Modular SBML

Proposal for an Extension of SBML towards level 2

Martin Ginkel
Max-Planck-Institute Dynamics of complex technical Systems,
Magdeburg, Germany
ginkel@mpi-magdeburg.mpg.de

10th June 2002

Contents

1 Introduction and Prerequisites **1**

2 Needed basic features **2**

3 Proposal of a Specification **3**

4 Example **8**

1 Introduction and Prerequisites

SBML level 1 [Hucka et al.(2001)Hucka, Finney, Sauro, and Bolouri] allowed to construct a structurally flat model of (bio-)chemical reaction networks with compartments as balanced volumes, species as representation for the nodes of the reaction network and reactions as edges. For a custom extension of the mathematical representation, system parameters (time-constant values) and rules (directly given mathematical relations) are part of the standard. Additional elements like annotations and units allow a documentation and perhaps automatic consistency checks of the models.
For the potential exchange and combination of partial models of different modelers it is desirable to introduce additions to SBML for the development of modular models. Main goals to achieve are therein:

- combination of modeling modules of different modelers without knowing all internals of the respective modules
- compatibility with modeling and simulation packages that already support modularity.
- well defined interfaces for the modules that allow a better understanding of the dependencies and function of the module and perhaps are compatible to the interfaces of modules in CellML.

Modeling packages for Systems Biology that use some kind of modularity so far are PROMOT and ECELL. Modules in PROMOT are logical, encapsulated groupings of modeling elements that may (but don't have to) represent compartments. For biological models, they contain mainly reactions, species and special signaling parts that can be represented as SBML rules. In fact reactions and species are modules in PROMOT too. As far as it is known to the author, ECELL uses an architecture where the complete model may be modularized through compartments. In this sense, modules must have some

physical border and are not only logical or functional groupings but represent an object in the physical topology of the cell. They also have an implementational meaning in the ECELL software architecture since they provide the basis of multiple solving modes in one simulated model (e. g. discrete programs, stochastic simulation and continuous ODE-solution).

2 Needed basic features

What is necessary in SBML to accomplish modular modeling? Currently one SBML document describes all components of a modeled system in one model. For modular modeling obviously a mechanism of aggregation is necessary to build up a hierarchy of aggregated modules. In the tradition of object-oriented modeling a simple approach is that one SBML document may contain one ore more models that are identified by an unique name. This list of models is indeed a list of modules (model classes) that may be used repeatedly by instantiation. Since recursive use of one model in itself should be impossible, forward definitions are not necessary and the list of models should be ordered in the way that dependencies (one model uses another) can be resolved directly. Inside of a model this models get instantiated with an instance name as a modelInstance. That instance name should not been mixed up with the class names of the models. Exactly one model in an sbmlDocument should be marked with a `isMainModel` attribute to state that this model should be the final simulation model and all the others are potential parts of it. Perhaps for the use of libraries a mechanism of including other SBML files at an specified URL can be helpful. But in this document the issue will not been addressed.
When instances of predefined models are created, these need to be parametrized. Otherwise one had to define a different model for each slightly different modelInstance even if only some numbers change. Parameterizations should override the default definitions given in the model definition. The possibility to set the initial amounts of substances and the values of kinetic and other parameters seems necessary. Since the modules may be nested internally, a hierarchical naming scheme is therefore required. To set for instance parameter `p` inside the sub-model `a` of the direct sub-model `b`, the modeler would define something like `<parameterSpec parameter="b.a.p" value="25.3">`. The idea is to provide a name delimiter that must not be used in normal SBML SNames. We propose the "." dot as the current way, as this is represented in PROMOT. ECELL uses slashes "/" and additionally colons for addressing internal attributes of an element. A possible extension of such parameterization scheme could be, to set the type of a submodel in the instantiated module.
Another very important issue is the definition of interfaces that allow the connection of sub-models without really touching the interior. This was a major drawback of the proposal [Finney(2000)] where models and modelinstances where used too, but connection and specification of modelinstances took place without defined interfaces. A favorable solution should be somehow compatible with the interface concepts of CELLML: to be able to connect modules described in SBML with "external"-declared modules written in CELLML. Since SBML level 1 uses a directed reference scheme from reactions to substances, the easiest way of an interface definition is a definition of public substances that may be used outside the module. The disadvantage of this very simple approach is, that it is necessary to build connections between the modules via single reactions on the same level of the aggregation hierarchy, as you cannot refer to e.g. the substrate input of a reaction inside the submodule. Also a definition of a complex process as a compound reaction is not possible with this approach.
A slightly more complicated way that overcomes these limitations, adopts "terminal" definitions similar to those in PROMOT. A terminal is a reference to a substance that is owned by the current module or by a sibling of the module outside. Terminals may be used similar to substances inside reactions and get connected outside a module through links.
Both kinds of interface definitions imply a direction of the data-flow during simulation. Although this is not state of the art in connection oriented modeling and simulation [Cellier and Elmquist(1993)] it

