Model Decomposition — decompose

Table of contents

decompose is a little utility designed to turn a <u>CellML 1.0</u> model into a <u>CellML 1.1</u> hierarchy of model components. This process also extracts model boundary and initial conditions embedded in the original model into new top level components in the 1.1 model hierarchy. While designed to work with single CellML 1.0 documents, decompose is not restricted to such and is capable of extracting the boundary and initial conditions from existing CellML 1.1 model hierarchies. decompose is built on top of the <u>CellML API</u> and some preliminary build instructions are <u>available</u>.

1 Model decomposition with decompose

In the world of CellML 1.0, models are developed in a single XML document which contains the entire encoded model, typically with boundary and initial conditions defined in-place as required. While this makes it easy to share the model, it makes it harder to reuse the model or parts of the model in different simulation scenarios. For example, you may have a model which you want to run with many different parameter sets. This is achieved in CellML 1.0 by simply copying the model for each parameter set and directly altering the appropriate variable values within each copy of the model.

A huge advantage of CellML 1.1 is that it allows for complete separation of the mathematical model description from the boundary and initial conditions. It also allows components and units to be imported from one model into another. With model decomposition we want to take a single CellML 1.0 model and turn it into a CellML 1.1 model hierarchy where each component is separated into its own model document and all the boundary and initial conditions are extracted into a new top level model and then connected back into the model components as required.

The way decompose does this is probably easiest explained with a (very simple) example...

1.1 Example

Here we look at a very minimal and simple example to illustrate the model decomposition process. All excess code been left out to hopefully aid clarity... Given the original source model below:

```xml
<model name="2010_electrical">
  <units name="mm">
    ...
  </units>
  ...
  <component name="INa">
    <variable name="INa" units="uApmmsq"/>
    <variable name="gNa_max" initial_value="0.3" units="mSpmmsq"/>
    ...
    <math>
      [INa = gNa_max * ...]
    </math>
  </component>
  <component name="membrane">
    <variable name="Vm" initial_value="-90" units="mV"/>
```

```
    ...
    <math>
      [d(Vm)/d(time) = ...]
    </math>
  </component>
</model>
```

`decompose` will take this model and decompose it into several new CellML 1.1 models. It will also extract out the `gNa_max` parameter and the initial condition for `Vm`. The initial value for the differential equation is extracted by creating a new variable, `Vm_initial`, and using that to set the initial value of the `Vm` variable. As shown in the new `membrane_model` below:

```
<model name="membrane_model">
  <import href="2010_electrical_units_model.xml">
    <units name="mm" units_ref="mm"/>
    ...
  </import>
  <component name="membrane">
    <variable name="Vm" initial_value="Vm_initial" units="mV"/>
    <variable name="Vm_initial" public_interface="in" units="mV"/>
    ...
    <math>
      [d(Vm)/d(time) = ...]
    </math>
  </component>
</model>
```

The `gNa_max` parameter, on the other hand, simply gets "passed" into the component from its new definition elsewhere...

```
<model name="INa_model">
  <import href="2010_electrical_units_model.xml">
    <units name="mm" units_ref="mm"/>
    ...
  </import>
  <component name="INa">
    <variable name="INa" units="uApmmsq"/>
    <variable name="gNa_max" public_interface="in" units="mSpmmsq"/>
    ...
    <math>
      [INa = gNa_max * ...]
    </math>
  </component>
</model>
```

In addition to the individual component models, `decompose` also creates models for the boundary/initial conditions, units, a single interface, and an example experiment model. The units model simply defines all model-scope units from the original model in order to provide a single definition of the units for reuse in all the other models created.

```
<model name="2010_electrical_units_model">
  <units name="mm">
```

```
      ...
    </units>
    ...
  </model>
```

The boundary and initial conditions model defines all the parameter values and initial conditions from the original model. In use, these can either be connected back to the decomposed model or supplanted by new values in new applications of the model.

```
<model name="2010_electrical_variable_values_model">
  <import href="2010_electrical_units_model.xml">
    <units name="mm" units_ref="mm"/>
    ...
  </import>
  <component name="parameters">
    <variable name="gNa_max" initial_value="0.3" units="mSpmmsq"/>
    ...
  </component>
  <component name="initial_values">
    <variable name="Vm_initial" initial_value="-90" units="mV"/>
    ...
  </component>
</model>
```

