Discovering How to Write Semantic Math with new Symbols

Eric Andrès
Competence Center Visu
Saarland University

Michael Dietrich
AG Siekmann
Saarland University

Paul Libbrecht
Competence Center for e-Learning
DFKI GmbH

Abstract

The ActiveMath learning environment is based on semantic mathematical formulæ encoded using OpenMath[1]. This gives it a chance to render formulæ on a variety of platforms, using cultural-dependent adaptations, and with added-value services that may help the learners in reading the formulæ.

The price to pay at authoring, however, is high since it requires encoding the meaning and not only the graphical presentation of the formulæ. Examples of challenges include the input of $\mathbf{K}[x_1, ..., x_n]$ which is well known to represent the ring of polynomials on n variables but which does not enjoy, yet, the support of *official* Content Dictionaries for the symbols.

In this paper, we explain methods we propose to discover the symbols needed to encode expressions, the typical expressions, and the ways to input this within the ActiveMath learning environment with jEditOQMath and to have it rendered. En passant, we describe requirements on the browsing and search methods for the presentations of OpenMath Content Dictionaries.

> **symbol**: from Latin symbolum token, sign, symbol, from Greek *symbolon*, literally, token of identity verified by comparing its other half, from *symballein* to throw together, compare, from *syn-* + *ballein*: to throw
> Merriam-Webster dictionary

One of the foundations of the manipulations of formulæ in the ActiveMath learning environment is that they are expressed semantically in OpenMath; that is their representation encodes their meaning. This is a strong requirement compared to the long tradition of encoding mathematical formulæ to obtain their typeset form. It brings along several features (such as a mathematical search, a rendering which may help the learner, or the facility to copy-and-paste)

[1] http://www.openmath.org

and it promises interoperability with the world. Learning to write semantically is not only an issue of technical knowledge, that is, of knowing a given syntax, it is also a discovery of the writing methods which other mathematics authors have been using to denote the same meaning, thus trying to use the same *symbols* for the same meaning.

In the very center of this approach lies the notion of OpenMath symbols, materialized by the declaration of an identifier within a resource called a Content Dictionary as specified by the OpenMath standards [BCC+04]: the symbols are declared in these Content Dictionaries (which are expected to be globally available and browsable). However, no thorough mathematical definition is provided, but a description comes along together with a set of formal properties (which should be always true). They may be supported by notations and types found close to them.

The Content Dictionaries symbols represent the *semantic hook* to which one *points to* when speaking about the symbol, which corresponds to the definition quoted above. This pointer is realized in XML by the `OMS` elements used in any OpenMath expression to denote the occurrence of the given symbol.

The purpose of our article is this discovery: from the search and browsing actions of others' writing towards the ability to input the formula one intended. This article focusses on authoring content for the ActiveMath learning environment which uses OpenMath through OMDoc [Koh06], has a rich rendering engine and provides a Web framework enabling the features described above.

Because authors of ActiveMath need to use content for their ActiveMath, they search symbols which they can use in this learning environment and which they can input in such. Thus, their interest is in the way the symbols are input, are rendered, are explained, and interoperate. The interest is, thus, in their conversion methods and they tend to wish to ignore their XML nature but the semantic value as well as the XML encoding stays present and central as the only cross-application encoding.

1 Overview of ActiveMath Authoring

ActiveMath has, currently, one authoring tool called jEditOQMath based on the source paradigm, whereby an author edits a readable plain text file and activates a *compilation process* converting it to the OMDoc format.

The readable sources files are in OMDoc format, which enables the underlying editor, jEdit, to offer its services for XML editing (structure awareness, live validation, authorized children list, ...). The XML documents, themselves, are tamed thanks to the usage of a DTD which hides from the normal author a large amount of *useless* attributes and the many namespace declarations.