should be used here for the sake of simplicity. The authors believe that this allows also the interfacing with CELLML. CELLML uses variables as the interface and defines a direction in which the variable is given across the module border (in vs. out).

Comment needed

For the connections in the model two forms are applicable: horizontal links between interfaces of modules on one level of hierarchy and connections between (direct-part) substances or reactions of the top level model and the interfaces of the sub-models. In the case of substances-as-interfaces no new language elements are needed but only the syntax of substance-references in the reactants and products at reactions must include the possibility of hierarchical names.
In the variant with terminals one can connect a complex reaction module with another module containing the necessary reactants. Therfore a special link element is needed that connects two or more terminals. This approach needs some checks of model consistency, since it will lead to an invalid model if two substances are connected through a link.

3 Proposal of a Specification

In the following a schema of the introduced data model is depicted that can be a basis for the discussion towards SBML level 2. This is based on Ideas out of [Finney(2000)] and the talk at the SBML workshop in June 2000. UML is used for the presentation of the ideas. Some of the diagrams get a little complicated and tricky since they contain a specification of somehow metacircular objects (Classes and Instances of them in one diagram). UML does not fit very well for this task. In the following, the more elaborate level with terminals is described.
Starting at the top level (Figure 1) of SBML there is a list of models where exactly (perhaps at most for libraries) one is marked by `isMainModel` as the main model that should be simulated. The interior of the models may contain all the elements of SBML level 1 and optionally some new ones. The model may have `modelInstances` that are instances of previously defined models They refer to their definition with the attribute `isA`. There may be terminals which define an interface of the current model. Any terminal may refer to a terminal of a modelinstance to propate this interface to the outside. Terminals are used in two subclasses: `reactionTerminals` and `specieTerminals`. Specieterminals refer by the attribute `specie` to one specie of the current model or by `terminal` to a specieterminal of a possible modelinstance inside the model. Specieterminals export species from the current model so that they can be used outside. Reactionterminals refer either to a reactionterminal of a submodel by terminal or contain no reference at all, in which case they act as a imported specie outside the current model, that is fully specified, when the model gets linked in a larger context. One speciereference of the current model can use this terminal as an equivalent for a normal specie.
A model can contain `Links` that connect one specie or specieterminal in the attribute specie with a number of `reactionTerminals`. The `specieReferences` of `Reactions` may refer to `Species`, `specieTerminals` of submodels and `reactionTerminals` of the current model. If links refer to terminals, then this terminals must be part of a direct submodel (modelinstance) in the current model.
As described above, modelinstances may contain further specifications and parametrizations of their inner parts. This is useful for predefining general models and specializing them in the model where they are applied. All specializations in the current model are named with XXXSpec, where XXX has to be replaced by one of Reaction, Specie, Compartment or Rule. Most of the specifications set or override numeric values of already known parts of the modelinstance. The main relations and references are described in Figure 2.
The simplest form is the `ParameterSpec`. It refers with the attribute `parameter` to a parameter

