Meeting Minutes 5 April 2001

Further Thoughts on MathML Subsetting

Author:
Warren Hedley (Bioengineering Research Group, University of Auckland)
Contributors:
David Bullivant (Bioengineering Research Group, University of Auckland)
Dan Gardner (Weill Medical College of Cornell University)
Alan Garny (University of Oxford)
Melanie Nelson (Physiome Sciences Inc.)
Poul Nielsen (Bioengineering Research Group, University of Auckland)

1 Introduction

On 4 April, I posted the previous two sets of meeting minutes relevant to this topic (from 27 March[1] and 2 April[2]) to the cellml-discussion[3] mailing list in an attempt to get feedback from the CellML community at large. The principle feedback I received (other than some additional typos that Melanie spotted) came from Dan Gardner and Alan Garny, who both get their own section devoted to their input.

In an attempt to improve on the draft CellML specification currently available, I've also proposed some rules regarding the correct way to call functions externally defined to CellML. This is discussed in Section 5. Note that I haven't gone into actually declaring those functions or how to embed them into MathML. I can postpone that because it doesn't involve MathML elements.

In Section 6, I've proposed a complete set of rules for the specification.

2 Dan Gardner's Feedback

The following is my attempt to summarise Dan Gardner's posts to the cellml-discussion mailing list in response to me sending out the meeting minutes. Dan thought that the compromise option (Section 3.3 in the 2 April meeting minutes[4]) wasn't too bad. However he was concerned that our subsets be designed to appeal to fields other than just cell biology. If "other specialized developers began using MathML for content manipulation, they too are likely to define subsets of MathML. In order for CellML not to be out of phase with this effort, perhaps the partition scheme should be designed to appeal to this wider audience and even submitted to the MathML working group." This might enable CellML implementers to make use of MathML capable software from other fields, and the community at large.

The idea of trying to achieve consensus on a sensible subsetting of MathML under the auspices of the MathML working group is a good one. However it currently appears that, on the timescale that we need to produce the CellML specification, it will not be possible to effectively collaborate with those without a direct interest in CellML. The W3C process generally involves lengthy debate (which is a good thing I feel, but a luxury we currently can't afford) and requires implementations before any sort of recommendation can be published.

[1] http://www.cellml.org/private/progress_reports/20010327_meeting_minutes.html
[2] http://www.cellml.org/private/progress_reports/20010402_meeting_minutes.html
[3] http://www.cellml.org/mailman/listinfo
[4] http://www.cellml.org/private/progress_reports/20010402_meeting_minutes.html

3 Alan Garny's Feedback

Alan Garny's feedback was delivered, in time-honoured tradition, orally (he's currently slaving away at the University of Auckland). Alan drew an interesting analogy between MathML and HTML. He pointed out that HTML 4.0 is a reasonably thorough specification that has been out for almost three years now, but you can't rely on any software that calls itself HTML 4.0 compliant to actually implement it properly. Why not, he suggested, forget subsetting the MathML elements altogether, say in the spec that everyone should implement everything, and take it for granted that they won't.

This wasn't quite what we were wanting to achieve with the CellML specification, so this suggestion was ignored. It's mentioned here for completeness.

Alan agreed strongly with Poul (so far the only one — well, he is foreign) that the burden of interoperability should stop with the implementor, and optimally, the user should never have to worry about this. Now consider the analogy between CellML processing software and web browsers. You fully expect all web browsers to be able to deal with all HTML 4.0 pages, and deal with them in the same way, but this is not the case. In CellML terms, you're not going to try and import and solve the HMT mechanics model using pathway editing software, but if you tried, you'd expect it to be able to do something — after all, the input is still valid CellML!

This point was a bit better, but not much.

Alan added one further analogy for good measure. HTML started out with a very small set of elements, and then further elements were added as implementors (go Mosaic) played around and developed new techniques. This might be a good approach to adopt for CellML he speculated. In CellML Version 1.0, specify a very small set of elements that we expect all software to be able to deal with, and that describes a large number of models. Implementors are free to add support for elements outside that set, but can not expect exported models to be portable to other applications. As implementors request new features, subsequent versions of CellML add to the set of MathML elements that we expect to be supported.

