Proposal for a Minimal SBML Level 2 Definition

Michael Hucka and Herbert Sauro

with material taken from other proposals by Andrew Finney, Victoria Gor, Eric Mjolsness and Hamid Bolouri

{mhucka,hsauro}@cds.caltech.edu
Systems Biology Workbench Development Group
ERATO Kitano Systems Biology Project
Control and Dynamical Systems, MC 107-81
California Institute of Technology, Pasadena, CA 91125, USA
http://www.cds.caltech.edu/erato

Principal Investigators: John Doyle and Hiroaki Kitano

June 28, 2002

1 Introduction

The SBML community is currently engaged in defining extensions to SBML Level 1 (Hucka et al., 2001) to support the representation of many new features: spatial geometry, model layout/diagrams, metadata, arrays, multistate biochemical entities, etc. All the features are important and will ultimately be necessary; however, for practical reasons, we propose here an alternative definition of Level 2 containing only a modest subset of features, with others left to SBML Level 3. The motivations for this proposal are the following:

1. A limited target for Level 2, containing a small set of features, can act as a concrete milestone on the road to developing a fuller Level 3 definition.
2. A smaller set of new SBML features to consider will enable the community to better evaluate the effects of the proposed changes and additions.
3. With a smaller set of changes to evaluate, it should be possible for the SBML community to reach consensus more quickly and define a working SBML Level 2 in short order.
4. There is currently an opportunity to contribute to the choice of exchange languages used by the DARPA BioSPICE effort. Discussions in the BioSPICE community has lead to the suggestion that SBML could be sufficient for their needs if it were extended to support the following: allowing user-defined functions, allowing discontinuous functions in formulas, and supporting simple metadata. This opportunity will exist only for a limited time; therefore, we propose to define Level 2 quickly, which is more likely to happen if the increment from Level 1 is kept small.

We propose that SBML Level 2 consist of four features added to SBML Level 1: (1) a convention for encoding arbitrary symbols in the `SName` syntax; (2) the provision for conditional expressions in formula strings; (3) the addition of user-defined functions for formulas; and (4) the addition of a basic set of metadata.

This proposal is partly based on a previous proposal by Finney et al. (2002). In order to provide readers with a self-contained proposal, we incorporate portions of Finney et al.'s document directly in this one.

2 Features Proposed for SBML Level 2

The following sections detail the proposed additions to SBML Level 1.

2.1 Encoding Arbitrary Symbols in `SName`

SBML Level 1 stipulates that symbol names must conform to a particular limited syntax containing only alphanumeric characters and the underscore ('_') character. This was chosen in an attempt to maximize compatibility between different software tools which may turn the symbols of an SBML model directly into symbols in (for example) a scripting language.

To extend the range of symbols that can be expressed in the very limited `SName` syntax, we propose the following encoding rule: for any character that is not one of '_', a-z, A-Z, or 0-9, express the character as the sequence __u*XXXX*__ (i.e., two underscores, followed by the letter 'u', followed by four hexadecimal numbers, following by two more underscores), where *XXXX* is the hexadecimal Unicode encoding of that character (Unicode Consortium, 1996, 2002). For example, the name IκB turns into `I__u03BA__B`, with `03BA` being the Unicode sequence for the Greek letter κ.

If a software program can display a symbol as the appropriate glyph, then it should translate names containing __u*XXXX*__ into the appropriate glyph on the screen. If a program cannot display the character, it should leave it as-is. In that case, users will still be able to manipulate the name, though the presentation will be less attractive.

2.2 Conditional Expressions in Mathematical Formulas

Finney et al. (2002) have already proposed an extension to SBML Level 1 formula strings for expressing discontinuous functions. We agree with that proposal, and reproduce the text here for easier reading. The rest of this section has been taken mostly verbatim from Section 4 of the document by Finney et al. (2002), with changed items shown in green.

2.2.1 New Operators

Table 1 shows a possible extended set of operators that could be available in SBML Level 2. New operators are shown in red and green. As in SBML Level 1, these operators work as defined in the programming language C, except that the types are always `double`. This means that "0" is interpreted as false and all nonzero numbers are interpreted as true.

