Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions

Michael Hucka (Chair)	*California Institute of Technology, USA*
Stefan Hoops	*Virginia Bioinformatics Institute, USA*
Sarah M. Keating	*California Institute of Technology, USA*
Nicolas Le Novère	*EMBL European Bioinformatics Institute, UK*
Sven Sahle	*University of Heidelberg, Germany*
Darren J. Wilkinson	*Newcastle University, UK*

sbml-editors@sbml.org

SBML Level 2 Version 5

Release 1

NN-NNNN-2014

Corrections and other changes to this SBML language specification may appear over time. Notifications of new releases are broadcast on the mailing list sbml-announce@sbml.org

The latest release of the SBML Level 2 Version 5 specification is available at
http://sbml.org/specifications/sbml-level-2/version-5/

This release of the specification is available at
http://sbml.org/specifications/sbml-level-2/version-5/release-1/

The list of known issues in all releases of SBML Level 2 Version 5 is available at
http://sbml.org/specifications/sbml-level-2/version-5/errata/

The XML Schema for SBML Level 2 Version 5 is available at
http://sbml.org/xml-schemas/

Contents

1 Introduction

We present the **S**ystems **B**iology **M**arkup **L**anguage (SBML) Level 2 Version 5 Release 1, a model representation format for systems biology. SBML is oriented towards describing systems of biochemical reactions of the sort common in research on a number of topics, including cell signaling pathways, metabolic pathways, biochemical reactions, gene regulation, and many others. SBML is defined in a neutral fashion with respect to programming languages and software encoding; however, it is primarily oriented towards allowing models to be encoded using XML, the eXtensible Markup Language (Bosak and Bray, 1999; Bray et al., 2000). This document contains many examples of SBML models written in XML, as well as the text of an XML Schema (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000) that defines SBML Level 2 Version 5. A downloadable copy of the XML Schema and other related documents and software are also available from the SBML project web site, `http://sbml.org/`.

The SBML project is not an attempt to define a universal language for representing quantitative models. The rapidly evolving views of biological function, coupled with the vigorous rates at which new computational techniques and individual tools are being developed today, are incompatible with a one-size-fits-all idea of a universal language. A more realistic alternative is to acknowledge the diversity of approaches and methods being explored by different software tool developers, and seek a common intermediate format—a *lingua franca*—enabling communication of the most essential aspects of the models.

The definition of the model description language presented here does not specify *how* programs should communicate or read/write SBML. We assume that for a simulation program to communicate a model encoded in SBML, the program will have to translate its internal data structures to and from SBML, use a suitable transmission medium and protocol, etc., but these issues are outside of the scope of this document.

1.1 Developments, discussions, and notifications of updates

SBML has been, and continues to be, developed in collaboration with an international community of researchers and software developers. As in many projects, the primary mode of interaction between members is electronic mail. Discussions about SBML take place on the mailing list `sbml-discuss@caltech.edu`. The mailing list archives and a web browser-based interface to the list are available at `http://sbml.org/forums/`.

A low-volume, broadcast-only mailing list is available where notifications of revisions to the SBML specification, notices of votes on SBML technical issues, and other critical matters are announced. This list is `sbml-announce@caltech.edu` and anyone may subscribe to it freely. This list will never be used for advertising and its membership list will never be disclosed. *It is vitally important that all users of SBML stay informed about new releases and other developments by subscribing to this list*, even if they do not wish to participate in discussions on the `sbml-discuss@caltech.edu` list. Please visit the SBML project web site, `http://sbml.org/`, for information about how to subscribe to `sbml-announce@caltech.edu` as well as for access to the list archives.

In Section 8.1, we attempt to acknowledge as many contributors to SBML's development as we can, but as SBML evolves, it becomes increasingly difficult to detail the individual contributions on a project that has truly become an international community effort.

1.2 SBML Levels, Versions, and Releases

Major editions of SBML are termed *levels* and represent substantial changes to the composition and structure of the language. The edition of SBML defined in this document, SBML Level 2, represents an evolution of the language resulting from the practical experiences of many users and developers working with SBML Level 1 since since its introduction in the year 2001 (Hucka et al., 2001, 2003). All of the constructs of Level 1 can be mapped to Level 2. In addition, a subset of Level 2 constructs can be mapped to Level 1. However, the levels remain distinct; a valid SBML Level 1 document is not a valid SBML Level 2 document, and likewise, a valid SBML Level 2 document is not a valid SBML Level 1 document.

Minor revisions of SBML are termed *versions* and constitute changes within a Level to correct, adjust, and refine language features. The present document defines SBML Level 2 Version 5. In Section 1.4.1 below

explains how color is used in this document to indicate changes; a separate document provides a detailed listing of the changes between versions of SBML Level 2 as well as between SBML Level 2 Version 5and SBML Level 2 Version 3.

Specification documents inevitably require minor editorial changes as its users discover errors and ambiguities. As a practical reality, these discoveries occur over time. In the context of SBML, such problems are formally announced publicly as *errata* in a given specification document. Borrowing concepts from the World Wide Web Consortium (Jacobs, 2004), we define SBML errata as changes of the following types: (a) formatting changes that do not result in changes to textual content; (b) corrections that do not affect conformance of software implementing support for a given combination of SBML Level and Version; and (c) corrections that *may* affect such software conformance, but add no new language features. A change that affects conformance is one that either turns conforming data, processors, or other conforming software into non-conforming software, or turns non-conforming software into conforming software, or clears up an ambiguity or insufficiently documented part of the specification in such a way that software whose conformance was once unclear now becomes clearly conforming or non-conforming (Jacobs, 2004). In short, errata do not change the fundamental semantics or syntax of SBML; they clarify and disambiguate the specification and correct errors. (New syntax and semantics are only introduced in SBML Versions and Levels.) An electronic tracking system for reporting and monitoring such issues is available at `http://sbml.org/issue-tracker`.

SBML errata result in new *Releases* of the SBML specification. Each release is numbered with an integer, with the first release of the specification being called release number 1. Subsequent releases of an SBML specification document contain a section listing the accumulated errata reported and corrected since the first release. A complete list of the errata for SBML Level 2 Version 5 since the publication of Release 1 is also made publicly available at `http://sbml.org/specifications/sbml-level-2/version-5/errata/`. Announcements of errata, releases of the SBML specification and other major changes are made on the `sbml-announce@caltech.edu` mailing list.

1.3 Language features and backward compatibility

Some language features of previous SBML Levels and Versions have been either deprecated or removed entirely in SBML Level 2 Version 5. For the purposes of SBML specifications, the following are the definitions of *deprecated feature* and *removed feature*:

Removed language feature: A syntactic construct that was present in previous SBML Levels and/or Versions within a Level, and has been removed beginning with a specific SBML Level and Version. Models containing such constructs do not conform to the specification of that SBML Level and Version.

Deprecated language feature: A syntactic construct that was present in previous SBML Levels and/or Versions within a Level, and while still present in the language definition, has been identified as non-essential and planned for future removal. Beginning with the Level and Version in which a given feature is deprecated, software tools should not generate SBML models containing the deprecated feature; however, for backward compatibility, software tools reading SBML should support the feature until it is actually removed.

As a matter of SBML design philosophy, the preferred approach to removing features is by deprecating them if possible. Immediate removal of SBML features is not done unless serious problems have been discovered involving those features, and keeping them would create logical inconsistencies or extremely difficult-to-resolve problems. The deprecation or outright removal of features in a language, whether SBML or other, can have significant impact on backwards compatibility. Such changes are also inevitable over the course of a language's evolution. SBML must by necessity continue evolving through the experiences of its users and implementors. Eventually, some features will be deemed unhelpful despite the best intentions of the language editors to design a timeless language.

Throughout the SBML specification, removed and deprecated features are discussed in the text of the sections where the features previously appeared. Appendix G lists the changes and describes their motivations in more detail.

1.4 Document conventions

In this section, we describe the conventions we use in this specification document in an effort to communicate information more effectively and consistently.

1.4.1 Color conventions

Throughout this document, we use coloring to carry additional information for the benefit of those viewing the document on media that can display color:

- We use red color in text and figures to indicate changes between this version of the specification (SBML Level 2 Version 5 Release 1) and the *most recent previous version* of the specification (which, for the present case, is SBML Level 2 Version 4 Release 1). The changes may be either additions or deletions of text; in the case of deletions, entire sentences, paragraphs or sections are colored to indicate a change has occurred inside them.

- We use blue color in text to indicate a hyperlink from one point in this document to another. Clicking your computer's pointing device on blue-colored text will cause a jump to the section, figure, table or page to which the link refers. (Of course, this capability is only available when using electronic formats that support hyperlinking, such as PDF and HTML.)

1.4.2 Typographical conventions for names

The following typographical notations are used in this document to distinguish objects and data types from other kinds of entities:

AbstractClass: Abstract classes are classes that are never instantiated directly, but rather serve as parents of other 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 are also hyperlinked to their definitions in the specification. For example, in the PDF and HTML versions of this document, clicking on the word ***SBase*** will send the reader to the section containing the definition of this class.

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 the specification. For example, in the PDF and HTML versions of this document, clicking on the word **Species** will send the reader to the section containing the definition of this class.

`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, but unfortunately, XML Schema 1.0 does not follow any convention and primitive XML types may either start with a capital letter (e.g,. `ID`) or not (e.g., `double`).

1.4.3 UML notation

Previous specifications of SBML used a notation that was at one time (in the days of SBML Level 1) fairly close to UML, the Unified Modeling Language (Eriksson and Penker, 1998; Oestereich, 1999), though many details were omitted from the UML diagrams themselves. Over the years, the notation used in successive specifications of SBML grew increasingly less UML-like. Beginning with SBML Level 2 Version 3, we have completely overhauled the specification's use of UML and once again define the XML syntax of SBML using, as much as possible, proper and complete UML 1.0. We then systematically map this UML notation to XML, using XML Schema 1.0 (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000) to express the overall syntax of SBML. In the rest of this section, we summarize the UML notation used in this document and explain the few embellishments needed to support transformation to XML form. A complete Schema for SBML is given in Appendix A.

We see three main advantages to using UML as a basis for defining SBML data objects. First, compared to using other notations or a programming language, the UML visual representations are generally easier to grasp by readers who are not computer scientists. Second, the notation is implementation-neutral: the

objects can be encoded in any concrete implementation language—not just XML, but C, Java and other languages as well. Third, UML is a de facto industry standard that is documented in many resources. Readers are therefore more likely to be familiar with it than other notations.

Object class definitions

Object classes in UML diagrams are drawn as simple tripartite boxes, as shown in Figure 1 (left). UML allows for operations as well as data attributes to be defined, but SBML only uses data attributes, so all SBML class diagrams use only the top two portions of a UML class box (see the right-hand diagram of Figure 1).

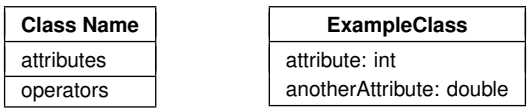


Figure 1: *(Left) The general form of a UML class diagram. (Right) Example of a class diagram of the sort seen in SBML. SBML classes never use operators, so SBML class diagrams only show the top two parts.*

As mentioned above, the names of ordinary (concrete) classes begin with a capital letter and are printed in an upright, bold, sans-serif typeface. The names of attributes begin with a lower-case letter and generally use a mixed case (sometimes called "camel case") style when the name consists of multiple words. Attributes and their data types appear in the part below the class name, with one attribute defined per line. The colon character on each line separates the name of the attribute (on the left) from the type of data that it stores (on the right). The subset of data types permitted for SBML attributes is given in Section 3.1.

In the right-hand diagram of Figure 1, the symbols `attribute` and `anotherAttribute` represent attributes of the object class **ExampleClass**. The data type of `attribute` is `int`, and the data type of `anotherAttribute` is `double`. In the scheme used by SBML for translating UML to XML, object attributes map directly to XML attributes. Thus, in XML, **ExampleClass** would yield an element of the form `<element attribute="42" anotherAttribute="10.0">`.

Notice that the element name is not `<ExampleClass ...>`. Somewhat paradoxically, the name of the element is *not* the name of the UML class defining its structure. The reason for this may be subtle at first, but quickly becomes obvious: object classes define the form of an object's *content*, but a class definition by itself does not define the *label* or symbol referring to an instance of that content. It is this label that becomes the name of the XML element. In XML, this symbol is most naturally equated with an element name. This point will hopefully become more clear with additional examples below.

Subelements

We use UML composite aggregation to indicate a class object can have other class objects as parts. Such containment hierarchies map directly to element-subelement relationships in XML. Figure 2 gives an example.

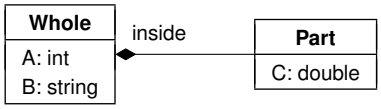


Figure 2: *Example illustrating composite aggregation: the definition of one class of objects employing another class of objects in a part-whole relationship. In this particular example, an instance of a* ***Whole*** *class object must contain exactly one instance of a* ***Part*** *class object, and the symbol referring to the* ***Part*** *class object is* `inside`*. In XML, this symbol becomes the name of a subelement and the content of the subelement follows the definition of* ***Part****.*

The line with the black diamond indicates composite aggregation, with the diamond located on the "container" side and the other end located at the object class being contained. The label on the line is the symbol used to refer to instances of the contained object, which in XML, maps directly to the name of an

XML element. The class pointed to by the aggregation relationship (**Part** in Figure 2) defines the *contents* of that element. Thus, if we are told that some element named `barney` is of class `Whole`, the following is an example XML fragment consistent with the class definition of Figure 2:

```
<barney A="110" B="some string">
    <inside C="444.4">
</barney>
```

Sometimes numbers are placed above the line near the "contained" side of an aggregation to indicate how many instances can be contained. The common cases in SBML are the following: `[0..*]` to signify a list containing zero or more; `[1..*]` to signify a list containing at least one; and `[0..1]` to signify exactly zero or one. The absence of a numerical label means "exactly 1". This notation appears throughout this specification document.

Inheritance

Classes can inherit properties from other classes. Since SBML only uses data attributes and not operations, inheritance in SBML simply involves data attributes from a parent class being inherited by child classes. Inheritance is indicated by a line between two classes, with an open triangle next to the parent class; Figure 3 illustrates this. In this example, the instances of object class **Child** would have not only attributes `C` and `D`, but also attributes `A` and `B`. All of these attributes would be required (not optional) on instances of class **Child** because they are mandatory on both **Parent** and **Child**.

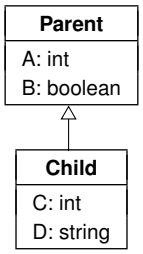


Figure 3: *Inheritance.*

Additional notations for XML purposes

Not everything is easily expressed in plain UML. For example, it is often necessary to indicate some constraints placed on the values of an attribute. In computer programming uses of UML, such constraints are often expressed using Object Constraint Language (OCL), but since we are most interested in the XML rendition of SBML, in this specification we use XML Schema 1.0 (when possible) as the language for expressing value constraints. Constraints on the values of attributes are written as expressions surrounded by braces ({ }) after the data type declaration, as in the example of Figure 4.