Mathematical objects in the jEditOQMath practice, could be written in OpenMath but this is typically very verbose, taking several lines for the simplest mathematical expressions. Another way is encouraged: the compilation process of jEditOQMath includes the OQMath process, itself encapsulating the QMath process [**?**], which converts a readable linear syntax, with various binary and

```
<symbol id="grad" kind="object" scope="global">
   <CMP>
    This symbol is used to represent the grad function. It takes one
    argument which should be a scalar valued function and returns a
    vector of functions. It should satisfy the defining relation:
    grad(F) = (\partial(F)/\partial(x_1), ... ,\partial(F)/partial(x_n))
   </CMP>
   <commonname>gradient</commonname>
   <commonname xml:lang="fr">gradient</commonname>
   <commonname xml:lang="ru">градиент</commonname>
</symbol>
```

Figure 1: The symbol for gradient

unary operator precedences, to OpenMath.

The compilation process enables jEditOQMath's principle *What You See is What You Check*: aside of converting the formulæ between $ signs, it publishes *the content collection* to the local ActiveMath server: the authors can use their content right away within the full learning environment; this is important since *using* is much more than viewing, the authoring results need to be checked for their hyperlinks, for the usage of their metadata (e.g. course generation), for their multiple access methods, for the rendering of all formulæ they contain...

Technically, the OMDoc documents are stored in a content-storage called *MBase*, which serves fragments of the OMDoc documents, which resolves references along imports, and which can be queried for relationships between fragments in both directions. The notation elements of the OMDoc documents are used to create the stylesheets which convert the fragments to HTML, XHTML+MathML, or PDF through TeX. Parts of the fragments are also sent to the computer algebra system, to the input-editor, or to the search engines.

2 Symbols in ActiveMath and their Relatives

In OMDoc, a symbol is an XML element, which has an identifier and lives within a theory; a symbol can be given titles (`commonname`) and descriptions (`CMP`). These can be complemented by examples and theorems, which represent the CMPs and their formal counterparts, the FMPs, of the OpenMath Content Dictionaries) [Koh06]. Figure **??** shows the symbol representing the mathematical concept of a gradient.

In ActiveMath, symbols are complemented by `symbolpresentation` elements which associate typical OpenMath expressions with their typical rendering. See [MLUM06] and [Lib07] for more information about them. Figure **??** shows a `symbolpresentation` for the gradient.

In jEditOQMath, symbols are written using the QMath syntax, which is extensible by the usage of new symbol declarations associating sequences of characters with the desired OpenMath symbol. QMath has symbols of various precedences to yield a readable syntax. Figure **??** shows the declaration of the character ∇ as a notation for the gradient.

```
<symbolpresentation id="veccalc1grad_pres1" for="grad">
  <notation precedence="500">
    <math><mrow><mo>∇</mo><mi am:precedence="300">f</mi></mrow></math>
    <OMOBJ>
      <OMA><OMS cd="veccalc1" name="grad"/>
        <OMV name="f"/>
      </OMA>
    </OMOBJ>
  </notation>
</symbolpresentation>
```

Figure 2: `symbolpresentation` for the gradient

```
Symbol: ∇      APPLICATION      "veccalc1:gradient"
```

Figure 3: Declaration of ∇ as notation for the gradient

3 The One Shot Myth

An eager author that just wishes to write content quickly expects almost all of the semantics to be invisible and have full automatisms, copying a formula somewhere and just pasting into his authoring tool.

Some effort in this direction is made in jEditOQMath with the built-in paste function: for example, this paste function will sniff the pasted content to see if it is an OpenMath expression and convert that to a QMath expression using the local notations. But that fails if there are unknown symbols. Moreover it may also be far from the notation the user expects.

Similarly, efforts are underway to use the canned notations embedded in tools such as WebEQ which can convert mathematical expressions available in presentation-MathML(MathML-P) to Content-MathML(MathML-C), provided the expressions in presentation actually are written with the anglo-saxon tradition of mathematical notation e.g., kgV(m,n) will not be convertible (kgV is the name of lcm in German), nor will the russian binomial coefficient notation.

As we see in the attempts above, hoping for a one shot transfer of mathematical notations from one place to another imposes very strong constraints on the compatibility of original and target contexts of the mathematical formulæ. Context is provided routinely in normal text-books by defining notations along the text, and in (La)TeX documents by macro definitions. The easy jump made possible by the world-wide web, however, allows unprecedented mixing.

