Meta-Stylesheets for the Conversion of Mathematical Documents into Multiple Forms

Bill Naylor (bill@orcca.on.ca)
and Stephen Watt (watt@orcca.on.ca)
Ontario Research Centre for Computer Algebra,
The University of Western Ontario,
London, Ontario. CANADA N6A 5B7

December 23. 2001

Abstract. It is sometimes necessary to convert mathematical documents into a number of different forms. This may be achieved by applying different XSLT stylesheets, one for each required form, to the document. We describe how *meta-stylesheets* may be used to automatically generate the stylesheets which will be applied to the mathematical documents to generate the multiple forms required. The resources required by the meta-stylesheets may be encoded in extended OpenMath Content Dictionaries. We discuss how the presentational information may be stored in the Content Dictionaries for various different notational styles.

Keywords: MathML, meta-stylesheets, OpenMath, Semantic Templates, XML, XSLT

1. Introduction

MathML [9] is an emerging standard from the W3C consortium for the representation of mathematical expressions. Its main design goals are:

1) to provide a means of representing "TEX quality" presentation (*presentation MathML*) of mathematics on the internet,

2) to provide a means of representing the semantic meaning of mathematics (*content MathML*), again with the major concern that this should be transferable over the internet.

MathML is an XML application, and as such may readily be converted into other document styles using XSLT [13] stylesheets. It has been designed primarily with the aim of representing mathematics to the level of *K-12*.
Since mathematical notations consist, in general, of a vocabulary of special symbols and a small number of relative positioning primitives, it is fairly well covered by MathML. Problems arise, however, in the representation of the meaning of general mathematical objects.
One of the limitations with MathML as it stands is that the meaning

of an element *resides* in the name of the tag. Therefore, to represent mathematical concepts not pre-defined in MathML it is necessary to refer to some external definitions. A resource which may be able to provide such a *definition server* may exist in the form of OpenMath [11] *Content Dictionaries.*

OpenMath is an XML application whose primary goal is to serve as a communication medium for the semantics of mathematical objects. One of its primary concerns is that it should be extensible, as it is a huge task to represent *all* of known mathematics, and indeed we may need to represent mathematics which is not at the time defined. Extensibility is provided via Content Dictionaries which are XML repositories of mathematical information.

As is described in our paper [4] we may use the OpenMath concept to provide a definition resource which may be used for extending MathML. Padovani et al. [5] discuss a more general framework, of which this is a particularly useful special case.

In this paper we describe how stylesheets may be automatically generated from extended OpenMath Content Dictionaries using meta stylesheets. Each of which will effectively define a new format of result document. It would define the context from which information is taken from the Content Dictionary and the context in which it will be included in the result document. This context information will be embodied by a stylesheet which is specific to the Content Dictionary, we shall call these stylesheets *annotation stylesheets.*

In this manner, a document written using an *extended MathML* set of symbols may be transformed into one of a number of different formats, for example:

1) presentation MathML, where the notation intended is specified in the Content Dictionary over which the meta-stylesheets are applied.

2) Mixed markup, using `semantic` elements specifying presentation MathML and references to OpenMath Content Dictionaries in `annotation-xml` children.

3) content MathML, using `csymbol` elements and definition URLs which point to symbols in OpenMath Content Dictionaries.

One of the major sources of ambiguity in mathematical notation is the ordering in which the arguments to an operator or function occur. This ordering of arguments should be defined somewhere in the Content Dictionary. We propose the declaration of a `Notation` element within the `Description` element of a symbol. This element will utilize `id` and `xref` attributes to specify the ordering intended, this concept shall be

made more precise in section 3.
It may also be necessary to cater for applications which do not have the ability to render or otherwise deal with MathML. In order to cater for these applications we also store TeX data and a graphical image associated with the notation (e.g. a URI for a `gif`, `jpeg` or `png` image).
In section 5 we describe a scheme we use for automatically creating stylesheets which translate documents between styles, these generated stylesheets utilize information within the `Notation` elements. The target style of these stylesheets will be dependant on the *meta-stylesheet* used for their creation.
In section 10 we shall show how we may automatically create configuration files for a notation selection tool which allows translation of content MathML into various presentation MathML variants, where the notational style may be chosen by a user [2].

2. Dealing with Notational Ambiguity

One of the major problems with mathematical presentation is that there is a many to many relationship between presentations of mathematical objects and semantic meanings. Certain presentations have multiple meanings, for example the symbol M^T could (amongst others) mean "The transpose of the matrix M" or "M to the power T".
Conversely we see that there are many presentations for some mathematical objects, for example the function describing the number of different ways in which m items may be selected from a set of n distinct objects, may be presented as:
n choose m, $\binom{n}{m}$, ${}_nC^m$, C^m_n, C^n_m, binomial(n,m) or Binomial[n,m].
These last two are the notational styles of the computer algebra systems Maple and Mathematica respectively.
This attribute of mathematical presentation (which has its root in the varied history of mathematics) makes extraction of semantic meaning from presentational markup a non-trivial task, due to the ambiguities that it implies. We shall propose a method in which one may specify markup which is both presentational in that it may be converted into presentation MathML (see section 7.1) and semantical in that it may be converted into content MathML (see sections 7.2 and 7.4). This technique utilises a `Notation` element, which we shall detail in the following section.