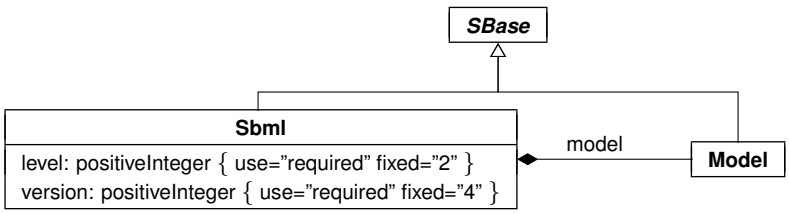


Figure 4: *A more complex example definition drawing on the concepts introduced so far in this section. Both **Sbml** and **Model** are derived from **SBase**; further, **Sbml** contains a single **Model** object named* `model`*. Note the constraints on the values of the attributes in **Sbml**; they are enclosed in braces and written in XML Schema language. The particular constraints here state that both the* `level` *and* `version` *attributes must be present, and that the values are fixed as indicated.*

In other situations, when something cannot be concisely expressed using a few words of XML Schema, we write constraints using English language descriptions surrounded by braces ({ }). To help distinguish these from literal XML Schema, we set the English text in a slanted typeface. The text accompanying all SBML component definitions provides explanations of the constraints and any other conditions applicable to the use of the components.

Compatibility issues and warnings

One important and confusing point that goes against the grain of XML must be highlighted: the order in which subelements appear within SBML elements *is* significant and *must* follow the order given in the corresponding object definition. This ordering is also difficult to express in plain UML, so we resort to using the approach of stating ordering requirements as constraints written in English and (again) enclosed in braces ({ }). Figure 8 on page 14 gives an example of this.

The ordering restriction also holds true when a subclass inherits attributes and elements from a base class: the base class attributes and elements must occur before those introduced by the subclass.

This ordering constraint stems from aspects of XML Schema beyond our control (specifically, the need to use XML Schema's `sequence` construct to define the object classes). It is an occasional source of software compatibility problems, because validating XML parsers will generate errors if the ordering within an XML element does not correspond to the SBML object class definition.

2 Overview of SBML

The following is an example of a simple network of biochemical reactions that can be represented in SBML:

$$S_1 \xrightarrow{k_1[S_1]/([S_1]+k_2)} S_2$$

$$S_2 \xrightarrow{k_3[S_2]} S_3 + S_4$$

In this particular set of chemical equations above, the symbols in square brackets (e.g., "$[S_1]$") represent concentrations of molecular species, the arrows represent reactions, and the formulas above the arrows represent the rates at which the reactions take place. (And while this example uses concentrations, it could equally have used other measures such as molecular counts.) Broken down into its constituents, this model contains a number of components: reactant species, product species, reactions, reaction rates, and parameters in the rate expressions. To analyze or simulate this network, additional components must be made explicit, including compartments for the species, and units on the various quantities.

SBML allows models of arbitrary complexity to be represented. Each type of component in a model is described using a specific type of data object that organizes the relevant information. The top level of an SBML model definition consists of lists of these components, with every list being optional:

beginning of model definition	
list of function definitions (optional)	(Section 4.3)
list of unit definitions (optional)	(Section 4.4)
list of compartment types (optional)	(Section 4.5)
list of species types (optional)	(Section 4.6)
list of compartments (optional)	(Section 4.7)
list of species (optional)	(Section 4.8)
list of parameters (optional)	(Section 4.9)
list of initial assignments (optional)	(Section 4.10)
list of rules (optional)	(Section 4.11)
list of constraints (optional)	(Section 4.12)
list of reactions (optional)	(Section 4.13)
list of events (optional)	(Section 4.14)
end of model definition	

The meaning of each component is as follows:

Function definition: A named mathematical function that may be used throughout the rest of a model.

Unit definition: A named definition of a new unit of measurement, or a redefinition of an SBML predefined unit. Named units can be used in the expression of quantities in a model.

Compartment Type: A type of location where reacting entities such as chemical substances may be located.

Species type: A type of entity that can participate in reactions. Typical examples of species types include ions such as Ca^{2+}, molecules such as glucose or ATP, and more.

Compartment: A well-stirred container of a particular type and finite size where species may be located. A model may contain multiple compartments of the same compartment type. Every species in a model must be located in a compartment.

Species: A pool of entities of the same *species type* located in a specific *compartment*.

Parameter: A quantity with a symbolic name. In SBML, the term *parameter* is used in a generic sense to refer to named quantities regardless of whether they are constants or variables in a model. SBML Level 2 provides the ability to define parameters that are global to a model as well as parameters that are local to a single reaction.

Initial Assignment: A mathematical expression used to determine the initial conditions of a model. This type of object can only be used to define how the value of a variable can be calculated from other values and variables at the start of simulated time.

Rule: A mathematical expression added to the set of equations constructed based on the reactions defined in a model. Rules can be used to define how a variable's value can be calculated from other variables, or used to define the rate of change of a variable. The set of rules in a model can be used with the reaction rate equations to determine the behavior of the model with respect to time. The set of rules constrains the model for the entire duration of simulated time.

Constraint: A means of detecting out-of-bounds conditions during a dynamical simulation and optionally issuing diagnostic messages. Constraints are defined by an arbitrary mathematical expression computing a true/false value from model variables, parameters and constants. An SBML constraint applies at all instants of simulated time; however, the set of constraints in model should not be used to *determine* the behavior of the model with respect to time.

Reaction: A statement describing some transformation, transport or binding process that can change the amount of one or more species. For example, a reaction may describe how certain entities (reactants) are transformed into certain other entities (products). Reactions have associated kinetic rate expressions describing how quickly they take place.

Event: A statement describing an instantaneous, discontinuous change in a set of variables of any type (species quantity, compartment size or parameter value) when a triggering condition is satisfied.

A software package can read an SBML model description and translate it into its own internal format for model analysis. For example, a package might provide the ability to simulate the model by constructing differential equations representing the network and then perform numerical time integration on the equations to explore the model's dynamic behavior. By supporting SBML as an input and output format, different software tools can all operate on an identical external representation of a model, removing opportunities for errors in translation and assuring a common starting point for analyses and simulations.

3 Preliminary definitions and principles

This section covers certain concepts and constructs that are used repeatedly in the rest of SBML Level 2.

3.1 Primitive data types

Most primitive types in SBML are taken from the data types defined in XML Schema 1.0 (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000). A few other primitive types are defined by SBML itself. What follows is a summary of the XML Schema types and the definitions of the SBML-specific types. Note that while we have tried to provide accurate and complete summaries of the XML Schema types, the following should not be taken to be normative definitions of these types. Readers should consult the XML Schema 1.0 specification for the normative definitions of the XML types used by SBML.

3.1.1 *Type* `string`

The XML Schema 1.0 type `string` is used to represent finite-length strings of characters. The characters permitted to appear in XML Schema `string` include all Unicode characters (Unicode Consortium, 1996) except for two delimiter characters, 0xFFFE and 0xFFFF (Biron and Malhotra, 2000). In addition, the following quoting rules specified by XML for character data (Bray et al., 2000) must be obeyed:

- The ampersand (`&`) character must be escaped using the entity `&`.
- The apostrophe (`'`) and quotation mark (`"`) characters must be escaped using the entities `&apos;` and `&quot;`, respectively, when those characters are used to delimit a string attribute value.

Other XML built-in character or entity references, e.g., `<` and `&x1A;`, are permitted in strings.

3.1.2 *Type* `boolean`

The XML Schema 1.0 type `boolean` is used as the data type for SBML object attributes that represent binary true/false values. XML Schema 1.0 defines the possible literal values of `boolean` as the following: "`true`", "`false`", "`1`", and "`0`". The value "`1`" maps to "`true`" and the value "`0`" maps to "`false`".

Note that there is a discrepancy between the value spaces of type `boolean` as defined by XML Schema 1.0 and MathML: the latter uses only "`true`" and "`false`" to represent boolean values and "`0`" and "`1`" are interpreted as numbers. Software tools should take care to not to use "`0`" and "`1`" as boolean values in MathML expressions. See further discussion in Section 3.4.4.

3.1.3 *Type* `int`

The XML Schema 1.0 type `int` is used to represent decimal integer numbers in SBML. The literal representation of an `int` is a finite-length sequence of decimal digit characters with an optional leading sign ("+" or "−"). If the sign is omitted, "+" is assumed. The value space of `int` is the same as a standard 32-bit signed integer in programming languages such as C, i.e., 2147483647 to −2147483648.

3.1.4 *Type* `positiveInteger`

The XML Schema 1.0 type `positiveInteger` is used to represent nonzero, nonnegative, decimal integers: i.e., 1, 2, 3, ... The literal representation of an integer is a finite-length sequence of decimal digit characters, optionally preceded by a positive sign ("+"). There is no restriction on the absolute size of `positiveInteger` values in XML Schema; however, the only situations where this type is used in SBML involve very low-numbered integers. Consequently, applications may safely treat `positiveInteger` as unsigned 32-bit integers.

3.1.5 *Type* `double`

The XML Schema 1.0 type `double` is the data type of floating point numerical quantities in SBML. It is restricted to IEEE double-precision 64-bit floating point type IEEE 754-1985. The value space of `double` consists of (a) the numerical values $m \times 2^x$, where m is an integer whose absolute value is less than 2^{53},

and x is an integer between -1075 and 970, inclusive, (b) the special value positive infinity (`INF`), (c) the special value negative infinity (`-INF`), and (d) the special value not-a-number (`NaN`). The order relation on the values is the following: $x < y$ if and only if $y - x$ is positive for values of x and y in the value space of `double`. Positive infinity is greater than all other values other than `NaN`. `NaN` is equal to itself but is neither greater nor less than any other value in the value space. (Software implementors should consult the XML Schema 1.0 definition of `double` for additional details about equality and relationships to IEEE 754-1985.)

The general form of `double` numbers is "x`e`y", where x is a decimal number (the mantissa), "`e`" is a separator character, and y is an exponent; the meaning of this is "x multiplied by 10 raised to the power of y", i.e., $x \times 10^y$. More precisely, a `double` value consists of a mantissa with an optional leading sign ("+" or "-"), optionally followed by the character `E` or `e` followed by an integer (the exponent). The mantissa must be a decimal number: an integer optionally followed by a period (`.`) optionally followed by another integer. If the leading sign is omitted, "+" is assumed. An omitted `E` or `e` and exponent means that a value of 0 is assumed for the exponent. If the `E` or `e` is present, it must be followed by an integer or an error results. The integer acting as an exponent must consist of a decimal number optionally preceded by a leading sign ("+" or "-"). If the sign is omitted, "+" is assumed. The following are examples of legal literal `double` values:

`-1E4`, `+4`, `234.234e3`, `6.02E-23`, `0.3e+11`, `2`, `0`, `-0`, `INF`, `-INF`, `NaN`

As described in Section 3.4, SBML uses a subset of the MathML 2.0 standard (W3C, 2000b) for expressing mathematical formulas in XML. This is done by stipulating that the MathML language be used whenever a mathematical formula must be written into an SBML model. Doing this, however, requires facing two problems: first, the syntax of numbers in scientific notation ("e-notation") is different in MathML from that just described for `double`, and second, the value space of integers and floating-point numbers in MathML is not defined in the same way as in XML Schema 1.0. We elaborate on these issues in Section 3.4.2; here we summarize the solution taken in SBML. First, within MathML, the mantissa and exponent of numbers in "e-notation" format must be separated by one `<sep/>` element. This leads to numbers of the form `<cn type="e-notation"> 2 <sep/> -5 </cn>`. Second, SBML stipulates that the representation of numbers in MathML expressions obey the same restrictions on values as defined for types `double` and `int` (Section 3.1.3).

3.1.6 *Type* `ID`

The XML Schema 1.0 type `ID` is identical to the XML 1.0 type `ID`. The literal representation of this type consists of strings of characters restricted as summarized in Figure 5.

```
NameChar   ::=  letter | digit | '.'  | '-' | '_' | ':'  | CombiningChar | Extender
ID         ::=  ( letter | '_' | ':'  )  NameChar*
```

Figure 5: *Type* `ID` *expressed in the variant of BNF used by the XML 1.0 specification (Bray et al., 2004). The characters* `(` *and* `)` *are used for grouping, the character* `*` *indicates "zero or more times", and the character* `|` *indicates "or". The production* `letter` *consists of the basic upper and lower case alphabetic characters of the Latin alphabet along with a large number of related characters defined by Unicode 2.0; similarly, the production* `digit` *consists of the numerals* `0..9` *along with related Unicode 2.0 characters. The* `CombiningChar` *production is a list of characters that add such things as accents to the preceding character. (For example, the Unicode character* `#x030A` *when combined with 'a' produces 'å'.) The* `Extender` *production is a list of characters that extend the shape of the preceding character. Please consult the XML 1.0 specification (Bray et al., 2004) for the complete definitions of* `letter`, `digit`, `CombiningChar`, *and* `Extender`.

In SBML, type `ID` is the data type of the `metaid` attribute on ***SBase***, described in Section 3.2. An important aspect of `ID` is the XML requirement that a given value of `ID` must be unique throughout an XML document. All data values of type `ID` are considered to reside in a single common global namespace spanning the entire XML document, regardless of the attribute where type `ID` is used and regardless of the level of nesting of the objects (or XML elements).

3.1.7 *Type* `SId`

The type `SId` is the type of the `id` attribute found on the majority of SBML components. `SId` is a data type derived from the basic XML type `string`, but with restrictions about the characters permitted and the sequences in which those characters may appear. The definition is shown in Figure 6 on the next page.

```
letter   ::=  'a'..'z','A'..'Z'
digit    ::=  '0'..'9'
idChar   ::=  letter | digit | '_'
SId      ::=  ( letter | '_' ) idChar*
```

Figure 6: *The definition of the type* `SId`. *(Please see the caption of Figure 5 for an explanation of the notation.)*

The equality of `SId` values is determined by an exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner. This applies to all uses of `SId`.

The `SId` is purposefully not derived from the XML `ID` type (Section 3.1.6). Using XML's `ID` would force all SBML identifiers to exist in a single global namespace, which would affect not only the form of local parameter definitions but also future SBML extensions for supporting model/submodel composition. Further, the use of the `ID` type for SBML identifiers would have limited utility because MathML 2.0 `ci` elements are not of the type `IDREF` (see Section 3.4). Since the `IDREF`/`ID` linkage cannot be exploited in MathML constructs, the utility of the XML `ID` type is greatly reduced. Finally, unlike `ID`, `SId` does not include Unicode character codes; the identifiers are plain text.

3.1.8 *Type* `UnitSId`

The type `UnitSId` is derived from `SId` (Section 3.1.7) and has identical syntax. The `UnitSId` type is used as the data type for the identifiers of units (Section 4.4.1) and for references to unit identifiers in SBML objects. The purpose of having a separate data type for such identifiers is enable the space of possible unit identifier values to be separated from the space of all other identifier values in SBML. The equality of `UnitSId` values is determined by an exact character sequence match; i.e., comparisons of these identifiers must be performed in a case-sensitive manner.