If transferring LaTeX expressions, the typical solution would be to obtain the appropriate version of the package or macro-definitions, a duty that may end-up being daunting but which otherwise allows verbatim reuse of expressions.

OpenMath expressions, and many XML-encoded expressions, have the potential of providing enough information to be exchanged unambiguously from one place to another without much other context. This transfer is generally not fully transparent and often an author still needs to read, write, or arrange

the OpenMath expressions: the verbosity of the XML language is clearly not something a normal user wishes to see everyday. XML can be easily and unambiguously parsed and processed so that it remains the best way to write in electronic communications that focus on the expressions.

In what follows below, an author will only be able to avoid reading OpenMath if he stays in a world restricted to ActiveMath authored using (O)QMath. Since any OpenMath expression would be written with the same QMath tool, it would be not too hard to reproduce the context of authoring by copying (QMath) notations and the QMath terms for each of them. However, the world of interoperable tools with OpenMath is far greater than the world of ActiveMath authoring converters.

4 Ideal Story

Suppose that we were authoring a book about an advanced mathematical topic, e.g., algebraic geometry. We would quickly come to a point where we recognize the need for a new symbol which is not yet available. For the sake of example, let's say we would need the symbol for the quotient ring. In an ideal authoring world we would now activate a special kind of search tool where we enter our search query - the string "quotient ring". The tool, equipped with a number of dictionaries, would start its search for the missing symbol within the local authoring environment, further extending it to known repositories on the internet as needed. This search tool would display its results as soon as the first hit is available to allow us to stop the possibly time consuming internet search. According to our preferences, possibly acquired through data mining on previous searches, the tool would rate the search results and deliver them in descending order. A part of that rating would be a qualitative assessment of the symbol's semantic annotation. This could be done for example by checking the number of languages the symbol is available in.

Starting with the first search result, we would get a list of relevant symbols. By clicking on an entry in the search result list the tool would show an example presentation containing the new symbol along with the underlying source. We would only need to select the symbol that is best suited for our needs. The search and authoring tools could perform all the required changes to our authoring environment and we could continue the authoring process, well informed on how to input our newly found symbol.

Sadly, we are not there yet. An analysis of the features offered by the wonderful tool described above reveals that the process of writing math with a new symbol can be broken down into distinct subtasks:

Find the symbol: We first need to look for the symbol to see if it already exists

Enable the symbol: Once we have located our symbol, we need to make it available to our authoring environment

Use the symbol: We certainly want to use the new symbol, but figuring out how to do so can be a non-trivial task.

The next three sections will cover each of these tasks in more detail. In what follows, we assume that jEdit is used as authoring environment. jEdit is an editor completely written in java, which has been enriched with a lot of functionalities for ActiveMath content authoring. Of course other editors may also be used for authoring, but some adjustments will be needed.

5 Methods to Find Symbols

Once we have identified the symbol we need, several questions come to mind. What is the name of the symbol? Where can we find it, if it exists? This section provides information on how to find symbols that are already existing. Obviously the first place to look for the symbol in question is our own ActiveMath installation, because all symbols that are present there can be used easily.

To find out which symbols can be used in the current buffer we can use the notation-list, a function which is a part of the OQMath Plugin for jEdit. Figure **??** shows the notation-list in more detail. It lists all symbols that are available in the currently opened OQMath or QMath file. These symbols are either defined explicitly or by imported context files. The author can now search the list for his symbol or in case the search yields no results read through the list from top to bottom. A huge advantage of the notation-list is that it makes the symbols imported by the contexts visible.

