Meeting Minutes 13 September 2000

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

1 Summary

On September 13, Warren and David sat down and discussed the differences in semantics between encapsulation and geometric relationships. The major thing to come out of this was a set of rules for encapsulation and a switch from connection "flags" to a grouping mechanism. The actual rules for geometry relationships were not finalised, but some interesting points were raised with regard to geometry and variable hierarchies. Finally, a variable role of `in-out` was finally given the go-ahead.

2 Encapsulation — A Clarification

The most important result of the September 13 meeting was a clarification of the role of encapsulation relationships in CellML, and the rules that go with an encapsulation relationship. A possible XML serialization of encapsulation definition was also proposed.

2.1 What Is Encapsulation

Encapsulation is a mechanism for limiting the external interface to a network of components to a single "root" component. The "root" component that provides the interface is the only component visible to components in the network in which it is placed. Similarly the root component, the other components encapsulated by the same root, and any components encapsulated by the current component are the only components visible to each component in the encapsulated network.

In less technical terms, encapsulation can be used to hide a complex network from the user, by putting all of the important inputs and outputs into a single component.

2.2 The Rules

The encapsulation relationship places some restrictions on the connections available to the root and child components.

- Each root component can only be the root component of one group of encapsulated components.
- Each child component can only be a child component in one group of encapsulated components.
- An encapsulated component can only be connected to its parent, siblings and children.

Note that encapsulation does not imply any inheritance behaviour as had been stated in early versions of the CellML specification. All variables with role `in` on both the root and child components must be explicitly mapped to variables with role `out` or role `in-out` in connections.

2.3 Grouping

In early versions of the CellML specification, it was proposed that encapsulation information should appear as a "*is-encapsulated-by*" flag on connections between the root and child components. It has been pointed out that, at least in an XML serialization using this method, it would not be a simple matter to work out which components were encapsulated by which — this process would involve processing all of the connections defined in the model. Editing of the model in its XML form would be unnecessarily error-prone as a result, because the relationship between components would not always be obvious.

By defining all of the components involved in a single encapsulation relationship in one place, some of this hassle can be removed. The grouping mechanism encapsulation information could also be used either together or separately for the definition of geometric hierarchies. This is best demonstrated by an example such as that in Figure 1, where sodium and calcium channels are encapsulated within a membrane. Note that a geometry flag is added to the grouping to indicate that the channels are also *inside* the membrane.

```
<grouping xmlns="http://www.cellml.org/2000/10/CELLML">
  <mode>encapsulation</mode>
  <mode>geometry</mode>
  <root>membrane</root>
  <child>I_Na</child>
  <child>I_Ca</child>
</grouping>
```

FIGURE 1: A possible XML serialization of the grouping mechanism used to store the geometric and encapsulation relationships between components. (Note that the `xmlns` attribute is the namespace mechanism used to set the default namespace for an XML element and all of its child elements.)

3 Geometry

It was generally agreed that if we to move to a grouping mechanism for defining encapsulation relationships, we should use the same mechanism for defining geometric and variable hierarchies. However the definition of a set of rules for the specification of geometry is not yet possible as there are still a few issues to sort out.

3.1 What Is Geometry?

We need to be able to indicate that one component is inside another using CellML. The definition of such a relationship between two components in a model does not add any information to the model and is independent of all other information in the model with the exception of encapsulation relationships. Specifically, it is invalid to have the root component of an encapsulation grouping inside one of its child components.

The very simple geometric relationship that we seek to define doesn't affect the underlying mathematical model in any way, so could be classified as rendering information. One could validly argue that, as a result, this is outside the scope of CellML, but this information is regarded as so vital it must be included in the CellML data model.

3.2 The Rules

The rules for geometry haven't been finalised yet. They may well prove to be more complex than the rules defined for encapsulation relationships because a modeller might reasonably want to define several overlapping geometric hierarchies on top of the same network. This is actually best demonstrated using an example from outside the realm of cellular modelling — anatomy! In a model of the human body, a typical geometric grouping of bones is not compatible with the typical geometric grouping of muscles, specifically because the muscles normally connect bones from different groups in the bone geometry hierarchy. As the CellML data model should work well at a range of resolutions, including the anatomy, there is no reason why this problem wouldn't occur at the cellular scale (although I can't think of a good example), so CellML must undoubtedly be able to deal with multiple geometry hierarchies.

The definition of multiple hierarchies over a network means that the grouping mechanism will require an identifier to allow the CellML processor to evaluate which groups belong to the same hierarchy. For the anatomical example above we might easily assign the identifier `bones` to one grouping hierarchy and the identifier `muscles` to the other. In the anatomical example the groupings would not share root or child components, but this possibility need not necessarily be eliminated.