3. Catalogs of Annotations for Mathematics

In order that we can disambiguate the ordering of arguments to a mathematical operator or function, we have introduced a "`Notation`" element for symbols defined in an extended OpenMath ContentDictionary.
The `Notation` element (in its present form) should satisy the dtd fragment:

```
<!ELEMENT Notation ((version)* , (semantic_template)+)>
```

Whilst the `version` element should satisy the dtd fragment:

```
<!ELEMENT version (image , tex , math)>
```

For our purposes, the different parts of these elements will have the following meanings:

– version:
  There should be one of these elements for every version of the notation (see section 2). It takes the required attribute `precedence`, to specify its order of precedence (this is an absolute value and must be standardized in some way) and the optional attribute `style` which is used as an index for the given style. If no `style` attribute is given (this case should only occur once within any given `Notation` element) then this is the default style, we shall discuss default styles further in section 6. The `version` element should have three children: `image`, `tex` and `math`.

– semantic_template :
  This should have one or more `OMOBJ` children and zero or more λ function bindings. (constructed within `OMBIND` elements). The element satisfies the dtd fragment:

  ```
  <!ELEMENT semantic_template ((OMOBJ)+ , (OMBIND)*)>
  ```

  This element is intended to form a template which may be used to endow the presentation MathML elements (within the `math` children of its `version` siblings) with semantic meaning. This is described in greater detail in section 4.

– image:
  This should have one `src` attribute whose value is the URI of a graphical image file illustrating the notation . This could be useful for display in pulldown menus (and the like) where it is unlikely that presentation MathML rendering will soon be available.

– tex:
  This should be a TEX-like representation of the notation. We shall describe this in more detail in section 9. It is included for automatic generation of TEX markup from Extended MathML markup.

– math:
  This should be presentation MathML markup defining the notation to be associated with the containing `version` element. The elements representing the arguments in this element should have `xref` attributes associated with them, these should have corresponding `id` attributes having the same values. These `id` attributes should be associated with the argument children of a `semantic_template` child of the `Notation` element.

We note that the `Notation` element could either be included in a file system written in an extension to the OpenMath Content Dictionary syntax, or they could be included in a parallel "`.ntn`" notation file system (analogous to the "`.sts`" file system used for types). For the purposes of this paper we shall assume that the `Notation` elements are included in Content Dictionaries.

Detailed Example

We now give a detailed example of a `Notation` element which has several `version` children.

EXAMPLE: 1. *We give the example of the* `Notation` *element for the* `choose` *symbol, where the presentations given represent styles selected from the notational styles given in section 2. We hypothesise a Content Dictionary* `"combinat2"` *containing this symbol. We note in particular the ambiguity that would otherwise be implied by the French versus the Russian styles* C_n^m, C_m^n *or* `style="French"`, `style="Russian"` *respectively:*

```
<Notation>
    <version precedence="1000">
      <!-- this is the default style-->
      <image src="choose2.gif"/>
      <tex>
        \left (\begin{array}{c}
          \xref{argChoose1}{n}\\ \xref{argChoose2}{m}\\
        \end{array} \right )
      </tex>
      <math><mrow>
        <mfenced><mtable>
          <mtr><mi xref="argChoose1">n</mi></mtr>
          <mtr><mi xref="argChoose2">m</mi></mtr>
        </mtable></mfenced>
      </mrow></math>
    </version>

    <version precedence="900" style="subnCsupm">
      <image src="choose3.gif"/>
      <tex>
        _\xref{argChoose1}{n}C^\xref{argChoose2}{m}
      </tex>
      <math><mrow>
        <mmultiscripts>
          <mi>C</mi>
          <none/>
          <mi xref="argChoose2">m</mi>
          <mprescripts/><mi xref="argChoose1">n</mi>
          <none/>
        </mmultiscripts>
      </mrow></math>
    </version>

    <version precedence="900" style="Russian">
      <image src="choose4.gif"/>
      <tex>
        C_\xref{argChoose2}{m}^\xref{argChoose1}{n}
      </tex>
      <math><mrow>
        <msubsup>
          <mi>C</mi>
          <mi xref="argChoose2">m</mi><mi xref="argChoose1">n</mi>
        </msubsup>
      </mrow></math>
    </version>

    <version precedence="900" style="French">
      <image src="choose5.gif"/>
      <tex>
        C_\xref{argChoose1}{n}^\xref{argChoose2}{m}
      </tex>
      <math><mrow>
        <msubsup>
          <mi>C</mi>
```

```
        <mi xref="argChoose1">n</mi><mi xref="argChoose2">m</mi>
      </msubsup>
    </mrow></math>
  </version>

  <version precedence="800" style="maple">
    <image src="choose6.gif"/>
    <tex>
      {\rm binomial}(\xref{argChoose1}{n},\xref{argChoose2}{m})
    </tex>
    <math>
      <mi>binomial</mi><mo>&ApplyFunction;</mo>
      <mfenced>
        <mi xref="argChoose1">n</mi><mi xref="argChoose2">m</mi>
      </mfenced>
    </math>
  </version>

  <semantic_template><OMOBJ><OMA>
    <OMS cd="combinat2" name="choose"/>
    <OMV name="n" id="argChoose1"/>
    <OMV name="m" id="argChoose2"/>
  </OMA></OMOBJ></semantic_template>
</Notation>
```

We may specify the presentation of a particular version of a symbol by giving a specific value to the `style` attribute of the extended MathML symbol, where the attributes value is the same as the value of the `style` attribute associated with the particular `version` element required.

4. Semantic Templates for Mathematical Objects

In order to specify notation for mathematical objects it is necessary to link the presentation information with some formal specification of the object, we express these formal specifications by means of *semantic templates*. These could possibly be expressed in a number of different languages, e.g. *Maple*, *Common Lisp* or *Scheme*. The language we use should have certain properties:

- Well developed semantics,
- Sufficient operations,
- Easy cross referencing with XML labels,

– An XML syntax would make integration with contemporary software simple.