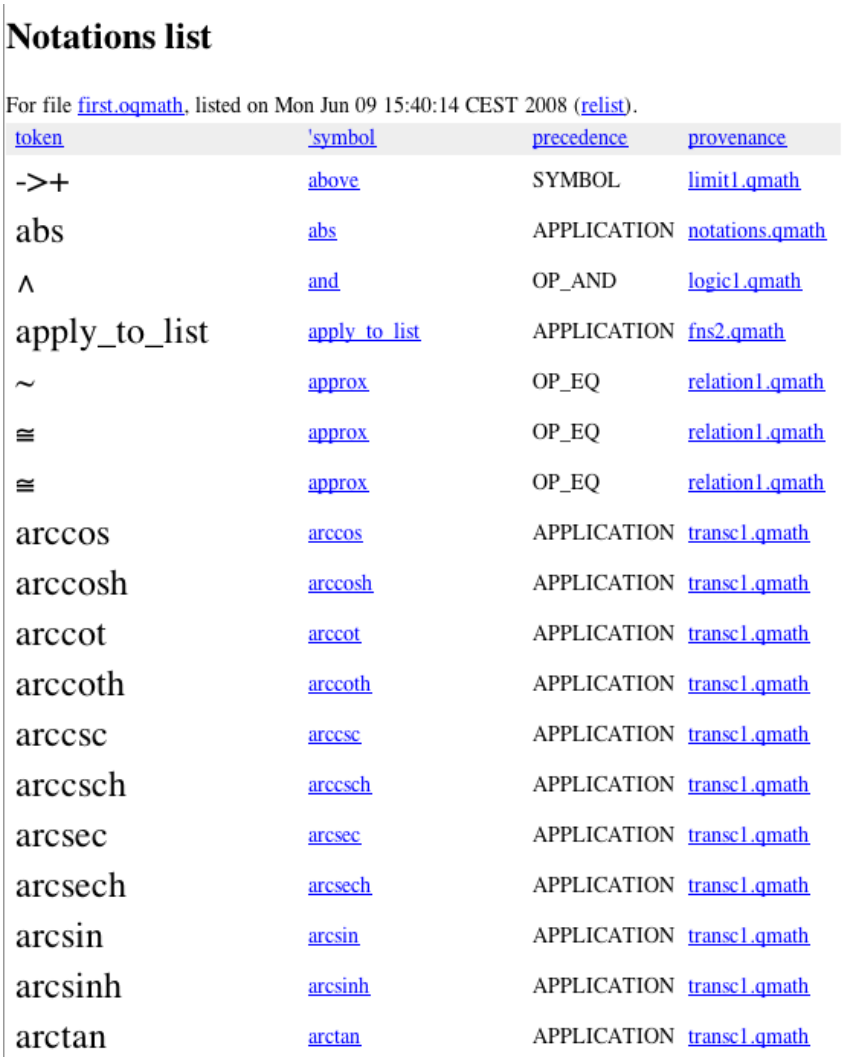
If the needed symbol is not contained in the notation-list, the next place is the Symbol Presentation List (which we will just refer to by presentation-list).

When logged into ActiveMath, an author finds the presentation-list in the 'Tools' menu bar.

In the left part of the screen we see all Collections that are loaded in this ActiveMath instance. Depending on the installation, the number of collections varies between 2-50(or more). A collection can be viewed by activating the checkbox left of the collection name and a click of the View-Button. To reduce the number of symbols presented, we may also specify the language of the symbols and add the requirement that the symbols must have a presentation, that means they can be rendered in a web-browser.

A fundamental collection is the `openmath-cds` collection which corresponds roughly to the collection of the official Content Dictionaries. It is a good idea to look in the `openmath-cds` collection for a symbol first. The search can then be extended to other collections as needed.

If the desired symbol is not present in the local ActiveMath, the next step in our search would be to visit `http://commons.activemath.org`, an ActiveMath installation which serves as a common collection repository and aims at containing as many as possible of the collections authored thus far. Hence, the presentation-list is very comprehensive, and the odds are good for a symbol to be found there if it is not too special.