At this point, I started taking Alan seriously. CellML is coming up to Version 1.0, and we don't expect every implementor to deal with all of MathML. So why not just define a single, small subset for now, and then expand on it later when implementation experience has improved, and perhaps we have more tools available for dealing with MathML.

4 New Subsetting Recommendation

Throwing out the subsetting altogether is quite attractive because it saves me a lot of writing. I'm sure there are plenty of other good reasons too. However, the original reason that subsetting was considered was the complexity that implementors of CellML face when contemplating having to deal with MathML. Alan's feedback led to the idea of having a single set of MathML elements that we require all CellML processing software to support in order to claim CellML 1.0 Level 1 compliance.

Personally, I'm not a big fan of this solution, because it is reasonably close to what Poul Nielsen originally suggested, and that just doesn't feel right.

The only question remaining then is what elements should go in the MathML set? One possible set is given in Figure 1, which is the same as that described in the 2 April meeting minutes[5] with the addition of the `<true>` and `<false>` elements.

There are probably going to be calls for the elements given in Figure 2 to be added to the basic set. This is currently open to debate, but not for long. I think that it would be fine to add these elements in CellML Version 1.1, when we have a bit more implementation experience available to draw on.

[5] http://www.cellml.org/private/progress_reports/20010402_meeting_minutes.html

<cn>, **<ci>**, **<apply>**, **<divide>**, **<minus>**, **<plus>**, **<times>**, **<power>**, **<root>**, **<abs>**, **<eq>**, **<neq>**, **<gt>**, **<lt>**, **<geq>**,**<leq>**, **<and>**, **<or>**, **<xor>**, **<not>**, **<diff>**, **<bvar>**, **<degree>**, **<piecewise>**, **<piece>**, **<otherwise>**, **<exp>**, **<log>**, **<ln>**, **<semantics>**, **<annotation>**, **<annotation-xml>**, **<max>**, **<min>**, **<floor>**, **<ceiling>**, **<sin>**, **<cos>**, **<tan>**, **<sec>**, **<csc>**, **<cot>**, **<sinh>**, **<cosh>**, **<tanh>**, **<sech>**, **<csch>**, **<coth>**, **<arcsin>**, **<arccos>**, **<arctan>**, **<arccosh>**, **<arccot>**, **<arccoth>**, **<arccsc>**, **<arccsch>**, **<arcsec>**, **<arcsech>**, **<arcsinh>**, **<arctanh>**, **<notanumber>**, **<pi>**, **<true>**, **<false>**

FIGURE 1: Yet another proposal for a *basic* set of MathML elements for use in CellML.

<declare>, **<lambda>**, **<csymbol>**, **<int>**, **<lowlimit>**, **<uplimit>**, **<partialdiff>**, **<arg>**, **<real>**, **<imaginary>**, **<conjugate>**, **<sep>**, **<imaginaryi>**, **<vector>**, **<matrix>**, **<matrixrow>**

FIGURE 2: Some extra MathML elements that people are sure to want added to the basic set.

5 Function Calling

It would be nice if I could hold off on dealing with functions altogether until a future version of the CellML specification. After all, the correct method of defining functions in MathML using the **<lambda>** and **<declare>** elements is described in the MathML specification, so it's covered by the catch all "follow the MathML spec" rule. As far as I can tell, functions defined using these elements can be called using a **<ci>** element, although I can't find an example of this in the specification. I can comfortably postpone a way of defining functions in Ecmascript and embedding them in CellML until a later edition of the CellML spec, contenting myself with the fact that people can do it using elements in an extension namespace if need be. However, I think I do have to recommend a means for handling the **<csymbol>** element, which is used to call functions defined in terms of some non-MathML syntax. Despite this element not being in the basic set defined in Figure 1, I think that it is important that we recommend how this be used now so that anyone who chooses to use it will use it wisely!

Figure 3 demonstrates what I think is the correct use of the **<csymbol>** element to reference a function defined in non-MathML syntax. The function itself is written in Ecmascript and defined inside a **<script>** element, which is placed in an extension namespace. Note that this element is not intended to represent the correct way of defining external functions in CellML — it is for demonstration purposes only.

