Internal Discussion Document
Possible extensions to the Systems Biology Markup Language

Andrew Finney

`afinney@cds.caltech.edu`
ERATO Kitano Systems Biology Workbench Development Group
Control and Dynamical Systems 107-81
California Institute of Technology, Pasadena, CA 91125

Version of November 27, 2000

Contents

1 Disclaimer

This document is intended for internal discussion amongst members of the ERATO Kitano Systems Biology Workbench Development Group and selected collaborators. It is very unlikely that any final version of this document will have any IPR or other release restrictions.

This document has not been reviewed in detail by the group and at this stage just describes the author's ideas. In addition this document is incomplete: it contains editorial notes which act as place holders for future work.

2 Introduction

This document proposes new features for inclusion in SBML [1]. These features include

- *Submodels* the ability to create groups of model components and then create multiple instances or copies of these groups.

- *Arrays* the ability to create and use arrays of model components of arbitrary dimension

- *Neighborhoods* the ability to define and use a set of neighbors to components in arrays

Appendix A contains a complete model using these extensions in XML using SBML.

The following features are being considered but are not covered by this document yet:

- *Partitions* the ability to label reactions and submodels for allocation to specific analyses

- *Species* the ability to define a set of states for a specie which are encoded in binary string form - after Thomas Shimizu

- *Default Kinetic Law* a definition of what law is used when a `kineticLaw` element is missing from a `reaction` element.

- *Diagrams* the ability to attach diagram information to components to enable SBML to used as a storage format for models created using diagram based graphical user interfaces.

- *ODE and PDE rules* the ability to create ODE and PDE equations as `Rule` elements

- *3D geometry* the ability to define 3D geometries for compartments

- *Reference and Author Data* the ability to attach reference and author data to any component

- *Component Identification* the ability attach standardized external identifiers to components

In the text and diagrams in this document the proposed new version of SBML is implicitly the same as the latest version of SBML [1]. Only changes to SBML are described or shown.

3 Submodels

The submodels feature allows the creation new component types out of assemblies of other components. Components include species, compartments, rules, parameters and reactions. New component types are created by defining models within a model. This hierarchy can have arbitrary depth. Figure 1 shows how this is represented.

A model includes zero or more models (submodels). These are just definitions and do not imply any occurrence of components inside the model. A model includes zero or more model instances. These create occurrences of the components in the referenced submodel within the model. An `modelInstance` element contains a name attribute `model` which refers to the submodel that should be instantiated in the model.

For example the following xml defines a submodel XX. The model creates two model instances p and q of the submodel XX.

