of the symbols for which special treatment is made during the presentation generation. So, whenever an author wants a special notation other than the default in his book, he can assign this attribute to the instance of the application object containing the particular symbol. In Fig. 5, an example of the `plus` notation for a specific occasion is shown.

```
<symbolpresentation id="arith1plus_3_88" xref="mbase://openmath-cds
/arith1/plus">
  <notation format="html|pmathml|TeX" precedence="110" lbrack="
  (" rbrack=")">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <mrow><mo>(</mo>
        <mi>a</mi>
        <mi>+</mi>
        <mi>b</mi>
        <mo>)</mo></mrow>
    </math>
    <OMOBJ xmlns="http://www.openmath.org/OpenMath">
      <OMATTR>
        <OMATP>
          <OMS cd="am.presentation" name="type"/>
          <OMV name="associative"/>
        </OMATP>
        <OMA>
          <OMS cd="arith1" name="plus" />
          <OMV name="a" />
          <OMV name="b" />
        </OMA>
      </OMATTR>
    </OMOBJ>
  </notation>
</symbolpresentation>
```

Fig. 5. Example of the `plus` symbol notation with `type` attribute

4.3 Knowledge Representation for the OpenMath Objects Annotations

Sometimes the authors wish to emphasize a sub-expression to elaborate a concept by annotating it with styles such as colors, borders, fonts etc or by adding a label. We have built a mechanism for attaching mathematical expressions to styles of a *stylesheet* using the attribution elements. The *stylesheet* contains the style definitions like color, background color, border, etc. Additional information can be stored in an attribution object. For instance, an author can assign an attribute *error* to OpenMath expression (as in Fig. 6) to highlight a mistake in an exercise step by rendering the background in the `red` color as is described Fig. 9.

We use the attribution object to define the additional information for the presentation systems. Adding attributes to the OpenMath objects does not change the meaning of the object. Moreover, OpenMath applications can ignore the attributes, if they do not know its meaning. Therefore attributes are the ideal place to store the data about the specialized presentation.

Presentation Attributes. In the following we propose a list of attributes to be used for storing the information about annotations:

```
<OMOBJ xmlns=”http://www.openmath.org/OpenMath”>
  <OMA>
    <OMS name=”plus” cd=”arith1” />
    <OMV name=”a”/>
    <OMATTR>
      <OMATP>
        <OMS name=”type” cd=”am.presentation”>
          <OMV name=”error” />
        </OMATP>
        <OMV name=”b” />
      </OMATTR>
    </OMA>
</OMOBJ>
```

Fig. 6. An example of stylistic annotation using an OpenMath attribution

type: This attribute assigns a type to an OpenMath object. The value of this variable can be any string in the `name` attribute of `<OMV>`. In our approach, we use the `type` attribute for defining multiple notations of a symbol (see Fig. 5).

The `type` attribute is also used to invoke the `stylesheet` for the presentation of a symbol; as in Fig. 6, `error` identifier is attributed to an OpenMath object.

label: Labels attach media information (images, text and math expression) with the mathematical sub-expressions to illustrate its underlying meaning. This attribute arranges labels into eight logical positions around the mathematical expressions as shown in the Fig. 7.

The logical position is assigned via the `orientation` attribute assigned to the `value` object of the `label` attribute. In the Fig. 8, the `label` attribute is assigned to the variable object x. The position of the text *Variable* is set `down`.

Stylesheet: A **stylesheet** stores style attributes, i.e color, font-size, font-style, background, border and border-color. These attributes are common for each output formats (HTML, MathML and LaTeX). The style attributes are grouped under `<styleset>` element with a logical name represented in `id` attribute, e.g. The `<styleset id=‘‘error’’>` style contains border and red background in the

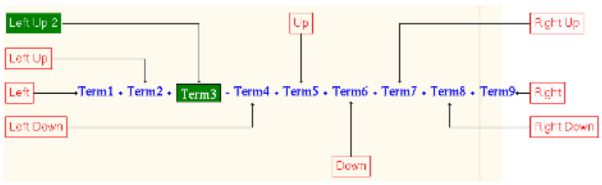

Left Up 2
Left Up
Left
Left Down
Up
Down
Right Up
Right
Right Down
Term1 + Term2 + Term3 - Term4 + Term5 + Term6 + Term7 + Term8 + Term9