In order to provide an easy-to-access handle for re-using the entire model, decompose also defines an interface model. The interface model defines a single component which has all parameter and initial condition variables coming in and all computed variables coming out. The interface model also imports all the models obtained from the decomposition of the original model and encapsulates them all beneath the interface component. In this manner, one simply needs to import the interface component in order to grab the entire original mathematical model. All connections amongst components are also defined, but not shown in the example below.

```
<model name="2010_electrical_interface_model">
  <import href="2010_electrical_units_model.xml">
    <units name="mm" units_ref="mm"/>
    ...
  </import>
  <import href="membrane_model.xml">
    <component name="membrane" units_ref="membrane"/>
  </import>
  <import href="INa_model.xml">
    <component name="INa" units_ref="INa"/>
  </import>
  ...
  <component name="2010_electrical_interface_component">
    <variable name="gNa_max" public_interface="in" units="mSpmmsq"/>
    <variable name="Vm_initial" public_interface="in" units="mV"/>
    <variable name="INa" public_interface="out" units="uSpmmsq"/>
    <variable name="Vm" public_interface="out" units="mV"/>
    ...
  </component>
  <group>
    <relationship_ref relationship="encapsulation"/>
    <component_ref component="2010_electrical_interface_component">
```

```
        <component_ref component="membrane"/>
        <component_ref component="INa"/>
        ...
      </component_ref>
    </group>
  </model>
```

The final model created is an experiment model, which illustrates how the interface model can be combined with the variable values model to define a specific instantiation of the mathematical model with a full parameter and initial condition set. This example experiment model should also completely recreate the original model.

```
<model name="2010_electrical_experiment_model">
  <import href="2010_electrical_interface_model.xml">
    <component name="2010_electrical_interface_component"
 units_ref="2010_electrical_interface_component"/>
  </import>
  <import href="2010_electrical_variable_values_model.xml">
    <component name="parameters" units_ref="parameters"/>
    <component name="initial_values" units_ref="initial_values"/>
  </import>
  ...
  <connection>
    <map_components component_1="2010_electrical_interface_component"
component_2="parameters"/>
    <map_variables variable_1="gNa_max" variable_2="gNa_max"/>
    ...
  </connection>
  <connection>
    <map_components component_1="2010_electrical_interface_component"
component_2="initial_values"/>
    <map_variables variable_1="Vm_initial" variable_2="Vm_initial"/>
    ...
  </connection>
</model>
```

2 Limitations

This particular tool is still in its infancy, having been initially developed to meet a particular objective of my work. As such, while the basic task of this utility is met there are a number of limitations resulting from the dodgy way I developed it to meet my requirements in the shortest possible time. It is probably also worth pointing out that I'm not yet convinced that there is an optimal decomposed model for any given source model, so I have gone with primarily separating out the parameter values and initial conditions and making sure all the appropriate connections are made and the example experiment model correctly reproduces the behaviour of the original model. The idea is that then a model author would manually arrange any extra encapsulation that they think best fits the model, as well as removing extraneous variables left over when the original encapsulation hierarchy got blown away. I will probably need to update this list as I remember more bits I left out, but here are the main points to consider.

- **Original encapsulation not maintained**: any encapsulation in the original model is ignored in the decomposed model. The decomposed model is a flat model under the interface component. This has the side effect of resulting in lots of variables defined in components which no longer need them as their previously encapsulated children have been raised to the sibling set.
- **Metadata**: all metadata, `cmeta:id`'s is currently not contained within the decomposed model. Adding `cmeta:id`'s is pretty straightforward, but need to think more about how to handle metadata. Probably easiest to output all metadata into a separate single document and then try to match up id's with the appropriate URL's of the new model documents.
- **Unique component names**: I'm assuming that all component names in the original model are unique. Probably a fairly safe assumption as the model should be a valid CellML 1.0 model, but might get tricky if people use `decompose` to extract parameters and initial values in CellML 1.1 model hierarchies.
- **Unique parameter and state variable names**: I'm assuming that all parameter names and state variables have unique names within the original model. Based on common usage this is also pretty safe, but it is easy to imagine a model for which this assumption doesn't hold true.
- **Component-scope units**: units which are defined within a component are copied to the matching component in the decomposed model. Variables calculated in the component which use such units are not exposed to the interface component. Parameters and initial values don't check this, so if any parameters or initial values use component-scope units then the resultant decomposed model has undefined behaviour. If the local units name matches that of a global units then the model will still be valid, but potentially incorrect. If the name doesn't match, then the decomposed model will be invalid and the user will need to manually touch up the units.
- **Reactions**: the reaction element is totally ignored.