Notations list

For file first.oqmath, listed on Mon Jun 09 15:40:14 CEST 2008 (relist).

token	'symbol	precedence	provenance
->+	above	SYMBOL	limit1.qmath
abs	abs	APPLICATION	notations.qmath
∧	and	OP_AND	logic1.qmath
apply_to_list	apply_to_list	APPLICATION	fns2.qmath
~	approx	OP_EQ	relation1.qmath
≅	approx	OP_EQ	relation1.qmath
≅	approx	OP_EQ	relation1.qmath
arccos	arccos	APPLICATION	transc1.qmath
arccosh	arccosh	APPLICATION	transc1.qmath
arccot	arccot	APPLICATION	transc1.qmath
arccoth	arccoth	APPLICATION	transc1.qmath
arccsc	arccsc	APPLICATION	transc1.qmath
arccsch	arccsch	APPLICATION	transc1.qmath
arcsec	arcsec	APPLICATION	transc1.qmath
arcsech	arcsech	APPLICATION	transc1.qmath
arcsin	arcsin	APPLICATION	transc1.qmath
arcsinh	arcsinh	APPLICATION	transc1.qmath
arctan	arctan	APPLICATION	transc1.qmath

Figure 4: jEditOQMath's notation-list

In both of these cases, using the presentation-list has the big advantage of letting the search be done visually, that is, by using the visual appearance as criterion. We have experienced that this is one of the favorite ways for authors to search for their symbols.

Another way to search for symbols in ActiveMath would be to use its search feature.[LM06]: ActiveMath authors have a slightly different search result. Additionally to the standard search result, their result contains symbols. Assuming we have author privileges, we have the possibility to use either simple or advanced search. Simple search is a text search using a lot of information retrieval techniques (like word stemming, fuzzy and phonetic search) to deliver an almost complete search result list. With the advanced search, more complex search queries are possible, including search for symbols containing a specific word and complex formulæ like assertions related to the needed symbol (deduced from a Content Dictionary's FMP).

The last place to search for the symbol before we are forced to create it ourselves is the set of all OpenMath symbols in published Content Dictionaries, for now this means visiting `http://www.openmath.org/`. We can browse through all symbols that are contained in the Content Dictionaries by visiting the **Index of all Symbols**. The browser's search functionality is really helpful here. It is also possible to list the symbols sorted by content dictionary which is useful if we know the content dictionary it is in or its topic area. Reading the Content Dictionary definition allows us to confirm that a given symbol has the semantic we wish. If the symbol is not present here, we must create the symbol by ourselves and ideally provide it for other authors to use.

6 Methods to Enable a Symbol Just Found

Once a symbol has been found, it needs to be enabled both for the author and the ActiveMath system. What needs to be done in detail largely depends on the location where the symbol has been found: it involves defining or importing a QMath notation for it. Ultimately, it amounts to putting the OMDoc representation of the symbol in a location where ActiveMath can load it. Furthermore, we need to declare that our collection depends on the collection the target symbol is in. This is done by declaring an "import" relationship. In what follows, we describe these processes in more detail, depending on the possible locations where a symbol can be found. At this point, we assume that the target symbol has no usable notation for the current buffer.

6.1 Symbols found in the local presentation-list or in a local example OMDoc

Every symbol that can be displayed in the local ActiveMath can be found in the local presentation-list, hence it is the perfect place to look for symbols. All these symbols are already available to ActiveMath, so all that is needed is to make the symbol available to the author, i.e., define a notation for it. In order to

Figure 5: ActiveMath's presentation-list

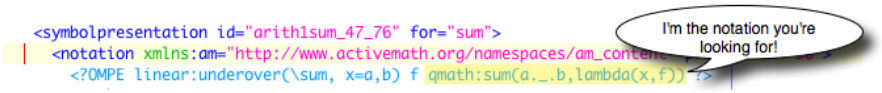


Figure 6: jEditOQMath showing the QMath prototype

do that, we first check whether there are already definitions of notations for the symbol. In the presentation-list, on each line describing a presentation, there is a link to a QMath notation, shown in Figure **??**

Following the link opens up the presentation's definition in jEditOQMath, this is shown in Figure **??**

We can find the QMath notation that has been used to generate the symbol's presentation in the notation line, behind the QMath prefix. In this example, it is `sum(a._.b, lambda(x,f))`. The link between the notation and the presentation is made more illustrated in Figure**??**. We now know how the notation is going to look like, but we also know that we can not use it yet - it is not yet declared for our buffer. We need to declare the notation, the required line inside the QMath processing instruction is:

```
Symbol: sum APPLICATION "arith1:sum"
```

This makes the notation available to the author. What we still need to do is to declare that our buffer now uses the symbol "sum" in theory "arith1", inducing a dependency on that theory. In order to do that, we declare an import:

```
<imports from="mbase://openmath-cd/arith1/>
```

With that information, we can now use the notation for the symbol, ActiveMath will also be able to interpret it's semantics. We are currently working on an enhancement of cut and paste for jEditOQMath (that could be called *paste-attempt*, which will make the necessary QMath and import declaration automatically.

