SBML Level 3 Package Specification

SBML Level 3 Package: Arrays (‘arrays’)

NEED SOME AUTHORS HERE

Version 1 (Draft)

September 11, 2013 (0.1α)

This is a draft specification for the package ‘arrays’ and not a normative document. Please send feedback to the Package Working Group mailing list at sbml-arrays@lists.sourceforge.net

The latest release, past releases, and other materials related to this specification are available at
http://sbml.org/Documents/Specifications/SBML_Level_3/Packages/Arrays

This release of the specification is available at
http://sbml.org/Documents/Specifications/arrays_Level_1_Version_1

Contents

1 Introduction and motivation

This document describes a package for arrays to be used with the Systems Biology Markup Language (SBML) Level 3 Version 1. This package enables SBML elements such as compartments, parameters, species, reactions, etc. to be arrays. While this package proposal is independent of all other package proposals, it is designed to work with and complement the hierarchical model composition and distributions packages. Indeed, the need to specify probability mass functions is a significant motivator for needing arrays.

1.1 Proposal corresponding to this package specification

NEED A PROPOSAL

1.2 Package dependecies

The arrays package has no dependencies on other SBML Level 3 packages. It is also designed with the goal of being able to work seamlessly with other SBML Level 3 packages. For example any objects are entirely encapsulated and any extensions to existing SBML classes are defined as optional.

1.3 Document conventions

Following the precedent set by the SBML Level 3 Core specification document, we use UML 1.0 (Unified Modeling Language; **??**) class diagram notation to define the constructs provided by this package. We also use color in the diagrams to carry additional information for the benefit of those viewing the document on media that can display color. The following are the colors we use and what they represent:

- ⬛ *Black*: Items colored black in the UML diagrams are components taken unchanged from their definition in the SBML Level 3 Core specification document.
- 🟩 *Green*: Items colored green are components that exist in SBML Level 3 Core, but are extended by this package. Class boxes are also drawn with dashed lines to further distinguish them.
- 🟦 *Blue*: Items colored blue are new components introduced in this package specification. They have no equivalent in the SBML Level 3 Core specification.

We also use the following typographical conventions to distinguish the names of objects and data types from other entities; these conventions are identical to the conventions used in the SBML Level 3 Core specification document:

AbstractClass: Abstract classes are classes that are never instantiated directly, but rather serve as parents of other object classes. Their names begin with a capital letter and they are printed in a slanted, bold, sans-serif typeface. In electronic document formats, the class names defined within this document are also hyperlinked to their definitions; clicking on these items will, given appropriate software, switch the view to the section in this document containing the definition of that class. (However, for classes that are unchanged from their definitions in SBML Level 3 Core, the class names are not hyperlinked because they are not defined within this document.)

Class: Names of ordinary (concrete) classes begin with a capital letter and are printed in an upright, bold, sans-serif typeface. In electronic document formats, the class names are also hyperlinked to their definitions in this specification document. (However, as in the previous case, class names are not hyperlinked if they are for classes that are unchanged from their definitions in the SBML Level 3 Core specification.)

`SomeThing`, `otherThing`: Attributes of classes, data type names, literal XML, and generally all tokens *other* than SBML UML class names, are printed in an upright typewriter typeface. Primitive types defined by SBML begin with a capital letter; SBML also makes use of primitive types defined by XML Schema 1.0 (**???**), but unfortunately, XML Schema does not follow any capitalization convention and primitive types drawn from the XML Schema language may or may not start with a capital letter.

For other matters involving the use of UML and XML, we follow the conventions used in the SBML Level 3 Core specification document.

2 Background

2.1 Problems with current SBML approaches

2.2 Past work on this problem or similar topics

2.3 Array notation used in this document

Elements of arrays will be referenced as A[i,j,...] in this document to refer to, i.e, the 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

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.

3 Proposed syntax and semantics

In this section, we define the syntax and semantics of the Arrays package for SBML Level 3 Version 1. We expound on the various data types and constructs defined in this package, then in Section 4 on page 10, we provide complete examples of using the constructs in example SBML models.