In the grouping mechanism that we have defined for encapsulation, a root node is required. One can imagine that this is not strictly necessary in a geometric grouping, where one might want to just reference some components and assign that grouping a name. When I want to define groups further up my geometry hierarchy, I only have to reference those groups. Consider again the anatomical example. With the previously defined grouping mechanism I would have to define a component called `lower_arm` for instance to group my radius and ulna bones. This component is not a useful addition to my model: it contains no information and is not connected to anything. As of September 13 this is still very much up in the air — maybe the geometric grouping mechanism is in fact different from the encapsulation grouping mechanism.

One particular feature that differentiates the geometric relationship from the encapsulation relationship is that connections may be made between components at all levels in a geometric hierarchy and to components outside of the hierarchy. This allows for instance the connection between the diatic subspace and the T-Tubule subspace shown in Figure 2. This would not be possible if the SR and diatic subspaces were encapsulated within the intra-cellular subspace, in which case communication would have to be through the intra-cellular subspace.

4 Variable Hierarchies

Variable hierarchies are another term we use for variable inheritance schemes. Inheritance is regarded by the Auckland group as a bit of a dirty word, because of the ambiguities it can raise in model definition. And the Physiome software team agrees that explicit mapping of variables makes life a lot easier for everyone. This is not to say that software can not give the appearance of having variable inheritance — the user may think they're defining temperature globally, but the CellML representation of that involves passing the variable around. Possibly this raises some difficulties when reading a network back in, but a variable could be flagged as global using metadata or an application specific extension.

Although global variables are not part of the CellML data model, they will almost undoubtedly be made available to the user in any cell-modelling software package, so we should consider carefully all of the issues relating to global variables. In general one could implement global variables by declaring a *world* component and making all of these variables visible there, and forming connections from any component that make use of the global variables to the *world* component. However any component that was encapsulated inside of another one would not be able to connect to the *world* component. This isn't a problem as such — the global variable can be passed through the encapsulator via a variable of role `in-out` — it just highlights a case where encapsulating a component may force changes to the external interface of the encapsulator (some people seem to care about this kind of thing.)

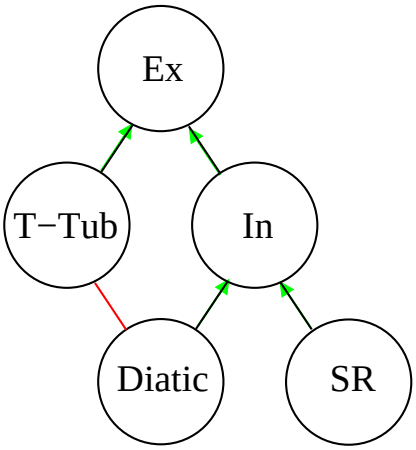

FIGURE 2: A simple electro-physiological with a geometric hierarchy as indicated by the green arrows. Note that the membranes/boundaries that would typically be inserted between some subspaces have not been included. The connection between the diatic and T-Tubule subspace (highlighted in red) is possible, even though it spans multiple branches and levels of the geometry hierarchy. This would not be possible if, for instance, the diatic and SR subspaces were encapsulated within the intra-cellular subspace.

The result of all this: we're not having any kind of explicit mechanism for implementing variable hierarchies/inheritance in CellML.

5 The Old In-Out

Although some have expressed some disquiet as to the choice of terminology (anyone seen "A Clockwork Orange" recently?), it appears that the old `in-out` is here to stay. Everybody in the Auckland team was finally forced to admit that some kind of variable pass-through mechanism was necessary and that the `in` to `in` and `out` to `out` mapping solutions that had been proposed on the 12th, and the variable renaming and equating solutions were in fact ridiculous. The main example that was used to illustrate the need for `in-out` was the requirement that encapsulated components need to be able to export and receive variables to and from components in the network outside the encapsulator without actually connecting to those components. Some concern was expressed that, when refining a model by adding extra encapsulated components, some variables in the external interface of the encapsulator may have to change their roles from `in` and `out` to `in-out`, but those concerned were quickly shown the error of their ways.

A quick formal specification: a variable with a role of `in-out` allows a variable to be imported from another component into the current component, used if desired (but not modified), and exported to other components. The value of the variable is obtained from a single variable with role `out` (or `in-out`) in another component via a mapping, and may be mapped to as many other variables with role `in` (or `in-out`) as the modeller wishes. Because a variable with role `in-out` can be mapped to another variable with role `in-out`, some validation is required to ensure that a variable with role `out` starts the chain. As each link in the chain is directional, this shouldn't be too hard to find, and software may choose to reference that variable directly at each point in the chain.