A number of reserved symbols are defined in the space of values of `UnitSId`. These reserved symbols are the list of base unit names defined in Table 1 on page 38, and the SBML predefined units "`substance`", "`volume`", "`area`", "`length`", and "`time`" listed in Table 2 on page 42. These symbols and their use is described in Section 4.4.

3.1.9 *Type* `SBOTerm`

The type `SBOTerm` is used as the data type of the attribute `sboTerm` on ***SBase***. The type consists of strings of characters matching the restricted pattern described in Figure 7.

```
digit     ::=  '0'..'9'
SBOTerm   ::=  'SBO:' digit digit digit digit digit digit digit
```

Figure 7: *The definition of* `SBOTerm`. *The* `SBOTerm` *type consists of strings beginning with* `SBO:` *and followed by seven decimal digits. (Please see the caption of Figure 5 for an explanation of the notation.)*

Examples of valid string values of type `SBOTerm` are "`SBO:0000014`" and "`SBO:0003204`". These values are meant to be the identifiers of terms from an ontology whose vocabulary describes entities and processes in computational models. Section 5 provides more information about the ontology and principles for the use of these terms in SBML models.

3.2 Type *SBase*

Nearly every object composing an SBML Level 2 model definition has a specific data type that is derived directly or indirectly from a single abstract type called ***SBase***. In addition to serving as the parent class for most other classes of objects in SBML, this base type is designed to allow a modeler or a software package to attach arbitrary information to each major element or list in an SBML model. The definition of ***SBase*** is presented in Figure 8 on the following page.

SBase contains two attributes and two subelements, all of which are optional: `metaid`, `sboTerm`, `notes` and `annotation`. These are discussed separately in the following subsections.

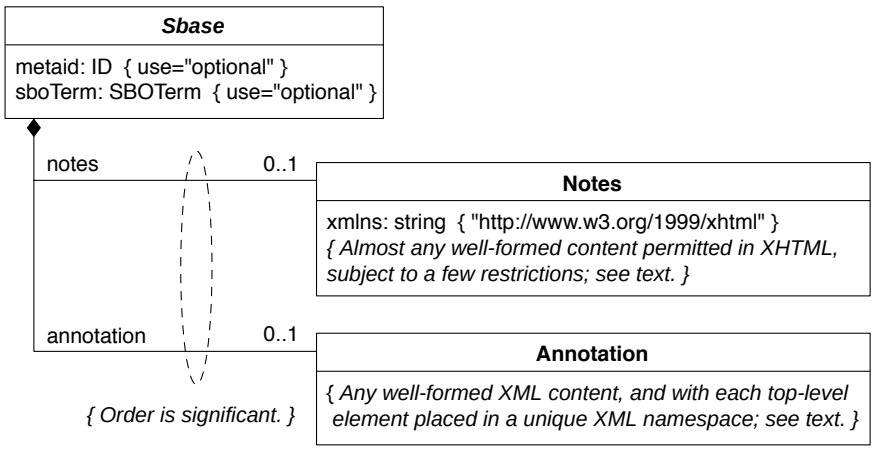


Figure 8: *The definition of* ***SBase****. Please refer to Section 1.4 for a summary of the UML notation used here. Note that the order of appearance of subelements* `notes` *and* `annotation` *is significant in instances of objects derived from* ***SBase****:* `notes` *must always come before* `annotation`*. (This requirement arises from XML Schema 1.0.)*

3.2.1 The `metaid` *attribute*

The `metaid` attribute is present for supporting metadata annotations using RDF (Resource Description Format; Lassila and Swick, 1999). It has a data type of XML `ID` (the XML identifier type; see Section 3.1.6), which means each `metaid` value must be globally unique within an SBML file. The `metaid` value serves to identify a model component for purposes such as referencing that component from metadata placed within `annotation` elements (see Section 3.2.4). Such metadata can use RDF `description` elements, in which an RDF attribute called "`rdf:about`" points to the `metaid` identifier of an object defined in the SBML model. This topic is discussed in greater detail in Section 6.

3.2.2 The `sboTerm` *attribute*

The attribute called `sboTerm` is provided on ***SBase*** to support the use of the Systems Biology Ontology (SBO; see Section 5). When a value is given to this attribute, it must conform to the data type `SBOTerm` (Sections 3.1.9). SBO terms are a type of optional annotation, and each different class of SBML object derived from ***SBase*** imposes its own requirements about the values permitted for `sboTerm`. Specific details on the permitted values are provided with the definitions of SBML classes throughout this specification document, and a broader discussion is provided in Section 5.

3.2.3 The `notes` *element*

The element `notes` in ***SBase*** is a container for XHTML 1.0 (Pemberton et al., 2002) content. It is intended to serve as a place for storing optional information intended to be seen by humans. An example use of the `notes` element would be to contain formatted user comments about the model element in which the `notes` element is enclosed. Every object derived directly or indirectly from type ***SBase*** can have a separate value for `notes`, allowing users considerable freedom when adding comments to their models.

XHTML 1.0 is simply a formulation of HTML 4 in XML 1.0. This means the full power of HTML formatting is available for use in `notes` content. The intention behind requiring XHTML (rather than, for example, plain HTML or plain text) for `notes` content is to balance several conflicting goals: (1) choosing a format for notes that is compatible with the XML form of SBML (plain HTML would not be); (2) supporting an international formatting standard so that users have more control over the appearance of notes and can predict to some degree how their notes will be displayed in different tools and environments (which argues against using plain-text notes); and (3) achieving these goals using an approach that is hopefully easy enough for software developers to support using off-the-shelf programming libraries. It is worth noting in passing that the requirement for XHTML does not *prevent* users from entering plain-text content with simple space/tab/newline formatting: it merely requires using the standard `<pre>...</pre>` element of (X)HTML.

Modern libraries for displaying and editing (X)HTML content are commonly available in contemporary software programming environments, and software developers may wish to avail themselves of these facilities rather than implementing their own XHTML support systems.

XML namespace requirements for `notes`

The XML content of **notes** elements must declare the use of the XHTML XML namespace. This can be done in multiple ways. One way is to place a namespace declaration for the appropriate namespace URI (which is `http://www.w3.org/1999/xhtml`) on the top-level **Sbml** object (see Section 4.1) and then reference the namespace in the **notes** content using a prefix. The following example illustrates this approach:

```
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5"
      xmlns:xhtml="http://www.w3.org/1999/xhtml">
  ...
  <notes>
    <xhtml:body>
      <xhtml:center><xhtml:h2>A Simple Mitotic Oscillator</xhtml:h2></xhtml:center>
      <xhtml:p>A minimal cascade model for the mitotic oscillator
      involving cyclin and cdc2 kinase</xhtml:p>
    </xhtml:body>
  </notes>
  ...
```

Another approach is to declare the XHTML namespace within the **notes** content itself, as in the following example:

```
...
<notes>
  <body xmlns="http://www.w3.org/1999/xhtml">

    <center><h2>A Simple Mitotic Oscillator</h2></center>

    <p>A minimal cascade model for the mitotic oscillator
    involving cyclin and cdc2 kinase</p>

  </body>
</notes>
...
```

The `xmlns="http://www.w3.org/1999/xhtml"` declaration on **body** as shown above changes the default XML namespace within it, such that all of its content is by default in the XHTML namespace. This is a particularly convenient approach because it obviates the need to prefix every element with a namespace prefix (i.e., **xhtml:** in the previous case). Other approaches are also possible.

The content of `notes`

SBML does not require the content of **notes** to be any particular XHTML element; the content can be almost any well-formed XHTML content. There are only two simple restrictions. The first restriction comes from the requirements of XML: the **notes** element must not contain an XML declaration nor a DOCTYPE declaration. That is, **notes** must *not* contain

```
<?xml version="1.0" encoding="UTF-8"?>
```

nor (where the following is only one specific example of a DOCTYPE declaration)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

The second restriction is intended to balance freedom of content with the complexity of implementing software that can interpret the content. The content of **notes** in SBML can consist only of the following possibilities:

1. A complete XHTML document (minus the XML and DOCTYPE declarations, of course), that is, XHTML content beginning with the **html** tag. The following is an example skeleton:

```
<notes>
    <html xmlns="http://www.w3.org/1999/xhtml">
      ...
    </html>
</notes>
```

2. The `body` element from an XHTML document. The following is an example skeleton:

```
<notes>
    <body xmlns="http://www.w3.org/1999/xhtml">
      ...
    </body>
</notes>
```

3. Any XHTML content that would be permitted within a `body` element. If this consists of multiple elements, each one must declare the XML namespace separately. The following is an example fragment:

```
<notes>
    <p xmlns="http://www.w3.org/1999/xhtml">
      ...
    </p>
    <p xmlns="http://www.w3.org/1999/xhtml">
      ...
    </p>
</notes>
```

Another way to summarize the restrictions above is simply to say that the content of an SBML `notes` element can be only be a complete `html` element, a `body` element, or whatever is permitted inside a `body` element. In practice, this does not limit in any meaningful way what can be placed inside a `notes` element; for example, if an application or modeler wants to put a complete XHTML page, including a `head` element, it can be done by putting in everything starting with the `html` container. However, the restrictions above do make it somewhat simpler to write software that can read and write the `notes` content. Appendix F describes one possible approach to doing just that.

3.2.4 *The* `annotation` *element*

Whereas the `notes` element described above is a container for content to be shown directly to humans, the `annotation` element is a container for optional software-generated content *not* meant to be shown to humans. Every object derived from ***SBase*** can have its own value for `annotation`. The element's content type is XML type `any`, allowing essentially arbitrary well-formed XML data content. SBML places only a few restrictions on the organization of the content; these are intended to help software tools read and write the data as well as help reduce conflicts between annotations added by different tools.

The use of XML namespaces in `annotation`

At the outset, software developers should keep in mind that multiple software tools may attempt to read and write annotation content. To reduce the potential for collisions between annotations written by different applications, SBML Level 2 Version 5 stipulates that tools must use XML namespaces (Bray et al., 1999) to specify the intended vocabulary of every annotation. The application's developers must choose a URI (*Universal Resource Identifier*; Harold and Means 2001; W3C 2000a) reference that uniquely identifies the vocabulary the application will use, and a prefix string for the annotations. Here is an example. Suppose an application uses the URI `http://www.mysim.org/ns` and the prefix `mysim` when writing annotations related to screen layout. The content of an annotation might look like the following:

```
<annotation>
    <mysim:nodecolors xmlns:mysim="http://www.mysim.org/ns"
        mysim:bgcolor="green" mysim:fgcolor="white"/>
</annotation>
```

In this particularly simple example, the content consists of a single XML element (`nodecolors`) with two attributes (`bgcolor`, `fgcolor`), all of which are prefixed by the string `mysim`. (Presumably this particular content would have meaning to the hypothetical application in question.) The content in this particular

example is small, but it should be clear that there could easily have been an arbitrarily large amount of data placed inside the `mysim:nodecolors` element.

The key point of the example above is that application-specific annotation data is entirely contained inside a single *top-level element* within the SBML `annotation` container. SBML Level 2 Version 4 places the following restrictions on annotations:

- Within a given SBML `annotation` element, there can only be one top-level element using a given namespace. An annotation element can contain multiple top-level elements but each must be in a different namespace.
- No top-level element in an `annotation` may use an SBML XML namespace, either explicitly by referencing one of the SBML XML namespace URIs or implicitly by failing to specify any namespace on the annotation. As of SBML Level 2 Version 5, the defined SBML namespaces are the following URIs:
  - `http://www.sbml.org/sbml/level1`
  - `http://www.sbml.org/sbml/level2`
  - `http://www.sbml.org/sbml/level2/version2`
  - `http://www.sbml.org/sbml/level2/version3`
  - `http://www.sbml.org/sbml/level2/version4`
  - `http://www.sbml.org/sbml/level2/version5`
  - `http://www.sbml.org/sbml/level3/version1/core`
  - `http://www.sbml.org/sbml/level3/version2/core`
- The ordering of top-level elements within a given `annotation` element is *not* significant. An application should not expect that its annotation content appears first in the `annotation` element, nor in any other particular location. Moreover, the ordering of top-level annotation elements may be changed by different applications as they read and write the same SBML file.

The use of XML namespaces in this manner is intended to improve the ability of multiple applications to place annotations on SBML model elements with reduced risks of interference or name collisions. Annotations stored by different simulation packages can therefore coexist in the same model definition. The rules governing the content of `annotation` elements are designed to enable applications to easily add, change, and remove their annotations from SBML elements while simultaneously preserving annotations inserted by other applications when mapping SBML from input to output.

As a further simplification for developers of software and to improve software interoperability, applications are only required to preserve other annotations (i.e., annotations they do not recognize) when those annotations are self-contained entirely within `annotation`, complete with namespace declarations. The following is an example:

```
<annotation>
    <topLevelElement xmlns="URI">
        ... content in the namespace identified by "URI"...
    </topLevelElement>
</annotation>
```

Some more examples hopefully will make these points more clear. The next example is invalid because it contains a top-level element in the SBML XML namespace—this happens because no namespace is declared for the `<cytoplasm>` element, which means by default it falls into the enclosing SBML namespace:

```
<annotation>
    <cytoplasm/>
</annotation>
```

The following example is also invalid, this time because it contains two top-level elements using the same XML namespace. Note that it does not matter that these are two different top-level elements (`<nodecolors>` and `<textcolors>`); what matters is that these separate elements are both in the same namespace rather than having been collected and placed inside one overall container element for that namespace.

```
<annotation>
    <mysim:nodecolors xmlns:mysim="http://www.mysim.org/ns"
        mysim:bgcolor="green" mysim:fgcolor="white"/>
    <mysim:textcolors xmlns:mysim="http://www.mysim.org/ns"
        mysim:bgcolor="green" mysim:fgcolor="white"/>
</annotation>
```

On the other hand, the following example is valid:

```
<annotation>
    <mysim:geometry xmlns:mysim="http://www.mysim.org/ns"
            mysim:bgcolor="green" mysim:fgcolor="white">
        <graph:node xmlns:graph="http://www.graph.org/ns"
            graph:x="4" graph:y="5" />
    </mysim:geometry>
    <othersim:icon xmlns:othersim="http://www.othersim.com/">
        WS2002
    </othersim:icon>
</annotation>
```

For completeness, we note that annotations can legally be empty:

```
<annotation />
```

It is worth keeping in mind that although XML namespace names must be URIs, they are (like all XML namespace names) *not required* to be directly usable in the sense of identifying an actual, retrieval document or resource on the Internet (Bray et al., 1999). URIs such as `http://www.mysim.org/` may appear as though they are (e.g.,) Internet addresses, but there are not the same thing. This style of URI strings, using a domain name and other parts, is only a simple and commonly-used way of creating a unique name string.

Finally, note that the namespaces being referred to here are XML namespaces specifically in the context of the `annotation` element on ***SBase***. The namespace issue here is unrelated to the namespaces discussed in Section 3.3.1 in the context of component identifiers in SBML.

Content of annotations and implications for software tools

The `annotation` element in the definition of ***SBase*** exists in order that software developers may attach optional application-specific data to the elements in an SBML model. However, it is important that this facility not be misused. In particular, it is *critical* that data essential to a model definition or that can be encoded in existing SBML elements is *not* stored in `annotation`. Parameter values, functional dependencies between model elements, etc., should not be recorded as annotations. It is crucial to keep in mind the fact that data placed in annotations can be freely ignored by software applications. If such data affects the interpretation of a model, then software interoperability is greatly impeded.

