Meeting Minutes 8 May 2002
Component Re-use: Initial Values

Author:
Autumn A Cuellar (Bioengineering Institute, University of Auckland)
Contributors:
David Bullivant (Bioengineering Institute, University of Auckland)
Warren Hedley (Alliance For Cellular Signaling, San Diego Supercomputer Center)
Catherine Lloyd (Bioengineering Institute, University of Auckland)
David Nickerson (Bioengineering Institute, University of Auckland)
Poul Nielsen (Bioengineering Institute, University of Auckland)

1 Introduction

While we believe CellML is complete in its ability to describe biological models as it is, it's becoming more and more obvious that the next step of CellML should be to incorporate component re-use for convenience. A long long time ago, the CellML team made an attempt to solve the component re-use problem as outlined in the Meeting Minutes for 31 October 2000[1]. The method they came up with was one that included copying parts of models into the current model and modifying those parts using XPath[2] to reference exactly what needs to be modified. The idea was a variation of the method introduced in the Low-level XML Re-use[3] document. We've recently discussed those methods and discarded them for reasons indicated in Section 3.

These minutes set forth the recent discussions concerning a simple case study: import of a file containing the initial values of a model. Examples are based on Catherine Lloyd's CellML description of Zeng *et al*'s modification of the Luo-Rudy II Model[4].

2 Same File Import of Initial Values

Simulation software demands initial values of every variable when simulating a model, and, for this reason, SBML (intended as an exchange language between software packages) requires that initial values be declared for every variable. CellML has avoided making this requirement by making a somewhat-hazy distinction between what is necessary to the description of a model and what is simulation information (a model can be sufficient without the simulation information). The idea has been that simulation software could also import a parameters file when importing CellML code. Now, how do we import those values; how do we connect the parameters to the rest of the model?

2.1 Using the Initial Value Attribute

Figure 1 shows the declaration of an initial value variable in one component and the reference to the initial value variable in another component using the **`initial_value`** attribute. `"V_init"`, the initial value variable, is declared as a constant would be. It is then mapped to the membrane component where it is referenced as the initial value of `"V"`, the variable that will be described mathematically as one that changes with respect to time. The obvious problem with this method is that it violates the CellML 1.0 Specification by giving the **`initial_value`** attribute a value of string and *not* a real number.

[1]http://www.cellml.org/private/progress_reports/20001031_meeting_minutes.html
[2]http://www.w3.org/TR/xpath
[3]http://www.cellml.org/private/documentation/component_reuse.html
[4]http://www.cellml.org/examples/repository/updated_LR_II_model_doc.html

```
<component name="initial_values">
    <variable
        name="V_init" public_interface="out"
        initial_value="-84.624" units="millivolt" />
  </component>

  <component name="membrane">
    <variable
        name="V" public_interface="out"
        initial_value="V_i" units="millivolt" />
    <variable name="V_i" public_interface="in" units="millivolt" />
  </component>

  <connection>
    <map_components
        component_1="membrane" component_2="init_values" />
    <map_variables variable_1="V_i" variable_2="V_init" />
  </connection>
```

FIGURE 1: The initial value is declared in a separate component from the one containing the mathematics describing "V".

2.2 Using Mathematics

One possible way of avoiding misuse of the **initial_value** attribute is by describing the initial value relationship mathematically, as shown in Figure 2. It'd be necessary to go outside the CellML subset of MathML for reasons described in the July 16, 2001, Meeting Minutes[5]. Two separate "for all" statements are used to describe the change of "V" with respect to time: the first sets the initial value, the second gives the differential equation explaining how "V" changes over time. Unfortunately, these equations will dramatically extend the length of a given model. Also, as Warren points out in the July 16 Meeting Minutes, there's not a clear way of associating the two "for all" statements.

The CellML Team has decided to use the **initial_value** attribute. The next version of CellML will state that the value of an **initial_value** attribute can be a real number OR match the value of a **name** attribute on a **<variable>** element declared in the current **<component>**.

