Meeting Minutes 19 June 2001

Dear CellML Guy — Questions About The CellML Spec

Author:
Warren Hedley (Bioengineering Institute, University of Auckland)

1 Introduction

In these meeting minutes, Warren (the CellML Guy) answers some of the more advanced questions put to him by those who have attempted the painful task of reading and/or understanding the 18 May 2001 Final Draft of the CellML 1.0 Specification. In some cases, the names and addresses of those asking the questions have been changed to mask their identity.

2 Equation Ordering

Andre Niquesans of *La Colline De L'un Arbre in the South Pacific* writes:

Dear CellML guy,

The CellML spec states that a processor cannot attach any significance to the ordering of the math in a CellML document which I'm interpreting as meaning that, assuming that all variables are resolved within the component, the processor is free to evaluate the equations in any order it likes. Does that sound about right?

The CellML Guy replies:

Dear Andre,

This is indeed an under-specified part of the CellML 1.0 specification. In the next edition of the specification you will find the following paragraph, which I think answers your question:

The mathematics in a model defined using CellML 1.0 consist of a static system of expressions, which are distributed over a network of components. CellML provides no facilities for specifying the order in which these expressions should be evaluated, as this is simulation information rather than model information. CellML processing software must not assume that the ordering of expressions within a component or within a CellML document has any significance, and may evaluate equations in any order when running simulations. For interoperability, it is recommended that CellML processing software evaluate expressions in an order which minimises the number of unknown variables in each expression.

3 Breaking pathways into components

Andrew the Beautiful from *the mainland* writes:

Dear CellML Guy,

I really enjoyed reading the CellML specification. I haven't read any work of fiction as fast paced and exciting since War and Peace. I failed to grasp some of the finer plot twists however and have some questions:

Given the following statements:

1. "Components are the smallest functional units in a model" (section 3.2.1, Definition of a component)
2. "Each variable in the model belongs to a single component" (section 3.2.3, Definition of variables)

and considering the following pathway:

```
A <-> B <-> C <-> D
```

how should this pathway be broken into components? In some cells `C` and `D` may not be present so by '1', `A <-> B` is a component. However, when using this component in a model in which `C` and `D` are present, does this `A <-> B` component remain, or are all four elements included (this would seem to violate '1')? If the former is true, is `C <-> D` now a separate component? What if `D` is absent? By '1' `A <-> B` should be a component but so too is `B <-> C`, but this violates '2'.

The CellML Guy replies:

Dear Andrew,

First of all, I must congratulate you: you've managed to write a paragraph more confusing than anything I've managed to put in the CellML specification. I am in awe of your obfuscatory powers.

To answer your question. Your pathway contains four species and three reactions. Each of these can be thought of as a functional unit, and therefore each gets its own component. A component that represents a species will generally contain nothing but a variable representing the concentration of that species, and an equation for the conservation of concentration that updates that variable based on what we call *delta* variables that are generated by reaction components. These delta variables contain the rate of change of a particular specie's concentration due to a particular reaction.

The components that correspond to a reaction define the reaction rate for use internally, and delta variables for all of the participating species. These delta variables are made available to the species components.

I'm sure you can see that this way of doing things satisfies both '1' and '2'. Taking this approach increases the chances for the re-use of reaction components. If a modeller isn't concerned about the re-use of components, it is possible to put parts or the entirety of a pathway inside a single component, but this is certainly not the recommended best practice.

4 How do I get the value of a variable in an encapsulated component?

Andrew the Beautiful from *the mainland* continues:

It appears that variable values can only be exported to the parent, sibling or encapsulated set of the component to which the variable belongs. If I find that a component in an encapsulated set requires a variable that belongs to a component that is not a parent, sibling or child, how should this be implemented?

The CellML Guy replies:

Encapsulation should be used by a modeller to hide the irrelevant detail of a complex system from the rest of a model. The component that encapsulates the complex subnetwork of components provides the sole interface between the subnetwork and the rest of the model. If a modeller finds that a component in the encapsulated subnetwork requires a variable from somewhere else in the model, he/she has three options:

1. **Add the variable to the interface** — The variable can be declared in the interface component with a inward public interface value and an outward private interface. Connections can then be created that pass the variable from the component to which it belongs to the interface component, and from there to the encapsulated component where it is needed.
2. **Break encapsulation** — If the interface component is serving to pass a large number of variables between the encapsulated network and the rest of the model, it may be more appropriate to break encapsulation and add the encapsulated components into the main model.
3. **Encapsulate the variable owner** — Based on the biological function of the encapsulated subnetwork, it may be appropriate to add the component with the variable of interest to the encapsulated subnetwork. This will generally involve re-wiring its connections with other components in the main model to pass through the interface component.