Here are examples of the kinds of data that may be appropriately stored in `annotation`: (a) information about the graphical layout of model components; (b) application-specific processing instructions that do not change the essential meaning of a model; (c) identification information for cross-referencing components in a model with items in a data resource such as a database; and (d) information about the model that cannot be readily encoded in existing SBML elements.

Standardized format for certain classes of annotations

For case (c) above (i.e., cross-references between model components and data resources), SBML Level 2 Version 5 recommends a standard format for use within `annotation` elements. It should be used in preference to proprietary syntaxes to maximize the likelihood that multiple software tools will converge on the same syntax for this kind of information. The recommended scheme is described in Section 6.

3.3 The `id` and `name` attributes on SBML components

As will become apparent below, most objects in SBML include two common attributes: `id` and `name`. These attributes are not defined on ***SBase*** (as explained in Section 3.3.3 below), but where they do appear, the common rules of usage described below apply.

3.3.1 The `id` attribute and identifier scoping

The `id` attribute is mandatory on most objects in SBML. It is used to identify a component within the model definition. Other SBML objects can refer to the component using this identifier. The data type of `id` is always either `Sid` (Section 3.1.7) or `UnitSId` (Section 3.1.8), depending on the object in question.

A model can contain a large number of components representing different parts. This leads to a problem in deciding the scope of an identifier: in what contexts does a given identifier X represent the same thing? The approaches used in existing simulation packages tend to fall into two categories which we may call global and local. The *global* approach places all identifiers into a single global space of identifiers, so that an identifier X represents the same thing wherever it appears in a given model definition. The *local* approach places symbols in separate identifier namespaces, depending on the context, where the context may be, for example, individual reaction rate expressions. The latter approach means that a user may use the same identifier X in different rate expressions and have each instance represent a different quantity.

The fact that different simulation programs may use different rules for identifier resolution poses a problem for the exchange of models between simulation tools. Without careful consideration, a model written out in SBML format by one program may be misinterpreted by another program. SBML Level 2 must therefore include a specific set of rules for treating identifiers and their scopes.

The scoping rules in SBML Level 2 are relatively straightforward and are intended to avoid this problem with a minimum of requirements on the implementation of software tools:

- The identifier (i.e., the value of the attribute `id`) of every **FunctionDefinition**, **CompartmentType**, **SpeciesType**, **Compartment**, **Species**, **Parameter**, **Reaction**, **SpeciesReference**, **ModifierSpeciesReference**, **Event**, and **Model**, must be unique across the set of all such identifiers in the model. This means, for example, that a reaction and a species definition cannot both have the same identifier.
- The identifier of every **UnitDefinition** must be unique across the set of all such identifiers in the model plus the set of base unit definitions in Table 1 on page 38. However, unit identifiers live in a separate space of identifiers from other identifiers in the model, by virtue of the fact that the data type of unit identifiers is `UnitSId` (Section 3.1.8) and not `SId`.
- Each **Reaction** instance (see Section 4.13) establishes a separate private local space for local **Parameter** identifiers. Within the definition of that reaction, local parameter identifiers override (shadow) identical identifiers (whether those identifiers refer to parameters, species or compartments) outside of that reaction. Of course, the corollary of this is that local parameters inside a **Reaction** object instance are not visible to other objects outside of that reaction.

The set of rules above can enable software packages using either local or global identifier spaces for parameters to exchange SBML model definitions. Software systems using local identifiers for parameters internally should, in principle, be able to accept SBML model definitions without needing to change component identifiers. Environments using a common global space of identifiers for parameters internally can perform manipulations of the identifiers of local parameters within reaction definitions to avoid identifier collisions.

The guidelines described here will hopefully provide a clean transition path to future levels of SBML, when submodels are introduced (Section 8.1). Submodels will provide the ability to compose one model from a collection of other models. This capability will have to be built on top of SBML Level 2's namespace organization. A straightforward approach to handling namespaces is to make each submodel's space be private. The rules governing identifier scoping within a submodel can simply be the Level 2 namespace rule described here, with each submodel having its own (to itself, global) namespace.

3.3.2 The `name` attribute

In contrast to the `id` attribute, the `name` attribute is optional and is not intended to be used for cross-referencing purposes within a model. Its purpose instead is to provide a human-readable label for the component. The data type of `name` is the type `string` defined in XML Schema (Biron and Malhotra, 2000; Thompson et al., 2000) and discussed further in Section 3.1. SBML imposes no restrictions as to the content of `name` attributes beyond those restrictions defined by the `string` type in XML Schema.

The recommended practice for handling `name` is as follows. If a software tool has the capability for displaying the content of `name` attributes, it should display this content to the user as a component's label instead of the component's `id`. If the user interface does not have this capability (e.g., because it cannot display or use special characters in symbol names), or if the `name` attribute is missing on a given component, then the user interface should display the value of the `id` attribute instead. (Script language interpreters are especially likely to display `id` instead of `name`.)

As a consequence of the above, authors of systems that automatically generate the values of `id` attributes should be aware some systems may display the `id`'s to the user. Authors therefore may wish to take some care to have their software create `id` values that are: (a) reasonably easy for humans to type and read; and (b) likely to be meaningful, e.g., the `id` attribute is an abbreviated form of the name attribute value.

An additional point worth mentioning is although there are restrictions on the uniqueness of `id` values (see Section 3.3.1 above), there are no restrictions on the uniqueness of `name` values in a model. This allows software packages leeway in assigning component identifiers.

3.3.3 *Why* `id` *and* `name` *are not defined on SBase*

Although many SBML components feature `id` and `name`, these attributes are purposefully not defined on ***SBase***. There are several reasons for this.

- The presence of an SBML identifier attribute (`id`) necessarily requires specifying scoping rules for the corresponding identifiers. However, the ***SBase*** abstract type is used as the basis for defining components whose scoping rules are in some cases different from each other. (See Section 3.3.1 for more details). If ***SBase*** were to have an `id` attribute, then the specification of ***SBase*** would need a default scoping rule and this would then have to be overloaded on derived classes that needed different scoping. This would make the SBML specification even more complex.
- Identifier are optional on some SBML components and required on most others. If `id` were defined as optional on ***SBase***, most component classes would separately have to redefine `id` as being mandatory—hardly an improvement over the current arrangement. Conversely, if `id` were defined as mandatory on ***SBase***, it would prevent it from being optional on components where it *is* currently optional.
- The ***SBase*** abstract type is used as the base type for certain objects such as **Sbml**, **AssignmentRule**, etc., which do not have identifiers because these components do not need to be referenced by other components. If ***SBase*** had a mandatory `id` attribute, *all* objects of these other types in a model would then need to be assigned unique identifiers. Similarly, because ***SBase*** is the base type of the `listOf`_____ lists, putting `id` on ***SBase*** would require all of these lists in a model to be given identifiers. This would be a needless burden on software developers, tools, and SBML users, requiring them to generate and store additional identifiers for objects that never need them.
- ***SBase*** does not have a `name` simply because such an attribute is always paired with an `id`. Without `id` on ***SBase***, it does not make sense to have `name`.

3.4 Mathematical formulas in SBML Level 2

Mathematical expressions in SBML Level 2 are represented using MathML 2.0 (W3C, 2000b). MathML is an international standard for encoding mathematical expressions using XML. There are two principal facets of MathML, one for encoding content (i.e., the semantic interpretation of a mathematical expression), and another for encoding presentation or display characteristics. SBML only makes direct use of a subset of the content portion of MathML. By borrowing a separately-developed XML standard, we can avoid having to define a specialized syntax for mathematical expressions in SBML and simultaneously leverage existing intellectual and technological work already done in the MathML community. However, it is not possible to produce a completely smooth and conflict-free interface between MathML and other standards used by SBML (in particular, XML Schema). Two specific issues and their resolutions are discussed in Sections 3.4.2.

The XML namespace URI for all MathML elements is `http://www.w3.org/1998/Math/MathML`. Everywhere MathML content is allowed in SBML, the MathML elements must be properly placed within the MathML 2.0

namespace. In XML, this can be accomplished in a number of ways, and the examples throughout this specification illustrate the use of this namespace and MathML in SBML. Please refer to the W3C document by Bray et al. (1999) for more technical information about using XML namespaces.

3.4.1 Subset of MathML used in SBML Level 2

The subset of MathML 2.0 elements used in SBML Level 2 is similar to that used by CellML (Hedley et al., 2001), another model definition language with similar goals as SBML. The subset of MathML elements used in SBML is listed below:

- *token*: `cn`, `ci`, `csymbol`, `sep`
- *general*: `apply`, `piecewise`, `piece`, `otherwise`, `lambda` (the last is restricted to use in **FunctionDefinition**)
- *relational operators*: `eq`, `neq`, `gt`, `lt`, `geq`, `leq`
- *arithmetic operators*: `plus`, `minus`, `times`, `divide`, `power`, `root`, `abs`, `exp`, `ln`, `log`, `floor`, `ceiling`, `factorial`
- *logical operators*: `and`, `or`, `xor`, `not`
- *qualifiers*: `degree`, `bvar`, `logbase`
- *trigonometric operators*: `sin`, `cos`, `tan`, `sec`, `csc`, `cot`, `sinh`, `cosh`, `tanh`, `sech`, `csch`, `coth`, `arcsin`, `arccos`, `arctan`, `arcsec`, `arccsc`, `arccot`, `arcsinh`, `arccosh`, `arctanh`, `arcsech`, `arccsch`, `arccoth`
- *constants*: `true`, `false`, `notanumber`, `pi`, `infinity`, `exponentiale`
- *annotation*: `semantics`, `annotation`, `annotation-xml`

The inclusion of logical operators, relational operators, `piecewise`, `piece`, and `otherwise` elements facilitates the encoding of discontinuous expressions. Note that MathML elements for representing partial differential calculus are not included. We anticipate that the requirements for partial differential calculus will be addressed in proposals for future SBML geometry representations (see Section 8.1).

As defined by MathML 2.0, the semantic interpretation of the mathematical functions listed above follows the definitions of the functions laid out by Abramowitz and Stegun (1977) and Zwillinger (1996). Readers are directed to these sources and the MathML specification for information about such things as which principal values of the inverse trigonometric functions to use.

Software authors should take particular note of the MathML semantics of the N-ary operators `plus`, `times`, `and`, `or` and `xor`, when they are used with different numbers of arguments. The MathML specification (W3C, 2000b) appendix C.2.3 describes the semantics for these operators with zero, one, and more arguments.

The following are the only attributes permitted on MathML elements in SBML (in addition to the `xmlns` attribute on `math` elements):

- `style`, `class` and `id` on any element;
- `encoding` on `csymbol`, `annotation` and `annotation-xml` elements;
- `definitionURL` on `ci`, `csymbol` and `semantics` elements; and
- `type` on `cn` elements.

Missing values for these attributes are to be treated in the same way as defined by MathML. These restrictions on attributes are designed to confine the MathML elements to their default semantics and to avoid conflicts in the interpretation of the type of token elements.

3.4.2 Numbers and `cn` elements

In MathML, literal numbers are written as the content portion of a particular element called `cn`. This element takes an optional attribute, `type`, used to indicate the *type* of the number (such as whether it is meant to be an integer or a floating-point quantity). Here is an example of its use:

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

```
        <apply>
            <times/> <cn type="integer"> 42 </cn> <cn type="real"> 3.3 </cn>
        </apply>
    </math>
```

The content of a `cn` element must be a number. The number can be preceded and succeeded by whitespace (see Section 3.4.5). The following are the only permissible values for the `type` attribute on MathML `cn` elements: "`e-notation`", "`real`", "`integer`", and "`rational`". The value of the `type` attribute defaults to "`real`" if it is not specified on a given `cn` element.

Value space restrictions on `cn` *content*

SBML imposes certain restrictions on the value space of numbers allowed in MathML expressions. According to the MathML 2.0 specification, the values of the content of `cn` elements do not necessarily have to conform to any specific floating point or integer representations designed for CPU implementation. For example, in strict MathML, the value of a `cn` element could exceed the maximum value that can be stored in a IEEE 64 bit floating point number (IEEE 754). This is different from the XML Schema type `double` that is used in the definition of floating point attributes of objects in SBML; the XML Schema `double` *is* restricted to IEEE double-precision 64-bit floating point type IEEE 754-1985. To avoid an inconsistency that would result between numbers elsewhere in SBML and numbers in MathML expressions, SBML Level 2 Version 5 imposes the following restriction on MathML content appearing in SBML:

- Integer values (i.e., the values of `cn` elements having `type`="`integer`" and both values in `cn` elements having `type`="`rational`") must conform to the `int` type used elsewhere in SBML (Section 3.1.3)
- Floating-point values (i.e., the content of `cn` elements having `type`="`real`" or `type`="`e-notation`") must conform to the `double` type used elsewhere in SBML (Section 3.1.5)

Syntactic differences in the representation of numbers in scientific notation

It is important to note that MathML uses a style of scientific notation that differs from what is defined in XML Schema, and consequently what is used in SBML attribute values. The MathML 2.0 type "`e-notation`" (as well as the type "`rational`") requires the mantissa and exponent to be separated by one `<sep/>` element. The mantissa must be a real number and the exponent part must be a signed integer. This leads to expressions such as

```
<cn type="e-notation"> 2 <sep/> -5 </cn>
```

for the number 2×10^{-5}. It is especially important to note that the expression

```
<cn type="e-notation"> 2e-5 </cn>
```

is *not valid* in MathML 2.0 and therefore cannot be used in MathML content in SBML. However, elsewhere in SBML, when an attribute value is declared to have the data type `double` (a type taken from XML Schema), the compact notation "`2e-5`" is in fact allowed. In other words, within MathML expressions contained in SBML (and *only* within such MathML expressions), numbers in scientific notation must take the form `<cn type="e-notation"> 2 <sep/> -5 </cn>`, and everywhere else they must take the form "`2e-5`".

This is a regrettable difference between two standards that SBML replies upon, but it is not feasible to redefine these types within SBML because the result would be incompatible with parser libraries written to conform with the MathML and XML Schema standards. It is also not possible to use XML Schema to define a data type for SBML attribute values permitting the use of the `<sep/>` notation, because XML attribute values cannot contain XML elements—that is, `<sep/>` cannot appear in an XML attribute value.

Units of numbers in MathML `cn` *expressions*

What units should be attributed to values appearing inside MathML `cn` elements? One answer is to assume that the units should be "whatever units appropriate in the context where the number appears". This implies that units can always be assigned unambiguously to any number by inspecting the expression in

which it appears, and this turns out to be false. Another answer is that numbers should be considered "dimensionless". Many people argue that this is the correct interpretation, but even if it is, there is an overriding practical reason why it cannot be adopted for SBML's domain of application: when numbers appear in expressions in SBML, they are *rarely intended* by the modeler to have the unit "`dimensionless`" even if the unit is not declared—the numbers are *supposed* to have specific units, but the units are usually undeclared. (Being "dimensionless" is not the same as having *undeclared* units!) If SBML defined numbers as being *by default* dimensionless, it would result in many models being technically incorrect without the modeler being aware of it unless their software tools performed dimensional analysis. Most software tools today still do not perform dimensional analysis, and so the inconsistency of units (and potential errors in the model) would not be detected until other researchers and database curators attempted to use the model in software packages that *did* check units. We believe the negative impact on interoperability and people's confidence in SBML as a reliable medium would be too high.