3.1 Namespace URI and other declarations necessary for using this package

Every SBML Level 3 package is identified uniquely by an XML namespace URI. For an SBML document to be able to use a given SBML Level 3 package, it must declare the use of that package by referencing its URI. The following is the namespace URI for this version of the Arrays package for SBML Level 3 Version 1:

“`http://www.sbml.org/sbml/level3/version1/arrays/version1`”

In addition, SBML documents using a given package must indicate whether understanding the package is required for complete mathematical interpretation of a model, or whether the package is optional. This is done using the attribute `required` on the `<sbml>` element in the SBML document. For the Arrays package, the value of this attribute must be set to “`true`”.

The following fragment illustrates the beginning of a typical SBML model using SBML Level 3 Version 1 and this version of the Arrays package:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level3/version1/core" level="3" version="1"
      xmlns:fbc="http://www.sbml.org/sbml/level3/version1/arrays/version1" arrays:required="true">
```

3.2 Primitive data types

Section 3.1 of the SBML Level 3 Version 1 Core specification defines a number of primitive data types and also uses a number of XML Schema 1.0 data types (?). More specifically we make use of `integer`, `double`, `string`, and `SIdRef`.

3.3 The Dimension class

QUESTION: if we use size instead of limits, are arrays 0 based or 1 based?

The arrays package extends several SBML L3V1 core elements with the addition of an ordered list of dimensions. An object with a list of dimensions is an array, and the number of dimensions is equivalent to the number of array indices; a vector would have one dimension statement, a matrix two, etc. Currently, objects are restricted to at most two dimensions. Each dimension of an arrays has a fixed size which is with a size attribute which is a parameterId indicated with the lowerLimit and upperLimit values. For example, a 10 by 10 array of compartments C, with array indices starting at 0, could be defined as:

```
<parameter id="n" value="10"/>
<compartment id="C" size="1.0" ...>
 <arrays:orderedListOfDimensions>
  <arrays:dimension id="i" size="n"/>
  <arrays:dimension id="j" size="n"/>
 </arrays:orderedListOfDimensions>
</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. 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 initialAssignment should be used.

For SBML L3V1 core, the following objects are extended to include an optional ordered list of dimensions:

- Parameters
- Compartments
- Species
- Reactions
- Species references
- Rules
- Initial assignments
- Events
- Constraints

Packages may choose to allow dimensions as desired. For example, comp will allow dimensions on subModels.

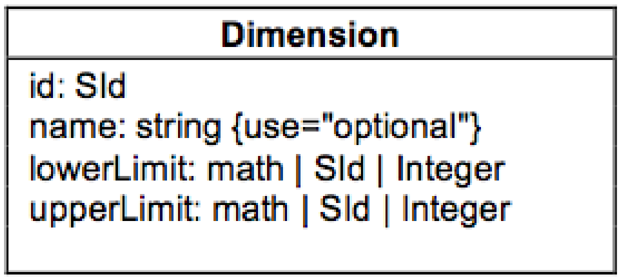
Dimension
id: SId
name: string {use="optional"}
lowerLimit: math | SId | Integer
upperLimit: math | SId | Integer

Figure 1: *A UML representation of the Arrays package classes. See Section 1.3* for conventions related to this figure.

3.4 The Index class

An assignment can include an ordered list of indices to specify the indices of the array element to assign to. Currently, this list can include at most two elements of the index class. When an element has dimensions but no index is provided, it is assumed that the entire array is referenced. When an element has two dimensions but only one index is provided, it is assumed that the corresponding row is being referenced (TODO: need a way to get one column). In all other cases, a scalar is returned. Each index class object is a mathematical function that is used to compute the index. The example below copies a reverse copy of the array in parameter x into parameter y. Basically, this example is equivalent to:

```
for (i=1; i < n; i++) {
  y(11 - i) = x(i)
}
```

```
<parameter id="n" value="10"/>
<parameter id="x" ...>
 <arrays:orderedListOfDimensions>
  <arrays:dimension id="i" size="n"/>
 </arrays:orderedListOfDimensions>
```

```
</parameter>
<parameter id="y" ...>
 <arrays:orderedListOfDimensions>
  <arrays:dimension id="i" size="n"/>
 </arrays:orderedListOfDimensions>
</parameter>
<assignmentRule variable=y>
 <arrays:orderedListOfDimensions>
  <arrays:dimension id="i" size="n"/>
 </arrays:orderedListOfDimensions>
 <arrays:orderedListOfIndices>
  <arrays:index>
   <math>
    <apply>
     <selector/>
      <ci>y</ci>
      <apply>
       <minus/>
        <cn>11</cn>
        <ci>i</ci>
      </apply>
    </apply>
   </math>
  </arrays:index>
 </arrays:orderedListOfIndices>
 <math>
  <apply>
   <selector/>
    <ci>x</ci>
    <cn>i</cn>
  </apply>
 </math>
</assignmentRule>
```

For SBML L3V1 core, the following objects are extended to include an optional ordered list of indices:

- Species references
- Rules
- Initial assignments
- Events assignments

Packages may choose to allow indicies as desired.

