Systems Biology Markup Language (SBML) Level 3 Proposal: Dynamic Arrays

Bruce E Shapiro, Victoria Gor, Eric Mjolsness
bshapiro@caltech.edu, victoria.gor@jpl.nasa.gov, emj@uci.edu

December 21, 2004

Contents

1. Introduction

This document describes proposed features for inclusion in Systems Biology Markup Language (SBML) Level 3. These features enable the inclusion of arrays of compartments, parameters, species, reactions, rules and events. These features are derived from the minimal array support requirements of (and are already used by) the Computable Plant Project (http://www.computableplant.org).

Although the features described herein are proposed to be a part of SBML Level 3, this is not a description of SBML Level 3. This proposal is an extension of SBML Level 2, and can be implemented as an extension of SBML Level 2. No additional features of SBML Level 3 are assumed beyond those that already exist in SBML Level 2 Version 1.

This proposal is independent of all other proposed extensions to SBML, and does not include any features that are designed to support other features, such as model composition or lists. However, this features proposed in this document do not exclude compatibility with such extensions.

For brevity this document only describes additional features that would be added to SBML by the proposal. All other features of SBML Level 2 are already assumed to exist; only the additions are described. All types proposed in this document are derived from the SBase type.

This document is a draft and comments and suggestions for improvement are welcome.

2. Arrays

The following objects can be defined as arrays: compartments, parameters, species, reactions, rules, and events.

Array
Dimension[0..n] exists: math {use="Optional"}

The Dimension field defines the number of indices and their allowed values; it is described in more detail in section 3.1.

The exists field is used to define conditional objects. It use is illustrated in section 6.

2.1. Array Notation Used in this Document

Elements of arrays will be referenced as `A[i,j,...]` in this document to refer to A_{ij}, i.e, the $i, j, \ldots$ th element of the mathematical matrix *A* with indices *i*, *j*, ...

In terms of SBML this means an array whose **`id`** is A whose first index has **`id`** i, second index **`id`** j, etc. This notation is used to describe the meaning of certain features, but it is not intended to be used explicitly anywhere in SBML.

In some cases we may refer to the ijk...th element of an array of objects (such as rules) that do not have object **`ids`** and can not be referenced in SBML.

Furthermore, we will sometimes use the same notation `A[i,j,...]` to refer to the array itself with indices i, j,... rather than a specific array element; the correct interpretation should be clear from the context in which it is used.

The notation `A[i..j, k..m, ..., p..q]` will be used to refer to the array

$$A_{\alpha\beta\ldots\xi}, \text{ where } i \le \alpha \le j,\ k \le \beta \le m, \ldots,\ p \le \zeta \le q$$

i.e, the array whose first dimension ranges from i to j, second dimension ranges from k to m, and whose last dimension ranges from p to q. For example `A[0..5,0..7]` refers to 6 by 8 array whose indices both start at 0.

2.2. Explicit Array Element Reference

Array elements can be referenced in two ways: explicitly and implicitly. Explicit array references allow users to specify precise array indices. Implicit references are discussed in section 2.3.

Explicit array references use the MathML **`selector`** operator. The following fragment of MathML represents the expression `0.1*s1[x]`, where x is an integer and s1 is a parameter of species array.

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
 <apply>
  <times/>
   <apply>
    <selector />
     <ci> s1 </ci>
```

```
    <ci> x </ci>
   </apply>
   <cn> 0.1 </cn>
 </apply>
</math>
```

Matrices can also be referenced in MathML using the **`selector`** operator. MathML does not provide for arrays with more than two indices. Rather than defining a special **`csymbol`** for this purpose, we allow the use of the selector operator for arrays with any number of indices, e.g.,

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
 <apply>
  <times/>
   <apply>
    <selector />
     <ci> s1 </ci>
     <ci> i </ci>
     <ci> j </ci>
     <ci> k </ci>
   </apply>
   <cn> 0.1 </cn>
 </apply>
</math>
```

would represent the expression `0.1*s1[i,j,k],` and so forth. Arguments of the selector operator starting with the second argument must be either integers, integer identifiers, or any expression that evaluates to an integer value.