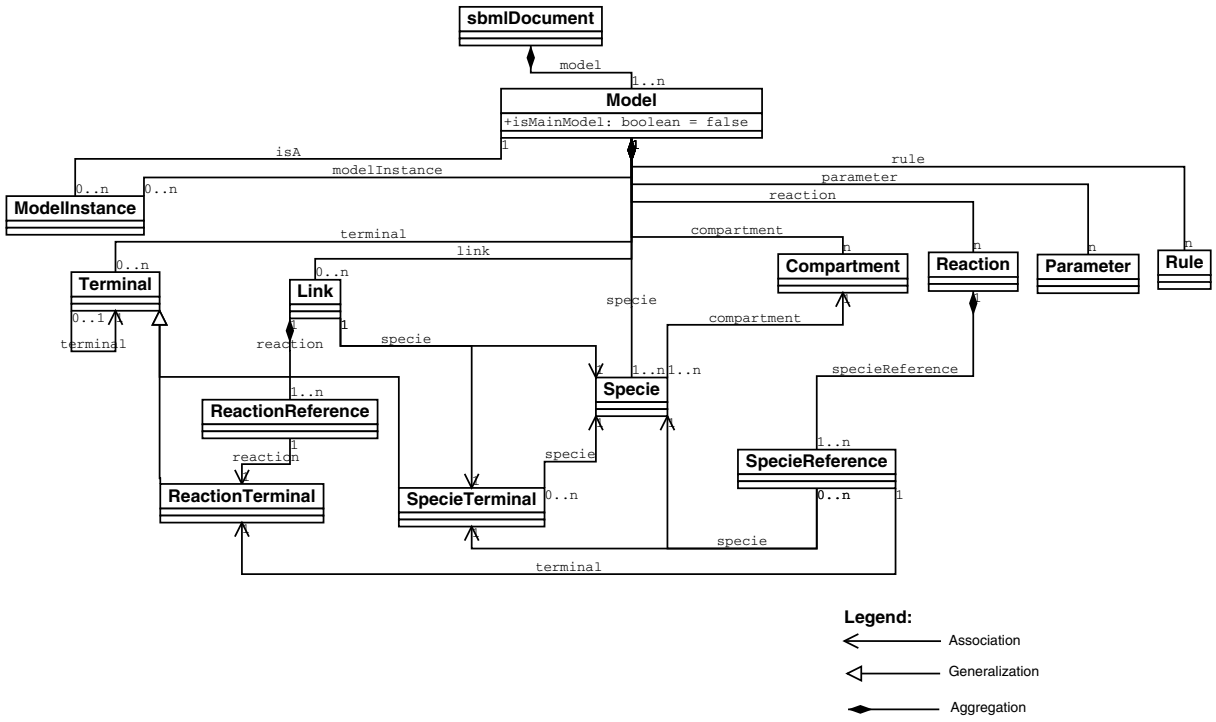

Figure 1: Augmented toplevel of the SBML and containment

that is defined for the modelinstance and with the attribute `value` it sets the value of that parameter. For changing species there is a `SpecieSpec` that refers with the attribute `specie` to an already known specie of the modelinstance and can set all the other attributes of a specie. Especially it can set the `compartment` to a globally accessible compartment of the model. This is important since in smaller modules most likely will comprise only species and reactions and no compartments. Another way to model the association of species in a modelinstance to compartments is with the `CompartmentSpec`. The description of the (sub)model contains a `Compartment` in this case. The species of the model refer to this compartment. When instanciating this model, the inner compartment is substituted by a compartment of the global model with the `CompartmentSpec`. Therfore the `CompartmentSpec` refers to the inner `Compartment` with `compartment` and to the outer `Compartment` with `isSubstitutedBy`. `RuleSpecs` specify the system behaviour of a `ModelInstance` by changing the formula of a internal rule. They refer to a rule of the instantiated model and contain a different string for the formula. `ReactionSpecs` change parameters of reactions in the modelinstance They refer to a reaction (with its `KineticLaw`) and may contain a list of `ParameterSpecs`. They may also change the formula of the kineticlaw, but not the structure of the reaction (substrates, products, enzymes or ligands).
A list of all new or changed elements and types is presented in the following:

- Type HName:= SName | **'parent'** {**'.'** SName}. The HName is the **H**ierarchical **Name** for accessing Elements of Submodels in arbitrary depth of the aggregation hierarchy. For the SNames of species, reactions, parameters, etc. applies the rule, that they are visible in the model they reside in. The **parent** is a new reserved word, referring to the model containing a modelInstance. This kind of backreference is especially necessary for substituting compartments or specifying the compartment attribute of species. In all formulae also HNames must be allowed as Names for species.