3.5 Extensions to the MathML subset

In order to support arrays, the following mathML elements need to be added to the supported MathML subset.

- 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**

4 Examples

This section contains a variety of examples of SBML Level 3 Version 1 documents employing the Arrays package.

4.1 Array of Reactions

This example creates an array `cell` of 100 compartments, arrays for species `A`, `B`, and `C` also of size 100 with each one placed in the corresponding compartment `cell(i)`, and an array of 100 reactions with one within each `cell(i)` converting `A(i)` plus `B(i)` into `C(i)`.

```
<listOfCompartments>
<!-- Specifies size of all arrays (i.e., n:=100) -->
<parameter id="n" value="100"...>
<!-- Create an array of n compartments -->
 <compartment id="cell"...>
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </compartment>
</listOfCompartments>
<listOfSpecies>
 <!-- Create array of n species A with A(i) implicitly placed in cell(i) -->
 <species id="A" compartment="cell" ... >
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </species>
 <!-- Create array of n species B with B(i) implicitly placed in cell(i) -->
 <species id="B" compartment="cell" ... >
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </species>
 <!-- Create array of n species C with C(i) implicitly placed in cell(i) -->
 <species id="C" compartment="cell" ... >
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </species>
</listOfSpecies>
<!-- Create array of n reactions r with r(i) converting A(i) and B(i) into C(i)-->
<listOfReactions>
 <reaction id="r" ...>
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
  <listOfReactants>
   <speciesReference species="A">
    <arrays:orderedListOfIndices>
     <arrays:index>
      <math><apply><selector/><ci>A<ci><ci>i</ci></apply></math>
     </arrays:index>
    </arrays:orderedListOfIndices>
   </speciesReference>
   <speciesReference species="B">
    <arrays:orderedListOfIndices>
     <arrays:index>
      <math><apply><selector/><ci>B<ci><ci>i</ci></apply></math>
     </arrays:index>
    </arrays:orderedListOfIndices>
   </speciesReference>
  </listOfReactants>
  <listOfProducts>
   <speciesReference species="C">
    <arrays:orderedListOfIndices>
     <arrays:index>
      <math><apply><selector/><ci>C<ci><ci>i</ci></apply></math>
     </arrays:index>
    </arrays:orderedListOfIndices>
   </speciesReference>
  </listOfProducts>
 </reaction>
</listOfReactions>
```

4.2 Array of Rate Rules

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

```
<listOfParameters>
 <!-- Create size variables for arrays -->
 <parameter id="n" value="8"/>
 <parameter id="m" value="5"/>
 <parameter id="o" value="3"/>
 <!-- Create array x of size n -->
 <parameter id="x" ...>
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </parameter>
 <!-- Create scalar parameter y -->
 <parameter id="y" .../>
</listOfParameters>
<listOfRules>
 <!-- Create rate rules dx(i)/dt = y for i = 1,2,3,4,5 -->
 <rateRule variable="x">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="m"/>
  </arrays:orderedListOfDimensions>
  <arrays:orderedListOfIndices>
   <arrays:index>
    <math><apply><selector/><ci>x<ci><ci>i</ci></apply></math>
   </arrays:index>
  </arrays:orderedListOfIndices>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
  <ci>y</ci></math>
 </rateRule>
 <!-- Create rate rules dx(i)/dt = 2*y for i = 6,7,8 -->
 <rateRule variable="x">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="o"/>
  </arrays:orderedListOfDimensions>
  <arrays:orderedListOfIndices>
   <arrays:index>
    <math>
    <apply>
     <selector/>
      <ci>x<ci>
      <apply>
       <plus/>
        <ci>i</ci>
        <ci>5</ci>
      </apply>
     </apply>
    </math>
   </arrays:index>
  </arrays:orderedListOfIndices>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply><times/>
   <cn type="integer">2</cn> <ci>y</ci>
  </apply></math>
 </rateRule>
</listOfRules>
```