3 Importing Initial Values from a Separate File

Importing the initial values from another file will be a little trickier and seems a convenient test case for component re-use. One of the main reasons the low-level component re-use[6] idea was tossed was that it relies too much on the component that is being re-used. The question arose: what if the original model (component, equation, etc.) changes? In re-using models and parts of models, the new model should have a certain amount of protection so that if the re-used portion changes, the changes do not adversely affect the new model.

It was recently proposed to use interfaces between models much like the interfaces between components through which variables are passed, but through which a model can pass not only variables, but components and parts of components, as well. Figure 3 shows one suggestion for how to handle the inclusion of initial

[5]http://www.cellml.org/private/progress_reports/20010716_meeting_minutes.html
[6]http://www.cellml.org/private/documentation/component_reuse.html

```
<component name="initial_values">
    <variable
        name="V_init" public_interface="out"
        units="millivolt" />
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply><eq />
        <ci> V_init </ci>
        <cn cellml:units="millivolt"> -84.624 </cn>
      </apply>
    </math>
  </component>

  <component name="membrane">
    <variable
        name="V" public_interface="out" units="millivolt" />
    <variable name="V_i" public_interface="in" units="millivolt" />
    <!--declaration of the other variables goes here-->
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply id="membrane_voltage_initial_condition"><forall />
        <bvar><ci> time </ci></bvar>
        <condition>
          <apply><eq />
            <ci> time </ci>
            <cn cellml:units="millisecond"> 0.0 </cn>
          </apply>
        </condition>
        <apply><eq />
          <ci> V </ci>
          <ci> V_i </ci>
        </apply>
      </apply>

      <apply id="membrane_voltage_diff_eq"><forall />
        <bvar><ci> time </ci></bvar>
        <condition>
          <apply><gt />
            <ci> time </ci>
            <cn cellml:units="millisecond"> 0.0 </cn>
          </apply>
        </condition>
        <apply><eq />
          <apply><diff />
            <bvar><ci> time </ci></bvar>
            <ci> V </ci>
          </apply>
          <apply><times />
            <apply><minus />
              <apply><divide />
                <cn cellml:units="dimensionless"> 1.0 </cn>
                <ci> Cm </ci>
              </apply>
            </apply>
            <apply><plus />
              <ci> i_Na </ci>
              <ci> i_Ca_L </ci>
              <ci> i_Ca_T </ci>
              <ci> i_Kr </ci>
              <ci> i_Ks </ci>
              <ci> i_K1 </ci>
              <ci> i_Kp </ci>
              <ci> i_NaCa </ci>
              <ci> i_p_Ca </ci>
              <ci> i_Na_b </ci>
              <ci> i_Ca_b </ci>
              <ci> i_NaK </ci>
              <ci> i_ns_Ca </ci>
              <ci> I_st </ci>
            </apply>
          </apply>
        </apply>
      </apply>
    </math>
  </component>

  <connection>
    <map_components
        component_1="membrane" component_2="init_values" />
    <map_variables variable_1="V_i" variable_2="V_init" />
  </connection>
```

FIGURE 2: Here there are two "for all" statements giving the equations for `"V"` with respect to time.

values from another file. The `<model_ref>` element gives the URI of the file containing the initial values that are being imported. The `name` attribute on this element gives a “nickname” for the file being referenced.

```
<model
    name="zeng_model_1995"
    xmlns="http://www.cellml.org/cellml/1.0#">
  <component name="membrane">
    <variable
        name="V_i" public_interface="in" units="millivolt" />
    <!--and so on, and so forth...-->
  </component>

  <model_ref name="initial_values_file" uri="initial_values_file.xml" />
  <connection>
    <map_models model_1="initial_values_file" model_2="zeng_model_1995" />
    <map_components component_1="initial_values" component_2="membrane" />
    <map_variables variable_1="V_init" variable_2="V_i" />
    <map_variables variable_1="constant_T" variable_2="T" />
    <map_variables variable_1="constant_F" variable_2="F" />
    <!--and so on, and so forth...-->
  </connection>
</model>
```

FIGURE 3: Import of initial values from a separate file. See text for details.