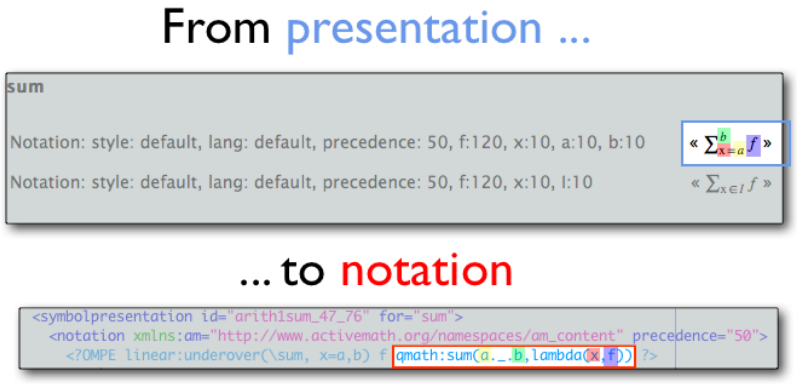


Figure 7: From presentation to notation

6.2 Symbols found on commons.activemath.org

As already described above, commons.activemath.org aims at being a central information hub for ActiveMath content. At this point, we assume that the desired symbol did not appear in the local presentation-list. Hence, we know that it is not yet available in our local ActiveMath installation. The first task is to figure out where to get the collection containing the symbol. If we are lucky, we will find a direct link from the presentation-list, to the collections main and development pages of the collection from where a download usually is available. Otherwise, `http://www.activemath.org/Content/` is a good starting place to look for it. Once found, the collection just needs to be installed in ActiveMath as usual,[2] and ActiveMath needs to be restarted. From that point on, the procedure to be followed is the same as in the previous paragraph.

6.3 Symbols found in official OpenMath Content Dictionaries with no presentation

The desired symbol may happen to be in an official OpenMath CD included in ActiveMath, but lack any useful presentation. In that case, we will need to define a presentation for it. A complete example for the scalar product of vectors is:

```
<symbolpresentation id="scalar_product_pres"
for="mbase://openmath-cds/linalg1/scalarproduct">
<notation>
    $a . b$
    <math><mrow><mi>a</mi><mo>.</mo><mi>b</mi></mrow></math>
```

[2]The installation of a collection for ActiveMath is described as a task at http://eds.activemath.org/en/inst-3

```
  </notation>
</symbolpresentation>
```

The QMath-expression after the notation-tag defines the prototypical QMath-expression that we want to use for referring to the scalarproduct symbol. In this case, it's the character '.' used as an infix operator. Below it, we put the MathML-P expression that will be used by ActiveMath to present this symbol. ActiveMath can automatically generate presentations for other output formats based on a MathML-P expression. In order for the presentation to work correctly, we need to define the operator '.' in the QMath processing instruction and to import linalg1 as well.

6.4 Symbols found in experimental Content Dictionaries

If the symbol is found in an experimental OpenMath CD or found out in the wild on the world-wide-web, it can not be used out of the box. We need to download the OCD-File from the web and transform it to OMDoc so we can make use of it. ActiveMath contains a stylesheet that does a reasonably good transformation of OCD to OMDoc as well as a stylesheet processor that can be used to do the transformation. The result of this transformation (if successful) could then be placed in ActiveMath's omdoc1/cd directory or, better, be placed in a collection that represents the source of this Content Dictionary (for example, the set of all Content Dictionaries made by the Riaca group at TU/e,[3] or those done for the project Intergeo[4]. After a restart of ActiveMath, the procedure from the previous paragraph is applicable.