2.3. Implicit Array Element References

Implicit references use the array identifier to implicitly define an array of objects. Implicit array references allow users to refer to a (possibly inclusive) leading subset of array indices using the array id only.

For example, array parameters A[1..3,1..3], v[1..3], and w[1..3], defined explicitly by

```
<listOfParameters>
<parameter id="A">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="3"/>
  <dimension id="j" lowerLimit="1" upperLimit="3"/>
 </listOfDimensions>
</parameter>
<parameter id="v">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="3"/>
 </listOfDimensions>
</parameter>
<parameter id="w">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="3"/>
 </listOfDimensions>
</parameter>
…
</listOfParameters>
```

Then the matrix product

```
w[i]=A[i,1]v[1]+A[i,2]v[2]+A[i,3]v[3]
```

can be written by implicitly referring to the first index an A:

```
<assignmentRule variable="w">
 <math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
```

```
    <plus/>
    <apply>
     <times/>
     <apply>
      <selector/>
      <ci>A</ci>
      <cn type="integer">1</cn>
     </apply>
     <apply>
      <selector/>
      <ci>v</ci>
      <cn type="integer">1</cn>
     </apply>
    </apply>
    <apply>
     <times/>
     <apply>
      <selector/>
      <ci>A</ci>
      <cn type="integer">2</cn>
     </apply>
     <apply>
      <selector/>
      <ci>v</ci>
      <cn type="integer">2</cn>
     </apply>
    </apply>
    <apply>
     <times/>
     <apply>
      <selector/>
      <ci>A</ci>
      <cn type="integer">3</cn>
     </apply>
     <apply>
      <selector/>
      <ci>v</ci>
      <cn type="integer">3</cn>
     </apply>
    </apply>
   </apply>
  </math>
 </assignmentRule>
 …
```

A simpler implementation would use the MathML scalarproduct function,

```
 <assignmentRule variable="w">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <apply>
    <ci>scalarproduct</ci>
    <ci>A</ci>
    <ci>v</ci>
   </apply>
  </math>
 </assignmentRule>
```

Additional examples are given in section 3.2 in the section on implicit array definitions.

3. Array Definitions

Objects can be defined as arrays in two ways, either ***explicitly*** or ***implicitly***. Any object can be defined as an array ***explicitly*** via a **`dimension`** object. Section describes explicit array definitions, and section 3.2 discusses implicit array definitions.

3.1 Explicit Array Definitions

An object can be explicitly declared to be an array by attaching a list of dimensions to the objects definition. The number of dimension statements is equal to the number of array indices; a vector would have one dimension statement, a matrix two, and so forth.

Dimension
id: SId name: string {use="optional"} lowerLimit: math | SId | Integer upperLimit: math | SId | Integer

Fixed length arrays are indicated with **`lowerLimit`** and **`upperLimit`** set to integer values. For example an a 10 by 10 array of compartments `C`, with array indices starting at 0, could be defined as

```
<compartment id="C">
 <listOfDimensions>
  <dimension id="i" lowerLimit="0" upperLimit="9"/>
  <dimension id="j" lowerLimit="0" upperLimit="9"/>
 </listOfDimensions>
</compartment>
```

The scope of the **`dimension id`** is local to the enclosing object definition (in this case the **`compartment`**) and is not visible outside the object (**`compartment)`** definition.

Dynamic arrays are defined by specifying one or more limits as a variable or with a MathML expression. Any previously defined variable may be referenced in **`upperLimit`** or **`lowerLimit`** of a **`dimension`**; however, all variables referenced in **`upperLimit`** or **`lowerLimit`** statements must be previously declared. Array limit variables can be reset by rules or events, and hence the arrays may become dynamic.

An array **`lowerLimit`** may not exceed its **`upperLimit`**. If a limit refers to a parameter with an undefined initial value than the initial value of that limit is assumed to be undefined.

Initial values in explicit declarations for compartments, parameters, and species are assumed to refer to every element of the array. To specify different initial values to different elements of an array an **`initialAssignmentRule`** should be used.