With these considerations, OpenMath is a good choice. In order to provide the cross referencing which is necessary to provide the presentation to content links, we use the XPath [12] `id/xref` linking mechanism. In most cases this is quite a simple procedure and is covered later in this section. However in some cases this process is not sufficient to specify all of the objects referred to in the notation, in this case extra technology is required, we cover this in section 8.
Within the presentation information which is represented by presentation MathML, there are elements representing the different parts of the object being represented. We attach `id` attributes to each of these elements.

EXAMPLE: 2. *In this example we give presentation MathML to represent the image:*

$$C_m^n$$

In the markup that follows,the 'm' subscript of 'C' is labeled with the attribute `xref` *which has a value of* `argChoose2` *whilst the superscript 'n' has an* `xref` *attribute with a value of* `argChoose1`.

```
<math><mrow>
  <msubsup>
    <mi> C </mi>
    <mi xref="argChoose2"> m </mi>
    <mi xref="argChoose1"> n </mi>
  </msubsup>
</mrow></math>
```

Now it is necessary to provide a *semantic template* that the parts of the presentational information can be linked to. The first step is to provide a *prototype object* which has *anchors* (i.e. the elements with the associated `id` attributes) which may be linked to by the presentation.

The most common type of object encountered in mathematical markup is that of a function being applied to a number of arguments. In this case the prototype is the OpenMath Application (`OMA`) of the function to OpenMath Variables (`OMV`s) in the place of its arguments. An `id` attribute is attached to each of the arguments. These attributes should have the same values as the `xref` attributes in the corresponding parts of the presentation MathML.

EXAMPLE: 3. *We give an example of a semantic template which could be used with example 2.*

```
<semantic_template><OMOBJ><OMA>
  <OMS cd="combinat2" name="choose"/>
  <OMV name="n" id="argChoose1"/>
  <OMV name="m" id="argChoose2"/>
</OMA></OMOBJ></semantic_template>
```

It should be noted that even though function application covers the majority of cases there are cases not covered. This shall be discussed further in section 8.

5. Automatic Creation of Stylesheets

In this section we describe how given a set of OpenMath Content Dictionaries, we may automatically create a set of stylesheets that may be used to transform mathematical documents. In our method, which we describe diagrammaticly in figure 1, we initially apply a *meta-stylesheet* to each one of a set of Content Dictionaries. This will result in a set of XSLT stylesheets, these are the annotated stylesheets. These stylesheets may be combined using a stylesheet (in our diagram we denote this by `comb`). Now if we apply the `comb` stylesheet to a document which is written in an extended (relative to the set of Content Dictionaries) MathML, we will result in a document in the required (relative to the meta-stylesheet) style.
The following points refer to figure 1.

- The Content Dictionaries are denoted `CD 1` to `CD n`.
- The meta-stylesheet (a different one for each format required) is denoted `META`.
- The annotated stylesheet that is the result of applying `META` to `CD` *i* is denoted `Annot` *i*.
- All the annotated stylesheets are imported by a combining stylesheet denoted `Comb`.
- Given the preceding, a document `Doc 1` which is written in the extended MathML implied by the Content Dictionaries `CD 1` to `CD n`, may be translated by the stylesheet `Comb` into a document `Doc 2` which is in the required format.