6.5 Activating a new Symbol

At this point, we assume that the new symbol is available. Now, we need to figure out how to put it to work. The information we need for this, among others, includes the number of arguments that the symbol takes as input and the nature of those arguments. Furthermore, we also need to check whether a correct presentation is available.

A first approach to tackle this problem is to just play with the symbol. We may try to use it in a way that seems intuitive, this usually works well with simple binary symbols, e.g. the intersection of sets. These can usually be defined as operators, which, of course, constrains usage scenarios. If the symbol we just imported needs to be declared as an application, things tend to get more complicated. It may be the case that the usage of the symbol at hand is still obvious, e.g., for the vanishing ideal, it is pretty clear that it takes one argument. However, if things are not so simple, we need to use other methods.

For symbols that we imported from the presentation-list, it is easy to find usage examples . The location where we found our notation usually is an example by itself, we should then be able to use the symbol in an analogous way. We

[3] See `http://riaca.win.tue.nl/`.
[4] See `http://www.inter2geo.eu/`)

may also use ActiveMath's semantic search feature to find learning objects that use the symbol. A click on the corresponding formula right in the browser should allow us to see the source in jEditOQMath, which we can also use as an example.

Symbols imported from the OpenMath Content Dictionaries are documented and usually come with examples, but those examples are provided in plain OpenMath. At least, it allows us to see how many arguments the symbol needs, and we can experiment with that. For this purpose jEditOQMath provides a trial tool, called the QMath-Experimenter, which shows right away the OpenMath result of the QMath expressions input according to the syntax defined in the (last edited) OQMath buffer.

The only thing that we still may need to fix is the presentation of the symbol, if none is available. For this, the procedure described in section 6.3 can be used.

Now that the symbol can be entered and displayed properly, one is only left to make sure that this symbol has the right title, that is, the string displayed to users when hovering around a symbol; this is done by modifying the symbol OMDoc element, adding the necessary commonname elements.

Further checks for interoperability could be made in order for the symbol to reach its full usage within ActiveMath:

- if one expects this symbol to be input by learners then one should care for their ability to input it: for the Wiris input editor [MEC+06] the *domain editor* should be used to enrich the palettes with the necessary button.This is not needed if this symbol should only be shown within the copy-and-paste (and be input copy-and-paste into as described in [LJ06] since the notations are then used).

- if one expects this symbol to be exchanged with a computer-algebra-system or with the plotter, each of these tools should be enriched to support the new symbol. This procedure is generally not specified although [DL08] explains the Content Dictionaries can be used.

7 Conclusion

In this conclusion, we present open research questions which this paper raises.

In this paper, we have introduced methods to re-use OpenMath symbols from different sources. Applying the methods above is a way to avoid the creation of new symbols. This freedom to create new symbols is clearly supported by the OpenMath standard but it breaks most tools into the unknown when encountering it thus making the content considerably uninteroperable.

In some situations, a *mild* creation of a new symbol should be possible. Examples of these include the specialization of the semantics where the symbol we want is somewhat more precise than another existing one. FMPs can support tools in order to realize that the new symbol is *just a special case*. This can be done in several ways, described in [DL08] but one still needs a `special-case`

symbol which would enable even the notations-processor to use the notations of the more general case if a notation for the special case is not available.

A recurring theme that has appeared in this paper is the ability to paste which should, if all the context is ready for it, simply be a *one shot action*; as we have described, this often fails but is still often attempted. Examples in OpenMath of the usages of a symbol as indicated in the previous section are, currently, already pastable in jEditOQMath which will try to be smart enough to convert it to QMath with the local notation. Since the context is often not the same, this often fails.

This issue raises the strong need for the management of the notations along with the content elements of management, in OMDoc, the theories and the imports; this has been attempted in [MK07], and, indeed, runs a high risk of conflicts of notations which, then, need to be managed.