The operators `&&` and `||` short-circuit the evaluation of their second operand depending on the value of the first operand. (This is an optimization that could be left optional, because the functions proposed in Section 2.3 cannot have side-effects and therefore short-circuiting cannot change the result value of an expression. However, if the definition of functions is changed to support multi-line functions with bodies and possible side-effects, it will then become necessary to require short-circuiting the evaluation of the operators `&&` and `||`.)

2.2.2 New Function: `switch`

In addition to the new operators described above, we propose to introduce an additional function called `switch`. The syntax of this function would be as follows:

```
switch(key, match1, result1, ... matchn, resultn, defaultResult);
```

This function would compare `key` to the values `match1...matchn` and returns the `result` value immediately following the matching `match`x argument. If none of the `match`x arguments are equal to `key`, then `switch` would return the value of the `defaultResult` argument.

The `switch` function is merely a convenient way of expressing long sequences of if-then conditions. The call

```
switch(k, m, r, d)
```

Tokens	Operation	Class	Precedence	Associates
name	symbol reference	operand	10	n/a
(*expression*)	expression grouping	operand	10	n/a
f (...)	function call	prefix	10	left
!	logical not	unary	9	right
not	logical not	unary	9	right
-	negation	unary	9	right
^	power	binary	8	left
*	multiplication	binary	7	left
/	division	binary	7	left
+	addition	binary	6	left
-	subtraction	binary	6	left
<	less than	binary	5	left
>	greater than	binary	5	left
>=	greater than or equal	binary	5	left
<=	less than or equal	binary	5	left
==	equality	binary	4	left
!=	inequality	binary	4	left
&&	logical and	binary	3	left
and	logical and	binary	3	left
||	logical or	binary	2	left
or	logical or	binary	2	left
? :	conditional	ternary	1	right

Table 1: *A table of the expression operators available in SBML, operators proposed in this document are shown in red. In the* **Class** *column, "operand" implies the construct is an operand, "prefix" implies the operation is applied to the following arguments, "unary" implies there is one argument, and "binary" implies there are two arguments. The values in the* **Precedence** *column show how the order of different types of operation are determined. For example, the expression* $a*b+c$ *is evaluated as* $(a*b)+c$ *because the* $*$ *operator has higher precedence. The* **Associates** *column shows how the order of similar precedence operations is determined; for example,* $a-b+c$ *is evaluated as* $(a-b)+c$ *because the* $+$ *and* $-$ *operators are left-associative. The precedence and associativity rules are taken from the C programming language (Harbison and Steele, 1995), except for the symbol ^, which is used in C for a different purpose.*

is equivalent to

```
(k == m) ? r : d
```

The call

```
switch(k, m1, r1, m2, r2, d)
```

is equivalent to

```
(k == m1) ? r1 : ((k == m2) ? r2 : d)
```

and so on.

2.2.3 Issues

Given that the function `switch` and the operators `<=`, `>=` and `!=` are redundant should they be incorporated into SBML Level 2?

2.3 User-Defined Functions in Mathematical Formulas

Finney et al. (2002) have already proposed an extension to SBML Level 1 to allow for user-defined functions. We agree with that proposal, and reproduce the text here with one alteration: the addition of two more questions for dicussion in Section 2.3.2 below. The rest of this section has been taken mostly verbatim from Section 5 of the document by Finney et al. (2002).

The proposed redefinition of the `Model` type is shown in UML in Figure 1. The change in this new definition is that a `Model` can now optionally contain a list of `Function` elements. The definition of `Function` is shown in Figure 2.

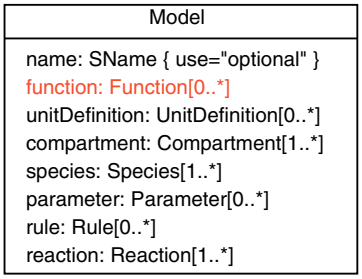

Figure 1: *The definition of the `Model` type.*

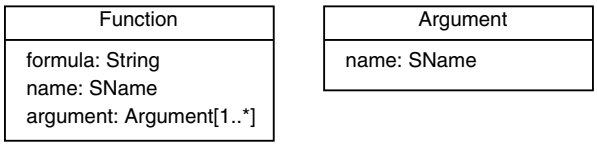

Figure 2: *The definition of the `Function` type.*

The `Function` data structure consists of the function's name (with a type of `SName`), a formula string, and a list of `Argument` structures. An `Argument` structure simply consists of the argument's name.