Any variable references must be previously defined in the SBML. The following example defines a **`parameter`** array `x[1..n, 0..2m+1],` where n is initially set to 10 and m is initial set 4.

```
<parameter id="m" value="4"/>
<parameter id="n" value="10"/>
<parameter id="x">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="n" />
  <dimension id="j" lowerLimit="0">
   <upperLimit>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
     <apply>
      <plus/>
       <apply>
        <times/>
         <cn type="integer">2</cn>
         <ci>m</ci>
        </apply>
       <cn type="integer">1</cn>
      </apply>
     </math>
    </upperLimit>
   </dimension>
  </listOfDimensions>
</parameter>
```

The first dimension statement

```
<dimension id="i" lowerLimit="1" upperLimit="n" />
```

can also be written as

```
<dimension id="i" lowerLimit="1">
 <upperLimit>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <ci>n</ci>
  </math>
 </upperLimit>
</dimension>
```

Because each element of an array is a different variable it is possible for different rules to refer to the same array. For example we can define an array of 8 rate rules

$$\frac{dx_i}{dt} = \begin{cases} y, \; i = 1,2,3,4,5 \\ 2y, \; i = 6,7,8 \end{cases}$$

with the following SBML

```
<rateRule variable="x">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="5"/>
 </listOfDimensions>
 <math xmlns="http://www.w3.org/1998/Math/MathML">
  <ci>y</ci>
 </math>
</rateRule>
<rateRule variable="x">
 <listOfDimensions>
  <dimension id="i" lowerLimit="6" upperLimit="8"/>
 </listOfDimensions>
 <math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
   <times/>
   <cn type="integer">2</cn>
   <ci>y</ci>
  </apply>
 </math>
 </rateRule>
```

Although this is an explicit declaration, it makes an implicit reference to the array x similar to the implicit rule declarations described in the following section.

An array of events can also be explicitly defined. For example, we can define the set of events

$$\text{If } x_i > 1 \text{ then set } x_i = \begin{cases} 0.5, \; i = 1,2,3,4,5 \\ 0.75, \; i = 6,7,8 \end{cases}$$

with the following SBML.

```
<event id="foosmall">
<listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="5"/>
 </listOfDimensions>
 <trigger>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <list>
    <apply>
     <gt/>
     <ci>x</ci>
     <cn type="integer">1</cn>
    </apply>
```

```
    </list>
   </math>
  </trigger>
  <listOfEventAssignments>
   <eventAssignment variable="x">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
     <cn type="real">0.5</cn>
    </math>
   </eventAssignment>
  </listOfEventAssignments>
 </event>
 <event id="foobig">
 <listOfDimensions>
   <dimension id="i" lowerLimit="6" upperLimit="8"/>
  </listOfDimensions>
  <trigger>
   <math xmlns="http://www.w3.org/1998/Math/MathML">
    <list>
     <apply>
      <gt/>
      <ci>x</ci>
      <cn type="integer">1</cn>
     </apply>
    </list>
   </math>
  </trigger>
  <listOfEventAssignments>
   <eventAssignment variable="x">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
     <cn type="real">0.75</cn>
    </math>
   </eventAssignment>
  </listOfEventAssignments>
 </event>
```

This set of event declarations implicitly refers to the array x.

3.2. Implicit Array Definitions

Arrays can be defined implicitly because variables "inherit" a dimension from certain "enclosing" or referenced objects, as described in the following sections.

3.2.1. Implied Compartment Arrays

If compartment B is inside of compartment A and A is an array, then B has the same implied dimensions as A. If B is an array inside A, then B has implied dimensions $\dim(\mathtt{B}) \times \dim(\mathtt{A})$. For example, to define an array of 100 cells, each of which contains a single nucleus and 20 organelles, one could implicitly define the compartments nucleus and organelle to inherit the dimensions of the enclosing compartment cell. The following example defines arrays of compartments `cell[1..100]`, `nucleus[1..100]`, and `organelle[1..100,1..25]`.

```
 <compartment id="cell">
  <listOfDimensions>
   <dimension id="i" lowerLimit="1" upperLimit="100"/>
  </listOfDimensions>
 </compartment>
 <compartment id="nucleus" outside="cell"/>
 <compartment id="organelle" outside="cell">
  <listOfDimensions>
   <dimension id="j" lowerLimit="1" upperLimit="20"/>
  </listOfDimensions>
 </compartment>
```