Model

name: Name
model[0..*]: Model
instance[0..*]: Instance

ModelInstance

name: Name {use = "optional"}
interface[0..*]: Interface
model : Name

InstanceSubstitute

insideInstance: Name
outsideInstance: Name

IntegerParameterSubstitute

insideIntegerParameter: Name
outsideIntegerParameter: Name

Substitute

SpecieSubstitute

insideSpecie: Name
outsideSpecie: Name

CompartmentSubstitute

insideCompartment: Name
outsideCompartment: Name

ParameterSubstitute

insideParameter: Name
outsideParameter: Name

Figure 1: *A diagram of the submodel class system (The notation used in the figures in this document is described in [2]).*

```
<sbml version="2">
    <model name="simpleinstances">
        <listOfModels>
            <model name="XX">
              <listOfCompartments>
                    <compartment name="x"/>
              </listOfCompartments>
              <listOfSpecies>
                    <specie name="a" compartment="x" initialAmount="1"/>
                    <specie name="b" compartment="x" initialAmount="1"/>
                    <specie name="c" compartment="x" initialAmount="1"/>
              </listOfSpecies>
              <listOfReactions>
                    <reaction name="s1">
                        <listOfReactants>
                            <specieReference specie="a"/>
                        </listOfReactants>
                        <listOfProducts>
                            <specieReference specie="b"/>
                        </listOfProducts>
                    </reaction>
                    <reaction name="s2">
                        <listOfReactants>
                            <specieReference specie="b"/>
                        </listOfReactants>
                        <listOfProducts>
                            <specieReference specie="c"/>
```
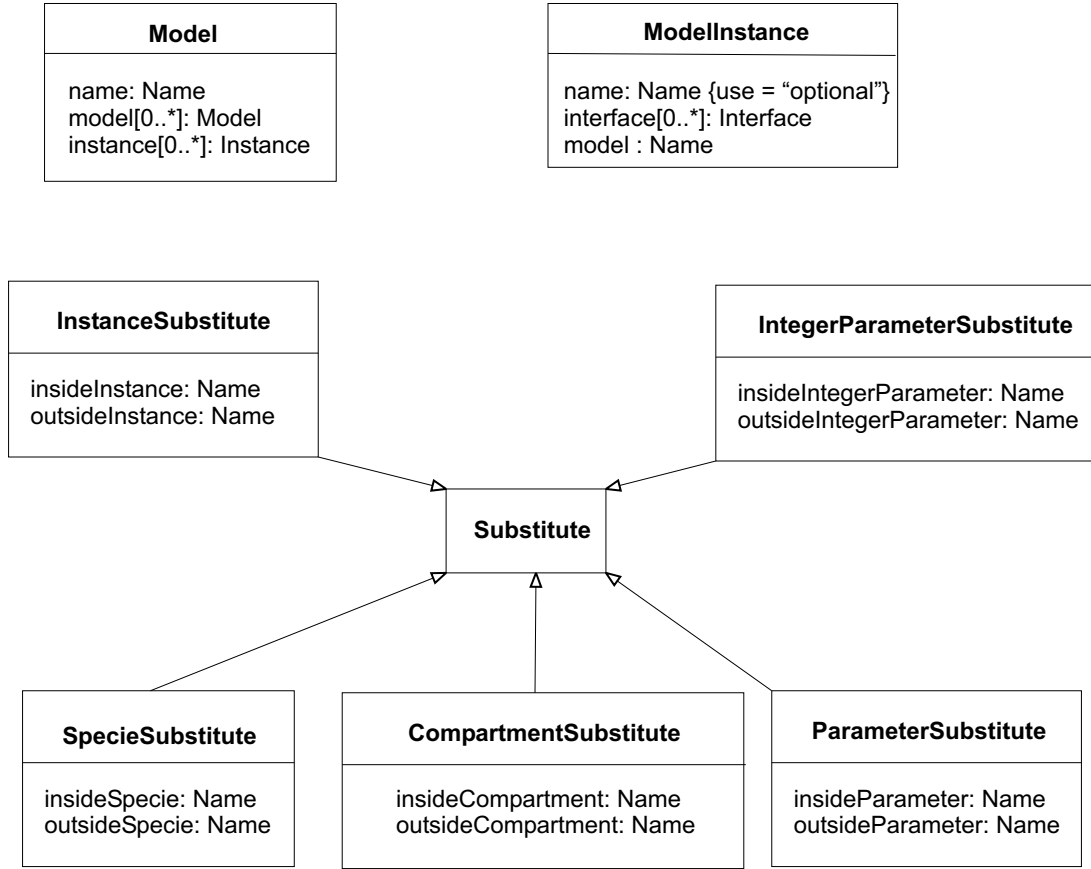
Figure 2: *A diagram of the simpleinstances model*

```
                    </listOfProducts>
                </reaction>
            </listOfReactions>
        </model>
    </listOfModels>
    <listOfModelInstances>
        <modelInstance name="p" model="XX"/>
        <modelInstance name="q" model="XX"/>
    </listOfModelInstances>
    </model>
</sbml>
```
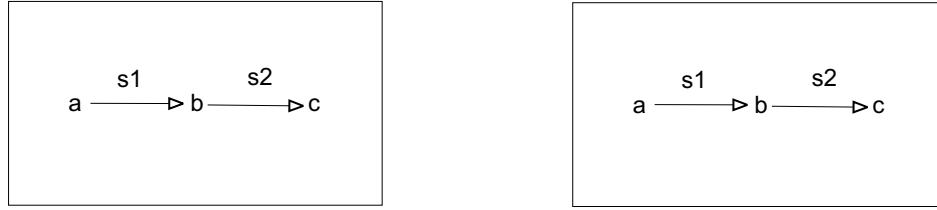
Figure 2 shows this model diagramatically.

In this model there are now two compartments, two species and two reactions. This is not meaningful as the species are not connected by a reaction.

3.1   Model Expansion and validation

This description of the submodel feature effectively defines a transformation from a model containing submodels and model instances to an equivalent model containing neither. This transformation is called *expansion*. The product of the transformation is called the *expanded* form of a model.

A model is valid if its expanded form is consistent (name attributes refer to other named elements of the correct type) and contains at least one compartment, one species and one reaction. The detail of a valid expanded form will require further work. In the meantime as a rough guide the expanded form should comply with the current SBML definition [1].

3.2   Referencing components inside model instances

Components inside model instances can be referenced in formulae and name attributes by using the form *instance.component* where *component* is the name of a component inside the model instance *instance*.

For example the following fragment shows an element referring to a species 's' inside model instance 'i'.

```
<specieReference specie="i.s"/>
```

The following model uses this technique to create a transport reaction between two compartments.

```
<sbml version="2">
    <model name="transport">
        <listOfModels>
            <model name="XX">
                <listOfCompartments>
                    <compartment name="inside"/>
                </listOfCompartments>
                <listOfSpecies>
                    <specie name="a" compartment="inside" initialAmount="1"/>
```

Figure 3: *A diagram of the transport model*

```
            </listOfSpecies>
        </model>
    </listOfModels>
    <listOfModelInstances>
        <modelInstance name="p" model="XX">
        <modelInstance name="q" model="XX">
    </listOfModelInstances>
    <listOfReactions>
        <reaction name="transport">
            <listOfReactants>
                <specieReference specie="p.a"/>
            </listOfReactants>
            <listOfProducts>
                <specieReference specie="q.a"/>
            </listOfProducts>
        </reaction>
    </listOfReaction>
    </model>
</sbml>
```

Figure 3 shows this model in diagram form.

This notation can be used to any depth. For example x.y.z refers to component z inside model instance y inside model instance x. This notation can only be used when referring to components - not declaring them.

3.3 Substitutes

Individual model instances can be made unique through the use of `substitute` elements, see Figure 1. An `modelInstance` element can have one or more `substitute` elements.

`Substitute` elements define how a component in a model ('outside') can be substituted for a component 'inside' a submodel for a given model instance. A submodel does not declare explicitly which components can be substituted. When a component is substituted the values and units associated with the outside component replace those of the inside component.

For example if we want to locate a model instance in a given compartment rather than the compartment that it was originally defined in by its submodel definition we can substitute another compartment for the original compartment as part of the `modelInstance` element. The original compartment is 'inside' the model instance and the substitute compartment is 'outside'.

This mechanism allows model instances to share common components by allowing a single component to be substituted into more than one model instance.