5 What's up with that unit coefficient in the Section 4.3?

Andrew the Beautiful from *the mainland* continues:

In Section 4.3[1], you wrote "The presence of the unit scale factor on the right hand side of the equation is needed for the equation to have consistent dimensions." and in Appendix C.4.4[2], "The first 1.0 in the equation is included specifically for units consistency. It would be possible to associate more complex units with the 0.1 in the numerator of the equation, but this would not accurately reflect the intent of the original model authors." I believe that adding an arbitrary dimensioned scalar is exactly the wrong thing to do, is unnecessary (in this example at least) and potentially leads to poor model equation design. In the original Hodgkin-Huxley paper they give the equation for the voltage-dependent reaction rates without (as far as I could see) specifying the units of each term in the equation. However, the equation (which does not include the "dimension-correcting term") must be dimensionally correct and therefore it is implicit that the first term of the numerator (almost certainly) has the units $mV^{-1}ms^{-1}$.

The CellML Guy replies:

Jeez Andrew, you do tend to go on a bit. However, I'll let you off, because it's probably going to bug some other people too. First of all, I'm sure that the only reason that the original model authors didn't specify the units for every term in their equations, was that they didn't have CellML! I have guessed at the intent of the model authors (in terms of units) based on the observation that the equation used in that example (and many of the similar equations) appear to have a repeated subexpression. In the denominator, the 0.1 must have units of mV^{-1}, and so I used the same units in the numerator. The unit coefficient is then needed to give the equation consistent dimensions.

In summary, the association of units with the various terms in the equations of legacy models is somewhat arbitrary. The CellML 1.0 specification does not require the equations in valid CellML documents to have consistent dimensions. However, the recommended best practice is for model authors and software to carefully consider the units associated with the terms in equations and to balance them across each equation.

6 A component can be in how many hierarchies?

Andrew the Beautiful continues:

I did not understand some of the Grouping section and this may relate to the previous question regarding passing variable references/values to members of the hidden set. What does it mean that "A component must only appear once within a given hierarchy, but may appear in multiple hierarchies if each of these hierarchies is of a different type." What are different types of hierarchies? Can a model be arranged into many different hierarchies, which can share components?

and **Wian Jang** of *Pittsburgh, USA* adds:

p53: Can there be recursive grouping?

p54: Can I not have a component in more than one hierarchies?

The CellML Guy replies:

The grouping section has certainly caused a bit of confusion. Sure, it sounds good when explained at conferences, but it's not until you see the specification, that you realise the real dark heart of CellML.

A model defined using CellML should be thought of fundamentally as a network. Any model can be reduced to a simple network of components where the values of variables are passed between the components along connections. It is sometimes convenient to logically group subsets of the components in a network together, a process which we've called *encapsulation*. A modeller might also want to specify some simple geometric relationships between the components, such as "A is inside B", a relationship that we've called

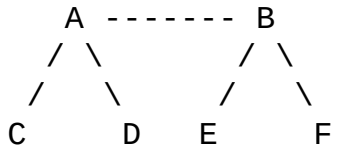
[1] http://www.cellml.org/public/specification/20010518/cellml_specification.html#sec_math_examples

[2] http://www.cellml.org/public/specification/20010518/cellml_specification.html#sec_units_examples_equation_dimension_checking

containment. In CellML, modellers may define numerous hierarchical arrangements of components, each one with a different type.

Both of these kind of relationships are implemented in CellML using a process called *grouping*. We could have called it hierarchying [sic], but we wanted to leave the option open to specify a hierarchy in pieces, so that for instance, the relationship between an encapsulating component and its encapsulated components can be defined in a separate part of a CellML document from other encapsulation relationships. The thing that links the groups together into a hierarchy is the *relationship type* — i.e., any two components that are linked by a given relationship type are parts of the same hierarchy. A given hierarchy itself may not be explicitly continuous. However by attaching an imaginary parent component to any component that doesn't have a parent component explicitly defined, we can form a single continuous hierarchy and check for loops, etc.

A model can contain at most one encapsulation *hierarchy*, which may be defined using several different **<group>** elements. Consider the elegantly rendered simple model below, where A encapsulates C and D, and B encapsulates E and F:



Encapsulation prevents connections between an encapsulated component and any components other than its parent, children or siblings (other components with the same parent). So, for instance, D cannot be connected to E. The fact that a encapsulation hierarchy affects connections means that we can only define a single encapsulation hierarchy in a given model — otherwise we'd need to define a whole new set of connections.

However, containment hierarchies have no effect on the model. This means that we could actually define multiple containment hierarchies over a single network of components. To distinguish these, CellML has an attribute allowing modellers to name geometric hierarchies, effectively setting up a number of geometric relationship types. Any component may appear in any number of hierarchies, but only once in each hierarchy.

7 The root element of a CellML document

Wian Jang continues:

The specification suggests that both the **<model>** and **<component>** elements can be the root element of CellML documents. Other languages have a consistent root element with the name of the language. Why not have a **<cellml>** element? Does this have something to do with namespaces?