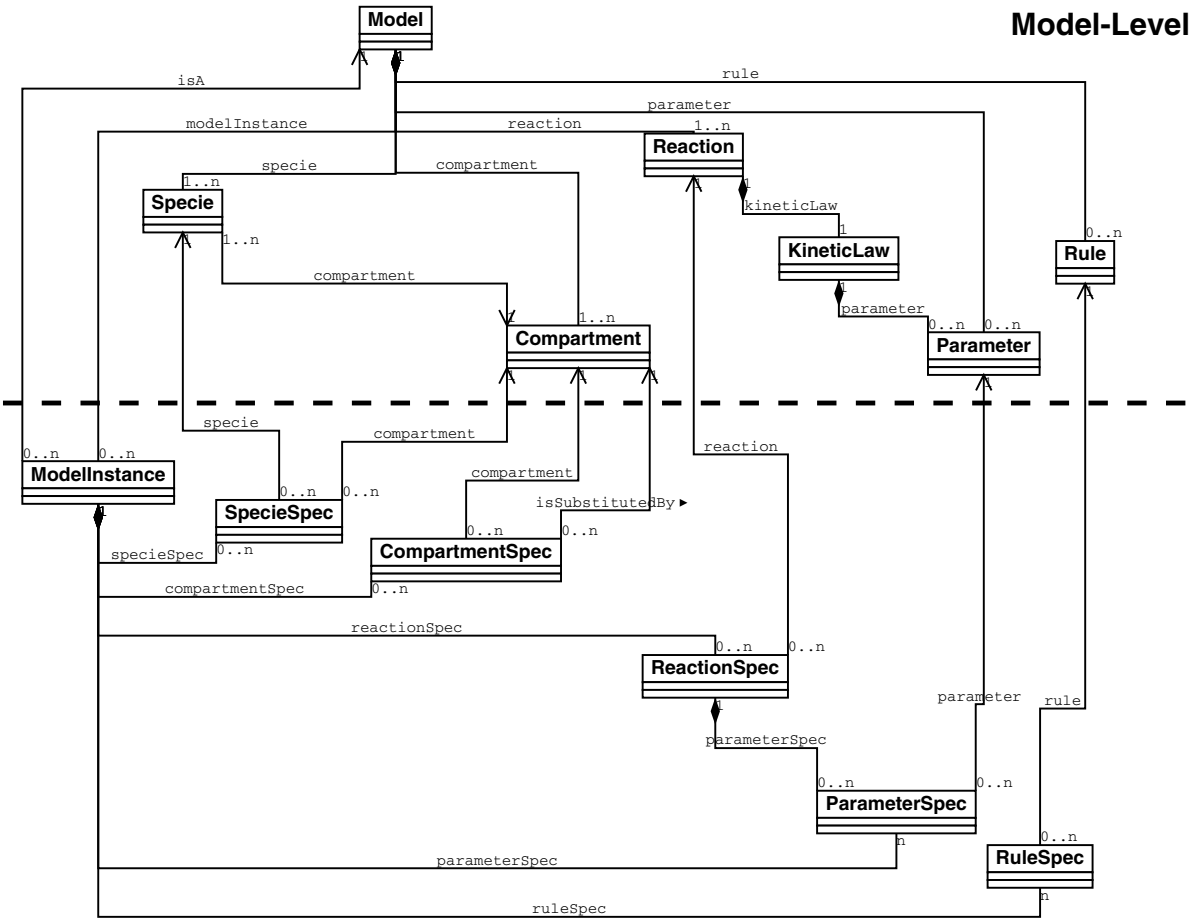

Figure 2: ModelInstances and Parametrizations

SbmlDocument
−model: Model +model: Model[1..*]

  There is now a list of models, model names are visible within the sbmlDocument, if a model is intended to be used in other models, it must be named. Models must be defined in an order, that no forward references are necessary.