There are several different subtypes of `Substitute` one for each type of component, see Figure 1. All these types have a similar form. These types have a name attribute of the form `insideX` which refers to a component of the respective type inside the model instance. The `substitute` elements have an attribute of the form `outsideX` which refers to a component in the model (as opposed to the submodel).

3.4 Missing attributes

It is not necessary for components inside a submodel to have a complete set of attribute values. In a model's expanded form these attributes should have acquired values through the use of `substitute` elements. For example all specie elements which don't have values for their `initialAmount` can only occur inside submodels. Instances of these submodels must have `substitute` elements which substitute species which do have values

Element	Attribute	Resolved Constraint
specie	compartment	use="required"
specie	initialAmount	use="required"
specie	boundaryCondition	use="default" value="false"
parameter	value	use="required"
integerParameter	value	eliminated by expansion
compartment	volume	use="default" value="1"
modelInstance	model	eliminated by expansion

Table 1: *Attributes that become optional in this proposed version. The constraint column shows the constraint on the attribute in the expanded form.*

for their `initalAmount` attributes.

Table 1 lists the attributes that are optional in the unexpanded form of SBML. These attributes have to be either resolved in the expanded form through substitution or defaulted.

The above example model "simpleinstances" can be modified allowing the two model instances to form a pathway as follows.

```
<sbml version="2">
    <model name="connectedinstances">
        <listOfModels>
            <model name="XX">
               <listOfCompartments>
                    <compartment name="inside"/>
               </listOfCompartments>
               <listOfSpecies>
                    <specie name="a"/>
                    <specie name="b" compartment="inside" initialAmount="1"/>
                    <specie name="c"/>
               </listOfSpecies>
               <listOfReactions>
                    <reaction name="s1">
                        <listOfReactants>
                            <specieReference specie="a"/>
                        </listOfReactants>
                        <listOfProducts>
                            <specieReference specie="b"/>
                        </listOfProducts>
                    </reaction>
                    <reaction name="s2">
                        <listOfReactants>
                            <specieReference specie="b"/>
                        </listOfReactants>
                        <listOfProducts>
                            <specieReference specie="c"/>
                        </listOfProducts>
                    </reaction>
               </listOfReactions>
            </model>
        </listOfModels>
        <listOfCompartments>
            <compartment name="outside"/>
        </listOfCompartments>
        <listOfSpecies>
            <specie name="aa" compartment="x" initialAmount="1"/>
            <specie name="ca" compartment="x" initialAmount="1"/>
            <specie name="cc" compartment="x" initialAmount="1"/>
        </listOfSpecies>
        <listOfModelInstances>
            <modelInstance name="p" model="XX">
                <listOfSubstitutes>
```
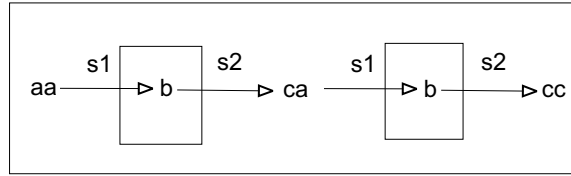
Figure 4: *A diagram of the connectedinstances model*

```
                        <speciesSubstitute insideSpecie="a" outsideSpecie="aa"/>
                        <speciesSubstitute insideSpecie="c" outsideSpecie="ca"/>
                    </listOfSubstitutes>
                </modelInstance>
                <modelInstance name="q" model="XX"/>
                    <listOfSubstitutes>
                        <speciesSubstitute insideSpecie="a" outsideSpecie="ca"/>
                        <speciesSubstitute insideSpecie="c" outsideSpecie="cc"/>
                    </listOfSubstitutes>
                </modelInstance>
            </listOfModelInstances>
        </model>
    </sbml>
```

Figure 4 shows this model in diagram form.

3.5   Missing components

A model can have zero or more of any compartments, species, model instances or reactions. The expanded form of a model should consist of at least one compartment, specie and reaction.

3.6   Units

[Ed - this section will be extended with examples in the future]

3.6.1   Built-in Units

A submodel can use different unit scales for component values of built-in units, eg volume, from those used by a model containing an instance of the submodel. When an `modelInstance` element has `substitute` elements which substitute components containing values at a different scales to the replaced components, the values from the substituted component are implicitly scaled to the scale used in the submodel.

3.6.2   Parameter Units

A similar scheme is used for `Parameter` elements. As for built-in types the scale of a substituted parameter value can be different from the scale of the replaced parameter. This assumes that the `Unit` of the two `Parameter` elements are equivalent [Ed - this needs careful definition].

In fact substituted `Parameter` elements must always have `Unit` elements equivalent to the `Parameter` elements that are replaced. This is a form of type checking. A `Parameter` element in a submodel which has no `Unit` can be replaced by any `Parameter` element through substitution. The opposite is not true.

4   Arrays of Components

It has become apparent that many systems biology simulations require arrays of components. The following section describes a scheme to support this.