The CellML Guy replies:

Dear Wian,

When the specification suggests that **<model>** and **<component>** are typical root elements, this should be viewed as just a recommendation for files. It is natural to think of CellML as purely a file format, but it is so much more! Consider a database designed to store cellular models, that exports CellML to applications based on queries it receives. The database may well export fragments of CellML that have **<reaction>** or **<connection>** as the root element. These are not necessarily invalid.

Adding a **<cellml>** element to the language to be the root of any CellML document or fragment is simply a waste of disk space, bandwidth and specification writing time! There are much better ways of identifying languages, and by far the best is to use namespaces. Making use of the CellML namespace allows processing software to recognise any CellML element at any point within any XML document. It is

important to realise that, as far as an XML parser is concerned, `<model>` (for example) is really only half of the element's name. The real name of the element is:

`<http://www.cellml.org/cellml/1.0#model>`

In a CellML document placing an element in the CellML namespace would typically look like this, where the attribute `xmlns` sets the default namespace for the current element and all of its children:

```
<model xmlns="http://www.cellml.org/cellml/1.0#" />
```

As you mentioned, many lesser languages have a single root element. This has occurred because namespaces are a relatively new concept. They are not really compatible with DTDs, which represent in many cases the entire specification for an XML application. Until recently, DTDs were the only widespread validation mechanism implemented in XML parsing software, which explains the slow acceptance of namespaces. With the XML Schema specification recently reaching Recommendation status at the W3C, I expect that namespaces will become more popular. Note that the DTD included in Appendix A of the CellML 1.0 specification actually includes limited support for namespaces, assuming that people follow the guidelines for defining namespaces recommended in the specification.

8 Why are reactions reversible by default?

Marilyn Mo of *North America* writes:

Dear CellML guy,

The CellML specification states that the default value of the `reversible` attribute on the `<reaction>` element is `"yes"`. In [the software that I use], the default value is `"no"`. Users of [the software that I use] have been happy with reactions being, by default, irreversible to this point. You should definitely change this.

The CellML Guy replies:

Dear Marilyn,

I detect that you are still bitter about the caption on that photo I took of you in New York. This kind of pointless stirring just wastes everyone's time!

The decision to make reactions reversible by default was made after extensive consultation between myself and representatives of a small biotech startup in New Jersey (who will remain nameless). I apologise profusely on behalf of those who chose not to include you in this consultative process — they will be sternly reprimanded. The decision was based on the observation that the majority of reactions were, in fact, reversible, and not based on current software practices.

To change this at this point would constitute a change to the syntax of CellML. I am no longer in a position to make such changes without the support of the wider CellML software development community. Did you realise that there could potentially be thousands of copies of the CellML specification in circulation at this point? To make changes at this late point would involve an intensive media campaign to ensure that the CellML community was aware that these changes had been made.

9 The reaction participant classifications don't handle ...

Marilyn Mo continues:

Section 7.4[3] of the CellML specification defines seven possible roles for reaction participants [`"reactant"`, `"product"`, `"catalyst"`, `"activator"`, `"inhibitor"`, `"modifier"`, and `"rate"`]. There is

[3] ${HTML_SPEC_20010518_ENTIRETY}#sec_rxn_basic_structure

another possibility: a lot of proteomic data (yeast 2-hybrid, etc) yield information on whether two proteins bind, but nothing about what the effect of binding is, or what the rate constant might be. Also `"rate"` is not independent of the reaction role e.g an activator can have a rate. I suggest replacing `"rate"` with `"binds"` and allowing a field [attribute? - ed] **`rate`** to be defined under any of `"catalyst"`, `"activator"`, `"inhibitor"`, `"modifier"` and `"binds"`, with the caveats of the mathematics as you've defined in section 7.5.6.

The CellML Guy replies:

A great man once told me: "show me a bunch of equations, and I'll show you true happiness". Or something like that. In terms of quantitative model simulation, the classification of reaction participants is really just metadata (the real data being model structure and mathematics). It has been included in the core of CellML because the reduction of a pathway model to straight mathematics may result in the loss of information that would enable the straightforward rendering of chemical expressions and pathways. The classification of reaction participants also allows the definition of simple qualitative models.

The choice of classification scheme is somewhat arbitrary, but as with the reversible reactions you mentioned in your last point, the current scheme is the result of an exhaustive consultation process. We realise that we can't make everybody happy here, and have aimed for a lowest common denominator in order to keep the standard simple for software developers to implement to. The `"modifier"` classification can be used by modellers to describe interactions that do not fall into any of the other categories. I expect that software will extend CellML in this area to add more detailed information, or store additional reaction information as metadata. As we get a better idea of the needs of the community, the most common extensions to CellML, and the most common types of metadata, will find their way into future versions of the CellML specification.