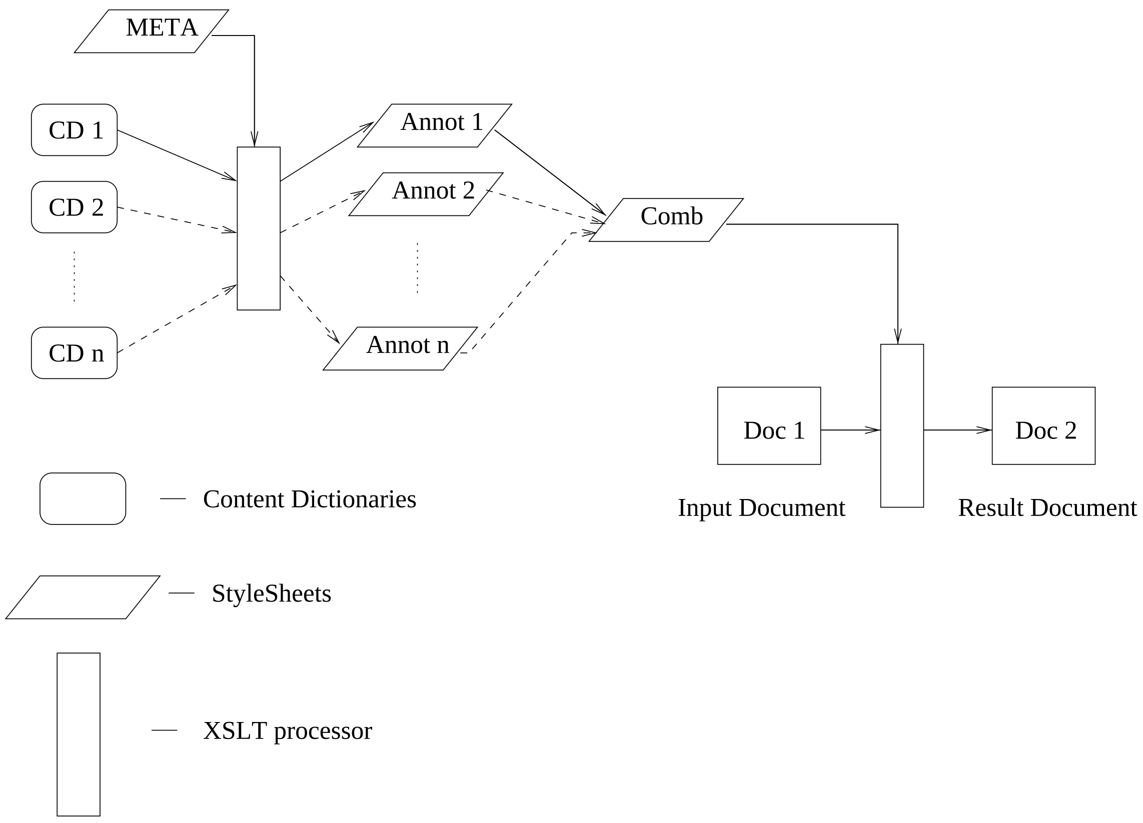


Figure 1. Automatic Production of Stylesheets

6. Default Notations

We wish the user to have full choice over which style is used within their document, whilst retaining the semantic content. This may be achieved using the `version` with `style` attributes and `semantic_template` elements described earlier. However we would like the user to have the option to relegate the styling decisions to some defaulting mechanisms. We specify a hierarchy of defaults which enable this via a set of priorities as follows:

- **priority 1**: Command line control,

  This is the level of control over notational defaults with the highest precedence. When the user applies the generated stylesheets to the input document, written in Extended MathML, default styles may be specified on the command line. These will override any defaults given at other levels of the default hierarchy.
  possible uses for this default level include:

1) production of copies of documents to be sent to different countries which have different or even conflicting styles in common use.
2) production of copies of documents to be submitted to separate journals which require differing styles.

- **priority 2**: Input Document defaults,

  This level of control over notational defaults has an intermediate precedence level. This allows an author to personalise a document with his own notational styles. To specify defaults at this level, an element would be included at the top level of the input document with the following syntax: [1]

```
<default_style>
  <for symbol="symbName" cd="cdName" style="rqdStyle"/>
   ...
</default_style>
```

  here the element `default_style` satisfies the dtd fragment:

```
<!ELEMENT default_style (for)*>
```

  and the element `for` satisfies the dtd fragment:

```
<!ELEMENT for EMPTY>
<!ATTLIST for symbol CDATA #REQUIRED
              cd CDATA #REQUIRED
              style CDATA #REQUIRED>
```

- **priority 3**: Meta-Stylesheet defaults,

  This level of control over notational defaults has a lower intermediate precedence level. It is unlikely to be used frequently. It could be used if a particular meta-stylesheet distribution was going out to a particular market which shared a common notational language. e.g. A meta-stylesheet going out to an engineering market might specify that the default symbol for the square root of -1 was j, whereas for mathematicians it might be i. It should be noted however that since the meta-stylesheets are intended to relegate the notational information to the Content Dictionaries, this level of defaulting is only for symbols which are *special* in some way.

[1] In this section the value `rqdStyle` should be read as meaning the value of the `style` attribute which is related to the notational style required (i.e. the style implied by the relevant `version` element in the appropriate Content Dictionary).

- **priority 4**: Content Dictionary defaults,

  This default level has the lowest priority but it is the level which must be adhered to if no other default is given and if no style has been set in the input document. These defaults are therefore required to be given.

7. The Different Formats we have Considered

We now make a more detailed presentation of some of the different formats we have considered by considering the translation of the application of the `choose` symbol to the variables n and m to each of the following formats:

- Extended MathML → Presentation MathML.
- Extended MathML → Content MathML with OpenMath references.
- Extended MathML → semantics elements with Presentation MathML and OpenMath annotations.
- Extended MathML → csymbol elements with OpenMath symbol URLs.

7.1. Extended MathML → Presentation MathML

In order to perform this transformation the annotated stylesheet must include the following constructs in the templates for each symbol:
We shall assume that the symbol which matches the template in question is `xmml:choose`[2].

```
<xsl2:choose>
  <xsl2:when test="@style=style1">
    -- presentation information taken from the 'style1'
       style of xmml:choose --
  </xsl2:when>

  ... one when element for each non-default style ...

  <xsl2:otherwise>
    -- presentation information taken from the default
```

[2] We use the xmml prefix here in order to emphasis the differences between the extended MathML symbol (`xmml:choose`) and the XSLT symbol (`xsl2:choose`)

```
      style of xmml:choose --
  </xsl2:otherwise>
</xsl2:choose>
```

Each of the sections specifying the 'presentation information' will include `apply-template` elements to specify the presentation of the arguments to the symbol, however these will not necessarily be in the order in which they appear in the presentation. They will be in the order in which the arguments are given in the `semantic_template` child of the `Notation` element (linked by the `xref`, `id` attributes). These stylesheets must be generated by the meta-stylesheets. We see that all of the information necessary to create the `xmml:choose` elements exists in the Content Dictionaries.

If a symbol exists in an OpenMath Content Dictionary with the name of `Symb`, the corresponding extended MathML element will have the same name. This element should be the first child of an `mrow` element. The following children should be the arguments of the function.

EXAMPLE: 4. *If we wish to specify, within some extended markup, the presentation of the choose function applied to two arguments n and m, we should use the following markup:*

```
...
<mrow>
  <xmml:choose/> <!-- no style attribute,
                      so the default is assumed -->
  <mi>n</mi> <!-- these could be arbitrary arguments -->
  <mi>m</mi>
</mrow>
...
```

N.B. the default referred to in the above depends on the default level which has been selected from the default hierarchy described in section 6. If the default is relegated to the Content Dictionary, on application of the stylesheet which has been automatically created from the Content Dictionary containing the `choose` *symbol, we obtain the markup:*

```
<mrow>
  <mfenced><mtable>
    <mtr><mi xref="argChoose1">n</mi></mtr>
    <mtr><mi xref="argChoose2">m</mi></mtr>
  </mtable></mfenced>
</mrow>
```

The rendering will be approximately $\binom{n}{m}$.

