Meeting Minutes 24 June 2003

Autumn A Cuellar

Table of Contents

Introduction

These minutes actually cover the 17 June, as well, and introduce yet another import method. We say now we're sticking to this one.

Correction from 10 June Meeting Minutes

Warren pointed out that the element names used in Figure 1 in the last meeting minutes [20030610_meeting_minutes.html] were a bit strange. Why break up the **<map_components>** element to use weird **<component_1>** and **<component_2>** elements, when what we were really going for was just a single **<component>** element. But then if you break up the **<map_variables>** element to two **<variable>** elements, as well, you have to imply meaning through the order of the elements which is kind of a messy way to do things. So we spent quite a bit of time working out alternatives to this scheme, using indices or another attribute. But all that conversation became moot when David B. came up with an alternative import scheme...

Import Method #672

David was still not happy with the way we were approaching this import feature for several reasons (the main one being that he still thought we should be able to get the part of a model that we wanted with necessary connections without getting the entire model tree). Then he was struck for a moment by genius. Well, maybe not, but we did all like his idea, which is best described through illustration (see Figure 1). The scenario is as follows: Model 2 wants to reuse components 1 & 2 from Model 1. Through David's new import scheme (shown in CellML format in Figure 2), by importing component 1, Model 2 gets an instance of the entire encapsulation tree of which component 1 is the parent. Therefore, in addition to getting component 1, Model 2 gets an instance of components 2 & 3, and their respective children 5 & 6. This means that if component 1 is dependent on values from components 2 & 3, you don't need to re-establish the connections between parent and child. However, since components 4, 7 & 8 are part of a different encapsulation hierarchy, a modeller would need to import them separately if he wanted access to them. Then to use component 2, Model 2 would need to import component 2 directly creating another instance, and again getting its entire tree which, in this case, includes component 5.

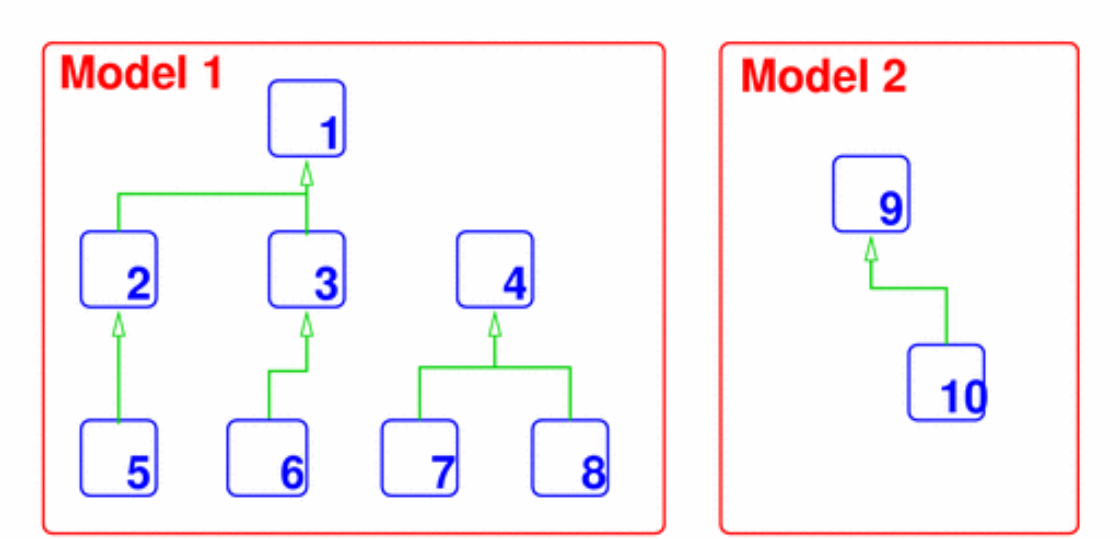

Figure 1. The encapsulation hierarchies of two models. Blue boxes represent components, green arrows represent an 'encapsulated by' relationship, thus component 5 is encapsulated by component 2, etc.

```
<import xlink:href="http://www.example.com/some_CellML_file.xml" xmlns:xlink="http://www.w3.org/1999/xlink">
  <component_ref component="component1" name="blahblah" />
  <component_ref component="component2" name="SchnizzleWizzle" />
  <units_ref name="cheese_weight_measure" units="pounds_of_fat" />
</import>
```

Figure 2. All components and units imported from a model are declared at the same time at the beginning of a file (best practice). The entire component tree is imported.

- Poul is still unhappy with the **`<import_model>`** element name. I still don't think it's really a big deal. We can shorten it to **`<import>`** to make him happy. Can we let it go now? If he changes his mind later, it will be hard for me not to sit sputtering in shock and disbelief.
- We can get rid of the **`xlink:title`** attribute because there should be no need to refer to the imported model within the importing model, and we're sticking to the one-CellML-model-per-URI rule.
- The units of Model 1 cannot be used in Model 2 unless stated explicitly with a **`<units_ref>`** element, as shown in Figure 2. However, imported components do have access to the units defined in Model 1, so as to give meaning to the units assigned to variables in the imported components. I'd imagine this will be true of units building on other defined units, as well.
- Of course, the **`<component_ref>`** element cannot be used in the import element unless we do change the **`<component_ref>`** element name in the grouping. Damn, looks like I might lose this argument, too. Hush, Andre.

Matt's Software Update

Matt has coded the CellML API in Python. He has two different sets of libraries - one that does the same as Andre's C++ version except also includes the reaction subset of CellML, the other implementing changes which he has suggested to us and hopes to be version 2.0 of CellML maybe. One of these recommended changes is to have the public and private interface information located in the connections instead of variables. You can correct me if I'm wrong, Matt, because I didn't take notes on this part and am going from memory instead.

Before making the Python libraries available Matt wants to clean them up and check them against the C++ implementation of the API. Matt also hopes to start using the libraries to develop a validator and graphic visualisation tool at the end of July.