To properly nail down how the **<mathml:csymbol>** element may be used within a CellML document, I want to introduce the following limitations.

- To call a function, the **<csymbol>** element is *applied* to the function arguments as if it were an operator. Therefore we introduce a rule that states that the **<csymbol>** element must only be used as the first child element of an **<apply>** element.
- The examples in the MathML spec involving the **<csymbol>** element generally feature MathML presentation elements inside the element. We do not want processors having to deal with presentation elements unless they're specifically interested in them — this is what the annotation elements are for. Therefore, after leading and trailing whitespace is removed, the content of a **<csymbol>** element must be a valid CellML identifier. This ensures that processing software has something useful to render.

- The content of a **<csymbol>** element must accurately represent the external function referenced. This is a bit of a vague assertion, but it will be fairly clear what the expected behaviour is. For instance, if the element references a function declared using a normal programming or scripting language, it should probably be the name of the function.
- The **<csymbol>** element declares two attributes, **definitionURL** and **encoding**. The **definitionUR** attribute contains a URI pointing to the definition of the symbol, which is marked up using the encoding specified by the **encoding** attribute. In MathML the **encoding** attribute is optional, but in CellML both the **definitionURL** and **encoding** attributes must be defined on a **<mathml:csymbol>** element. It is anticipated that, when the CellML specification comes to define a scheme for specifying functions and scripts, the **encoding** attribute will be moved into the CellML namespace and required to assume a value from a controlled vocabulary.

```
<component
    name="component_with_script"
    xmlns="http://www.cellml.org/2001/03/cellml#">
  <variable name="A" units="metre" />
  <variable name="B" units="metre" />
  <variable name="C" units="metre" />

  <ext:script
      id="add_two_numbers"
      num_args="2"
      type="text/ecmascript"
      xmlns:ext="http://www.software.com/cellml_extension">
    function add_two_numbers(a, b) {
      return(a + b);
    }
  </ext:script>

  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply><eq />
      <ci> A </ci>
      <apply>
        <csymbol
            definitionURL="#add_two_numbers"
            encoding="text/ecmascript">
          add_two_numbers
        </csymbol>
        <ci> B </ci>
        <ci> C </ci>
      </apply>
    </apply>
  </math>
</component>
```

FIGURE 3: A demonstration of the correct use of the **<csymbol>** element to call an external function. In this instance, the function is defi ned using a **<script>** element in an extension namespace, that is defi ned within the parent **<component>** element. It is assumed that the **id** attribute on the **<script>** element is of XML type ID, allowing the **<csymbol>** to reference it using a URI. See text for more details.

6 New Rules for the Mathematics Section

It seems that it's about time to start considering a complete set of rules for the processing of MathML elements in CellML. A proposed set of rules is given below.

6.1 Rules for CellML Documents

6.1.1 The `<mathml:math>` element

1. **Allowed use of the `<mathml:math>` element**

   - The **`<mathml:math>`** element must only appear as a child of **`<cellml:component>`** or **`<cellml:role>`** elements.

     [In this and subsequent rules, the use of the `mathml` and `cellml` prefixes indicates that elements and attributes are in the MathML and CellML namespaces, respectively.]
   - All elements in the MathML namespace that are within a **`<mathml:math>`** element and not within a **`<mathml:annotation>`** or **`<mathml:annotation-xml>`** element must be taken from the set of MathML elements defined in {wherever the list given in Figure 1 is defined in the spec} ***(Level 1)***, or the complete set of MathML content markup elements, as defined in Chapter 4 of the MathML 2.0 recommendation[6] ***(Level 2)***.

     [CellML only makes use of the content markup elements from MathML. However presentation markup elements may be used within the annotation elements to associate equation rendering information with equations.]
   - The contents of a **`<mathml:math>`** element must conform to the Mathematical Markup Language (MathML) Version 2.0 recommendation[7] from the W3C.