The following components can occur in arrays: model instances, compartments, geometry, mapping, species,
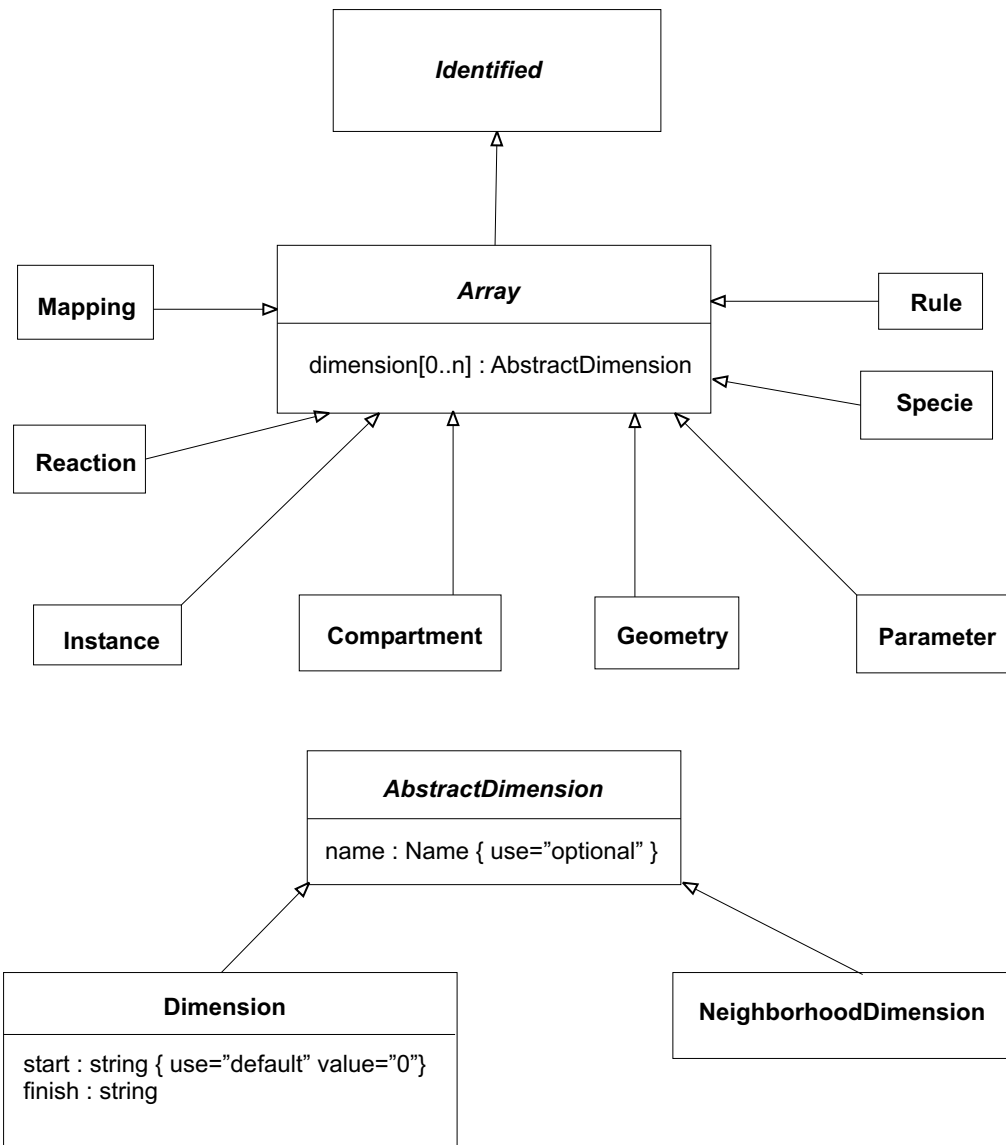
Figure 5: *The Dimension class*

parameters, rules and reactions. Each of these elements can contain a list of `dimension` elements, see Figure 5. This diagram shows the `NeighborhoodDimension` class which is detailed in the next section.

A dimension element has a `name` name attribute and two integer expression attributes `start` and `finish` which define the range of the array in that dimension.

The name attribute declares the given name as a symbol that can be used in integer expressions.

For example here's a two dimensional array of compartments:

```
<compartment name="cell" volumne="1">
    <listOfDimensions>
        <dimension name="x" start="0" finish="9"/>
        <dimension name="y" start="0" finish="9"/>
    </listofDimensions>
</compartment>
```

4.1 Expansion and validity

The principle of expansion defined in section 3.1 applies to arrays. The expanded form of a model doesn't contain any `dimension` elements. There is no difference in the validity of the expanded form of models containing arrays or submodels.

4.2 Refering to array elements

Names can be followed by an array element postfix operator `[]` where the brackets contain a comma separated sequence of integer expressions. The number of integer expressions should be the number of dimensions of the element declaring the symbol. This notation can only be used in name fields and symbols in formulae refering to components.

For example here's a species placed inside an element of the previous array of compartments.

```
<specie name="s" initialAmount="1" compartment="cell[0,0]"/>
```

As another example here's an array of species placed inside the previous array of compartments.

```
<specie name="s" initialAmount="1" compartment="cell[xd,yd]">
    <listOfDimensions>
        <dimension name="xd" start="0" finish="9"/>
        <dimension name="yd" start="0" finish="9"/>
    </listofDimensions>
</specie>
```

For example here's a rule applied to a species array.

```
<listOfSpecies>
    <specie name="s" initialAmount="1" compartment="cell">
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </specie>
</listOfSpecies>
<listOfRules>
    <specieRule specie="z" formula="t-s[i]"/>
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </specieRule>
</listOfRules>
```

This notation for referring to components inside array can be combined with the model instance component reference notation. For example accessing a specie inside a array of model instances:

```
<listOfModels>
    <model name="submodel">
        <listOfSpecies>
            <specie name="s" initialAmount="1" compartment="cell"/>
        </listOfSpecies>
    </model>
</listOfModels>
<listOfModelInstances>
    <modelInstance name="I" model="submodel">
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </modelInstance>
</listOfModelInstances>
```

```
<listOfRules>
    <specieRule specie="z" formula="t-I[i].s"/>
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </specieRule>
</listOfRules>
```

