Meeting Minutes 10 June 2003

Autumn A Cuellar

Table of Contents

Introduction

Wow! We're already halfway through the year, and this is the first progress report of the year. Someone's been slacking. Alright. First the update of what's been going on in the past six months. Team CellML's main focus has been the development of the CellML ontologies. We've pretty much let Matt and Poul run wild with their ideas, and only time will tell if that was a good thing *wink, wink*. Matt has been keeping those meeting minutes on his portal site, and shortly he or I will put together a summary of those minutes. If you haven't had a look at the model repository search [http://www.bioeng.auckland.ac.nz/physiome/php/repository_search.php], check it out to find a model that you're looking for quickly and easily. And just for your information, the *cellml.org* site is moving from a server in Princeton, New Jersey to a local server here in New Zealand to make it easier for site maintenance, but I cannot tell you when that move will be completed. Hopefully you won't notice a difference.

However, the reason I was motivated to write these meeting minutes is that we've begun to reapproach the import and reuse features introduced in the CellML 1.1 (6 November 2002) Draft Specification [../../public/specification/20021106/index.html]. A recent meeting with the SBML folks set Poul to pondering, and after having a six month break from the 1.1 stuff and gaining fresh perspective in the form of Matt, we were all able to critically examine the import method described in the draft spec. Two problems came to mind: we never really discussed how the model tree is handled by applications (described in the section “What happens to the model tree...”), and the entire model is imported, connections and everything (see the section “Importing the Entire Model”).

What happens to the model tree...

From Poul's e-mail:

> I am still not happy with the way importing models is handled. It seems to me that the semantics of nested models is too tightly coupled to the way models are imported. In particular, if I have a model tree that I wish to edit and consolidate as a single model (or rearrange to form a different model tree) the semantics of the allowed coupling between variables/components/units will change (because they are dependent upon the particular nesting of models).
>
> One way around this undesirable situation is to change the specification of **<import_model>** to restrict any connections to child models to be made via the private interface of components in the parent model. With this approach all components in child models will behave as encapsulated components in the parent model, es-

> sentially using the component-level mechanism of encapsulation for information hiding. One obvious advantage of this approach is that the encapsulation induced by importing models is semantically identical to information hiding via encapsulation, and thus independent of the model hierarchy. This approach also relegates model importation to the role of a syntactic convenience, and retains encapsulation as the sole mechanism for information hiding.

Matt's argument (in my very simplified versioning) was that it depends on what you are importing for, which eventually lead to the discussions presented in the section “Importing the Entire Model”. Other than that, no one vehemently objected to Poul's recommendation, and, since I don't think his concerns are cleared up with the following solution, we might adopt his proposal. Of course, you all will correct me if I'm wrong.

Importing the Entire Model

I never was happy with the idea of importing an entire model, connections and all. In order for the method to be successful at all, modellers have to be too forward thinking. As we discovered last July [20020726_meeting_minutes.html], you can't just import any model without worrying about the consequence of the connections. So to avoid those problems with the connections, a modeller must anticipate that someone else may want to reuse any of his components and split a possible model into many separate component-models. As seen by my demonstration using the Zeng *et al* modification of the Luo-Rudy II model [../../examples/examples/CellML_1.1/reuse_models_doc/zeng_reuse_doc.html], the current method of importing can be unwieldy, a situation we are trying to avoid.

We went back to the marker board. Poul's immediate suggestion was to add two new elements using similar semantics as the **<import_model>** element. The first, an **<import_component>** element, would add the imported model's component to the importing model but no connections or groups. Beyond the **xlink:title** and **xlink:href** attributes, you would need an extra attribute such as **component** to indicate which component you're importing. The second would be an **<import_units>** element and correspondingly would import the units from a model but no components, connections, and groups, and would feature an extra attribute such as **units** for clarification. Poul added that this could all be done under a single **<import>** element with the meaning of the import implied by the attributes present.

The extra benefit of Poul's suggested method of import is that we could possibly get rid of the horrid **model_1** and **model_2** attributes on the **<map_components>** element of a connection because theoretically you could directly connect to the imported component (although I'd imagine you'd still want to be able to connect to components in a fully imported model). The drawback is that the specifically imported components or units might be dependent on other units which means the modeller might have to have a lot of imports or we apply some messy rules about inference.

Matt quietly interjected at this point - what we really want to get rid of is the connections. How about introducing a **with_connections** attribute to the **<import_model>** element that has a boolean value? Yeah, seems simple and clear now, but hindsight blah, blah, blah. So giving a **with_connections** attribute a value of **"no"** on an **<import_model>** element would indicate that you are importing an entire model without the connections (Captain Obvious).

We were all quite happy with Matt's solution, but there were two dilemmas we were facing towards the end of the meeting. The first went back to Poul's distaste of the **model_1** and **model_2** attributes, and importing a model without connections does not eliminate this problem. If you'll remember, we resigned ourselves to this method after discarding a couple of other dysfunctional ideas last July (see Section 5) [20020726_meeting_minutes.html]. Again Matt offered a suggestion (shown in Figure 1). Poul sat mulling over Matt's example. Then he turns and asks David B, “Why didn't we do it like that in the first place?” David quips, “I don't know. Warren advocated that method, but you objected for some reason.” “Oh,” says Poul. Heehee. I haven't researched if David's right, but it's funny knowing that if you argue with Poul on an issue, he'll probably come back in a few months time and argue your position for you.

```
<connection>
 <map_components>
  <component_1 component="calcium_channel" model="imported_model" />
  <component_2 component="environment" />
 </map_components>
 <map_variables variable_1="time" variable_2="time" />
</connection>
```

Figure 1. Matt was only talking about breaking down the **<map_components>** element, but we'd probably want to be consistent and break down the **<map_variables>** element, as well.

I disagreed with Matt on his solution because if we changed the spec in such a way, it means that every CellML 1.0 model will no longer be valid CellML 1.1. David B and Poul argued that we could quickly transform all the old models to be correct to this method, but the thing is that the **model_1** and **model_2** attributes, ugly as they are, aren't exactly wrong. I once thought that backwards compatibility was important to more people than just me.

The second dilemma concerns whether we should also break up the encapsulation hierarchy when we import without connections. Andre says yes, we should because what's the point of breaking all the connections if you don't get direct access to all the components? I say why is it such a necessity to get direct access to all the components when you have to re-do all the connections anyways? Surely a modeller adds the encapsulation to structure his model in meaningful ways; wouldn't you want that structure maintained when it's imported? Fodder for another week (or month).

Miscellaneous

- Poul suggested we revisit the "**<model>** purely as a container" discussion. We talked about it a bit but still remained divisive (though I'm not sure how because I'm certain we all agreed on it the last time it was brought up).
- Matt wants the **<component_ref>** element changed because component_ref means something different in whatever logic he uses than the way we use it. I will continue to object to the change because he hasn't yet convinced me that his is not a semantical argument - more philosophical than practical. He will probably argue that I'm not the one he needs to convince (true), but I don't think Team CellML should take the issue of changing element names lightly. Michael diplomatically suggested that we deprecate the use of **<component_ref>** in favour of whatever Matt wants, which is a better idea, if it needs to be changed at all (but I still don't think it does).

I'll leave you with my new favourite quote, "In meetings, [s]he who writes the minutes determines the outcome." Ha. Said Rinaldo [http://www.netfunny.com/rhf/jokes/89q1/rinald.298.html].