We’re still discussing whether it is best to have the file in need of the initial values import the file containing the initial values file or vice versa, or have a third file import them both simultaneously. There’s a concern that in the way described above, only one file can be imported. In other words perhaps it’s the file name that should change if you want to supply a different set of parameters, instead of the actual file (moving and re-naming files so that you always import an `"initial_values_file.xml"`).

It may be more useful to have a second `<model>` element encapsulating the model with the name `"zeng_model_1995"`, as shown in Figure 4. This method may be much more flexible and robust, as both “models” are completely separate entities, and this would easily allow for connections between multiple models.

In this case, would it be better to have an abstract interface component in each model which encapsulates everything else so that everything must be passed through this interface component? Figure 5 shows a `<component>` named `"interface"` that receives all the initial value variables from another file with a `public_interface` attribute with a value of `"in"`. It then passes the variables to its children (all other components in the model) through a private interface. The point of doing this is to make it obvious to the user what this model is lacking. With the interface component, machine or human need only look at the encapsulating component to see what variables the model requires from elsewhere. Of course, the drawback is that with all the extra connections that need to be made, a large model will be made even larger.

```
<model name="zeng_plus_initial_values">

  <model
    name="zeng_model_1995"
    xmlns="http://www.cellml.org/cellml/1.0#">
    <component name="membrane">
      <variable
        name="V_i" public_interface="in" units="millivolt" />
      <!--and so on, and so forth...-->
    </component>
    ...

  </model>

  <model_ref name="initial_values_file" uri="initial_values_file.xml" />

  <connection>
    <map_models model_1="initial_values_file" model_2="zeng_model_1995" />
    <map_components component_1="initial_values" component_2="membrane" />
    <map_variables variable_1="V_init" variable_2="V_i" />
    <map_variables variable_1="constant_T" variable_2="T" />
    <map_variables variable_1="constant_F" variable_2="F" />
    <!--and so on, and so forth...-->
  </connection>

</model>
```

FIGURE 4: The `"zeng_plus_initial_values"` model encapsulates the `"zeng_model_1995"`. All connections between the `"zeng_model_1995"` and the referenced model `"initial_values_file"` are made in the encapsulating model.

```
<model
    name="zeng_model_1995"
    xmlns="http://www.cellml.org/cellml/1.0#">

  ...

  <component name="interface">
    <variable
        name="time_i" public_interface="in"
        private_interface="out" units="millisecond" />
    <variable
        name="R" public_interface="in"
        private_interface="out" units="joule_per_kilomole_kelvin" />
    <variable
        name="T" public_interface="in"
        private_interface="out" units="kelvin" />
    <variable
        name="F" public_interface="in"
        private_interface="out" units="coulomb_per_mole" />
    <!--etc.-->
  </component>

  ...

  <group>
    <relationship_ref relationship="encapsulation" />
    <component_ref component="interface">
      <component_ref component="fast_sodium_current">
        <component_ref component="fast_sodium_current_m_gate" />
        <component_ref component="fast_sodium_current_h_gate" />
        <component_ref component="fast_sodium_current_j_gate" />
      </component_ref>
      <component_ref component="L_type_Ca_channel">
        <component_ref component="L_type_Ca_channel_d_gate" />
        <component_ref component="L_type_Ca_channel_f_gate" />
        <component_ref component="L_type_Ca_channel_f_Ca_gate" />
      </component_ref>
    </component_ref>
  </group>
</model>
```

FIGURE 5: The `"interface"` component encapsulates all other components in the model. All variables from an outside file are passed through this component to its children.