The opposite access method is possible as well, accessing a array of species inside a model instance:

```
<listOfModels>
    <model name="submodel">
        <listOfSpecies>
            <specie name="s" initialAmount="1" compartment="cell">
                <listOfDimensions>
                    <dimension name="i" start="0" finish="9"/>
                </listOfDimensions>
            </specie>
        </listOfSpecies>
    </model>
</listOfModels>
<listOfModelInstances>
    <modelInstance name="I" model="submodel"/>
</listOfModelInstances>
<listOfRules>
    <specieRule specie="z" formula="t-I.s[i]"/>
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </specieRule>
</listOfRules>
```

5   Neighborhoods

To further improve the modeling of arrays of biological entities we propose the concept of neighborhoods. A `neighborhood` is a mapping from an integer co-ordinate, *source*, to a set of integer co-ordinates, *connections*. A `neighborhood` can be defined for any number of dimensions.

The structure of `neighborhood` is shown in figure 6. A `neighborhood` element contains a number of `dimension` elements that define the range of possible connections in each dimension. These elements do not define the set of all connections but just restrict the possible connections to this range. The name attribute on `dimension` elements inside a `neighborhood` is redundant.

A `neighborhood` is defined by a set of `connectionGroup` elements. A `connectionGroup` consists of a set of `connection` elements. A `connection` element is a co-ordinate relative to a source and is comprised of a list of `connectionDimension` elements, one for each `dimension` of the `neighborhood`.

A `connectionGroup` has a `pattern` element for each dimension that describes which source co-ordinates the `connectionGroup` is applied to. The `pattern` string - integer formula - attributes `start`, `finish` and `step` define a loop. Each iteration of the loop defines a location where the `connectionGroup` source occurs.

The set of unbounded connections at a given location is the union of the set of `connection` elements of those `connectionGroup` elements that have a source that occurs at that location. The source co-ordinate is added to the given connection co-ordinate. The actual set of connections at a source is the intersection of the unbounded connections with the range defined by the `dimension` elements enclosed in the `neighborhood` element.

Each `pattern` and `connectionDimension` element corresponds to one dimension of the co-ordinate system that the `neighborhood` can be applied to. All lists containing `dimension`, `pattern` and `connectionDimension` elements should have the same number of elements when enclosed in the same neighborhood element
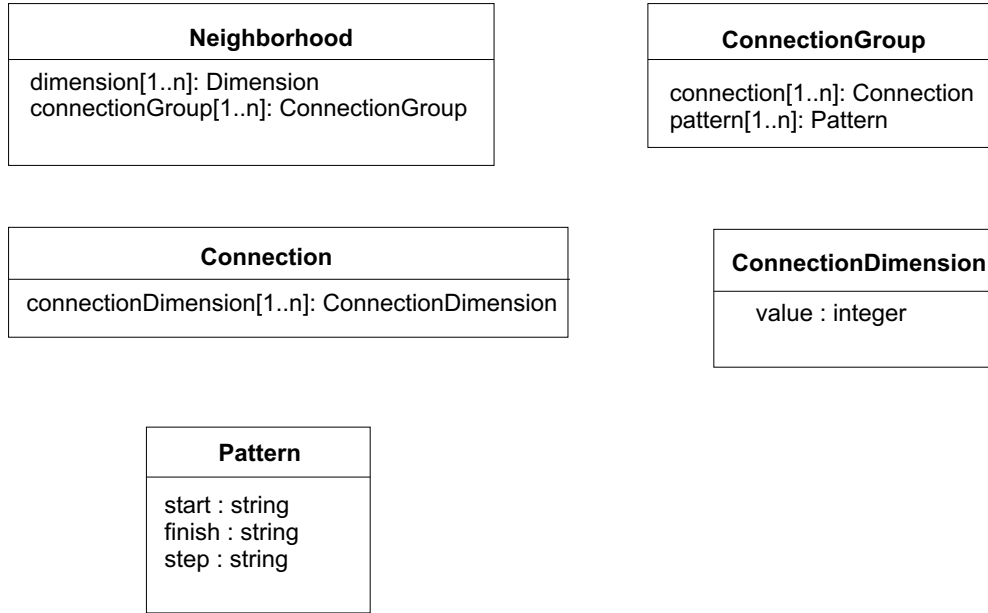
Figure 6: *Neighborhood and related classes*

A `neighborhood` consists of several `connectionGroup` elements so that its possible to define hexagonal connections in a 2D grid.

For example here's a simple 1 dimensional neighborhood:

```
<neighborhood name="one_d">
    <listOfDimensions>
        <dimension start="0" finish="8">
    </listOfDimensions>
    <listOfConnectionGroups>
        <connectionGroup>
            <listOfPatterns>
                <pattern start="0" finish="8" step="1"/>
            </listOfPatterns>
            <listOfConnections>
                <connection>
                    <listOfConnectionDimensions>
                        <connectionDimension value="1"/>
                    </listOfConnectionDimensions>
                </connection>
            </listOfConnections>
        </connectionGroup>
    </listOfConnectionGroups>
</neighborhood>
```

In the example given a co-ordinate of x in the range 0 to 8 is mapped to the set consisting of one integer x + 1. Any value outside that range results in an empty set. Obviously this example doesn't amount to much however it is possible to create hexagon patterns using this scheme see appendix A.