As a result, the current approach in SBML is to leave the default units of literal numbers in MathML content undefined. Software packages and modelers are encouraged to explicitly add unit declarations to numbers. There is a simple mechanism in SBML for associating units with numbers: do not use literal numbers at all; instead, define **Parameter** objects (Section 4.9) for every quantity, declare units for each such parameter value in its definition, and then insert the parameters in place of numbers in expressions. This leads to mathematical formulas whose units can be fully determined, permitting software tools to perform dimensional analysis and potentially report problems with a model.

In summary: literal numbers appearing within MathML content in SBML have no declared units.

3.4.3 Use of `ci` elements in MathML expressions in SBML

The content of a `ci` element must be an SBML identifier that is declared elsewhere in the model. The identifier can be preceded and succeeded by whitespace. The set of possible identifiers that can appear in a `ci` element depends on the containing element in which the `ci` is used:

- If a `ci` element appears in the body of a **FunctionDefinition** object (Section 4.3), the referenced identifier must be either (i) one of the declared arguments to that function, or (ii) the identifier of a previously defined **FunctionDefinition** object in the model.
- Otherwise, the referenced identifier must be that of a **Species**, **Compartment**, **Parameter**, **FunctionDefinition**, or **Reaction** object defined in the model. The following are the only possible interpretations of using such an identifier in SBML:
  - *Species identifier*: When a **Species** identifier occurs in a `ci` element, it represents the quantity of that species in units of either *amount of substance* or units of *concentration*, depending on the species' definition; see Section 4.8.5.
  - *Compartment identifier*: When a **Compartment** identifier occurs in a `ci` element, it represents the size of the compartment. The units of measurement associated with the size of the compartment are those given by the **Compartment** instance's `units` attribute value; see Section 4.7.5.
  - *Parameter identifier*: When a **Parameter** identifier occurs in a `ci` element, it represents the numerical value assigned to that parameter. The units associated with the parameter's value are those given by the **Parameter** instance's `units` attribute; see Section 4.9.3.
  - *Function identifier*: When a **FunctionDefinition** identifier occurs in a `ci` element, it represents a call to that function. Function references in MathML occur in the context of using MathML's `apply` and often involve supplying arguments to the function; see Section 4.3. The units associated with the value returned by the function call are the overall units of the mathematical expression contained in the function definition.
  - *Reaction identifier*: When a **Reaction** identifier occurs in a `ci` element, it represents the rate of that reaction as defined by the `math` expression in the **KineticLaw** object within the **Reaction**. The units associated with that rate are *substance*/*time*, where the *substance* and *time* units established by the values of the SBML predefined units "`substance`" and "`time`", respectively. These units may be redefined globally in the model; see Section 4.4.3. If a **Reaction** instance has no **KineticLaw**,

its reaction identifier has no mathematical definition within the model (perhaps indicating that the model is incomplete).

The content of `ci` elements in MathML formulas outside of a **KineticLaw** or **FunctionDefinition** must always refer to objects declared in the top level global namespace; i.e., SBML uses "early binding" semantics. Inside of **KineticLaw**, a `ci` element can additionally refer to local parameters defined within that **KineticLaw** instance; see Section 4.13.5 for more information.

3.4.4 Interpretation of boolean values

As noted already in Section 3.1.2, there is another unfortunate difference between the XML Schema 1.0 and MathML 2.0 standards that impacts mathematical expressions in SBML: in XML Schema, the value space of type `boolean` includes "`true`", "`false`", "`1`", and "`0`", whereas in MathML, only "`true`" and "`false`" count as boolean values.

The impact of this difference thankfully is minimal because the XML Schema definition is only used for attribute values on SBML objects, and those values turn out never to be accessible from MathML content in SBML—values of boolean attributes on SBML objects can never enter into MathML expressions. Nevertheless, software authors and users should be aware of the difference and in particular that "`0`" and "`1`" are interpreted as numerical quantities in mathematical expressions. There is no automatic conversion of "`0`" or "`1`" to boolean values in contexts where booleans are expected. This allows stricter type checking and unit verification during the validation of mathematical expressions.

3.4.5 Handling of whitespace

MathML 2.0 defines "whitespace" in the same way as XML does, i.e., the space character (Unicode hexadecimal code 0020), horizontal tab (code 0009), newline or line feed (code 000A), and carriage return (code 000D). In MathML, the content of elements such as `cn` and `ci` can be surrounded by whitespace characters. Prior to using the content, this whitespace is "trimmed" from both ends: all whitespace at the beginning and end of the content is removed (Ausbrooks et al., 2003). For example, in `<cn> 42 </cn>`, the amount of white space on either side of the "`42`" inside the `<cn> ... </cn>` container does not matter. Prior to interpreting the content, the whitespace is removed altogether.

3.4.6 Use of `csymbol` elements in MathML expressions in SBML

SBML Level 2 uses the MathML `csymbol` element to denote certain built-in mathematical entities without introducing reserved names into the component identifier namespace. The `encoding` attribute of `csymbol` must be set to "`text`". The `definitionURL` should be set to one of the following URIs defined by SBML:

- `http://www.sbml.org/sbml/symbols/time`. This represents the current simulation time. See Section 3.4.7 for more information. The units of the current time entity is determined from the built-in `time` of Table 2 on page 42.
- `http://www.sbml.org/sbml/symbols/delay`. This represents a delay function. The delay function has the form $delay(x, d)$, taking two MathML expressions as arguments. Its value is the value of argument x at d time units before the current time. There are no restrictions on the form of x. The units of the d parameter are determined from the built-in `time`. The value of the d parameter, when evaluated, must be numerical (i.e., a number in MathML real, integer, or "e-notation" format) and be greater than or equal to 0. The *delay* function is useful for representing biological processes having a delayed response, but where the detail of the processes and delay mechanism is not relevant to the operation of a given model. See Section 3.4.7 below for additional considerations surrounding the use of this `csymbol`.

The following examples demonstrate these concepts. The XML fragment below encodes the formula $x + t$, where t stands for time.

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
        <plus/>
        <ci> x </ci>
        <csymbol encoding="text" definitionURL="http://www.sbml.org/sbml/symbols/time">
            t
        </csymbol>
    </apply>
</math>
```

In the fragment above, the use of the token `t` is mostly a convenience for human readers—the string inside the `csymbol` could have been almost anything, because it is essentially ignored by MathML parsers and SBML. Some MathML and SBML processors will take note of the token and use it when presenting the mathematical formula to users, but the token used has no impact on the interpretation of the model and it does *not* enter into the SBML component identifier namespace. In other words, the SBML model cannot refer to `t` in the example above. The content of the `csymbol` element is for rendering purposes only and can be ignored by the parser.

As a further example, the following XML fragment encodes the equation $k + delay(x, 0.1)$ or alternatively $k_t + x_{t-0.1}$:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
        <plus/>
        <ci> k </ci>
        <apply>
            <csymbol encoding="text" definitionURL="http://www.sbml.org/sbml/symbols/delay">
                delay
            </csymbol>
            <ci> x </ci>
            <cn> 0.1 </cn>
        </apply>
    </apply>
</math>
```

Note that the URI in the value of `definitionURL`, as all URIs, is intended to serve as a unique identifier and is not intended to be dereferenced as an Internet address. There is nothing actually located at the address `http://www.sbml.org/sbml/symbols/delay`.

3.4.7 Simulation time

The principal use of SBML is to represent quantitative dynamical models whose behaviors manifest themselves over time. In defining an SBML model using constructs such as reactions, time is most often implicit and does not need to be referred to in the mathematical expressions themselves. However, sometimes an explicit time dependency needs to be stated, and for this purpose, the *time* `csymbol` (described above in Section 3.4.6) may be used. This *time* symbol refers to "instantaneous current time" in a simulation, frequently given the literal name t in one's equations.

An assumption in SBML is that "start time" or "initial time" in a simulation is zero, that is, if t_0 is the initial time in the system, $t_0 = 0$. This corresponds to the most common scenario. Initial conditions in SBML take effect at time $t = 0$. There is no mechanism in SBML for setting the initial time to a value other than 0. To refer to a different time in a model, one approach is to define a **Parameter** for a new time variable and use an **AssignmentRule** in which the assignment expression subtracts a value from the `csymbol` *time*. For example, if the desired offset is 2 time units, the MathML expression would be