These same arrays could also be defined explicitly, as follows:

```
<compartment id="cell">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="100"/>
 </listOfDimensions>
</compartment>
<compartment id="nucleus">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="100"/>
 </listOfDimensions>
</compartment>
<compartment id="organelle">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="100"/>
  <dimension id="j" lowerLimit="1" upperLimit="20"/>
 </listOfDimensions>
</compartment>
```

However the explicit declaration looses the topological information indicated by the outside field. If the outside field were included along with the explicit definition, the meaning would be different. In the following example, we define 100 cells `cell[1..100]`, each will 100 nuclei, `nucleus[1..100,1..100]` and 2000 organelles, `organelle[1..100,1..100,1..20]`.

```
<listOfCompartments>
<compartment id="cell">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="100"/>
 </listOfDimensions>
</compartment>
<compartment id="nucleus" outside="cell">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="100"/>
 </listOfDimensions>
</compartment>
<compartment id="organelle" outside="cell">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="100"/>
  <dimension id="j" lowerLimit="1" upperLimit="20"/>
 </listOfDimensions>
</compartment>
</listOfCompartments>
```

3.2.2. Implied Parameter Arrays

Parameters can be associated with compartments using the **`foreach`** field.

Parameter
foreach: SId {use="optional"}

For example, given

```
<compartment id="cell">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="100"/>
 </listOfDimensions>
</compartment>
```

then the following will define an array of 100 parameters, `radius[1..100]`, and an array of 100 vectors, `position[1..100,1..3]`,

```
<listOfParameters>
<parameter id="radius" foreach="cell"/>
<parameter id="position" foreach="cell">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="3"/>
 </listOfDimensions>
</parameter>
```

```
</listOfParameters>
```

The equivalent explicit declarations would be

```
<listOfParameters>
 <parameter id="radius">
  <listOfDimensions>
   <dimension id="i" lowerLimit="1" upperLimit="100"/>
  </listOfDimensions>
 </parameter>
 <parameter id="position">
  <listOfDimensions>
    <dimension id="i" lowerLimit="1" upperLimit="100"/>
    <dimension id="j" lowerLimit="1" upperLimit="3"/>
   </listOfDimensions>
  </parameter>
 </listOfParameters>
```

3.2.3. Implied Rule Arrays

Arrays of rules can be defined by making an implicit reference to an array variable, without explicitly stating a variable's index. If more than one array variable are referenced in a rule they must have the same dimensions and are assumed to refer to the same (implied) index. For example, given

```
<listOfParameters>
 <parameter id="x">
  <listOfDimensions>
   <dimension id="i" lowerLimit="1" upperLimit="100"/>
  </listOfDimensions>
 </parameter>
<parameter id="y">
  <listOfDimensions>
   <dimension id="i" lowerLimit="1" upperLimit="100"/>
  </listOfDimensions>
 </parameter>
<parameter id="z">
  <listOfDimensions>
   <dimension id="i" lowerLimit="1" upperLimit="100"/>
   <dimension id="j" lowerLimit="1" upperLimit="5"/>
  </listOfDimensions>
 </parameter>
</listOfParameters>
```

Then the following example defines an array of 100 rate rules $x_i' = x_i y_i$.

```
<rateRule variable="x">
 <math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
   <times/>
   <ci>x</ci>
   <ci>y</ci>
  </apply>
 </math>
</rateRule>
```

Implicit rule definitions can also make explicit array references. The following defines an array of 100 assignment rules `x[i]=x[i]y[i]+z[i,3]`

```
<assignmentRule variable="y">
 <math xmlns="http://www.w3.org/1998/Math/MathML">
 <apply>
  <plus/>
   <apply>
    <times/>
     <ci>x</ci>
     <ci>y</ci>
```

```
    </apply>
    <apply>
     <selector/>
      <ci>z</ci>
      <cn type="integer">3</cn>
     </apply>
    </apply>
  </math>
</assignmentRule>
```

Note that the implied index i is not referenced anywhere in this example. To refer to a specific index a **`dimension`** statement should be included.