[Ed - Add description of algorithm for mapping a given co-ordinate to a the set of co-ordinates that is the set of connections for the given co-ordinate]
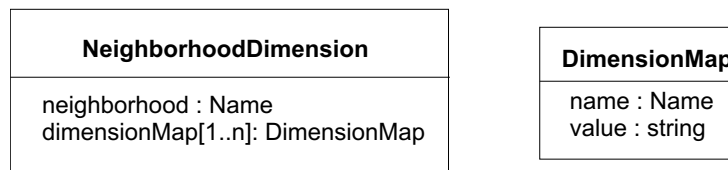
NeighborhoodDimension
neighborhood : Name dimensionMap[1..n]: DimensionMap

DimensionMap
name : Name value : string

Figure 7: `NeighborhoodDimension` and related classes

5.1 Using neighborhoods for creating arrays

A neighborhood can be referenced as a way to create an array that has a variable number of elements in one dimension. The number of elements at given co-ordinate is the the number of connections present in neighborhood at that location. This is achieved by using a `neighborhoodDimension` element, see figure 7, as a dimension in place of a `dimension` element. Like a `dimension` element a `neighborhoodDimension` element has a name attribute which declares a integer symbol.

A `neighborhoodDimension` element contains a list of dimensionMap elements. Each `dimensionMap` instantiates one integer symbol through its name attribute and supplies the value for the source co-ordinate in the corresponding dimension through its value attribute which is an integer expression.

For example here's an array of reactions for each connection in the previous xml fragment:

```
<listOfSpecies>
    <specie name="s" initialAmount="1" compartment="cell[i]">
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </specie>
</listOfSpecies>
....
<reaction name="transport" reversible="true">
    <listOfDimensions>
        <dimension name="i" start="0" finish="9"/>
        <neighborhoodDimension name="j" neighborhood="one-d">
            <dimensionMap name="k" value="i"/>
        </neighborhoodDimension>
    </listOfDimensions>
    <listOfProducts>
        <speciesReference specie="s[i]"/>
    </listOfProducts>
    <listOfReactant>
        <speciesReference specie="s[k]"/>
    </listOfReactants>
</reaction>
```

6 Math for arrays and neighborhoods

6.1 Matrix functions

Matrix functions can appear in float formulas. A scheme for product and sum functions is described here but other types of matrix functions are probably possible with further thought.

The sequence of the arguments to these functions are of the form $integerDeclaration_0, \ldots, integerDeclaration_n, f$ where

- $integerDeclaration_i := simpleDeclaration_i | mappedDeclaration_i$

- f is a float expression that can contain integer symbols declared in $simpleDeclaration$ and $mappedDeclaration$

- $simpleDeclaration_i$ is i_i, is_i, if_i

- i_i is a new integer symbol,

- is_i and if_i are integer expressions which define the range of i_i

- $mappedDeclaration_i$ is $neighborhood\ neighborhoodname[j_i], k_{i,0}, l_{i,0} \dots , k_{i,m}, l_{i,m}$

- $neighborhood$ is literal

- $neighborhoodname$ is the name of a `neighborhood` element

- j_i is an new integer symbol which runs over a set of connections in the given neighborhood,

- $k_{i,h}$ is a new integer symbol mapped by the given neighborhood

- $l_{i,h}$ is an integer expression which is the input co-ordinate to the neighborhood

There is a pair of $k_{i,h}$ and $l_{i,h}$ for each `dimension` in the `neighborhood` refereed to by the preceding $neighborhoodname$

For example consider a reaction affected by product of the concentrations of a set of species:

```
<listOfSpecies>
    <specie name="s" initialAmount="1" compartment="cell"/>
    <specie name="p" initialAmount="0" compartment="cell"/>
    <specie name="c" initialAmount="1" compartment="cell">
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </specie>
</listOfSpecies>
....
<reaction name="complex" reversible="true">
    <listOfProducts>
        <speciesReference specie="p"/>
    </listOfProducts>
    <kineticLaw formula="k * product(i, 0, 9, c[i])">
        <listOfParameters>
            <parameter name="k" value="0.25"/>
        </listOfParameter>
    </kineticLaw>
    <listOfReactant>
        <speciesReference specie="s"/>
    </listOfReactants>
</reaction>
```

As another example we add a similar reaction to the previous example using neighborhoods:

```
<listOfSpecies>
    <specie name="s" initialAmount="1" compartment="cell">
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </specie>
    <specie name="p" initialAmount="1" compartment="cell">
        <listOfDimensions>
            <dimension name="i" start="0" finish="9"/>
        </listOfDimensions>
    </specie>
</listOfSpecies>
....
<reaction name="z" reversible="true">
    <listOfDimensions>
        <dimension name="i" start="0" finish="9"/>
    </listOfDimensions>
    <listOfProducts>
```

```
            <speciesReference specie="s[i]"/>
        </listOfProducts>
        <kineticLaw formula="product(neighborhood one-d, i, connection, s[connection])"/>
        <listOfReactant>
            <speciesReference specie="p[i]"/>
        </listOfReactants>
    </reaction>
```

6.2 Integer Parameters

A model can have a list of integer parameters. An `integerParameter` element has a name attribute and an integer `value` attribute. The name values on these elements can be used as symbols in integer expressions in which case the value of `value` attribute is substituted. An expanded form of model must not contain `integerParameter` elements.

6.3 Integer Expressions

Integer expressions occur in various places in the above scheme. Integer expressions are similar to float expressions except that only $+, -, *$ and $/$ operations are available and the symbols are the names of either `dimension` or elements associated with the formula, the name of `integerParameter` elements or declared by an sum or product function call.