Model
name: SName {use=optional} +isMainModel: boolean {use=default value="false"} unitDefinition: UnitDefinition[0..*] compartment: Compartment[0..*] specie: Specie[0..*] parameter: Parameter[0..*] rule: Rule[0..*] reaction: Reaction[0..*] +modelInstance: ModelInstance[0..*] +link: Link[0..*] +terminal: Terminal[0..*]

  An additional difference to SBML 1 is, that a model is not required to have at least one specie, reaction and compartment, since it can be connected with others, that contain e.g. the necessary specie. However, the (compound) main model still must contain at least one of each.

  Perhaps also an attribute inCompartment can be included for pathway models that contain no compartments but instead get instantiated inside a compartment (all species inside get calculated with respect to that compartment).

ReactionReference
reaction: HName

  This class is only necessary to form lists of reaction names inside Links.

Link
name: SName {use=optional} specie: HName reaction: ReactionReference[1.*]

  The specie attribute of a link can be the SName of a specie in the current model or the HName of a SpecieTerminal. Species inside Submodels should not be accessed directly.

ModelInstance
name: SName {use=optional} isA: SName ruleSpec: RuleSpec[0..*] specieSpec: SpecieSpec[0..*] reactionSpec: ReactionSpec[0..*] parameterSpec: ParameterSpec[0..*] compartmentSpec: CompartmentSpec[0..*]

  A modelinstance may be named and must refer to a defined model, that gets instantiated. It may contain specifications for the contained elements.

Terminal
name: SName terminal: HName {use=optional}

  The terminal is a superclass for reactionterminals and specieterminals.

ReactionTerminal

  Reactionterminals act as a placeholder for a substance inside the model and as an interface to a reaction of a modelinstance.

SpecieTerminal
specie: SName {use=optional}

  Specieterminals allow accessing a specie inside a modelinstance and have no special function inside the model that defines them.

SpecieReference
specie: SName {use=optional} +terminal: HName {use=optional} stoichiometry: integer {use=default, value="1"} denominator: integer {use=default, value="1"}

  The speciereference can either refer to a specie or a reactionterminal of the current model with the attribute terminal. The same terminal may only occur in one speciereference for consistency.

ParameterSpec
parameter: HName value: double

  This type allows the adjustment of parameter-values of a modelinstance or reaction.

RuleSpec
rule: HName formula: string

  With the rulespec, the formula of the rule can be changed. Since it is not intended to change the structure of a modelinstance, the rulespec does not allow to change the references in SpecieConcentrationRules, CompartmentVolumeRules or ParameterRules.

ReactionSpec
reaction: HName formula: string parameter: ParameterSpec[0..*]

  For reactions it is possible to change the formula of the KineticLaw or to add Parameterspecs for defined local parameters of that reaction.

CompartmentSpec
compartment: HName isSubstitutedBy: HName volume: double outside: HName

  With the compartmentspec, either a local compartment in a submodel can be substituted by a global comparment or the attributes volume and outside of the local compartment can be changed.

SpecieSpec
specie: HName compartment: HName initialAmount: double

  Species in a submodel can be specialized with a compartment that is valid in the constructed model and with a different initial amount.

4 Example

The following simple example facilitates a modular structure and models a linear pathway with simple Michaelis Menten reactions. It contains three model definitions 'twostep', 'path' and 'whole' that are aggregated into each other. A diagram with module borders is shown in Figure 3.