The `Function` structure defines a new function that can be used in any formula located after the definition of that `Function` structure in the text of an SBML model. The only symbols that can be used in the `formula` string of `Function` are the following:

- any of the names given in the `Argument` list;
- any of the names given in preceding `Function` structures;
- any of the names of built-in functions.

These restrictions mean that functions cannot be recursive or mutually recursive, thus ensuring that function "calls" can be expanded in place. Models using functions can therefore be mapped to SBML Level 1.

A `Function` formula simply returns a `double` value. All the arguments to a function are of type `double`. These functions share the same namespace as other model-level components; this means that, for example, a function can't have the same name as a specie or a reserved name.

2.3.1 Example

The following is a simple example of a `Function` structure in a `Model` structure:

```
<model name="cell">
    <listOfFunctions>
        <function name="pow3" formula="x^3">
            <listOfArguments>
                <argument name="x"/>
            </listOfArguments>
        </function>
    </listOfFunctions>
    ...
</model>
```

2.3.2 Issues

- Is the scope of symbols in formulas in functions too restrictive?
- Do we need to be able to handle objects other than `double` values?
- Should we actually introduce two kinds of user-defined functions, macros and true functions? The `Function` proposed above are actually macros and can be implemented as text substitutions. True functions would support more than a single `formula` expression. It may be useful to have the ability to define functions as a list of expressions, so that functions could compute intermediate values or subexpressions within their body. Without this, it will not be possible to split up a complicated expression except by using multiple `Function` definitions.

  Here is an example of what a true function might look like:

```
<listOfFunctions>
  <function name="sillyfunction">
    <listOfArguments>
      <argument name="x"/>
      <argument name="y"/>
    </listOfArguments>
    <listOfParameters>
      <parameter name="a" value="0.5"/>
    </listOfParameters>
    <listOfStatements>
      <statement lhs="a" rhs="cos(x)"/>
      <statement lhs="y" rhs="sin(a)"/>
    </listOfStatements>
  </function>
</listOfFunctions>
```

2.4 Metadata

The CellML group has worked out an approach to incorporating metadata in CellML models (Cuellar et al., 2002). For SBML Level 2, we propose to use a subset of the metadata components used in CellML, specifically RDF and the Dublin Core Metadata Element Set only. We leave the remaining aspects of CellML's metadata definition to SBML Level 3.

2.4.1 Motivations for the Approach

There are several motivations for taking his approach:

- RDF plus Dublin Core is not complicated. If the SBML group were to define their own metadata tags, for example, it would undoubtedly end up very similar to the Dublin Core set (modulo differences in tag names and XML namespaces).
- RDF and Dublin Core are gaining ground as standards. It is preferable to reuse standards when possible rather than proposing unique new definitions. Among other things, this makes it more likely to find off-the-shelf tools that work with the standard representations. There currently exist several free RDF parser implementations for Java, C, Perl, and other languages.
- A number of other XML-based languages also use RDF for metadata; for example, CellML and CML

(Chemical Markup Language; `www.xml-cml.org`). Settling on the same metadata representation would provide greater interoperability as well as opportunities for synergies that the SBML community cannot yet anticipate.

- SBML can be specialized to the business of defining biochemical models, because SBML is intended to be used by software created for the purpose of handling biochemical models. However, metadata about an SBML model is intended to be used not just by these software tools but by other software that is *not* specialized to biochemical modeling—for example, search engines and database systems. Thus, it behooves us to use a more general framework that is accepted by a broader range of software systems.
- The CellML group has researched the metadata topic extensively. The SBML community can leverage their work and save considerable (and quite likely needless) effort.
- Using the same basic metadata framework for both CellML and SBML would allow the two dominant biochemical model representation languages to be described in common ways, benefitting biologists and other users.