Assignment rules, algebraic rules, and initial assignment rules may also be defined in this manner.

3.2.4 Implied Reaction Arrays

Arrays of reactions may be defined implicitly by referring to any array species without explicitly referring to an array index. If `A[1..100]`, `B[1..100]`, and `C[1..100]` are arrays of species of the same dimension,

```
<listOfCompartments>
 <compartment id="cell">
  <listOfDimensions>
   <dimension id="i" lowerLimit="1" upperLimit="100"/>
  </listOfDimensions>
 </compartment>
</listOfCompartments>
<listOfSpecies>
 <species id="A" compartment="cell" />
 <species id="B" compartment="cell" />
 <species id="C" compartment="cell" />
</listOfSpecies>
```

then $A + B \rightarrow C$ is an implied array of reactions

```
<reaction id="reaction1">
 <listOfReactants>
  <speciesReference species="A"/>
  <speciesReference species="B"/>
 </listOfReactants>
 <listOfProducts>
  <speciesReference species="C"/>
 </listOfProducts>
</reaction>
```

If A, B, and C do not have compatible dimensions then this would produce an error.

3.2.5. Implied Event Arrays

An event array may be declared implicitly by referring to a trigger expression that evaluates to an array. For example, suppose that

```
<listOfCompartments>
 <compartment id="cell">
  <listOfDimensions>
   <dimension id="i" lowerLimit="1" upperLimit="100"/>
  </listOfDimensions>
 </compartment>
</listOfCompartments>
...
<listOfParameters>
  <parameter id="mass" foreach="cell">
</listOfParameters>
```

defines the variable `mass[1..50]`. In the following implied array of events, if any one of the `mass` variables is greater than 50, it will cause that particular `mass` to split in half as a result of the event's triggering.

```
<event id="divideMass">
 <trigger>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <list>
    <apply>
     <gt/>
     <ci>mass</ci>
     <cn type="integer">50</cn>
    </apply>
   </list>
  </math>
 </trigger>
 <listOfEventAssignments>
  <eventAssignment variable="mass">
   <math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
     <times/>
     <ci>mass</ci>
     <apply>
      <power/>
      <cn type="integer">2</cn>
      <cn type="integer">-1</cn>
     </apply>
    </apply>
   </math>
  </eventAssignment>
 </listOfEventAssignments>
</event>
```

4. Initial Assignment Rules

In initial value can be set to every element of a **`parameter`** array using the **`value`** field, to every element of a **`compartment`** array using the **`size`** field, or to every element of a **`species`** array with the **`initialAmount`** or **`initialConcentration`** field. However, these fields force the user to set every element of the array to the same value.

To set specific values of an array one must use an **`initialAssignmentRule`**. Initial assignment rules are a new class of rules that are similar to assignment rules, except that they are only evaluated only once, when the model is read into the system, and never again. They are used to set initial values to an array. Values set in an **`initialAssignmentRule`** override any initial values that have been previously set in the parameter, compartment, or species declaration.

initialAssignmentRule
index[0..n]: (Math Integer | Math Identifier) Index {use="optional"} variable: SId value: String | math

The **`id`** of the variable being assigned must be specified in the **`variable`** field.

The **`value`** field must evaluate to a numeric quantity.

The **`index`** allows one to assign values to specific elements of an array.

