Meeting Minutes 24 August 2000

Author:
Warren Hedley (Bioengineering Institute, University of Auckland)
Contributors:
Poul Nielsen (Bioengineering Institute, University of Auckland)
David Bullivant (Bioengineering Institute, University of Auckland)

1 Summary

At the CellML meeting on August 24 2000, Warren presented a notation convention that might make subsequent CellML discussion and examples easier to understand when thought of in terms of the CellML 99 system. The basis for the CellML data model was discussed, before a debate about the *role* of variables interrupted that discussion. Some doubt was expressed over the necessity of inheritance functionality in the connection datatype. The definition of a very simple EP model and a very simple pathway model using the current CellML system was discussed.

2 Graphical Notation

Warren presented a possible standard notation for the simple rendering of models in CellML related discussion. This notation is shown in Figure 1. The symbols in the design are completely arbitrary and the colours used are purely a byproduct of the colours of the whiteboard markers available.

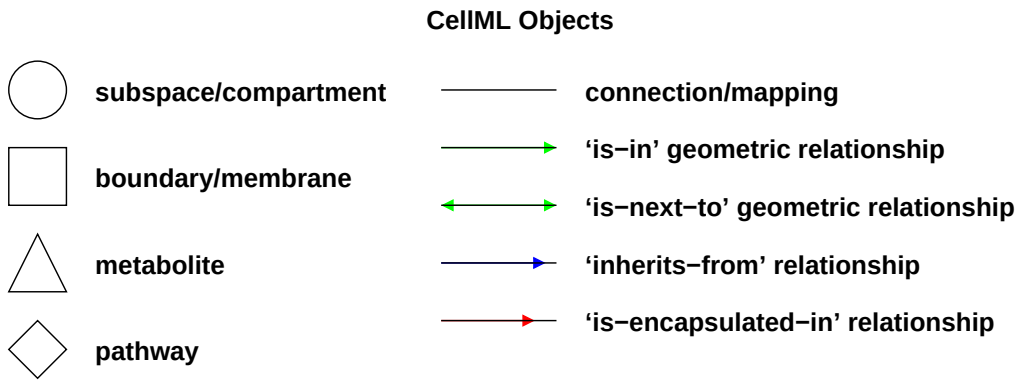

FIGURE 1: A notation proposed for the rendering of models for the examples in the CellML specification and associated documentation. The notation includes four possible representations of the standard CellML **component** data type.

Currently we are only concerned with two geometric relationships: '*is-in*' and '*is-next-to*'. The first is directed, so the arrow points from the child to the parent, whereas the second is not, so there are arrowheads at both end of the connection.

3 Variable Role

Warren had originally proposed separating the role of a variable into two parts: the first would be a boolean determining if the variable was declared inside the current component or was a required input, and the

second a scope flag with possible values of public and private. David and Poul had then suggested that the two were quite intimately connected: that a component should have variables of type *in* (required variables), *out* (public variables), *in+out* and *none*. At this meeting it was proposed the the *in+out* value was perhaps unnecessary, and probably complicated things for code generation. There were two other ways to pass variables across multiple components: the first involves just adding connections from all components that require a given variable to the components where a variable is declared, and the second is to provide two different versions of a variable (one of type *in* and one of type *out*) in a component and provide an identifying equation in the component that equates the two.

4 Examples

In order to demonstrate how the CellML data-model could be effectively applied to cellular modelling, we decided we needed some simple examples. The first of these is a small subset of a typical electro-physiological model, and is shown in Figure 2. The model consists of two "subspaces" (extra- and intra-cellular) and the boundary between the two (a membrane).

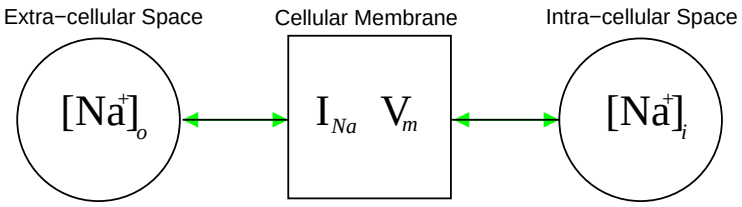

FIGURE 2: An example network featuring a small subset of a typical electro-physiological model.

This model is primarily concerned with the fast sodium current from the DiFrancesco-Noble model. The extra-cellular subspace contains a variable representing the concentration of extra-cellular sodium $[Na^+]_o$ which is required by the membrane boundary for its calculation of the channel reversal potential. This means that $[Na^+]_o$ must have role *out* in the extra-cellular subspace, and must be declared with role *in* in the boundary. The fast sodium current I_{Na} is calculated in the membrane boundary, along with the membrane voltage V_m but both variables are used in the extra-cellular subspace for calculating, among other things, the conservation of sodium concentration. Obviously these variables will be declared in both components with role attributes of type *in*.

Connections are made between the extra- and intra-cellular subspaces and the boundary. These connections contain mappings between the sodium concentrations in the subspaces and in the boundary, and between the boundary's variables and variables with the same name in both subspaces. The connections also contain geometric relationship information of the "*is-next-to*" variety.

The second example is the pathway model from Section 4.1 of the CellML requirements document, which demonstrates how adding encapsulation relationship information to connections can be used to hide unnecessary information from the user, effectively creating different layers of complexity in the model. At the topmost level, shown in Figure 3, the user sees a pathway with metabolites **A**, **B** and **C** as inputs, and **AB** and **D** as outputs.

However this simple pathway is actually hiding several more complex sub-processes. These are included in the network by adding '*is-encapsulated-by*' relationship information to the connections between the levels of complexity. The '*is-encapsulated-by*' identifier allows the modeller to collapse a complex network into a single component which is placed into another network, providing an interface between the different layers of complexity. It is invalid for components on different levels of complexity to be connected through

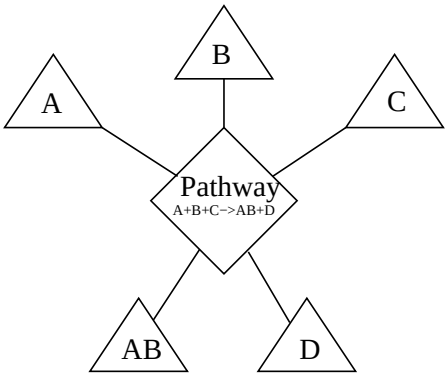

FIGURE 3: The simplest view available to the user of the pathway model from Section 4.1 of the CellML requirements document.

any component other than the encapsulating interface component. The pathway in Figure 3 is actually made up of a network with sub-pathways and intermediate metabolites, as shown in Figure 4.

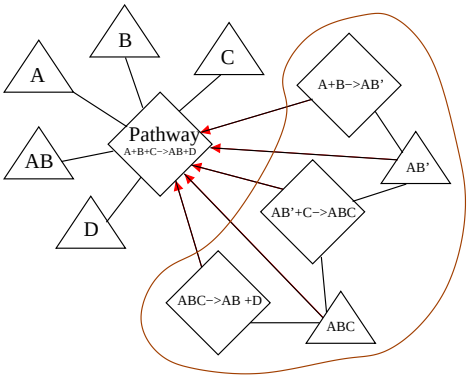

FIGURE 4: The full pathway model from Section 4.1 of the CellML requirements document. The pathway shown in Figure 3 has been expanded to show the underlying network encapsulated within (the second layer of complexity is the network inside the brown line). Note that the only connections crossing between the layers of complexity have the '*is-encapsulated-by*' identifier, as indicated by the red arrowhead.