The reader is referred back to section 6 for a discussion on how the default style may be altered.
We now give an example where one particular style is required.

EXAMPLE: 5. *If we required markup to display the same application as in example 4 but in style "subnCsupm", for example, we could use the markup:*

```
<mrow>
  <xmml:choose style="subnCsupm"/>
  <mi>n</mi>
  <mi>m</mi>
</mrow>
```

This would be converted by the same stylesheet as used in example 4 to the following:

```
<mrow>
  <mmultiscripts>
    <mi>C</mi>
    <none/>
    <mi xref="argChoose2">m</mi>
    <mprescripts/><mi xref="argChoose1">n</mi>
    <none/>
  </mmultiscripts>
</mrow>
```

The rendering will be approximately "${}_nC^m$".

7.2. Extended MathML → Content MathML with OpenMath references

In order to perform this transformation, not much extra information is required, the required form is the following (this is based on the `semantics` element, a mapping from some markup to an external definition resource):

```
<apply>
  <semantics>
    <!--extended MathML symbol-->
    <annotation-xml>
      <!--OpenMath symbol-->
    </annotation-xml>
  </semantics>
  <!--arguments-->
</apply>
```

The extended MathML symbol will be the same as in the input document, so that can just be copied through. This can also be done for the arguments. The only section which needs any extra information is the OpenMath reference. This will be an `<OMS cd="?" name="?"/>` element, the `name` attribute value may be determined trivially from the extended MathML symbol name, and the `cd` attribute value may be read off from the Content Dictionary, as it must be given as one of the first level children elements (as specified by the OpenMath Content Dictionary DTD). The only remaining task is to wrap the determined elements in `semantics`, `annotation-xml` and `apply` elements where appropriate.

If a symbol exists in an OpenMath Content Dictionary with the name of `Symb`, the corresponding extended MathML element will be `Symb`. This element should be the first child of an `apply` element. The following children should be the arguments of the function.

EXAMPLE: 6. *If we wish to specify within some extended content markup the application of the choose function to two arguments n and m, we should use the following markup:*

```
...
<apply>
  <xmml:choose/>
  <ci>n</ci>
  <ci>m</ci>
</apply>
...
```

The resulting markup that we would obtain after application of the stylesheet that has been automatically created from the Content Dictionary which contains the `choose` *symbol (which for arguments sake was the Content Dictionary* `combinat2.ocd`*) is:*

```
...
<apply>
  <semantics>
    <xmml:choose/>  <!-- extended MathML symbol -->
    <annotation-xml encoding="OpenMath">
      <!-- OpenMath symbol-->
      <om:OMS cd="combinat2" name="choose"/>
    </annotation-xml>
  </semantics>
  <ci>n</ci>        <!-- arguments -->
  <ci>m</ci>
```

```
</apply>
...
```

7.3. Extended MathML → semantics elements with Presentation MathML and OpenMath

In order to perform this transformation essentially we only need to perform a combination of the transformations of sections 7.1 and 7.2, to obtain a document with the structure:

```
  <semantics>
    <!-- MathML Presentation of the application -->
    <annotation-xml>
      <!-- OpenMath application -->
    </annotation-xml>
  </semantics>
```

7.4. Extended MathML → csymbol elements and OpenMath symbol URLs

The only information that is required for this transformation is a URI of the symbol, this can be determined from one of the top level children in the Content Dictionaries, the `CDURL` element and the name of the symbol, (this problem was solved in section 7.2). This URI must be given as the value of the `definitionURL` attribute of the `csymbol` element.
The syntax with which we wish to specify the extended content MathML shall be the same as in example 6.

EXAMPLE: 7. *We shall also use the same input as in example 6. However we should expect to get output of the following form:*[3]

```
<apply>
  <csymbol definitionURL=
"http://orcca.on.ca/~bill/MyCombFnDist/html/cd/series.html#choose"/>
  <ci>n</ci>
  <ci>m</ci>
</apply>
```

[3] The URL specified is fictitious and serves the purpose of explanation only.

8. Semantic Templates which Include Extra Information

There are cases where sub-elements of the presentation do not correspond to specific elements in any generic formulation of the prototype object (typical examples are n-ary functions, matrices and partial differentiations). We propose a scheme whereby we may prescribe complete (in that they include all the information required to construct the given notations), unambiguous templates that include anchors for the presentation sub-elements. The scheme we use is to follow the prototype object with OpenMath specifications of functions which calculate the required values (In the case of n-ary functions this would be the elements of the n'tuple, in the case of a matrix, the elements of the matrix, partial differentiation is more complex (see [14]):
we shall call these functions *template functions*. The idea is that the `OMBIND` element enclosing the function has an `id` attribute which may be linked to from the presentation. The expression part of the function definition will link to the prototype in the usual way using `xref` attributes.

8.1. Extracting the Template Function call from the Presentation

It is necessary to include the arguments to the template functions in the document, where their positions relative to the element which specifies the function call (via an `xref` attribute) are constant. The arguments to the template function shall be taken as the children of this element in the order given in the document.