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
    <apply>
        <minus/>
        <csymbol encoding="text" definitionURL="http://www.sbml.org/sbml/symbols/time"> t
        </csymbol>
        <cn> 2 </cn>
    </apply>
</math>
```

SBML's assignment rules (Section 4.11.3) can be used to express mathematical statements that hold true at all moments, so using an assignment rule with the expression above will result in the value being equal to $t - 2$ at every point in time. A parameter assigned this value could then be used elsewhere in the model, its value could be plotted by a simulator, etc.

3.4.8 Initial conditions and special considerations

The identifiers of **Species**, **Compartment**, **Parameter**, and **Reaction** object instances in a given SBML model refer to the main variables in a model. Depending on certain attributes of these objects (e.g., the attribute `constant` on species, compartments and parameters—this and other conditions are explained in the relevant sections elsewhere in this document), some of the variables may have constant values throughout a simulation, and others' values may change. These changes in values over time are determined by the system of equations constructed from the model's reactions, initial assignments, rules, and events.

As described in Section 3.4.7, an SBML model's simulation is assumed to begin at $t = 0$. The availability of the *delay* `csymbol` (Section 3.4.6) introduces the possibility that at $t \geq 0$, mathematical expressions in a model may draw on values of model components from time *prior* to $t = 0$. A simulator may therefore need to compute the values of variables at time points $t_i \leq 0$ to allow the calculation of values required for the evaluation of delay expressions in the model for $t \geq 0$. If there are no delays in the model, then $t_i = 0$.

The following is how the definitions of the model should be applied:

1. At time t_i:
   - Every **Species**, **Compartment**, and **Parameter** whose definition includes an initial value is assigned that value. If an element has `constant`="`false`", its value may be changed by other constructs or reactions in a model according to the steps below; if `constant`="`true`", only an **InitialAssignment** can override the value.
   - All **InitialAssignment** definitions take effect, overriding initial values on any **Species**, **Compartment** or **Parameter**.
   - All **AssignmentRule** and **AlgebraicRule** definitions take effect. These rules also override any initial values of any **Species**, **Compartment**, or **Parameter** in the model. Only elements set `constant`="`false`" can be affected in this way. (Note there cannot be both an **AssignmentRule** and an **InitialAssignment** for the same identifier, nor may an **AlgebraicRule** determine the value of any element that has an **InitialAssignment**; see Section 4.11.)
   - The identifier of any **Reaction** is the value of its **KineticLaw**. This cannot yet affect the **Species** referenced by the **Reaction**, but the identifier itself may appear in the math of other elements calculated above.
2. For time $t_i < t < 0$
   - Any **Species**, **Compartment**, or **Parameter** with no **InitialAssignment** or ***Rule*** that targets it continues to have its initial value, as defined by the relevant attribute.
   - Any **InitialAssignment** definition continues to take effect. Since these contain mathematical formulas, different values may be computed at each time step t in $t_i \leq t \leq 0$.
   - Any **AssignmentRule** or **AlgebraicRule** definition continues to take effect, and may not be overridden. Again, different values may be computed at each time step t in $t_i \leq t \leq 0$.
   - The identifier of any **Reaction** continues to be the value of its **KineticLaw**. Again, different values may be computed at each time step t in $t_i \leq t \leq 0$.
3. At time $t = 0$:
   - The identifier of any **Reaction** continues to be the value of its **KineticLaw**, but may begin to affect its referenced **Species**. A **Reaction** with a `fast` attribute of "`false`" (the default) affects its **Species** at a rate, that is, an amount per time. As the time that has passed is still zero, the change in the referenced **Species** is also zero. But reactions with a `fast` attribute of "`true`" *do* affect the levels of their referenced **Species**, regardless of the timeframe involved, and thus may change the value of their referenced **Species** even at time $t = 0$.

- Any **Species**, **Compartment**, or **Parameter** with no **InitialAssignment** or ***Rule*** that targets it continues to have its initial value, as defined by the relevant attribute, but may now be overridden by any fast **Reaction**, executed as above.
- Any **InitialAssignment** definition continues to take effect, but may now be overridden by any fast **Reaction**, executed as above.
- Any **AssignmentRule** or **AlgebraicRule** definition continues to take effect, and may not be overridden.
- **Event** definitions triggered by a change wrought by a fast **Reaction** (as above) can begin to take effect. As events are only triggered when their trigger condition changes from $false$ to $true$, fast reactions are the only SBML construct that can cause this to happen at time $t = 0$, though an **Event** cascade may also result (that is, a fast **Reaction** causing a change that triggers an **Event**, whose **EventAssignment** triggers another **Event**, etc.).
- **Constraint** definitions begin to take effect (and a constraint violation may result; see Section 4.12).

4. For time $t > 0$:
   - The value of any element with mathematical meaning may now be overridden by any construct in SBML (though it may retain its original value if no such constructs apply).
   - The value of any element with an **InitialAssignment** may also now be overridden by any construct in SBML (though it may retain the value set by the **InitialAssignment** if no such constructs apply).
   - Any **AssignmentRule** or **AlgebraicRule** definition continues to take effect, and still may not be overridden by any other SBML construct.
   - Any **Reaction** can begin to affect its referenced **Species**. Its identifier continues to be the value of its **KineticLaw**.
   - **RateRule** definitions can begin to take effect.
   - All **Event** definitions can begin to take effect. (Note that an **Event** cannot be defined to change the value of a variable that is also the subject of an **AssignmentRule**; see Section 4.14.)
   - System simulation proceeds.

To reiterate: in modeling situations that do not involve the use of the *delay* `csymbol`, $t_i = 0$, but this does not alter the steps above.

3.4.9 Underdetermined models

SBML models may not be *overdetermined*, that is, one cannot define a model with multiple constructs that each define their own way of establishing the value of a symbol. (The exception to this rule is that one may have one element with an attribute defining its initial value which is overruled by an **InitialAssignment** or ***Rule***.) Such models are inherently self-contradictory, and thus not valid. However, it is perfectly legal to define and exchange an *underdetermined* model, that is, a model with one or more symbols that have no way of establishing their values (such as a model with a **Parameter** with no `initialValue`, no **InitialAssignment**, and no relevant ***Rule***), or a model with multiple correct solutions (such as a model with an **AlgebraicRule** that could be used to determine either one but not both of two different symbols, or an **AlgebraicRule** with multiple solutions, like $0 = x^2 - 4$). Such models cannot be simulated without extra information being added in some way, but they are *incomplete*, not *self-contradictory*, and therefore valid.

There are a number of reasons one may wish to create an underdetermined SBML model. At the most basic level, one may be in the process of creating a fully-determined model, but are not yet finished doing so, either as a work in progress on one tool, or as part of a model-creating pipeline across multiple tools. Similarly, one may be interested in creating a model that reflects the current state of knowledge about a biological system, and that knowledge itself may be incomplete. One may also be interested in performing a type of analysis other than simulation, such as a structural analysis, which does not require all symbols to be numerically defined.

Different simulation software tools approach the problem of underdetermined models in different ways when asked to perform a simulation. Some simply refuse, requiring more information from the user before proceeding. Others provide defaults (typically telling the user they are doing so) for symbols whose values are not established by the model, using values of '1' or '0', depending on the element type. In the case of encountering an **AlgebraicRule** with multiple solutions, some software tools allow the use of **Constraint** elements to choose one solution over another ($0 = x^2 - 4; x < 0$). All of these approaches are valid responses to encountering an underdetermined SBML model, but no one solution is established cannonically as being 'correct', as different situations warrant different responses.

3.4.10 MathML expression data types

MathML operators in SBML each return results in one of two possible types: boolean and numerical. By *numerical* type, we mean either (1) a number in MathML real, integer, rational, or "e-notation" format; or (2) the `csymbol` for delay or the `csymbol` for time described in Section 3.4.6. The following guidelines summarize the different possible cases.

The relational operators (`eq`, `neq`, `gt`, `lt`, `geq`, `leq`), the logical operators (`and`, `or`, `xor`, `not`), and the boolean constants (`false`, `true`) always return boolean values. As noted in Section 3.4.4, the numbers `0` and `1` do not count as boolean values in MathML contexts in SBML.

The type of an operator referring to a **FunctionDefinition** is determined by the type of the top-level operator of the expression in the `math` element of the **FunctionDefinition** instance, and can be boolean or numerical.

All other operators, values and symbols return numerical results.

The roots of the expression trees used in the following contexts must yield boolean values:

- the arguments of the MathML logical operators (`and`, `or`, `xor`, `not`);
- the second argument of a MathML `piece` operator;
- the `trigger` element of an SBML **Event**; and
- the `math` element of an SBML **Constraint**.

The roots of the expression trees used in the following contexts can optionally yield boolean values:

- the arguments to the `eq` and `neq` operators;
- the first arguments of MathML `piece` and `otherwise` operators; and
- the top level expression of a function definition.

The roots of expression trees in other contexts must yield numerical values.

The type of expressions should be used consistently. The set of expressions that make up the first arguments of the `piece` and `otherwise` operators within the same `piecewise` operator should all return values of the same type. The arguments of the `eq` and `neq` operators should return the same type.

3.4.11 Consistency of units in mathematical expressions and treatment of unspecified units

Strictly speaking, physical validity of mathematical formulas requires not only that physical quantities added to or equated with each other have the same fundamental dimensions and units of measurement; it also requires that the application of operators and functions to quantities produces sensible results. Yet, in real-life models today, these conditions are often and sometimes legitimately disobeyed.

In a public vote held in late 2007, the SBML community decided to revoke the requirement (present up through Level 2 Version 3) for strict unit consistency in SBML. As a result, Level 2 Version 4 follows this decision; the units on quantities and the results of mathematical formulas in a model *should* be consistent, but it is not a strict error if they are not. The following are thus formulated as recommendations that *should* be followed except in special circumstances.

Recommendations for unit consistency of mathematical expressions

The consistency of units is defined in terms of dimensional analysis applied recursively to every operator and function and every argument to them. The following conditions should hold true in a model (and software developers may wish to consider having their software warn users if one or more of the following conditions is not true):

1. All arguments to the following operators should have the same units (regardless of what those units happen to be): `plus`, `minus`, `eq`, `neq` `gt`, `lt`, `geq`, `leq`.

2. The units of each argument in a call to a **FunctionDefinition** should match the units expected by the `lambda` expression within the `math` expression of that **FunctionDefinition** instance.

3. All of the possible return values from `piece` and `otherwise` subelements of a `piecewise` expression should have the same units, regardless of what those units are. (Otherwise, the `piecewise` expression would return values having different units depending on which case evaluated to true.)

4. For the *delay* `csymbol` (Section 3.4.6) function, which has the form $delay(x, d)$, the second argument d should match the model's unit of *time* (i.e., the "`time`" predefined unit).

5. The units of each argument to the following operators should be "`dimensionless`": `exp`, `ln`, `log`, `factorial`, `sin`, `cos`, `tan`, `sec`, `csc`, `cot`, `sinh`, `cosh`, `tanh`, `sech`, `csch`, `coth`, `arcsin`, `arccos`, `arctan`, `arcsec`, `arccsc`, `arccot`, `arcsinh`, `arccosh`, `arctanh`, `arcsech`, `arccsch`, `arccoth`.

6. The two arguments to `power`, which are of the form $power(a, b)$ with the meaning a^b, should be as follows: (1) if the second argument is an integer, then the first argument can have any units; (2) if the second argument b is a rational number n/m, it should be possible to derive the m-th root of $(a\{\text{units}\})^n$, where $\{\text{units}\}$ signifies the units associated with a; otherwise, (3) the units of the first argument should be "`dimensionless`". The second argument (b) should always have units of "`dimensionless`".

7. The two arguments to `root`, which are of the form $root(n, a)$ with the meaning $\sqrt[n]{a}$ and where the degree n is optional (defaulting to "`2`"), should be as follows: (1) if the optional degree qualifier n is an integer, then it should be possible to derive the n-th root of a; (2) if the optional degree qualifier n is a rational n/m then it should be possible to derive the n-th root of $(a\{\text{units}\})^m$, where $\{\text{units}\}$ signifies the units associated with a; otherwise, (3) the units of a should be "`dimensionless`".

8. Since the units of literal numbers cannot be specified directly in SBML (see below), it is possible for the units of a **FunctionDefinition** object's return value to be effectively different in different contexts where it is called. If a **FunctionDefinition**'s mathematical formula contains literal constants (i.e., numbers within MathML `cn` elements), the units of the constants should be identical in all contexts the function is called.

The units of other operators such as `abs`, `floor`, and `ceiling`, can be anything.

The final bulleted item above, regarding **FunctionDefinition**, warrants additional elaboration. An example may help illustrate the problem. Suppose the formula $x + 5$ is defined as a function, where x is an argument. The literal number 5 in SBML has unspecified units. If this function is called with an argument in moles, the only possible consistent unit for the return value is mole. If in another context in the same model, the function is called with an argument in seconds, the function return value can only be treated as being in seconds. Now suppose that a modeler decides to change all uses of seconds to milliseconds in the model. To make the function definition return the same quantity in terms of seconds, the 5 in the formula would need to be changed, but doing so would change the result of the function everywhere it is called—with the wrong consequences in the context where moles were intended. This illustrates the subtle danger of using numbers with unspecified units in function definitions. There are at least two approaches for avoiding this: (1) define separate functions for each case where the units of the constants are supposed to be different; or (2) declare the necessary constants as **Parameter** objects in the model (with declared units!) and pass those parameters as arguments to the function, avoiding the use of literal numbers in the function's formula.

Treatment of unspecified units

There are only two ways to introduce numbers with unspecified units into mathematical formulas in SBML: using literal numbers (i.e., numbers enclosed in MathML `cn` elements), and using **Parameter** objects defined without unit declarations. All other quantities, in particular species and compartments, always have unit declarations (whether explicit or the defaults).

If an expression contains literal numbers and/or **Parameter** objects without declared units, the consistency or inconsistency of units may be impossible to determine. In the absence of a verifiable *inconsistency*, an expression in SBML is accepted as-is; the writer of the model is assumed to have written what they intended. Nevertheless, this is *not* equivalent to assuming the expression *does* have consistent units.

In some cases, it may be possible to determine that expressions containing unspecified units are inconsistent regardless of what units would be attributed to the unspecified quantities. For example, the expression

$$\frac{dX}{dt} = \frac{[Y] \cdot [Z]^n}{[Z]^m + 1} \cdot V$$

with X, Y and Z in units of substance, V in units of volume, and $m \neq n$, cannot ever be consistent, no matter what units the literal 1 takes on. (This also illustrates the need not to stop verifying the units of an expression immediately upon encountering an unspecified quantity—the rest of the expression may still be profitably evaluated and checked for inconsistency.)

We advise modelers and software tools to declare the units of all quantities in a model, insofar as this is possible, using the various mechanisms provided for this in SBML. Fully declared units can allow software tools to perform dimensional analysis on the units of mathematical expressions, and such analysis can be valuable in helping modelers produce correct models. In addition, it can allow model-wide operations such as conversion or rescaling of units. The lack of declared units on quantities in a model does not render it invalid SBML, but it reduces the types of consistency checks and useful operations (such as conversions and translations) that software systems can perform.

3.4.12 SBML does not define implicit unit conversions

Implicit unit conversions do not exist in SBML. Consider the following example. Suppose that in some model, a species S_1 has been declared as having a mass of 1 kg, and a second species S_2 has been declared as having a mass of 500 g. What should be the result of evaluating an expression such as $S_1 > S_2$? If the numbers alone are considered,

$$1 > 500$$

would evaluate to "`false`", but if the units were implicitly converted by the software tool interpreting the model,

$$1\ kg > 500\ g$$

would evaluate to "`true`". This is a trivial example, but the problem for SBML is that implicit unit conversions of this kind can lead to controversial situations where even humans do not agree on the answer. Consequently, SBML only requires that mathematical expressions be evaluated numerically. It is up to the model writer to ensure that the units on both sides of an expression match, by inserting explicit unit conversion factors if necessary.

It is simple enough to avoid expressions with mixed units such as in the example above: a modeler or a software tool can define a parameter that acts as a conversion factor, and then multiply one of the terms by this parameter. Thus, a model could include a parameter "*g_per_kg*" with a value of 1000, and the expression could be written as

$$1 * g\_per\_kg > 500$$

which will then evaluate to "`true`".

4 SBML components

In this section, we define each of the major components of SBML. We use the UML notation described in Section 1.4.3 for defining classes of objects. We also illustrate the use of SBML components by giving partial model definitions in XML. Section 7 provides many full examples of SBML in XML.

4.1 The SBML container

All well-formed SBML documents must begin with an *XML declaration*, which specifies both the version of XML assumed and the document character encoding. The declaration begins with the characters `<?xml` followed by the XML `version` and `encoding` attributes. SBML Level 2 uses XML version 1.0 and requires a document encoding of UTF-8. Following this XML declaration, the outermost portion of a model expressed in SBML Level 2 Version 5 consists of an object of class **Sbml**, defined in Figure 9. This class contains three required attributes, for the `xmlns` and the SBML `level` and `version`, and one required subelement called `model` whose class is **Model**.

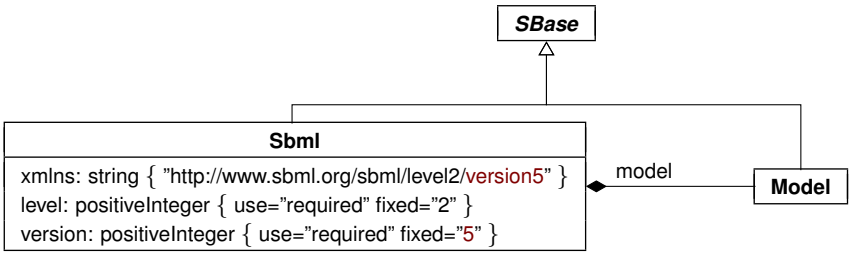


Figure 9: *The definition of **Sbml** for SBML Level 2 Version 5. The class **Model** is defined in Section 4.2. Note that both **Sbml** and **Model** are subclasses of **SBase**, and therefore inherit the attributes of that abstract class.*

The following is an abbreviated example of these XML elements for an SBML Level 2 Version 5 document:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
  ...
</sbml>
```

The attribute `xmlns` declares the default XML namespace used within the `sbml` element. The URI for SBML Level 2 Version 5 is `http://www.sbml.org/sbml/level2/version5`. All elements must be placed in this namespace either by assigning the default namespace as shown above, or using a tag prefix on every element.

An SBML XML document must not contain elements or attributes in the SBML namespace that are not defined in this SBML Level 2 Version 5 Release 1 specification. Documents containing unknown elements or attributes placed in the SBML namespace do not conform to this SBML specification.

Readers may wonder why the SBML top-level XML element uses both a namespace URI identifying the SBML level and version, as well as separate XML attributes giving the level and version. Why is the information duplicated? There are several reasons. First, XML is only one possible serialization of SBML (albeit an extremely popular one at this time). Though most of this document is written with XML in mind, it is the intention behind the design of SBML that its object structure should be implementable in other languages and software systems. Programmatic access is easier if the level and version information are accessible directly as data rather than have to be extracted from a string. Second, generic high-level XML parsers may not give their calling programs access to the value of the `xmlns` attribute. Providing the information via separate attributes is a good backup measure. And finally, earlier in the history of SBML, it was expected that only the level needed to be encoded as part of the namespace URI (e.g., `http://www.sbml.org/sbml/level1`) because it was hoped that changes within levels would not require XML Schema changes. This has proven to be false, but SBML Level 1 (both versions) and the first version of SBML Level 2 still subscribe to this principle. This means that for these variants of SBML, software tools must look for a `version` attribute on the top-level element. For backwards compatibility with software that expects this, it makes more sense to keep the version and level attributes.

4.2 Model

The definition of **Model** is shown in Figure 10 on the next page. Only one instance of a **Model** object is allowed per instance of an SBML Level 2 Version 5 Release 1 document or data stream, and it must be located inside the `<sbml> ... </sbml>` element as described in Section 4.1.

The **Model** object has an optional attribute, `id`, used to give the model an identifier. The identifier must be a text string conforming to the syntax permitted by the `SId` data type described in Section 3.1.7. **Model** also has an optional `name` attribute, of type `string`. The `name` and `id` attributes must be used as described in Section 3.3.