Appendix

A   A complete model using model instances and arrays

This model is comprised of 10000 'cells' arranged in 2 dimensions as a square 100 cells on a side. The cells are connected to six nearest neighbors to simulate a hexagonal tessalation of the cells.

A.1 Creating a neighborhood for a hexagonal tessalation

The method for creating a neighborhood which simulates a hexagonal tessalation is shown in figure 8. For this figure you can see that this requires 2 sets of connections: one for even columns and the other for odd columns. This is represented in the SBML model as two `connectionGroup` elements.

A.2 Reactions

The reactions in this model are shown figure 9. The model is not biologically meaningful. The duplicate reactions in the adjacent cell are not shown. None of the reactions are reversible.

A.3 The Model in XML

```
<sbml version="2">

    <model name="SIMPLE_HEXAGON">
        <listOfModels>

            <model name="CELL">

                <listOfCompartments>
                    <compartment name="cell">
                </listOfCompartments>

                <listOfSpecies>
                    <specie name="Sin" compartment="cell" initialAmount="1"/>
                    <specie name="S" compartment="cell" initialAmount="0"/>
```

A Hexagonal Tesslation mapped onto a 2D grid

Figure 8: *Mapping a hexagonal connection scheme onto connections in a 2D array*

```
        <specie name="Sinhib" compartment="cell" initialAmount="0"/>
        <specie name="Pout" compartment="cell" initialAmount="0"/>
    </listOfSpecie>

    <listOfReactions>

        <reaction name="J1">
            <listOfReactants>
                <speciesReference name="Sin">
            </listOfReactants>
            <listOfProducts>
                <speciesReference name="S">
            </listOfProducts>
            <kinecticLaw formula="uui(Sin, Vm, Km)"/>
            <listOfParameters>
                <parameter name="Vm" value="1"/>
                <parameter name="Km" value="1"/>
            </listOfParameters>
        </reaction>

        <reaction name="J2">
            <listOfReactants>
                <speciesReference name="S">
            </listOfReactants>
            <listOfProducts>
                <speciesReference name="Pout">
            </listOfProducts>
            <kinecticLaw formula="unii(S, Sinhib, V, Ki)"/>
            <listOfParameters>
                <parameter name="V" value="1"/>
                <parameter name="Km" value="1"/>
                <parameter name="Ki" value="1"/>
            </listOfParameters>
```
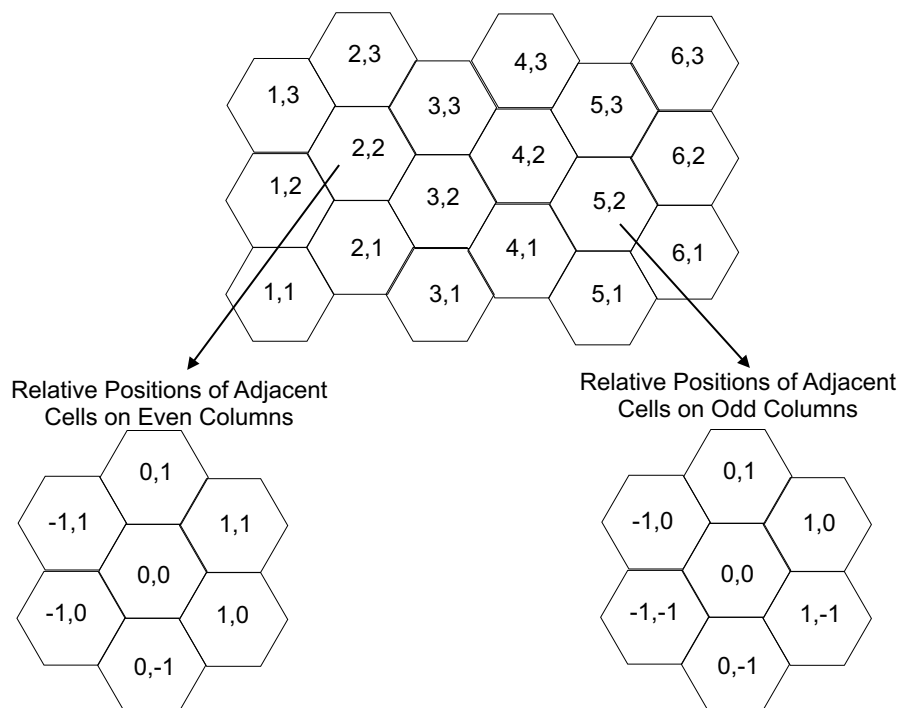
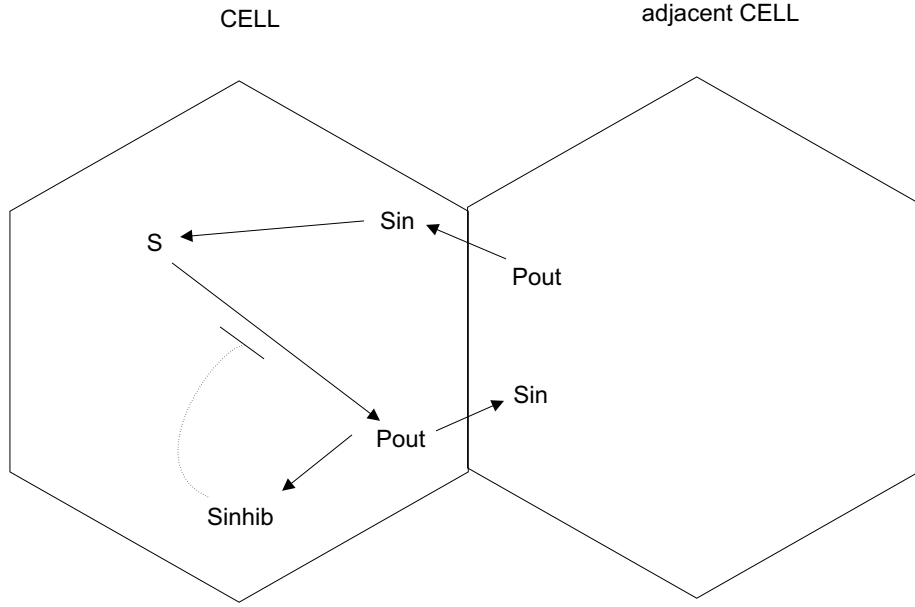**Figure 9:** *The reaction pathway modeled in the SIMPLE_HEXAGON model*

```
            </reaction>

            <reaction name="J3">
                <listOfReactants>
                    <speciesReference name="Pout">
                </listOfReactants>
                <listOfProducts>
                    <speciesReference name="Sinhib">
                </listOfProducts>
                <kinecticLaw formula="uui(S, Vm, Km)"/>
                <listOfParameters>
                    <parameter name="Vm" value="1"/>
                    <parameter name="Km" value="1"/>
                </listOfParameters>
            </reaction>

        </listOfReactions>
    </model>
</listOfModels>

<listOfIntegerConstants>
    <integerConstant name="size" value="100" />
<listOfIntegerConstants>

<listOfNeighborhoods>
    <neighborhood name="hexagon">
        <listOfDimensions>
            <dimension start="1" finish="size"/>
            <dimension start="1" finish="size"/>
        </listOfDimensions>
        <listOfConnectionGroups>
            <connectionGroup name="odd_columns">
                <listOfPatterns>
                    <pattern start="1" finish="size" step="2"/>
                    <pattern start="1" finish="size" step="1"/>
                </listOfPatterns>
                <listOfConnections>
                    <connection>
                        <listOfConnectionDimensions>
                            <connectionDimension value="0">
```

```xml
                <connectionDimension value="1">
            </listOfConnectionDimensions>
        </connection>