6.1.2 The `<mathml:ci>` element

1. **Allowed use of the `<mathml:ci>` element**

   - After leading and trailing whitespace is removed, the content of a **`<mathml:ci>`** element must match the value of the name of a variable declared within the current component.

     [The **`<mathml:ci>`** element is used to reference variables from inside equations. It is valid to add leading and trailing whitespace to a variable's name to make the MathML more readable. The handling of whitespace in MathML is described in more detail in Section 2.4.6 of the MathML recommendation[8].]

6.1.3 The `<mathml:cn>` element

1. **Allowed use of the `<mathml:cn>` element**

   - A **`<mathml:cn>`** element must define a **`cellml:units`** attribute.

     [All bare numbers in MathML content markup are enclosed in a **`<cn>`** element in the MathML namespace.]

2. **Allowed values of the `cellml:units` attribute**

[6] http://www.w3.org/TR/2001/REC-MathML2-20010221/chapter4.html
[7] http://www.w3.org/TR/2001/REC-MathML2-20010221
[8] http://www.w3.org/TR/2001/REC-MathML2-20010221/chapter2.html#fund_collapse

- The value of the **cellml:units** attribute must be taken from the standard dictionary of units given in {the units part of the spec}, or be the value of the **name** attribute on a **<units>** element defined in the current **<component>** or **<model>** element.

6.1.4 The <mathml:csymbol> element *(Level 2)*

1. **Allowed use of the <mathml:csymbol> element**

   - The **<mathml:csymbol>** element must only appear as the first child element within a **<mathml:app** element.

     [A **<mathml:csymbol>** element must only be used as an operator, which is "applied" to some arguments.]
   - After leading and trailing whitespace is removed, the content of a **<mathml:csymbol>** element must be a valid CellML identifier as discussed in {the identifier part of the spec}. This identifier must accurately represent the external function referenced.

     [The content of a **<mathml:csymbol>** element should preferably be a human-readable identifier for the external function. If the function is defined using a common scripting or programming language, then this identifier should be the name of the function.]
   - Each **<mathml:csymbol>** element must define a **definitionURL** and an **encoding** attribute.

2. **Allowed values of the definitionURL attribute**

   - The value of the **definitionURL** attribute must be a valid URI that identifies a single resource containing the definition of the external function.

3. **Allowed values of the encoding attribute**

   - The value of the encoding attribute must indicate to processing software the format of the externally defined function.

     [Note that in the version of the specification that describes how external functions are to be defined within the CellML framework, the **encoding** attribute will be moved into the CellML namespace and will be required to take a value from an enumerated list.]

6.2 Rules for Processor Behaviour

6.2.1 Meaning of MathML content markup elements

CellML processing software must interpret all MathML content markup elements according to the meanings specified in Chapter 4 of the Mathematical Markup Language (MathML) Version 2.0 recommendation[9].

6.2.2 Processing of the <mathml:ci> element

CellML processing software must resolve references from **<mathml:ci>** elements to variables. The **<mathml:ci>** element must only be used to reference variables declared within the current component.

[9] http://www.w3.org/TR/2001/REC-MathML2-20010221/chapter4.html

6.2.3 Processing of annotations

The correct way to embed alternate representations of equations inside MathML elements is to make use of the **<mathml:semantics>**, **<mathml:annotation>** and **<mathml:annotation-xml>** elements. CellML processing software must interpret any MathML elements that occur inside **<mathml:semantics>** normally, but may ignore the contents of any **<mathml:annotation>** and **<mathml:annotation-xml>** elements.

6.2.4 Modification of variables

Expressions defined using MathML must only specify modifications to the values of variables that are not declared with a **public_interface** or **private_interface** attribute value of "in" in the current component.

6.2.5 Order of calculations

Any component may contain multiple blocks of equations that call multiple externally defined functions. The order of execution of equations in any block is not necessarily the order in which they appear. All non-differential equations will be calculated before differential equations. This allows the evaluation of parameters before they are input into differential equations. Processing software must determine the appropriate ordering for the calculation of equations across multiple blocks of equations and across the model in general.

Functions are assumed to execute *instantly* with respect to any independent variables, making it possible for the integrator to ignore their execution.