4.3 Array of Events

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

```
<listOfParameters>
 <!-- Create size variables for arrays -->
 <parameter id="n" value="8"/>
 <parameter id="m" value="5"/>
 <parameter id="o" value="3"/>
 <!-- Create array x of size n -->
 <parameter id="x" ...>
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
</listOfParameters>
<listOfEvents>
 <!-- Create events to set x(i) to 0.5 when x(i)>1 for i = 1,2,3,4,5 -->
 <event id="eventSmall">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="m"/>
  </arrays:orderedListOfDimensions>
  <trigger>
   <math xmlns="http://www.w3.org/1998/Math/MathML">
    <list>
     <apply>
      <gt/>
       <apply>
        <selector/>
        <ci>x</ci>
        <ci>i</ci>
       </apply>
      <cn type="integer">1</cn>
     </apply>
    </list>
   </math>
  </trigger>
  <listOfEventAssignments>
   <eventAssignment variable="x">
    <arrays:orderedListOfIndices>
     <arrays:index>
      <math><apply><selector/><ci>x<ci><ci>i</ci></apply></math>
     </arrays:index>
    </arrays:orderedListOfIndices>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
     <cn type="real">0.5</cn>
    </math>
   </eventAssignment>
  </listOfEventAssignments>
 </event>
 <!-- Create events to set x(i) to 0.75 when x(i)>1 for i = 6,7,8 -->
 <event id="eventBig">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="o"/>
  </arrays:orderedListOfDimensions>
  <trigger>
   <math xmlns="http://www.w3.org/1998/Math/MathML">
    <list>
     <apply>
      <gt/>
       <apply>
        <selector/>
        <ci>x</ci>
        <apply>
         <plus/>
          <ci>i</ci>
          <cn>5</cn>
        </apply>
       </apply>
      <cn type="integer">1</cn>
     </apply>
    </list>
   </math>
  </trigger>
  <listOfEventAssignments>
   <eventAssignment variable="x">
    <arrays:orderedListOfIndices>
     <arrays:index>
      <math>
       <apply>
        <selector/>
        <ci>x<ci>
        <apply>
         <plus/>
          <ci>i</ci>
          <cn>5</cn>
        </apply>
       </apply>
      </math>
     </arrays:index>
    </arrays:orderedListOfIndices>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
     <cn type="real">0.75</cn>
    </math>
   </eventAssignment>
  </listOfEventAssignments>
 </event>
</listOfEvents>
```

4.4 Initial Assignment Arrays

This will set an the same initial value to all 10 elements of the x array.

```
<listOfParameters>
 <!-- Set size n=10 -->
 <parameter id="n" value="10"/>
 <!-- Set array parameters x(i)=5.7 for all i=1,...,10 -->
 <parameter id="x" value="5.7"...>
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n">
  </arrays:orderedListOfDimensions>
 </parameter>
</listOfParameters>
```

This could also be done with an initial assignment.

```
<listOfParameters>
 <!-- Set size n=10 -->
 <parameter id="n" value="10"/>
 <!-- Create an array x of size n -->
 <parameter id="x"...>
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </parameter> ...
</listOfParameters> ...
<listOfInitialAssignments>
```

```
 <!-- Set array parameters x(i)=5.7 for all i=1,...,10 -->
 <initialAssignment variable="x">
  <math>
   <cn>5.7<cn>
  </math>
 </initialAssignment>
</listOfInitialAssignments>
```

