Meeting Minutes 9 April 2001
The Use of MathML in CellML

Author:
Warren Hedley (Bioengineering Institute, University of Auckland)
Contributors:
David Bullivant (Bioengineering Institute, University of Auckland)
Melanie Nelson (Physiome Sciences Inc.)
Poul Nielsen (Bioengineering Institute, University of Auckland)

1 Introduction

In this document I present and compare some of the alternatives that were considered for specifying the use of MathML within CellML. I intend to justify the final decision on the subsetting of MathML for use within the CellML specification.

In Section 3, I revisit the issue of CellML conformance levels (as described in Section 1.2.2 of the 2001-03-02 draft of the CellML specification) in relation to some issues that David Bullivant raised with respect to the handling of the subsetting of MathML. I'll also briefly review the CellML requirements that fall into the Level 2 category.

2 The Use Of MathML In CellML

2.1 Background

The CellML development team has always intended to make use of MathML's facilities for describing expression trees to define equations within CellML documents. During the development of CellML version 1.0 we have watched MathML become more complex and better defined as it developed from Version 1.0 (April 1998, revised July 1999) to Version 2.0 (February 2001). In fact it's become so complex that no-one really wants to implement the whole thing. This one of the biggest complaints we've heard from developers who wish to implement software that can read and write CellML. So in March 2001, I started looking at possible ways to break down MathML.

2.2 The Options

Options for the subsetting of MathML ranged between between specifying that software correctly interpret all MathML content elements, and specifying nothing and allowing software to interpret any MathML elements it likes. Neither of these options is workable. The first simply puts too great a burden on the implementors of CellML-compliant software and the second makes it unlikely that CellML software would ever be interoperable.

Somewhere between these extremes, we considered a solution where we split up the entire set of MathML content elements into a number of problem-type specific sets. The CellML specification would require that software be able to correctly interpret a single basic set, and would define other sets, that software could use to correctly advertise its capabilities. This was judged unworkable because of its complexity, and the fact that it only marginally improved interoperability.

The final scheme that we arrived at was to increase the size of the basic set a little, adding elements that we felt were essential, simple to interpret, or needed for consistency reasons, and not bother defining further sets. CellML-compliant software would be required to interpret all elements in this basic set, and was free to read and write other MathML content elements, but could not expect interoperability in this event.

It is anticipated that, as developers gain experience with MathML and the mathematical requirements of the modelling community become more clear, future versions of CellML will add additional MathML elements into the basic set. I like to think that the development of CellML will be analogous to the development of HTML, where elements and functionality are added to the specification in a backwards compatible manner based on implementation experience and experimentation. For instance, CellML V2.0 compliant software may be expected to interpret all of MathML's integration related content elements.

2.3 The Final Solution

CellML-compliant software will be required to correctly interpret all of the MathML content markup elements given in Figure 1.

In general, MathML that appears in CellML documents is only governed by the rules laid out in the MathML specification. However, to ensure that the MathML can be properly interpreted with respect to the CellML component it is embedded in, we need to specify some additional restrictions on certain elements.

3 Conformance Levels Issues Relating To MathML

- *token elements*

  `<cn>`, `<ci>`
- *basic content elements*

  `<apply>`, `<piecewise>`, `<piece>`, `<otherwise>`
- *relational operators*

  `<eq>`, `<neq>`, `<gt>`, `<lt>`, `<geq>`, `<leq>`
- *arithmetic operators*

  `<plus>`, `<minus>`, `<times>`, `<divide>`, `<power>`, `<root>`, `<abs>`, `<exp>`, `<ln>`, `<log>`, `<min>`, `<max>`, `<floor>`, `<ceiling>`, `<factorial>`
- *logical operators*

  `<and>`, `<or>`, `<xor>`, `<not>`
- *calculus elements*

  `<diff>`
- *qualifier elements*

  `<degree>`, `<bvar>`, `<logbase>`
- *trigonometric operators*

  `<sin>`, `<cos>`, `<tan>`, `<sec>`, `<csc>`, `<cot>`, `<sinh>`, `<cosh>`, `<tanh>`, `<sech>`, `<csch>`, `<coth>`, `<arcsin>`, `<arccos>`, `<arctan>`, `<arccosh>`, `<arccot>`, `<arccoth>`, `<arccsc>`, `<arccsch>`, `<arcsec>`, `<arcsech>`, `<arcsinh>`, `<arctanh>`
- *constants*

  `<true>`, `<false>`, `<notanumber>`, `<pi>`, `<infinity>`, `<exponentiale>`
- *semantics and annotation elements*

  `<semantics>`, `<annotation>`, `<annotation-xml>`

FIGURE 1: The CellML basic set, grouped according to function.