EXAMPLE: 8. *In this example we consider presentations for the symbol* `matrix` *from* `linalg1`*. The presentation we shall consider is:*

$$\begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}$$

where x_{11}, x_{1n}, x_{m1} and x_{mn} denote the top left, top right, lower left and lower right elements of the matrix respectively.
In order to provide anchors for the markup to be meaningful, as part of the `semantic_template` *we require a function taking two variables i, j (in fact since this presentation requires the two variables to be in an* `mrow`*, this will take the place of one argument and thus we require the template function argument to be a two element list. This is reflected in the presentation of the* `semantic_template` *given later). The return value of this function should be the element indexed by i and j. In this case, the top left, top right, lower left and lower right elements. Presentation MathML which could be used for this presentation follows:*

```
<mrow>
  <mo> ( </mo>
  <mtable>
    <mtr>
      <msub xref="matelt">
        <mi>x</mi> <mrow><mn>1</mn> <mn>1</mn></mrow>
      </msub>
      <mi>&hellip;</mi>
      <msub xref="matelt">
        <mi>x</mi> <mrow><mn>1</mn> <mi>n</mi></mrow>
      </msub>
    </mtr>
    <mtr>
      <mi>&vellip;</mi><mi>&dtdot;</mi><mi>&vellip;</mi>
    </mtr>
    <mtr>
      <msub xref="matelt">
        <mi>x</mi> <mrow><mn>m</mn> <mn>1</mn></mrow>
      </msub>
      <mi>&hellip;</mi>
      <msub xref="matelt">
        <mi>x</mi> <mrow><mn>m</mn> <mi>n</mi></mrow>
      </msub>
    </mtr>
  </mtable>
  <mo> ) </mo>
</mrow>
```

(The entities `&hellip;`*,* `&vellip;` *and* `&dtdot;` *expand to the characters* $\cdots$, $\vdots$ *and* $\ddots$ *respectively.)*

Now the `semantic_template` *element, for this case should be the following:*

```
<semantic_template><OMOBJ>
  <!-- the following variable represents the matrix -->
  <OMV name="M" id="matrix"/>
  <!-- the following function specification takes a matrix and a
       tuple (row,col) (expressed as a list). It returns the row,col th
       element of the matrix -->
  <OMBIND id="matelt"><OMS cd="fns1" name="lambda"/>
    <OMBVAR>
      <OMV name="M" xref="matrix"/>
      <OMV name="t"/>
    </OMBVAR>
    <OMA><OMS cd="linalg1" name="matrix_selector"/>
      <OMA><OMS cd="list2" name="list_selector"/>
        <OMI>1</OMI>
        <OMV name="t"/>
      </OMA>
      <OMA><OMS cd="list2" name="list_selector"/>
        <OMI>2</OMI>
        <OMV name="t"/>
      </OMA>
      <OMV name="M" xref="matrix"/>
    </OMA>
  </OMBIND>
</OMOBJ></semantic_template>
```

The reason why the coordinates of the element in the matrix is expressed as a tuple, is to bring the correspondence of the `semantic-template` *in line with the presentation markup (in which case this object must be given as an* `mrow`*).*
To see the full `notation` *element see the citation [15]*

8.2. Multiple Templates

There are certain situations where the expressibility of an OpenMath symbol is so broad that the same OpenMath symbol may be used to represent structurally disparate concepts. In order to deal with these incongruous symbols, it is necessary to provide more than one `semantic_template` child of the `Notation` element. It should be considered an error to have a `version` element which contains references to different `semantic_template` elements.

EXAMPLE: 9. *The example we shall look at is that of definite integration. This concept is represented by the OpenMath symbol* `defint` *from the Content Dictionary* `calculus1`*. The symbol may be applied over a "range (e.g. a set) of integration". This leaves the freedom that the symbol may be applied over an interval (with upper and lower bounds a,b respectively) in which case we might intend that it should be displayed as $\int_b^a f(x)dx$, or it may be applied over a set (denoted by S) in which case we may intend that it should be displayed as $\int_{x\in S} f(x)dx$. In order to provide anchors for both these notations, we need different prototype objects, we can do this by having more than one* `semantic_template` *element.* `semantic_template` *elements for these cases follow.*
For the case $\int_b^a f(x)dx$:

```
<semantic_template><OMOBJ>
```

```
  <OMA>
    <OMS cd="calculus1" name="defint"/>
    <OMA>
      <OMS cd="interval1" name="interval"/>
      <OMV name="b" id="intervalDefintArg1"/>
      <OMV name="a" id="intervalDefintArg2"/>
    </OMA>
    <OMBIND>
      <OMS cd="fns1" name="lambda"/>
      <OMBVAR>
        <OMV name="x" id="intervalDefintVar"/>
      </OMBVAR>
      <OMA>
        <OMV name="f"/>
        <OMV name="x"/>
      </OMA>
    </OMBIND>
  </OMA>
</OMOBJ></semantic_template>
```

In the presentation MathML for $\int_b^a f(x)dx$, the superscript, subscript for the integral and variable of integration (viz. a, b and x respectively) should have `xref` *attributes with values of (in this case)* `intervalDefintArg1`, `intervalDefintArg2` *and* `intervalDefintVar` *respectively.*

For the case $\int_{x\in S} f(x)dx$:

```
<semantic_template><OMOBJ>
  <OMA>
    <OMS cd="calculus1" name="defint"/>
    <OMV name="S" id="setDefintArg"/>
    <OMBIND>
      <OMS cd="fns1" name="lambda"/>
      <OMBVAR>
        <OMV name="x" id="DefintVar"/>
      </OMBVAR>
      <OMA>
        <OMV name="f"/>
        <OMV name="x"/>
      </OMA>
    </OMBIND>
  </OMA>
</OMOBJ></semantic_template>
```