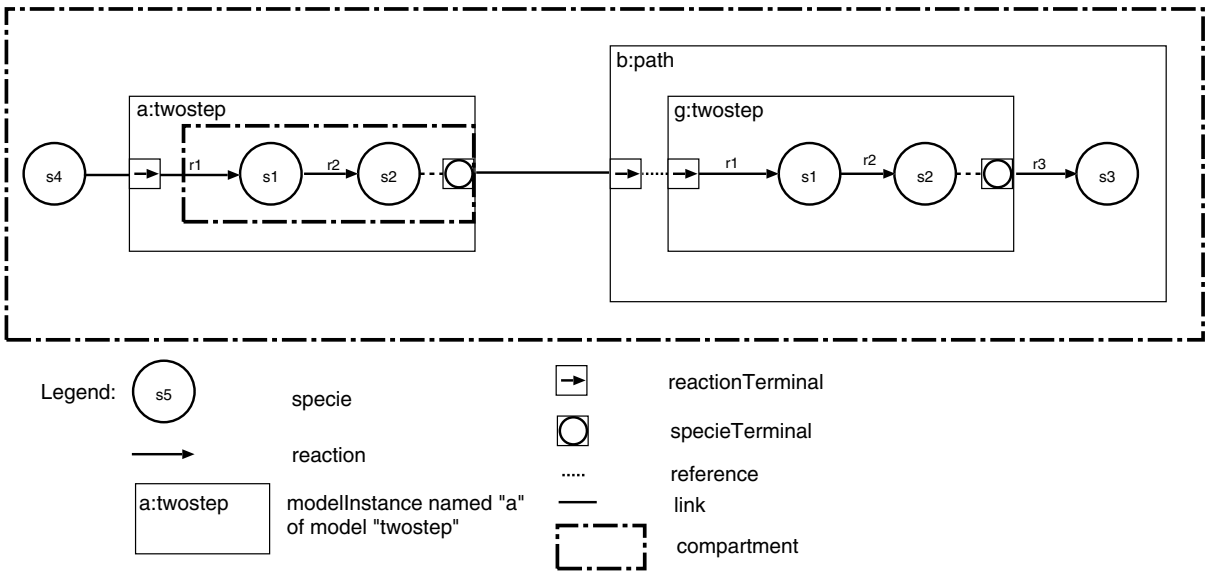

Figure 3: Schema of the modular example

```
<sbml version="1" level="2">
  <listOfModels>
    <model name="twostep">
    <notes>
```

```
    <body xmlns="http://www.w3.org/1999/xhtml">
      Simple 2 Reaction model with 2 Michaelis-Menten
      Reactions.  Input is the reactionTerminal r1_t, the
      specie s2 is public through the specieTerminal
      s2_t. Everything takes place in the compartment c1.
    </body>
  </notes>
    <listOfCompartments>
      <compartment name="c1" volume="10">
    </listOfCompartments>
    <listOfSpecies>
      <specie name="s1" compartment="c1" initialamount="1.0"/>
      <specie name="s2" compartment="c1" initialamount="1.1"/>
    </listOfSpecies>
    <listOfTerminals>
      <reactionTerminal name="r1_t"/>
      <specieTerminal name="s2_t" specie="s2"/>
    </listOfTerminals>
    <listOfReactions>
      <reaction name="r1">
        <listOfReactants >
          <specieReference terminal="r1_t"/>
        </listOfReactants>
        <listOfProducts>
          <specieReference specie="s1" stoichiometry="2"/>
        </listOfProducts>
        <kineticLaw formula="uui(r1_t,vm,km)">
          <listOfParameters>
            <parameter name="km" value="1000"/>
            <parameter name="vm" value="10"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
      <reaction name="r2">
        <listOfReactants >
          <specieReference specie="s1" stoichiometry="2"/>
        </listOfReactants>
        <listOfProducts>
          <specieReference specie="s2"/>
        </listOfProducts>
        <kineticLaw formula="uui(r1_t,vm,km)">
          <listOfParameters>
            <parameter name="km" value="1000"/>
            <parameter name="vm" value="10"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
    </listOfReactions>
```

```
</model>

<model name="path">
  <notes>
    <body xmlns="http://www.w3.org/1999/xhtml">
      The model path extends the twostep model by one
      reaction. It uses the s2_t terminal in a
      specieReference and propagates (exports) the r1_t
      terminal as its input.  ‘ It uses one compartment and
      integrates also the twostep modelInstance into this
      compartment.
    </body>
  </notes>

  <listOfCompartments>
    <compartment name="c1"/>
  <listOfCompartments>
  <listOfModelInstances>
    <modelInstance name="g" isa="twostep">
      <listOfCompartmentSpecs>
        <compartmentSpec compartment="c1"
                         isSubstitutedBy="parent.c1"/>
      </listOfCompartmentSpecs>
      <listOfSpecieSpecs>
        <specieSpec specie="s1" initialamount="100"/>
        <specieSpec specie="s2" initialamount="1"/>
      </listOfSpecieSpecs>
      <listOfReactionSpecs>
        <reactionSpec reaction="r1">
          <parameterSpec parameter="vm" value="300"/>
        </reactionSpec>
      </listOfReactionSpecs>
    </modelInstance>
  </listOfModelInstances>
  <listOfSpecies>
    <specie name="s3" initialAmount="1.0" compartment="c1"/>