The following assigns the value of 5.7 to every element of an array x.

```
<listOfParameters>
 <parameter id="x">
  <listOfDimensions>
```

```
    <dimension id="i" lowerLimit="0" upperLimit="9">
   </listOfDimensions>
  <parameters>
...
</listOfParameters>
...
<listOfRules>
 <initialAssignmentRule variable="x" value="5.7">
 ...
```

The following SBML fragment has the identical result.

```
<listOfParameters>
 <parameter id="x" value="5.7">
  <listOfDimensions>
   <dimension id="i" lowerLimit="0" upperLimit="9">
  </listOfDimensions>
 <parameters>
...
</listOfParameters>
```

The following assigns different values to x[0..4]=5.7 and x[5..9]=3.2

```
<initialAssignmentRule variable="x" value="5.7">
 <listOfDimensions>
   <dimension id="i" lowerLimit="0" upperLimit="4">
 </listOfDimensions>
</initialAssignmentRule>
<initialAssignmentRule variable="x" value="3.2">
 <listOfDimensions>
   <dimension id="i" lowerLimit="5" upperLimit="9">
 </listOfDimensions>
</initialAssignmentRule>
```

An equivalent **`initialAssignmentRule`** could use the MathML `vector`:

```
<initialAssignmentRule variable="x">
 <math>
  <vector>
   <cn>5.7</cn>
   <cn>5.7</cn>
   <cn>5.7</cn>
   <cn>5.7</cn>
   <cn>5.7</cn>
   <cn>3.2</cn>
   <cn>3.2</cn>
   <cn>3.2</cn>
   <cn>3.2</cn>
   <cn>3.2</cn>
  </vector>
 </math>
</initialAssignmentRule>
```

The `MathML` matrix and `matrixRow` fields can also be used; for example, a 3 by 3 identity matrix can be defined with

```
<parameter id="Ident" value="0">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="3">
  <dimension id="j" lowerLimit="1" upperLimit="3">
 </listOfDimensions>
</parameter>
...
<initialAssignmentRule variable="Ident">
 <math>
```

```
  <matrix>
   <matrixrow>
    <cn>1</cn>
    <cn>0</cn>
    <cn>0</cn>
   </matrixrow>
   <matrixrow>
    <cn>0</cn>
    <cn>1</cn>
    <cn>0</cn>
   </matrixrow>
   <matrixrow>
    <cn>0</cn>
    <cn>0</cn>
    <cn>1</cn>
   </matrixrow>
  </matrix>
 </math>
</initialAssignmentRow>
```

Since MathML only specifically supports vector and matrix arrays but does not support higher dimensional arrays (arrays with three or more indices), the only way to assign values to specific elements is via the **`index`** field. The **`index`** field can also be used for vectors and matrices. For example, the following sets the initial value of `x[4,9,42]=14.`

```
<initialAssignmentRule variable="x" value="14">
 <listOfIndices>
  <cn>4</cn>
  <cn>9</cn>
  <cn>42</cn>
 </listOfIndices>
<initialAssignemntRule>
```

5. Sparse Arrays

Sparse arrays can be defined by first defining a default value of zero to every element of an array, and then assigning specific values to the non-zero elements. These specific assignments override the default values for those locations only. In the following example we define a 3 by 3 identity matrix by first setting every element of the array to zero in the **`parameter`** statement, and then by resetting the diagonal values to 1 in **`initialAssignmentRules`**:

```
<parameter id="Ident" value="0">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="3">
  <dimension id="j" lowerLimit="1" upperLimit="3">
 </listOfDimensions>
</parameter>
...
<initialAssignmentRule variable="Ident" value="1">
 <listOfIndices>
  <cn>1</cn>
  <cn>1</cn>
 </listOfIndices>
 </initialAssignmentRule>
<initialAssignmentRule variable="Ident" value="1">
 <listOfIndices>
  <cn>2</cn>
  <cn>2</cn>
 </listOfIndices>
</initialAssignmentRule>
<initialAssignmentRule variable="Ident" value="1">
 <listOfIndices>
```

```
  <cn>3</cn>
  <cn>3</cn>
 </listOfIndices>
</initialAssignmentRule>
```

This defines an array of 9 elements `Ident[0..3, 0..3]` in which every element is assigned a value.

It should be noted that sparse arrays are different from conditional objects, in which some elements are not defined. If there is no need to every know the values of the zero-elements – of if the off-diagonal elements never exist – one should use a conditional object defined by an **`exist`** statement instead of a sparse array.

6. Conditional Objects

We allow any array object to have an optional **`exists`** field to indicate that some values of an array are undefined and never need to be considered.