Here is an example where half of the array is assigned 5.7 and the other half is 3.2.

```
<listOfParameters>
 <!-- Set size n=10 -->
 <parameter id="n" value="10"/>
 <!-- Set size m=5 -->
 <parameter id="m" value="5"/>
 <!-- Create an array x of size n -->
 <parameter id="x"...>
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </parameter> ...
</listOfParameters> ...
<listOfInitialAssignments>
 <!-- Set array parameters x(i)=5.7 for i=1,...,5 -->
 <initialAssignment variable="x">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="m"/>
  </arrays:orderedListOfDimensions>
  <arrays:orderedListOfIndices>
   <arrays:index>
    <math><apply><selector/><ci>x<ci><ci>i</ci></apply></math>
   </arrays:index>
  </arrays:orderedListOfIndices>
  <math><cn>5.7<cn></math>
 </initialAssignment>
 <!-- Set array parameters x(i)=3.2 for i=6,...,10 -->
 <initialAssignment variable="x">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="m"/>
  </arrays:orderedListOfDimensions>
  <arrays:orderedListOfIndices>
   <arrays:index>
    <math>
     <apply>
      <selector/>
      <ci>x<ci>
      <apply>
       <plus/>
       <ci>i</ci>
       <ci>m</ci>
      </apply>
     </apply>
    </math>
   </arrays:index>
  </arrays:orderedListOfIndices>
  <math><cn>3.2<cn></math>
 </initialAssignment>
</listOfInitialAssignments>
```

This could also be done using the `vector` operator.

```
<initialAssignment 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>
</initialAssignment>
```

The `matrix` and `matrixrow` operators can also be used for initial assignments.

```
<listOfParameters>
 <!-- Create size variable n=3 -->
 <parameter id="n" value="3">
 <!-- Create a two dimensional array of size n by n -->
 <parameter id="Ident" value="0">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
   <arrays:dimension id="j" size="n"/>
  </arrays:orderedListOfDimensions>
 </parameter>
</listOfParameters>
<listOfInitialAssignments>
 <!-- Assign Ident to the identity matrix -->
 <initialAssignment 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>
 </initialAssignment>
</listOfInitialAssignments>
```

Here is an example to assign a single value.

```
<initialAssignment variable="Ident">
 <arrays:orderedListOfIndices>
  <arrays:index>
```

```
   <math><apply><selector/><ci>Ident<ci><ci>2</ci></apply></math>
  </arrays:index>
  <arrays:index>
   <math><apply><selector/><ci>Ident<ci><ci>1</ci></apply></math>
  </arrays:index>
 </arrays:orderedListOfIndices>
 <math>
  <cn>14<cn>
 </math>
<initialAssignment>
```

4.5 Examples for array referencing

Here is an example array reference using `selector`.

$$0.1 * s1[x]$$

```
<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>
```

Here is a more complicated example of array referencing.

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

```
<listOfParameters>
 <!-- Create size variable n=3 -->
 <parameter id="n" value="3".../>
 <!-- Create 2-dimensional array A of size n by n -->
 <parameter id="A">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
   <arrays:dimension id="j" size="n"/>
  </arrays:orderedListOfDimensions>
 </parameter>
 <!-- Create an array v of size n -->
 <parameter id="v">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </parameter>
 <!-- Create an array w of size n -->
 <parameter id="w">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
 </parameter>
</listOfParameters>
<listOfRules>
 <!-- w(i) = A(i,1)v(1) + A(i,2)v(2) + A(i,3)v(3) -->
 <assignmentRule variable="w">
  <arrays:orderedListOfDimensions>
   <arrays:dimension id="i" size="n"/>
  </arrays:orderedListOfDimensions>
  <arrays:orderedListOfIndices>
   <arrays:index>
    <math><apply><selector/><ci>w<ci><ci>i</ci></apply></math>
   </arrays:index>
  </arrays:orderedListOfIndices>
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <apply><plus/>
    <apply><times/>
     <apply><selector/><ci>A</ci><ci>i</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><ci>i</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><ci>i</ci><cn type="integer">3</cn></apply>
     <apply><selector/><ci>v</ci><cn type="integer">3</cn></apply>
    </apply>
   </apply>
  </math>
 </assignmentRule>
</listOfRules>
```

This could also be done with `scalarproduct` as follows:

```
<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>
```

4.6 Array references in functions

Functions can also make reference to array variables, but in this case, 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,

$$f(x, y) \quad = \quad x[2]\,y[1] - y[2]\,x[1]$$

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

```
<!-- f(x,y) = x(2)y(1) - y(2)x(1) -->
<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>
```

5 Best practices

In this section, we recommend a number of practices for using and interpreting various constructs in the arrays package. These recommendations are non-normative, but we advocate them strongly; ignoring them will not render a model invalid, but may reduce interoperability between software and models.

Acknowledgments

THANK SOME PEOPLE