  </listOfSpecies>
  <listOfTerminals>
    <reactionTerminal name="r1_t" terminal="g.r1_t"/>
  </listOfTerminals>
  <listOfReactions>
    <reaction name="r3">
      <listOfReactants >
        <specieReference terminal="g.s2_t"/>
      </listOfReactants>
      <listOfProducts>
        <specieReference specie="s3"/>
      </listOfProducts>
```

```
          <kineticLaw formula="uui(g.s2_t,vm,km)">
            <listOfParameters>
              <parameter name="km" value="2000"/>
              <parameter name="vm" value="30"/>
            </listOfParameters>
          </kineticLaw>
        </reaction>
      </listOfReactions>
    </model>

    <model name="whole" isMainModel="true">
      <notes>
        <body xmlns="http://www.w3.org/1999/xhtml">
          The model whole connects one specie with a twostep (a)
          and a path (b) modelInstance. Therefore links a.r1_t
          with the specie s4 and a.s2_t with b.r1_t.  The path b
          is integrated into the large compartment c1 but the
          compartment inside a exists further and is only marked
          as being inside of c1.
        </body>
      </notes>

      <listOfCompartments>
        <compartment name="c1" volume="10.0"/>
      <listOfCompartments>
      <listOfModelInstances>
        <modelInstance name="a" isa="twostep">
          <listOfCompartmentSpecs>
            <compartmentSpec compartment="c1" volume="0.1"
                             outside="parent.c1"/>
          </listOfCompartmentSpecs>
        </modelInstance>
        <modelInstance name="b" isa="path">
          <listOfCompartmentSpecs>
            <compartmentSpec compartment="c1" isSubstitutedBy="c1"/>
          </listOfCompartmentSpecs>
          <listOfReactionSpecs>
            <reactionSpec reaction="g.r1">
              <parameterSpec parameter="vm" value="15"/>
            <reactionSpec>
          </listOfReactionSpecs>
        </modelInstance>
      </listOfModelInstances>
      <listOfSpecies>
        <specie name="s4" initialAmount="5" compartment="c1"/>
      </listOfSpecies>
      <listOfLinks>
        <link specie="s4">
```

```
            <reactionReference reaction="a.r1_t"/>
          </link>
          <link specie="a.s2_t">
            <reactionReference reaction="b.r1_t"/>
          </link>
        </listOfLinks>
      </model>
    </listOfModels>
  </sbml>
```

References

[Cellier and Elmquist(1993)] F. Cellier and H. Elmquist. Automated manipulation supports object-oriented continuous-system modeling. *IEE Control Systems*, 13(2):28–38, 1993.

[Finney(2000)] A. Finney. Possible extensions to the systems biology markup language. Technical report, Control and Dynamical Systems, Caltech, Pasadena, CA, USA, 2000.

[Hucka et al.(2001)Hucka, Finney, Sauro, and Bolouri] M. Hucka, A. Finney, H. Sauro, and H. Bolouri. Systems biology markup language (SBML) level 1: Structures and facilities for basic model definitions. Technical report, Control and Dynamical Systems, Caltech, Pasadena, CA, USA, 2001.