2.4.2 Metadata Components for SBML Level 2

As mentioned above, we propose using a subset of the CellML metadata definition, specifically the use of RDF and "simple" Dublin Core (i.e., the DC Metadata Element Set). We also propose that the use of these components in SBML be consistent with their use in the CellML metadata framework.

Here is an example of XML metadata expressed using this approach (taken from the CellML metadata specification):

```
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="#toon_times">
    <dc:title>Toonville Times</dc:title>
    <dc:creator>R.J. Gopher</dc:creator>
    <dc:date>2001-10-18</dc:date>
  </rdf:Description>
</rdf:RDF>
```

In the example above, `dc` is the prefix for Dublin Core elements. Dublin Core defines the following terms:

`Title`	`Creator`	`Subject`
`Description`	`Publisher`	`Contributor`
`Date`	`Type`	`Format`
`Identifier`	`Source`	`Language`
`Relation`	`Rights`	`Coverage`

For information about the specific meanings of these terms, we refer readers to the CellML Metadata 1.0 Specification document (Cuellar et al., 2002).

2.4.3 Metadata Components for SBML Level 3 and Beyond

We propose the possible adoption of the remainder of the CellML metadata elements (e.g., the use of vCard, BQS, etc.) be left to SBML Level 3. The goal here is to strike a balance between a larger metadata framework (namely, the whole of the CellML metadata definition) and none at all. Hopefully, RDF and Dublin Core can serve as a small-enough working set for the immediate future.

3 Discussion

An alternative to the approach taken here is to modularize the definition of SBML Level 2. In this alternative approach, each feature (such as support for arrays, support for spatial geometries, etc.) would be encapsulated as a separate add-on, and each SBML Level 2 model would declare which particular feature set it uses.

A given simulation tool could key off the feature set declaration in an SBML model file to determine whether it can interpret the model correctly or whether the model uses features that it cannot handle. This is a valid alternative and could be implemented using a variety of possible mechanisms (e.g., XML namespaces, or a list of feature tags in an SBML model file header).

Such a modular Level 2 definition may offer advantages in terms of backwards compatibility. In particular, note that the additions described in Sections 2.1 and 2.4 (i.e., name encoding and metadata) are backwards compatible with SBML Level 1. If a feature-set approach were used for Level 2, and if a model definition used only these two Level 2 extensions, the resulting model definition could in fact be read by an application that only understands SBML Level 1. This is not true for the current non-modular Level 2 proposal: without feature-set tags, an application would either have to assume that it could not handle the model (because the model says it is a Level 2 definition), or it would have to parse the whole model definition to evaluate whether it could handle it.

The present Level 2 proposal does not use this approach primarily for the following reason: our instinct is that a single cohesive SBML Level 2 definition, rather than a modular one, will be simpler to evaluate by the community and can be adopted (or rejected) more quickly.

Nevertheless, we propose that this issue (whether to have a singular Level 2 definition or a modular one) be left to the community to discuss and evaluate. This document proposes one a single (non-modular) definition for Level 2, but the definition could easily be turned into a version employing feature sets.

References

Cuellar, A. A., Nelson, M., and Hedley, W. (2002). The CellML metadata 1.0 specification working draft—16 January 2002. Available via the World Wide Web at `http://cellml.org/public/metadata/cellml_metadata_specification.html`.

Finney, A., Gor, V., Mjolsness, E., and Bolouri, H. (2002). Systems Biology Markup Language (SBML) Level 2 Proposal: Miscellaneous features.

Harbison, S. P. and Steele, G. L. (1995). *C: A Reference Manual*. Prentice-Hall.

Hucka, M., Finney, A., Sauro, H. M., and Bolouri, H. (2001). Systems Biology Markup Language (SBML) Level 1: Structures and facilities for basic model definitions. Available via the World Wide Web at `http://www.sbml.org/`.

Unicode Consortium (1996). *The Unicode Standard, Version 2.0*. Addison-Wesley Developers Press, Reading, Massachusetts.

Unicode Consortium (2002). Unicode home page. Available via the World Wide Web at `http://www.unicode.org/`.