Fig. 7. Various positions for the presentation of labels

```
$label(orientation(down),"Variable",x)$

<OMOBJ xmlns="http://www.openmath.org/OpenMath">
  <OMATTR>
    <OMATP>
      <OMS name="label" cd="am.presentation">
      <OMATTR>
        <OMATP>
          <OMS name="orientation" cd="am.presentation">
          <OMV name="down" />
        </OMATP>
        <OMSTR>Variable</OMSTR>
      </OMATTR>
    </OMAT>
    <OMV name="x" />
  </OMATTR>
</OMOBJ>
```

Fig. 8. An example OpenMath formula with attribution to denote a label. Above it is the OQMath formula that produces it: this is what we expect the author to input.

```
<Styles>
  <styleset id="error">
    <style name="border" value="solid" />
    <style name="background" value="red" />
  </styleset>
</Styles>
```

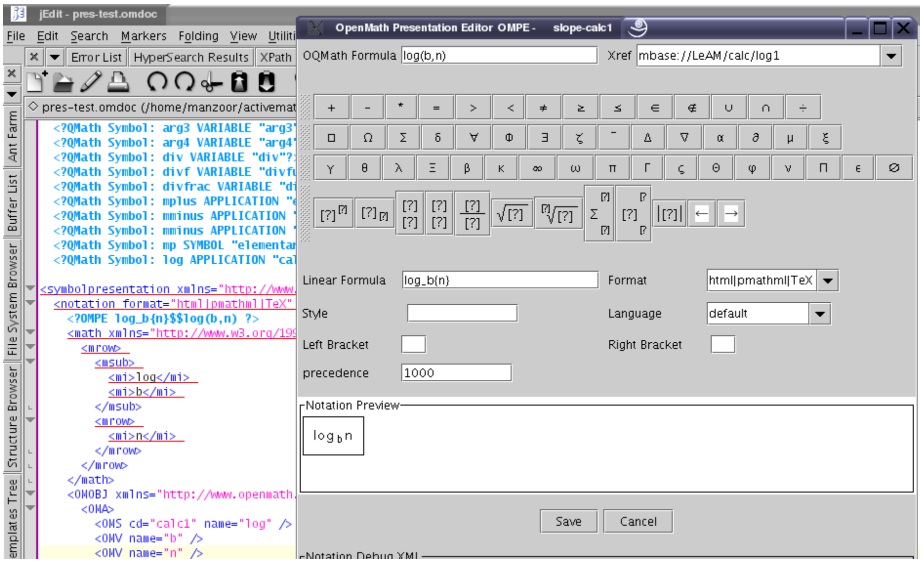
Fig. 9. Definition of Error Style

stylesheet (an example is shown in Fig. 9). The XSLT presentation system of each output format applies these logical styles to the mathematical expression by translating the attributes into their native implementations. For example, the `background` attribute , MathML has the `mathbackground` attribute of the `<style>` element, in HTML, it has `background-color` in CSS and in LaTeX it is implemented via the `colorbox` macro. The style definition (`<styleset>`) is only instantiated by matching `id` of the style definition with the `type` attribute of the OpenMath objects.

From the authoring point of view, the `stylesheet` offers an easy way to write stylistic information by only requiring the definition of the `<styleset>` element and the assignment of the `type` attribute to the OpenMath objects.

4.4 OpenMath Presentation Editor (OMPE)

OMPE is an authoring tool for the symbol notation (`<symbolpresentation>`) editing. It accpets a linear syntax for both OpenMath and PMML expressions, and, reduces the burden of writing long complex XML expressions: the syntax