For example, the following defines an array that has two indices but whose off diagonal elements are undefined and can be ignored by the program. Note that this is not the same object as the sparse diagonal array defined in the previous section.

```
<parameter id="I" value="1">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="3">
  <dimension id="j" lowerLimit="1" upperLimit="3">
 </listOfDimensions>
 <exists>
  <math>
   <apply>
    <eq/>
    <ci>i</ci>
    <ci>j</ci>
   </apply>
  </math>
 </exists>
</parameter>
```

This has the same effect of setting `I[1,1]=I[2,2]=I[3,3]=1`, but has a different effect on the off-diagonal elements: it tells us that elements like `I[2,3]` do not exist, rather than being set to zero.

7. Connection Rules

Connection rules allow users to define different types of geometric interactions.

ConnectionRule
id: SId from: SId to: SId math: math indicesFrom[1..n]: (Math Integer | Math Identifier) indicesTo[1..n]: (Math Integer | Math Identifier)

AssignmentRule
connection: SId connectionIndexType: "from" | "to" indicesFrom[1..n]: (Math Integer | Math Identifier) indicesTo[1..n]: (Math Integer | Math Identifier)

InitialAssignmentRule
connection: SId connectionIndexType: "from" | "to" indicesFrom[1..n]: (Math Integer | Math Identifier) indicesTo[1..n]: (Math Integer | Math Identifier)

RateRule
connection: SId connectionIndexType: "from" | "to" indicesFrom[1..n]: (Math Integer | Math Identifier) indicesTo[1..n]: (Math Integer | Math Identifier)

A **`connectionRule`** is a rule that operates on compartment array indices that evaluates to a boolean value. It is different from other rule types in that the **`id`** field is required, because it is referred to in the **`connection`** field of other rules. It also has two fields from and to which refer to the **`id`** fields of compartments. The compartments referred to must be single-dimensional arrays (vectors).

Suppose that `cell` is a compartment array. Then the following defines a **`connectionRule`** `xneighbor` from `cell[from]` to `cell[to]` whenever $|x[from]-x[to]|<radius[from]$, i.e., whenever two cells are close along the x-axis.

```
<listOfCompartments>
<compartment id="cell" spatialDimensions="2">
 <listOfDimensions>
  <dimension id="i" lowerLimit="1" upperLimit="100"/>
 </listOfDimensions>
</compartment>
</listOfCompartments>
...
<listOfParameters>
 <parameter id="x" foreach="cell" />
 <parameter id="radius" foreach="cell" />
</listOfParameters>
...
<listOfRules>
<connectionRule id="xneighbor" from="cell" to="cell">
  <listOfIndicesFrom>
   <ci>from</ci>
  </listOfIndicesFrom>
  <listOfIndicesTo>
   <ci>to</ci>
  </listOfIndicesTo>
 <math xmlns='http://www.w3.org/1998/Math/MathML'>
 <apply>
  <abs/>
  <apply>
   <lt/>
   <apply>
    <plus/>
    <apply>
     <selector/>
     <ci>x</ci>
     <ci>from</ci>
    </apply>
    <apply>
     <times/>
     <cn type='integer'>-1</cn>
     <apply>
      <selector/>
      <ci>x</ci>
```

```
      <ci>to</ci>
     </apply>
    </apply>
   </apply>
   <apply>
    <selector/>
    <ci>radius</ci>
    <ci>from</ci>
   </apply>
  </apply>
 </apply>
</math>
</connectionRule>
...
</listOfRules>
```

A second **connectionRule** neighbor might define two cells as neighbors if their distance (in a two dimensional simulation) is less than some parameter value radius[from]:

$$\sqrt{(x[from]-x[to])^2+(y[from]-y[to])^2} < radius[from]$$

The **connection** field of an assignment, initial assignment, or rate rule determines if two compartments are connected, and if so, defines that rule that operates on their indices. Suppose, for example, that we also have the parameter definition

```
<parameter id="p" foreach="cell"/>
```

added to the previous example. Then the following defines the rate rules

$$\frac{d}{dt}p[from]=5$$

if cell[from] is connected to cell[to] via connection rule neighbor and also the rule

$$\frac{d}{dt}p[from]=x[from]$$

if cell[from] is connected to cell[to] via connection rule xneighbor, and also the rule. If both rules are true then the right hand sides of the two differential equations in the different rate would be are added, i.e.,

$$\frac{d}{dt}p[from]=x[from]+5$$

The field **connectionIndexType** tells a rate, assignment, or initialAssignmentRule whether the rule applies to the variable associated with the "from" end of the connection or the "to" end of the connection. The only allowed values are the strings "from" and "to".