In the presentation MathML for $\int_{x\in S} f(x)dx$, the subscript for the integral (viz. S) and the variable of integration (viz. x) should have `xref` *attributes with values of (in this case)* `setDefintArg setDefintVar` *respectively.*
The `semantic_template` *elements will be children of the* `Notation` *element for the* `defint` *symbol.*

8.3. Stylesheets Generated from Template Functions

The expressibility of OpenMath is very broad. The scheme we use for marking up template functions utilises the OpenMath markup for functions, this has a similar (though slightly more restrictive) expressibility. For a meta-stylesheet to create the correct annotation style-sheets, it is necessary that it encapsulates the functionality implicit in any template function it encounters, because of the above problem, we see that this is a very hard and indeed, for a generic template function, an impossible task. However certain values are returned from template functions with such frequency that it may be a good idea to implement meta-stylesheets implementing the calculation of these particular objects. The most common of these are:

– the selection of the i-th element from a list or collection of ordered objects.

– the selection of the i, j-th element of an array of values or a matrix.

– some object which specifies the repetition of a sequence of commands over a collection of object. It seems that it would not be possible to specify how to display every part of an aggregate object without this sort of return value.

These issues reflect current research (see for example [3]).

9. Specification of the TEX-like representation

Though the markup discussed in this section is sufficient for converting from extended MathML to TEX, it is not sufficient for converting from TEX to MathML. Work is in progress in order to address this problem, see [7]. We shall look at the markup for the specification of conversions from extended MathML to TEX. The information shall be held as a string of *character data* held in `tex` elements. This character data will be expected to be TEX source, except for the parts which are designated as representing the arguments of the function. For each argument we must store three fields, these are the following:

field 1: This is equivalent to the `xref` attribute in the `math` element. It should hold the same value.

field 2: This holds an arbitrary value which may be used to present the object in a generic setting.

field 3: This holds zero or more arguments which will be interpreted as the arguments for template functions, if field 1 is pointing to a template function.

This may be done by specifying the TEX template in the following manner:

```
<tex>
     ⋆⋆⋆\xref{val1}{default1}[arg₁1,···,arg₁m₁]⋆⋆⋆ ···
     ···  ⋆⋆⋆\xref{valn}{defaultn}[argₙ1,···, argₙmₙ]⋆⋆⋆
</tex>
```

In the above: $\star\star\star$ represents arbitrary TEX source, `val`i is the value of the i-th `xref` attribute, `default`i is the i-th arbitrary value and the values `arg`$_i j$ are arguments to template functions (if the link is not to a template function then the third field will be empty).

10. Applications

In the next section we shall consider a tool which has been written and for which our method finds useful application.

10.1. A Notation Selection Tool

We consider a notation selection tool for mathml [2]. This tool is an extensible notation selection tool, which provides a graphical user interface (GUI) to allow the user to select his/her notational conventions while transforming content MathML to presentation MathML.
The work on this is ongoing and details may vary in a future version. This, however, serves as a good example of the use of the concepts outlined in this paper.
The tool provides an interface which allows the user to specify an *Input File* (see section 10.2). An interface based on this is then created which allows the user to choose, from a set of menus and radio buttons, a graphical image of a notation for each mathematical concept handled by the notation files (see figure 2). The user may then input an (extended) content MathML file, this may then be displayed in the desired

notation, or alternatively a presentation MathML file may be created. The tool is extensible in that it references an XML file to extract graphical information, and XSLT templates in performing its transformation. Our method may be used to create these files (modulo the relevant `Notation` elements being available).

10.2. Structure of the Input File for the Notation Selection Tool

The XML document that is required by the Notation Selection Tool (we shall denote these *Notation Input Documents*) must have the following structure (In this example we assume that the relevant Content Dictionary is called `Arithmetic` containing symbols; for example `DIVISION`).

```
<mnotations>
  <catalog>  <!-- Arithmetic -->
    <name> Arithmetic </name>
    <itemlist>
      <item>
        <keyword> DIVISION </keyword>
        <choicelist>
          <choice>
            <image src="div1.gif"/>
            <keyvalue>1</keyvalue>
            <presentation>
              <!-- XSLT templates for this notation -->
            </presentation>
          </choice>
            ... <!-- other choices for DIVISION -->
        </choicelist>
      </item>
        ... <!-- other items of Arithmetic -->
    </itemlist>
  </catalog>
    ... <!-- other catalogs -->
</mnotations>
```

We may construct a `META` stylesheet which produces a Notation Input Document containing one catalog element for each Content Dictionary. The correspondance we make between the elements of the Content Dictionary and the elements of the Notation Input Document are the following:

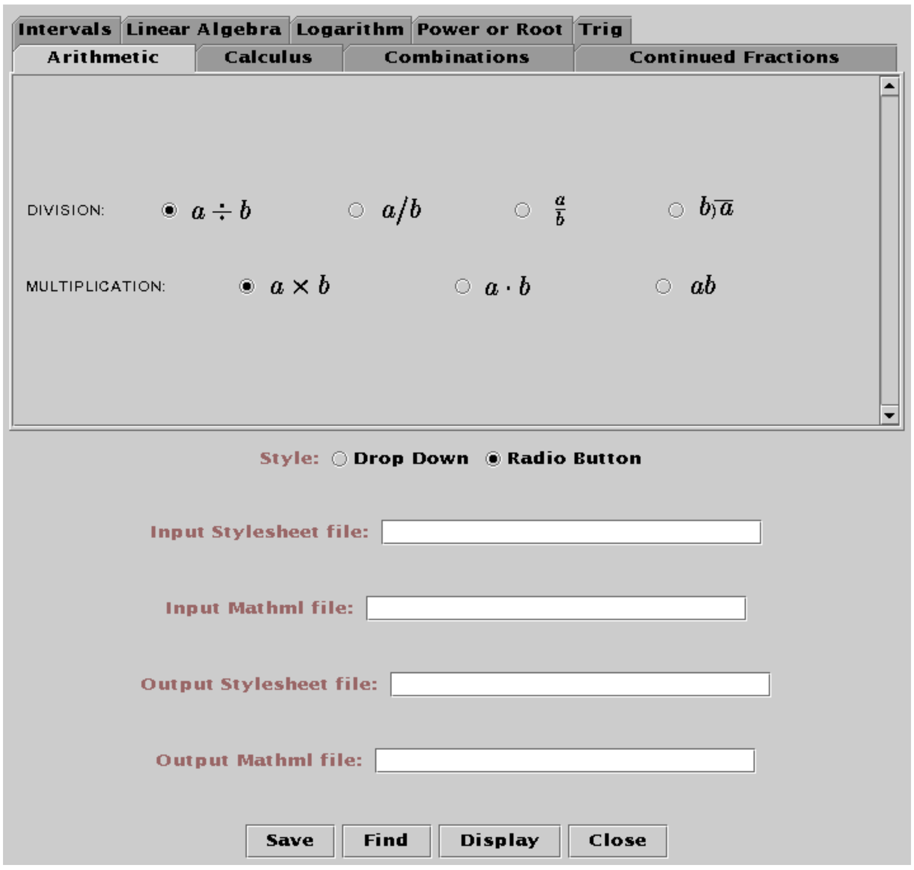
Content Dictionary	Notation Input Document
CD	catalog
CDName	name
CDDefinition	item
Name	keyword
Notation	choicelist
version	choice
image	image
math/semantic_template	presentation

It must be noted that the information which resides in the `presentation` element of the Notation Input Document will originate from three sources, the `semantic_template` elements of the Content Dictionaries, the `math` elements of the Content Dictionaries and the `META` stylesheets. The one element of the Notation Input Document that we have not dealt with here is the `keyvalue` element. The function of this element is to identify the particular style of notation required, this information is held in the `style` attribute of the `version` element under our scheme. We show a screen shot of the tool in action in figure 2.

11. Further directions

The problems which become apparent when we try to specify notations with an arbitrary number of elements, e.g. an arbitrary length vector or matrix, are not addressed in this work, we refer the reader to the paper [3] for discussion and progress in this area. A second issue relates to notations which are not local tree transformations of the semantic markup, e.g. $\sin^2 x$ rather than $(\sin\ x)^2$ or $x^2 - 3yx + 1$ rather than $x^2+(-3y)x+1$. We view these problems as building on the current work but under a separate mechanism. A separate project is the translation of TEX to MathML respecting macro definitions. The work of this paper can be used to generate stylesheets which give a direct correspondence between TEX macros and XML markup (MathML or XSLT templates), thereby retaining semantics which would be lost on expanding macros to low level TEX.

It would also be possible to write a tool which could translate between Extended MathML or OpenMath and LATEX by utilizing the information held in the `tex` child of the `Notation` element.



DIVISION: [✓ $a \div b$] a/b $\frac{a}{b}$ $b\overline{)a}$

MULTIPLICATION: [✓ $a \times b$] $a \cdot b$ ab

Style: Drop Down [✓ Radio Button]

Input Stylesheet file:

Input Mathml file:

Output Stylesheet file:

Output Mathml file:

Figure 2. A screen shot showing possible choices for division and multiplication

12. Conclusion

We conclude that the OpenMath Content Dictionary concept is a useful carrier to support notation databases in a form adaptable to automatic translation tools. This can allow a great flexibility in the use of representations and notations for mathematical objects and a simple mechanism for converting documents between these.

The specific form of the notation database shown here illustrates the ideas of the paper, but it is certain that collaboration with standards and bodies (most notably, the OpenMath consortium [11], the W3C working group on mathematics [9] and the OMDoc project [10]) will occur, in particular with regard to finalizing the names of various elements and attributes.

References

1. M.Abramowitz and A.Stegun: Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables.
2. Dicheng Liu: A notation selection tool for MathML stylesheets, University of Western Ontario MSc Project, 2001.
3. Bill Naylor: Mappings between presentation markup and semantic markup for variable size objects, presented at MathML 2002, Chicago, Illinois.
4. Bill Naylor, Stephen Watt: On the relationship between OpenMath and MathML, 2001 Workshop on Internet Accessible Mathematical Computation: http://icm.mcs.kent.edu/research/iamc2001.papers/nay.ps.gz
5. Luca Padovani, Irene Schena and Stephen Watt: Stylesheets for Mathematics in the Semantic Web (preprint).
6. Igor Rodionov: Tools for MathML, University of Western Ontario MSc Thesis, 2001.
7. Igor Rodionov and Stephen Watt: Tool for Translating TeX/LaTeX to MathML, presented as a Poster at MathML 2002, Chicago, Illinois.
8. Amaya: http://www.w3.org/Amaya
9. MathML: http://www.w3.org/TR/MathML2/
10. OMDoc: http://www.mathweb.org/omdoc/
11. OpenMath: http://www.openmath.org
12. XPath: http://www.w3.org/TR/xpath
13. XSLT: http://www.w3.org/TR/xslt.html
14. http://www.orcca.on.ca/MathML/texmmlom/partialDiffExample/
15. http://www.orcca.on.ca/MathML/texmmlom/matrixExample/