Moreover, this raises the need for a concretization of any notation information to enter the XML (and OMDoc) reference architecture. This is the case of the `symbolpresentation` elements of ActiveMath already. However, even though best-practice dictates that OQMath files are OMDoc files, the current OQMath tool is currently purely text-oriented; it is, thus, almost impossible to query the OQMath notation that has been used to encode a given formula.

An important architectural issue we have also met lies in the architectural separation of the authoring tool, expected to run on the client, and the ActiveMath server: indeed, currently, it is not possible for an author to use the mathematical syntax of his authoring tool, for example, to search for formulæ in the search tool; only the input editor is available in this tool. This is due to the fact that the ActiveMath server, although mostly running on the same host as the authoring tool client, ignores the architectural details of authoring tool. We shall explore web-service based methods to enable such a context-aware conversion, maybe even relying on the browser communicating directly to the authoring tool.

The final challenge of an author that has gone his way into using a new symbol is probably to cleanly expose his contribution as a retro-contribution to the community:

- having re-used the OMDoc symbol of an existing ActiveMath collection, contact should be made so that it can be contributed to the existing collection. This raises the necessity of a publicly visible *development web-space* for each content-collection where a community is reachable.

- having enriched a Content Dictionary, the result of this enrichment should be submitted for enhancement of the ActiveMath software (as an issue of its issue tracker) and/or of the Content Dictionaries on openmath.org/cd (probably as a post to the om-discuss@openmath.org mailing list).

References

[BCC+04] Stephen Buswell, Olga Caprotti, David Carlisle, Mike Dewar, Marc Gaëtano, and Michael Kohlhase. The openmath standard, version 2.0. Technical report, The OpenMath Society, June 2004. Available at `http://www.openmath.org/`.

[DL08] James H. Davenport and Paul Libbrecht. The freedom to extend openmath and its utility. *Journal of Computer Science and Mathematics*, 2008.

[Koh06] Michael Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents [version 1.2]*, volume 4180/2006 of *LNCS*. Springer Verlag Heidelberg, 2006. See `http://www.mathweb.org/omdoc`.

[Lib07] Paul Libbrecht. Content dictionary notations. In *Proceedings of the OpenMath Workshop, Linz Austria, 2007*, 2007. See `http://jem-thematic.net/node/167`.

[LJ06] Paul Libbrecht and Dominik Jednoralski. Drag and Drop of Formulae from a Browser. In *Proceedings of MathUI'06*, August 2006. Available from `http://www.activemath.org/~paul/MathUI06/`.

[LM06] Paul Libbrecht and Erica Melis. Methods for Access and Retrieval of Mathematical Content in ActiveMath. In Nobuki Takayama, Andres Iglesias, and Jaime Gutierrez, editors, *Proceedings of ICMS-2006*, number 4151 in LNCS. Springer Verlag GmbH, september 2006. Available from `http://www.activemath.org/pubs/bi.php?id=Libbrecht-Melis-Access-and-Retrieval-ActiveMath-ICMS-2006`.

[MEC+06] Daniel Marquès, Ramon Eixarch, Glòria Casanellas, Bruno Martínez, and Tim Smith. WIRIS OM Tools a Semantic Formula Editor. In *Proceedings of MathUI'06*, August 2006. Available from `http://www.activemath.org/~paul/MathUI06/`.

[MK07] Normen Müller Michael Kohlhase, Christine Müller. Documents with flexible notation contexts as interfaces to mathematical knowledge. In *Proceedings of MathUI 07, Linz.*, 2007. See `http://www.activemath.org/workshops/MathUI/07/proceedings/Kolhase-et-al-DocumentNotations.html`.

[MLUM06] S. Manzoor, P. Libbrecht, C. Ullrich, and E. Melis. Authoring presentation for OpenMath. In Michael Kohlhase, editor, *Mathematical Knowledge Management: 4th International Conference, MKM 2005, Bremen, Germany, July 15-17, 2005, Revised Selected Papers*, volume 3863 of *LNCS*, pages 33–48, Heidelberg, 2006. Springer.

[Pal06] Alberto González Palomo. QMath: A human-oriented language and batch formatter for OMDoc. [Koh06], chapter 26.2. See `http://www.mathweb.org/omdoc`.