         <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="1">
                <connectionDimension value="1">
            </listOfConnectionDimensions>
        </connection>
        <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="1">
                <connectionDimension value="0">
            </listOfConnectionDimensions>
        </connection>
        <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="0">
                <connectionDimension value="-1">
            </listOfConnectionDimensions>
        </connection>
        <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="-1">
                <connectionDimension value="0">
            </listOfConnectionDimensions>
        </connection>
        <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="-1">
                <connectionDimension value="1">
            </listOfConnectionDimensions>
        </connection>
    </listOfConnections>
</connectionGroup>
<connectionGroup name="even_columns">
    <listOfPatterns>
        <pattern start="0" finish="size" step="2"/>
        <pattern start="2" finish="size" step="1"/>
    </listOfPatterns>
    <listOfConnections>
        <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="0">
                <connectionDimension value="1">
            </listOfConnectionDimensions>
        </connection>
         <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="1">
                <connectionDimension value="1">
            </listOfConnectionDimensions>
        </connection>
        <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="1">
                <connectionDimension value="0">
            </listOfConnectionDimensions>
        </connection>
        <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="0">
                <connectionDimension value="-1">
            </listOfConnectionDimensions>
        </connection>
        <connection>
            <listOfConnectionDimensions>
                <connectionDimension value="-1">
                <connectionDimension value="0">
```

```
                            </listOfConnectionDimensions>
                        </connection>
                        <connection>
                            <listOfConnectionDimensions>
                                <connectionDimension value="-1">
                                <connectionDimension value="1">
                            </listOfConnectionDimensions>
                        </connection>
                    </listOfConnections>
                </connectionGroup>
            </listOfConnectionGroups>
        </neighborhood>
    </listOfNeighborhood>

    <listOfModelInstances>
        <modelInstance name="cell" model="CELL">
            <listOfDimensions>
                <dimension start="1" finish="size"/>
                <dimension start="1" finish="size"/>
            </listOfDimensions>
        </modelInstance>
    </listOfModelInstances>

    <listOfReactions>

        <reaction name="between_cells">

            <listOfDimensions>
                <dimension name="x" start="1" finish="size"/>
                <dimension name="y" start="1" finish="size"/>
                <neighborhoodDimension neighborhood="hexagon">
                    <listOfDimensionMaps>
                        <dimensionMap name="xc" value="x"/>
                        <dimensionMap name="yc" value="y"/>
                    </listOfDimensionMaps>
                </neighborhoodDimension>
            </listOfDimensions>

            <listOfReactants>
                <specieReference specie="cell[xc, yc].Pout"/>
            </listOfReactants>

            <listOfProducts>
                <specieReference specie="cell[x, y].Sin"/>
            </listOfProducts>

            <kinecticLaw formula="uui(S, Vm, Km)"/>

            <listOfParameters>
                <parameter name="Vm" value="1"/>
                <parameter name="Km" value="1"/>
            </listOfParameters>

        </reaction>

    </listOfReactions>

    </model>

</sbml>
```

References

[1] Andrew Finney, Herbert Sauro, Michael Hucka, and Hamid Bolouri. An xml-based model description language for systems biology simulations. September 2000.

[2] Michael Hucka. A notation for describing data representations intended for xml encoding. September 2000.