All connection indices referenced in the MathML must be defined by in a **listOfIndicesFrom** or **listOfIndicesTo** field. These field declare the indices that will be used to refer to the from and to ends of the connection. These are lists to allow for multi-dimensional arrays.

The rate rules for this example are given in the following SBML fragment:

```
<rateRule variable="p" connection="neighbor" connectionIndexType="from">
 <listOfIndicesFrom>
   <ci>from</ci>
 </listOfIndicesFrom>
 <math xmlns="http://www.w3.org/1998/Math/MathML">
    <cn type="integer">5</cn>
 </math>
</rateRule>
<rateRule variable="p" connection="xneighbor" connectionIndexType="from">
```

```
 <listOfIndicesFrom>
   <ci>from</ci>
 </listOfIndicesFrom>
<math xmlns="http://www.w3.org/1998/Math/MathML">
<apply>
   <selector/>
   <ci>x</ci>
   <ci>from</ci>
  </apply>
 </math>
</rateRule>
```

8. Array References in Functions

We also require that functions can make reference to array variables. It is not necessary to declare arrays as such within the function or to declare the array indices or limits within functions.

The following defines a function on two vectors,

```
f(x,y)=x[2]y[1]-y[2]x[1]
```

The arguments are declared as vectors using the **`type`** field of the **`ci`** command.

```
<functionDefinition id="f" />
 <math xmlns="http://www.w3.org/1998/Math/MathML">
  <lambda>
   <bvar>
    <ci>x</ci>
   </bvar>
   <bvar>
    <ci>y</ci>
   </bvar>
   <apply>
    <plus/>
    <apply>
     <times/>
     <apply>
      <ci>selector</ci>
      <ci>x</ci>
      <cn type="integer">2</cn>
     </apply>
     <apply>
      <ci>selector</ci>
      <ci>y</ci>
      <cn type="integer">1</cn>
     </apply>
    </apply>
    <apply>
     <times/>
     <cn type="integer">-1</cn>
     <apply>
      <times/>
      <apply>
       <ci>selector</ci>
       <ci>x</ci>
       <cn type="integer">1</cn>
      </apply>
      <apply>
       <ci>selector</ci>
       <ci>y</ci>
       <cn type="integer">2</cn>
      </apply>
     </apply>
```

```
    </apply>
   </apply>
  </lambda>
 </math>
</functionDefinition>
```

9. Array Math

Under this proposal the following set of MathML elements would be incorporated into SBML math elements:

- constructors: **`matrix`**, **`matrixrow`**, **`vector`**
- element referenced operator: **`selector`**
- qualifier components: **`bvar`**, **`lowlimit`**, **`uplimit`**, **`interval`**, **`condition`**
- linear algebra operators: **`vectorproduct`**, **`scalarproduct`**, **`outerproduct`**, **`transpose`**
- sum product operators: **`sum`**, **`product`**
- quantifier operators: **`forall`**, **`exists`**

10. Acknowledgements

This proposal is heavily indebted to the authors of the earlier array proposal (Finney et all, 2003b). Many thanks to Andrew Finney for a detailed reading and critique of this proposal.

This work was supported by the US National Science Foundation Frontiers in Integrative Biological Research Program under award number 0330786.

11. References

Finney A, Hucka M (2003a) Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitons, SBML Level 2, Version 1 (Final) (June 28, 2003) http://www.sbml.org/documents

Finney A, Gor V, Bornstein E, Mjolsness E (2003b) Systems Biology Markup Language (SBML) Level 3 Proposal: Array Features (May 1, 2003) http://sbml.org/wiki/Arrays