jEdit - pres-test.omdoc
File Edit Search Markers Folding View Utiliti
Error List HyperSearch Results XPath
pres-test.omdoc (/home/manzoor/activemat
Ant Farm
Buffer List
File System Browser
Structure Browser
emplates Tree
OpenMath Presentation Editor OMPE - slope-calc1
Save
Cancel
Notation Debug XML

```
<?QMath Symbol: arg3 VARIABLE "arg3"
<?QMath Symbol: arg4 VARIABLE "arg4"
<?QMath Symbol: div VARIABLE "div"?>
<?QMath Symbol: divf VARIABLE "divfu
<?QMath Symbol: divfrac VARIABLE "di
<?QMath Symbol: mplus APPLICATION "e
<?QMath Symbol: mminus APPLICATION "
<?QMath Symbol: mminus APPLICATION "
<?QMath Symbol: mp SYMBOL "elementar
<?QMath Symbol: log APPLICATION "cal

<symbolpresentation xmlns="http://www.
  <notation format="html|pmathml|TeX"
    <?OMPE log_b{n}$$log(b,n) ?>
    <math xmlns="http://www.w3.org/199
      <mrow>
        <msub>
          <mi>log</mi>
          <mi>b</mi>
        </msub>
        <mrow>
          <mi>n</mi>
        </mrow>
      </mrow>
    </math>
    <OMOBJ xmlns="http://www.openmath.
      <OMA>
        <OMS cd="calc1" name="log" />
        <OMV name="b" />
        <OMV name="n" />
```

OQMath Formula: log(b,n)
Xref: mbase://LeAM/calc/log1

Linear Formula: log_b{n}
Format: html|pmathml|TeX
Style:
Language: default
Left Bracket:
Right Bracket:
precedence: 1000

Notation Preview: $\log_b n$

Fig. 10. Screenshot of OMPE editor

for the OpenMath input is based on OQMath and the PMML input is similar to LaTeX.

The editor also provides a tool bar, containing buttons for the maths notations, operators and symbols to help in editing the linear expression for PMML.

OMPE is developed on top of the Java OpenMath Editor (JOME).[3] Originally, JOME was capable of producing a limited set of OpenMath symbols and content MathML from a linear syntax. We have extended it to produce PMML, and the notation of the symbol in the `<symbolpresentation>` tag. OMPE runs as a plug-in of jEditOQMath as shown in Fig. 10.

To build a new notation, an author has to invoke the editor from an OQMath document where mathematical input notations are already defined. There, he provides the required information in the input boxes and then saves it. The new notation is pasted at the location of the cursor in the document. Alternatively, the author has to move his cursor inside the `<symbolpresentation>` element in the document and then invoke the OMPE editor from the plug-in menu. The editor loads the notation in its environment and gives the author the opportunity to edit it. After editing, the author saves the notation, OMPE replaces the old notation with the edited one.

5 Automatic Presentation Generation

The content presentation process in ActiveMath consists of series of steps making a presentation pipeline [ULWM04]. It is a 2-stage process. In the first stage,

[3] Please see `http://jome.sourceforge.net/`.

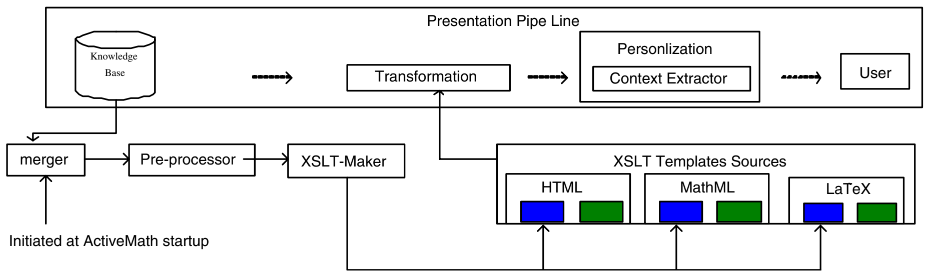


Fig. 11. Phase II: Presentation generation for the symbols

contents fragments are fetched from the knowledge base, with some intermediate preprocessing, and then transformed into output format by using XSLT; no adaptivity information is known, at this stage, except language. In the second stage, the fragments are combined to form a complete output page and enriched with user and context-specific information with the help of velocity code (Velocity[4] is a high performance template language to generate dynamic web pages).

The presentation generation comes in the phase II of our authoring process. It is started automatically in the ActiveMath environment. This phase deals with the adaptation that consists in selecting a notation among the several ones available using contextual information. It is made of the following two processes:

5.1 XSLT Generation for the Notations

The generation of the XSLT code from `<symbolpresentation>` element is a three step process (illustrated in Fig. 11). In the first step, the `merger` extracts all the `<symbolpresenttion>` elements from the knowledge base and groups the notations in one `<symbolpresentation>` element for each symbol. In the second step, the merged file is passed to the `pre-processor`, which decorates the notations code with styles, XPath and conditions if required. In the last step, the decorated notations are passed to the XSLT-maker, which finally generates XSLT templates for the three output formats. The `blue` box in the figure represents, the generated XSLT code for the symbols.

Below are examples of XSLT code generated depending on the `<notation>` elements for each symbol available in the system.

Example 1. The XSLT code for the `plus` symbol for two cases: 1) having `associative` attribute for explicit brackets printing, 2) or default.

```
<xsl:template match="om:OMA[om:OMS[@name='plus' and @cd='arith1']]">
  <xsl:choose>
    <xsl:when test="om:OMATTR[om:OMATP[om:OMS[@name='type' and
    cd='am.presentation'and following−sibling::om:OMV[position() =1
          and @name='associative']]]]">
```

[4] Please see `http://jakarta.apache.org/velocity/`.

```
      <mrow>
        <mo>(</mo>
        <xsl:apply−templates select=”∗[2]” />
        <mo>+</mo>
        <xsl:apply−templates select=”∗[3]” />
        <mo>)</mo>
      </mrow>
    </xsl:when>
    <xsl:otherwise>
      <mrow>
        <xsl:apply−templates select=”∗[2]” />
       <mo>+</mo>
       <xsl:apply−templates select=”∗[3]” />
      </mrow>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Example 2. In this example, the language adaptivity (German, i.e. `de`, on line 3) is checked at XSLT level. Further adaptation is made for the author style (`m-notation`) in the default language case (on line 11). This XSLT template produces Velocity code (e.g. on lines 12, 19 and 27) where branching between the notations is done by an `#if` condition.

```
1   <xsl:template match=”om:OMA[om:OMS[@name=’slope’ and @cd=’calc1’]]”>
2     <xsl:choose>
3       <xsl:when test=”$language=’de’”>
4         <mrow><msub><mo>steigung</mo>
5           <xsl:apply−templates select=”∗[2]” /></msub>
6           <mrow><mo>(</mo>
7           <xsl:apply−templates select=”∗[3]” />
8           <mo>)</mo></mrow>
9         </mrow>
10      </xsl:when>
11      <xsl:otherwise>
12        <xsl:text>#if ($style.contains(”m−notation”)) then </xsl:text>
13          <mrow><msub><mo>m</mo>
14            <xsl:apply−templates select=”∗[2]” /></msub>
15            <mrow><mo>(</mo>
16            <xsl:apply−templates select=”∗[3]” />
17            <mo>)</mo></mrow>
18          </mrow>
19        <xsl:text> #else</xsl:text>
20          <mrow><mo>slope</mo>
21            <mo>(</mo>
22            <xsl:apply−templates select=”∗[2]” />
23            <mo>,</mo>
24            <xsl:apply−templates select=”∗[3]” />
25            <mo>)</mo>
26          </mrow>
```

```
            <xsl:text> #end</xsl:text>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:template>
```

Example 3. The example for the `sum` symbol with different arguments patterns, i.e `set` and default.

```
<xsl:template match=”om:OMA[om:OMS[@name=’sum’ and @cd=’arith1’]]”>
  <xsl:choose>
    <xsl:when test=”om:OMA[position()=2 and om:OMS[@name=’integer_interval’
      and @cd=’interval1’]]”>
      –
    </xsl:when>
    <xsl:otherwise>
      –
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

5.2 Notation Generation in Output Formats

Current approaches for the notation generation that we reviewed in Sec. 2 only deal with XSLT generation. In order to adaptively present mathematical notations, information from the context is needed which can only be acheived by server side scripting languages such as Servelts, PHP and Velocity. In ActiveMath, this context information is available at Velocity interpretation time, i.e. in the personalization phase.

The presentation generation pipeline (shown in Fig. 11) is initiated with a user request from the browser. At the stage of transformation, the `annotation interpreter` (a set of XSLT templates, shown in green box in the Fig. 11) checks whether the OpenMath objects (in the contents) are attributed. In that case, it applies the `<styleset>` from the **stylesheet** document or generate supporting code for the labels. The `interpreter` implementation is specific to each output format. The labeling annotation can not be fully handled at the XSLT level, because XSLT does not know about the dimension and position of the expression. For this, we have written output format-specific routines that executes at final rendering level: for HTML & MathML, they are executed in the client browser, for LaTeX-based output-formats (currently PDF), a LaTeX package was written. The labeling, in MathML, requires a DOM facility and thus is restricted to the Mozilla browser currently. At the stage of `Personalization` in the pipeline, the Velocity code for multiple notations (discussed in Sec. 5.1) is executed by matching the *styles list* returned from our `ContextExtractor`. The `ContextExtractor` should retrieve the styles list based on the priority of the context. Currently, it only extracts style at the `collection` level.

6 Conclusion and Future Work

In this paper, we have described a solution to the authoring problems that we have identified in our work with the ActiveMath learning environment. We believe that other mathematical web presentation systems could benefit from our work as well.

We are polishing the tools so that authors adopt it as soon as possible, in particular in the LeActiveMath EU project.[5] Further work is also being done in order to make notations user-visible aside of the description of a symbol.

Currently, adaptivity of a notation is only acheived at the collection level. Using a notation selection tool similar to[6] we intend to offer the management of notations to a system: it will be of use to authors and editors to edit their books' notations, to teachers to edit the classes' notations, and, potentially, to individual users. This will allow the adaptation of a notation as discussed in Sec.4.2.

Moreover, the notations selected for a presentation have to be synchronized with presentation of mathematical tools. For instance, an input editor will be integerated into LeActiveMath; it will enable learners to input mathematical formulæ into, e.g, the search interface or exercise interactions. The Wiris input editor[7] chosen for this task allows the definition of *domain files* which describe the notations for each symbol: an export from the `<symbolpresentation>` elements to domain files is under work. The integeration of the input editor within ActiveMath will take advantage of the notations available for the context and will, thus, enable learners to input mathematical formulæ using notations consistent with the mathematical content that is presented to them.

References

[Car00] David Carlisle. Openmath, MathML, and XSL. In *ACM SIGSAM Bulletin*, volume 34, number 2, pages 6–11, June 2000. ISSN:0163-5824.

[Koh04] Michael Kohlhase. OMDoc an open markup format for mathematical documents (version 1.2), 2004. Manuscript, http://www.mathweb.org/omdoc/.

[Lib04] P. Libbrecht. Authoring web content in activemath: From developer tools and further. In Alexandra Christea and Franca Garzotto, editors, *Proceedings of the Second International Workshop on Authoring Adaptive and Adaptable Educational Hypermedia, AH-2004: Workshop Proceedings, Part II, CS-Report 04-19*, pages 455–460. Technische Universiteit Eindhoven, 2004.

[LW05] "Paul Libbrecht and Stefan Winterstein". "internationalizing leactivemath". Technical report, LeActiveMath consortium, "http://www.leactivemath.org", "2005".

[5] Please see `http://www.leactivemath.org/`.

[6] Please see Notation Selection Tool at `http://www.orcca.on.ca/MathML/NotationSelectionTool/`.

[7] See `http://www.wiris.com/`.

[MAB$^+$01] E. Melis, E. Andrès, J. Büdenbender, A. Frischauf, G. Goguadze, P. Libbrecht, M. Pollet, and C. Ullrich. Activemath: A generic and adaptive web-based learning environment. *International Journal of Artificial Intelligence in Education*, 12(4):385–407, 2001.

[NW01] Bill Naylor and Stephen Watt. Meta style sheets for the conversion of mathematical documents into multiple forms. In *International Workshop on Mathematical Knowledge Management*, September 2001. http://www.emis.de/proceedings/MKM2001.

[ULWM04] C. Ullrich, P. Libbrecht, S. Winterstein, and M. Mühlenbrock. A flexible and efficient presentation-architecture for adaptive hypermedia: Description and technical evaluation. In Kinshuk, C. Looi, E. Sutinen, D. Sampson, I. Aedo, L. Uden, and E. Kähkönen, editors, *Proceedings of the 4th IEEE International Conference on Advanced Learning Technologies (ICALT 2004)*, pages 21–25, 2004.