Model serves as a container for components of classes **FunctionDefinition**, **UnitDefinition**, **CompartmentType**, **SpeciesType**, **Compartment**, **Species**, **Parameter**, **InitialAssignment**, ***Rule***, **Constraint**, **Reaction** and **Event**. Instances of the classes are placed inside instances of classes **ListOfFunctionDefinitions**, **ListOfUnitDefinitions**, **ListOfCompartmentTypes**, **ListOfSpeciesTypes**, **ListOfCompartments**, **ListOfSpecies**, **ListOfParameters**, **ListOfInitialAssignments**, ***ListOfRules***, **ListOfConstraints**, **ListOfReactions**, and **ListOfEvents**. The "list" classes are defined in Figure 10. All of the lists are optional, but if a given list container is present within the model, the list must not be empty; that is, it must have length one or more. The resulting XML data object for a full model containing every possible list would have the following form:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model id="My_Model">
        <listOfFunctionDefinitions>
            one or more <functionDefinition> ...  </functionDefinition> elements      } optional
        </listOfFunctionDefinitions>
        <listOfUnitDefinitions>
            one or more <unitDefinition> ...  </unitDefinition> elements              } optional
        </listOfUnitDefinitions>
        <listOfCompartmentTypes>
            one or more <compartmentType> ...  </compartmentType> elements            } optional
        </listOfCompartmentTypes>
        <listOfSpeciesTypes>
            one or more <speciesType> ...  </speciesType> elements                    } optional
        </listOfSpeciesTypes>
        <listOfCompartments>
            one or more <compartment> ...  </compartment> elements                    } optional
        </listOfCompartments>
        <listOfSpecies>
            one or more <species> ...  </species> elements                            } optional
        </listOfSpecies>
        <listOfParameters>
            one or more <parameter> ...  </parameter> elements                        } optional
        </listOfParameters>
        <listOfInitialAssignments>
            one or more <initialAssignment> ...  </initialAssignment> elements        } optional
        </listOfInitialAssignments>
        <listOfRules>
            one or more elements of subclasses of Rule                                } optional
        </listOfRules>
        <listOfConstraints>
            one or more <constraint> ...  </constraint> elements                      } optional
        </listOfConstraints>
        <listOfReactions>
            one or more <reaction> ...  </reaction> elements                          } optional
        </listOfReactions>
        <listOfEvents>
            one or more <event> ...  </event> elements                                } optional
        </listOfEvents>
    </model>
</sbml>
```

Although all the lists are optional, there are dependencies between SBML components such that defining some components requires defining others. An example is that defining a species requires defining a compartment, and defining a reaction requires defining a species. The dependencies are explained throughout the text.

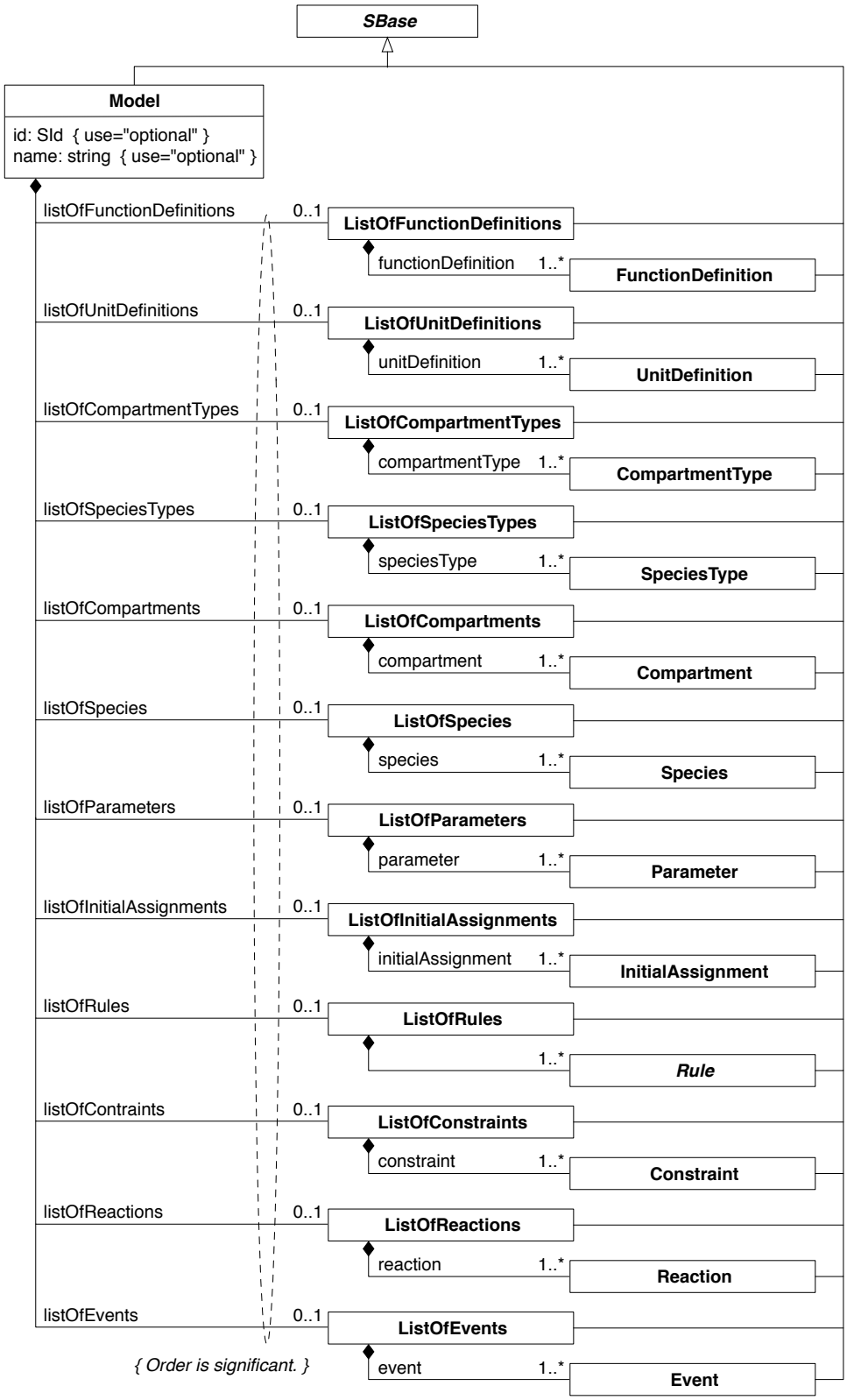


Figure 10: *The definition of* ***Model*** *and the many helper classes* ***ListOfFunctionDefinitions***, ***ListOfUnitDefinitions***, ***ListOfCompartmentTypes***, ***ListOfSpeciesTypes***, ***ListOfCompartments***, ***ListOfSpecies***, ***ListOfParameters***, ***ListOfInitialAssignments***, ***ListOfRules***, ***ListOfConstraints***, ***ListOfReactions***, *and* ***ListOfEvents****.*

4.2.1 The ListOf container classes

The various **ListOf**____ classes defined in Figure 10 are merely containers used for organizing the main components of an SBML model. All are derived from the abstract class ***SBase*** (Section 3.2), and inherit ***SBase***'s various attributes and subelements such as `metaid` and `annotation`, although in SBML Level 2 Version 5 Release 1 there are no defined SBO terms for the `sboTerm` attribute. The **ListOf**____ classes do not add any attributes of their own.

Readers may wonder about the motivations for using the **ListOf**____ containers. A simpler approach in XML might be to place the components all directly at the top level. We chose instead to group them within XML elements named after `listOfClassNames`, in part because we believe this helps organize the components and makes visual reading of models in XML easier. More importantly, the fact that the container classes are derived from ***SBase*** means that software tools can add information about the lists themselves into each list container's `annotation`, a feature that a number of today's software tools exploit.

4.2.2 The `sboTerm` attribute

The **Model** object has an optional `sboTerm` attribute of type `SBOTerm` (see Sections 3.1.9 and 5). Values of this attribute should be chosen from identifiers referring to an interaction defined in SBO (i.e., terms derived from `SBO:0000231`, "interaction"). The SBO term chosen should be the most precise (narrow) term that defines the overall process or phenomenon represented by the overall SBML model.

Prior to SBML Level 2 Version 4, the SBML specifications stipulated that the SBO branch for **Model** had be the mathematical framework branch of SBO. This turned out to be confusing and problematic. A realization also occurred in the SBML community that a model is, ultimately, always a representation of some process or phenomenon involving different entities, making the SBO branch of `SBO:0000231`, "interaction", an appropriate one for the `sboTerm` value on an SBML **Model**.

4.3 Function definitions

The **FunctionDefinition** object associates an identifier with a function definition. This identifier can then be used as the function called in subsequent MathML `apply` elements. **FunctionDefinition** is shown in Figure 11.

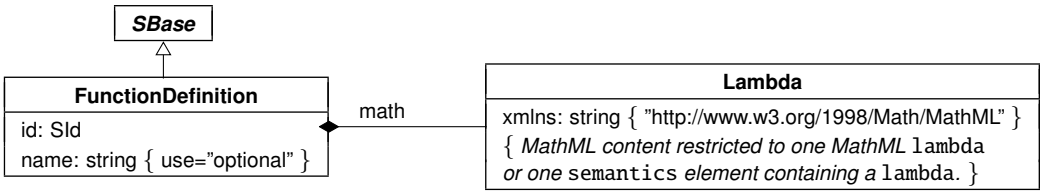


Figure 11: *The definition of class **FunctionDefinition**. The contents of the **Lambda** class is a single MathML* `lambda` *expression (or a* `lambda` *surrounded by a* `semantics` *element). A function definition must contain exactly one* `math` *element defined by the **Lambda** class; note also that **Lambda** is not derived from **SBase**, which means that the attributes defined on **SBase** are* not *available on the* `math` *element. A sequence of one or more instances of **FunctionDefinition** objects can be located in an instance of **ListOfFunctionDefinitions** in **Model**, as shown in Figure 10.*

Function definitions in SBML (also informally known as "user-defined functions") have purposefully limited capabilities. As is made more clear below, a function cannot reference parameters or other model quantities outside of itself; values must be passed as parameters to the function. Moreover, recursive and mutually-recursive functions are not permitted. The purpose of these limitations is to balance power against complexity of implementation. With the restrictions as they are, function definitions could be implemented as textual substitutions—they are simply macros. Software implementations therefore do not need the full function-definition machinery typically associated with programming languages.

4.3.1 The `id` and `name` attributes

The `id` and `name` attributes have types `SId` and `string`, respectively, and operate in the manner described in Section 3.3. MathML `ci` elements in an SBML model can refer to the function defined by a **FunctionDefinition** using the value of its `id` attribute.

4.3.2 The `math` element

The `math` element is a container for MathML content that defines the function. The content of this element can only be a MathML `lambda` element or a MathML `semantics` element containing a `lambda` element. The `lambda` element must begin with zero or more `bvar` elements, followed by any other of the elements in the MathML subset listed in Section 3.4.1 *except* `lambda` (i.e., a `lambda` element cannot contain another `lambda` element). This is the only place in SBML where a `lambda` element can be used.

A further restriction on the content of `math` is that it cannot contain references to variables other than the variables declared to the `lambda` itself. That is, the contents of MathML `ci` elements inside the body of the `lambda` can only be the variables declared by its `bvar` elements, or the identifiers of other **FunctionDefinition**s defined in the same model. This restriction also applies to the `csymbol` for *time* and to the `csymbol` for *delay*. Functions must be written so that all variables or parameters used in the MathML content are passed to them via their function parameters.

4.3.3 The `sboTerm` attribute

FunctionDefinition inherits an optional `sboTerm` attribute of type `SBOTerm` from its parent class ***SBase*** (see Sections 3.1.9 and 5). When a value is given to this attribute in a **FunctionDefinition** instance, it should be an SBO identifier referring to a mathematical expression (i.e., terms derived from `SBO:0000064`, "mathematical expression"). The relationship is of the form "the function definition *is a* X", where X is the SBO term. The term chosen should be the most precise (narrow) one that captures the role of the function in the model.

As discussed in Section 5, SBO labels are optional information on a model. Applications are free to ignore `sboTerm` values. A model must be interpretable without the benefit of SBO labels.

4.3.4 Calling user-defined functions

Within MathML expressions in an SBML model, all calls to a function defined by a **FunctionDefinition** must use the same number of arguments as specified in the function's definition. The number of arguments is equal to the number of `bvar` elements inside the `lambda` element of the function definition.

Note that **FunctionDefinition** does not have a separate attribute for defining the units of the value returned by the function. The units associated with the function's return value, when the function is called from within MathML expressions elsewhere in SBML, are simply the overall units of the expression in **FunctionDefinition**'s `math` when applied to the arguments supplied in the call to the function. Ascertaining these units requires performing dimensional analysis on the expression. (Readers may wonder why there is no attribute. The reason is that having a separate attribute for declaring the units would not only be redundant, but also lead to the potential for having conflicting information. In the case of a conflict between the declared units and those of the value actually returned by the function, the only logical resolution rule would be to assume that the correct units are those of the expression anyway.)

4.3.5 Examples

The following abbreviated SBML example shows a **FunctionDefinition** object instance defining `pow3` as the identifier of a function computing the mathematical expression x^3, and after that, the invocation of that function in the mathematical formula of a rate law. Note how the invocation of the function uses its identifier.

```
<model>
    ...
    <listOfFunctionDefinitions>
        <functionDefinition id="pow3">
            <math xmlns="http://www.w3.org/1998/Math/MathML">
```

```
                    <lambda>
                        <bvar><ci> x </ci></bvar>
                        <apply>
                            <power/>
                            <ci> x </ci>
                            <cn> 3 </cn>
                        </apply>
                    </lambda>
                </math>
            </functionDefinition>
        </listOfFunctionDefinitions>
        ...
        <listOfReactions>
            <reaction id="reaction_1">
                ...
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <ci> pow3 </ci>
                            <ci> S1 </ci>
                         </apply>
                    </math>
                </kineticLaw>
                ...
            </reaction>
        </listOfReactions>
        ...
    </model>
```

4.4 Unit definitions

Units of measurement may be supplied in a number of contexts in an SBML model. The units of the following mathematical entities can be specified explicitly: the size of a **Compartment**, the initial amount of a **Species**, and the units of constant and variable **Parameter** values. The overall units of any mathematical formula appearing in SBML are those that arise naturally from the components and mathematical expressions comprising the formula, or in other words, the units obtained by doing dimensional analysis on the formula.

Rather than requiring a complete unit definition on every object, SBML provides a facility for defining units that can be referenced throughout a model. In addition, every kind of SBML mathematical entity has units assigned to it from a set of predefined defaults (see Section 4.4.3 below, and also Sections 4.7.5, 4.8.5 and 4.13.5). By redefining these predefined units, it is possible to change the units used throughout a model in a simple and consistent manner. The SBML unit definition facility uses two classes of objects, **UnitDefinition** and **Unit**. Their definitions are shown in Figure 12 and explained in more detail in Sections 4.4.1 and 4.4.2 below.

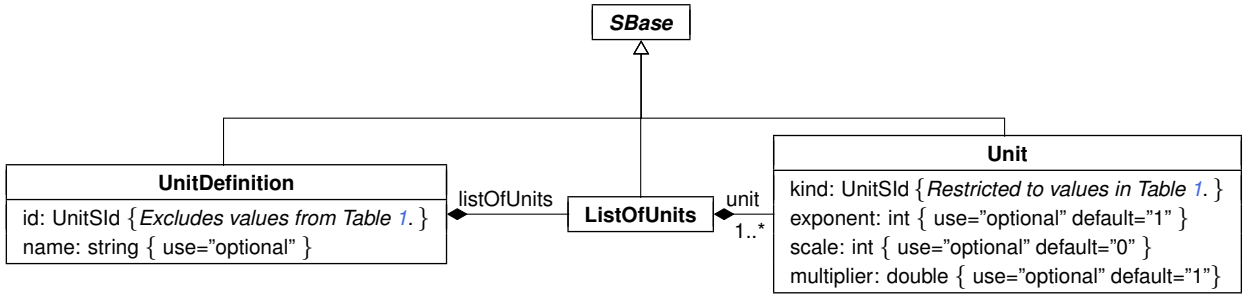


Figure 12: *The definition of classes* ***UnitDefinition*** *and* ***Unit****. A sequence of one or more instances of* ***UnitDefinition*** *can be located in an instance of* ***ListOfUnitDefinitions*** *in* ***Model*** *(Figure 10).* ***ListOfUnits*** *has no attributes (beyond those it inherits from class* ***SBase****); it merely acts as a container for one or more instances of* ***Unit*** *objects. Note that the only permitted values of* `kind` *on* ***Unit*** *are the reserved words in Table 1 on page 38, but these symbols are* excluded *from the permitted values of* ***UnitDefinition****'s* `id` *because SBML's unit system does not allow redefining the base units.*

The approach to defining units in SBML is compositional; for example, *meter* $second^{-2}$ is constructed by combining a **Unit** object representing *meter* with another **Unit** object representing $second^{-2}$. The combination is wrapped inside a **UnitDefinition**, which provides for assigning an identifier and optional name to the combination. The identifier can then be referenced from elsewhere in a model.

The vast majority of modeling situations requiring new SBML unit definitions involve simple multiplicative combinations of base units and factors. An example of this might be "moles per litre per second". What distinguishes these sorts of simpler unit definitions from more complex ones is that they may be expressed without the use of an additive offset from a zero point. The use of offsets complicates all unit definition systems, yet in the domain of SBML the real-life cases requiring offsets are few (and in fact, to the best of our knowledge, only involve temperature). Consequently, the SBML unit system has been consciously designed in a way that attempts to simplify implementation of unit support for the most common cases in systems biology, at the cost of requiring units with offsets to be handled explicitly by the modeler.

4.4.1 UnitDefinition

A unit definition in SBML consists of an instance of a **UnitDefinition** object, shown in Figure 12.

The `id` *and* `name` *attributes*

The required attribute `id` and optional attribute `name` have data types `UnitSId` and `string`, respectively. The `id` attribute is used to give the defined unit a unique identifier by which other parts of an SBML model definition can refer to it. The `name` attribute is intended to be used for giving the unit definition an optional human-readable name; see Section 3.3.2 for more guidelines about the use of names.

There are two important restrictions and guidelines about the use of unit definition `id` values:

1. The `id` of a **UnitDefinition** must *not* contain a value from Table 1, the list of reserved base unit names. This constraint simply prevents the redefinition of base units.

2. There is a set of reserved identifiers for the predefined units in SBML; these identifiers are "`substance`", "`volume`", "`area`", "`length`", and "`time`". Using one of these values for `id` in a **UnitDefinition** has the effect of redefining the model-wide default units for the corresponding quantities. We discuss this in more detail in Section 4.4.3.

The list of ***Units***

A **UnitDefinition** object must contain one or more **Unit** objects inside a **ListOfUnits** container. Section 4.4.2 explains the meaning and use of **Unit**.

Example

The following skeleton of a unit definition illustrates an example use of **UnitDefinition**:

```
<model>
    <listOfUnitDefinitions>
        <unitDefinition id="unit1">
            <listOfUnits>
                ...
            </listOfUnits>
        </unitDefinition>
        <unitDefinition id="unit2">
            <listOfUnits>
                ...
            </listOfUnits>
        </unitDefinition>
    </listOfUnitDefinitions>
    ...
</model>
```

4.4.2 Unit

A **Unit** object represents a (possibly transformed) reference to a base unit chosen from the list in Table 1. The attribute `kind` indicates the chosen base unit, whereas the attributes `exponent`, `scale`, and `multiplier` define how the base unit is being transformed. These various attributes are described in detail below.

In SBML Level 2 Version 1, **Unit** had an additional attribute called `offset`. This attribute has been removed entirely in Level 2 Versions 2–4. Modelers and software authors are instead directed to use other methods of encoding units requiring offsets. The reasons for this change, and some suggestions for how to achieve equivalent effects of unit offsets, are discussed in more detail below. Another change in Version 3 and 4 is the removal of the enumeration `UnitKind` and the redefinition of `UnitSId` to include the previous `UnitKind` values as reserved symbols. This change has no net effect on permissible models or their representation.

The `kind` *attribute*

The **Unit** object class has one required attribute, `kind`, whose value must be taken from the list of reserved words given in Table 1. These reserved symbols are in the value space of `UnitSId` (Section 3.1.8).

ampere	gram	katal	metre	second	watt
becquerel	gray	kelvin	mole	siemens	weber
candela	henry	kilogram	newton	sievert	
coulomb	hertz	litre	ohm	steradian	
dimensionless	item	lumen	pascal	tesla	
farad	joule	lux	radian	volt	

Table 1: *Base units defined in SBML. These symbols are predefined values of type* `UnitSId` *(Section 3.1.8). All are names of base or derived SI units (Bureau International des Poids et Mesures, 2000), except for "*`dimensionless`*" and "*`item`*", which are SBML additions important for handling certain common situations. "*`Dimensionless`*" is intended for cases where a quantity is a ratio whose units cancel out, and "*`item`*" for expressing such things as "N items" (e.g., "100 molecules"). Also, note that the gram and litre are not strictly part of SI; however, they are frequently used in SBML's areas of application and therefore are included as predefined unit identifiers. (The standard SI unit of mass is in fact the kilogram, and volume is defined in terms of cubic metres.) Comparisons of these values, like all values of type* `UnitSId`*, must performed in a case-sensitive manner.*

Note that the set of acceptable values for the attribute `kind` does *not* include units defined by **UnitDefinition** object. This means that the units definition system in SBML is not hierarchical: user-defined units cannot be built on top of other user-defined units, only on top of base units. SBML differs from CellML (Hedley et al., 2001) in this respect; CellML allows the construction of hierarchical unit definitions.

The `exponent`, `scale` *and* `multiplier` *attributes*

The optional `exponent` attribute on **Unit** represents an exponent on the unit. Its default value is "`1`" (one). A **Unit** object also has an optional `scale` attribute; its value must be an integer exponent for a power-of-ten multiplier used to set the scale of the unit. For example, a unit having a `kind` value of "`gram`" and a `scale` value of "`-3`" signifies $10^{-3} \times gram$, or milligrams. The default value of `scale` is "`0`" (zero), because $10^0 = 1$. Lastly, the optional `multiplier` attribute can be used to multiply the `kind` unit by a real-numbered factor; this enables the definition of units that are not power-of-ten multiples of SI units. For instance, a `multiplier` of 0.3048 could be used to define "`foot`" as a measure of length in terms of a metre. The `multiplier` attribute has a default value of "`1`" (one).

The unit system allows model quantities to be expressed in units other than the base units of Table 1. For analyses and computations, the consumer of the model (be it a software tool or a human) will want to convert all model quantities to base SI units for purposes such as verifying the consistency of units throughout the model. Suppose we begin with a quantity having numerical value y when expressed in units $\{u\}$. The relationship between y and a quantity y_b expressed in base units $\{u_b\}$ is

$$y_b\,\{u_b\} = y\,\{u\}\left(\frac{w\,\{u_b\}}{\{u\}}\right) \tag{1}$$

The term in the parentheses on the right-hand side is a factor w for converting a quantity in units $\{u\}$ to another quantity in units $\{u_b\}$. The ratio of units leads to canceling of $\{u\}$ in the equation above and leaves a quantity in units $\{u_b\}$. It remains to define this factor. In terms of the SBML unit system, it is:

$$\{u\} = (\texttt{multiplier} \cdot 10^{\texttt{scale}}\,\{u_b\})^{\texttt{exponent}} \tag{2}$$

where the dot ($\cdot$) represents simple scalar multiplication. The variables `multiplier`, `scale`, and `exponent` in the equation above correspond to the attributes with the same names in the **Unit** object defined in Figure 12. The exponent in the equation above may make it more difficult to grasp the relationship immediately; so let us suppose for the moment that `exponent`="`1`". Then, it is easy to see that

$$\{u\} = \texttt{multiplier} \cdot 10^{\texttt{scale}}\,\{u_b\}$$

Dividing both sides by $\{u\}$ produces the ratio in the parenthesized portion of Equation 1, which means that $w = \texttt{multiplier} \cdot 10^{\texttt{scale}}$. To take a concrete example, one foot expressed in terms of the metre (a base unit) requires `multiplier`="`0.3048`", `exponent`="`1`", and `scale`="`0`":

$$\text{foot} = 0.3048 \cdot 10^{0} \cdot \text{metre}$$

leading to a conversion between quantities of

$$y_b\,\text{metres} = 0.3048 \cdot y\,\text{feet}$$

Given a quantity of, say, $y = 2$, the conversion results in $y_b = 0.6096$. To relate this to SBML terms more concretely, the following fragment of SBML illustrates how this is represented using the **Unit** and **UnitDefinition** constructs:

```
<listOfUnitDefinitions>
    <unitDefinition id="foot">
        <listOfUnits>
            <unit kind="metre" multiplier="0.3048"/>
        </listOfUnits>
    </unitDefinition>
</listOfUnitDefinitions>
```

The case above is the simplest possible situation, involving the transformation of quantities from a single defined unit $\{u\}$ into a quantity expressed in a single base unit $\{u_b\}$. If, instead, multiple base units $\{u_{b_1}\}, \{u_{b_2}\}, \ldots, \{u_{b_n}\}$ are involved, the following equation holds (where the m_i terms are the `multiplier` values, the s_i terms are the `scale` values, and the x_i terms are the `exponent` values):

$$\begin{aligned}\{u\} &= (m_1 \cdot 10^{s_1}\{u_{b_1}\})^{x_1} \cdot (m_2 \cdot 10^{s_2}\{u_{b_2}\})^{x_2} \cdot \ldots \cdot (m_n \cdot 10^{s_n}\{u_{b_n}\})^{x_n} \\ &= m_1^{x_1} \cdot m_2^{x_2} \cdot \ldots \cdot m_n^{x_n} \cdot 10^{(s_1x_1 + s_2x_2 + \ldots + s_nx_n)}\{u_{b_1}\}^{x_1}\{u_{b_2}\}^{x_2}\ldots\{u_{b_n}\}^{x_n}\end{aligned} \tag{3}$$

Software developers should take care to track the exponents carefully because they can be negative integers. The overall use of Equation 3 is analogous to that of Equation 2, and leads to the following final expression. First, to simplify, let

$$\begin{aligned} m &= m_1^{x_1} \cdot m_2^{x_2} \cdot \ldots \cdot m_n^{x_n} \\ p &= s_1x_1 + s_2x_2 + \ldots + s_nx_n \end{aligned}$$

then,

$$y_b\,\{u_{b_1}\}^{x_1}\{u_{b_2}\}^{x_2}\ldots\{u_{b_n}\}^{x_n} = y\,\{u\}\left(\frac{m \cdot 10^{p}\{u_{b_1}\}^{x_1}\{u_{b_2}\}^{x_2}\ldots\{u_{b_n}\}^{x_n}}{\{u\}}\right) \tag{4}$$

Some additional points are worth discussing about the unit scheme introduced so far. First, and most importantly, the equations above are formulated with the assumption that the base units do not require

an additive offset as part of their definition. *When temperature values in units other than kelvin are being considered, then a different interpretation must be made*, as discussed below.

A second point is that care is needed to avoid seemingly-obvious but incorrect translations of units described in textbooks. The scheme above makes it easy to formulate statements such as "1 foot = 0.3048 metres" in the most natural way. However, the most common expression of the relationship between temperature in Fahrenheit and kelvin, "$T_{Fahrenheit} = 1.8 \cdot (T_{kelvin} - 273.15) + 32$" might lead one to believe that defining Fahrenheit degrees in terms of kelvin degrees involves using `multiplier`="`1.8`". *Not so*, when degree changes are being considered and not temperature values. Converting *temperature values* is different from expressing a relationship between degree measurements. The proper value for the multiplier in the latter case is 5/9, i.e., `multiplier`="`0.555556`" (where we picked an arbitrary decimal precision). If, on the other hand, the actual temperature is relevant to a quantity (e.g., if a model uses a quantity that has particular values at particular temperatures), then offsets are required in the unit calculations and a formula must be used as discussed above.

Handling units requiring the use of offsets in SBML Level 2 Version 5

Unit definitions and conversions requiring offsets cannot be done using the simple approach above. The most general case, involving offsets, multipliers and exponents, requires a completely different approach to defining units than what has been presented up to this point.

In previous versions of SBML, not only was the general case incorrectly presented (i.e., in the same terms described above, when in reality a different approach is required), but few if any developers even attempted to support offset-based units in their software. In the development of SBML Level 2 Version 2, a consensus among SBML developers emerged that a fully generalized unit scheme is so confusing and complicated that it actually impedes interoperability. SBML Level 2 Versions 2–4 acknowledge this reality by reducing and simplifying the unit system, specifically by removing the `offset` attribute on **Unit** and Celsius as a predefined unit, and by describing approaches for handling Celsius and other temperature units. This is a backwards-incompatible change relative to SBML Level 2 Version 1 and SBML Level 1 Version 2, but it is believed to have limited real-life impact because so few tools and models appeared to have employed this feature anyway. By simplifying the unit system to the point that it only involves multiplicative factors as described above, we expect that more software tools will be able to support the SBML unit system from this point forward, ultimately improving interoperability.

We first address the question of how to handle units that *do* require offsets:

- *Handling Celsius.* A model in which certain quantities are temperatures measured in degrees Celsius can be converted straightforwardly to a model in which those temperatures are in kelvin. A software tool could do this by performing a straightforward substitution using the following relationship:

$$T_{kelvin} = T_{Celsius} + 273.15 \tag{5}$$

  In every mathematical formula of the model where a quantity (call it x) in degrees Celsius appears, replace x with $x_k + 273.15$ where x_k is now in kelvin. An alternative approach would be to use a **FunctionDefinition** to define a function encapsulating this relationship above and then using that in the rest of the model as needed. Since Celsius is a commonly-used unit, software tools could help users by providing users with the ability to express temperatures in Celsius in the tools' interfaces, and making substitutions automatically when writing out the SBML.

- *Handling other units requiring offsets.* The only other units requiring offsets in SBML's domain of common applications are other temperature units such as Fahrenheit. Few modern scientists employ Fahrenheit degrees; therefore, this is an unusual situation. The complication inherent in converting between degrees Fahrenheit and kelvin is that both a multiplier and an offset are required:

$$T_{kelvin} = \frac{T_F + 459.67}{1.8} \tag{6}$$

  One approach to handling this is to use a **FunctionDefinition** to define a function encapsulating the

relationship above, then to substitute a call to this function wherever the original temperature in Fahrenheit appears in the model's mathematical formulas. Here is a candidate definition as an example:

```
<functionDefinition id="Fahrenheit_to_kelvin">
    <math xmlns="http://www.w3.org/1998/Math/MathML">
        <lambda>
            <bvar><ci> temp_in_fahrenheit </ci></bvar>
            <apply>
                <divide/>
                <apply>
                    <plus/>
                    <ci> temp_in_fahrenheit </ci>
                    <cn> 459.67 </cn>
                </apply>
                <cn> 1.8 </cn>
            </apply>
        </lambda>
    </math>
</functionDefinition>
```

An alternative approach not requiring the use of function definitions is to use an **AssignmentRule** for each variable in Fahrenheit units. The **AssignmentRule** could compute the conversion from Fahrenheit to (say) kelvin, assign its value to a variable (in Kelvin units), and then that variable could be used elsewhere in the model. Still another approach is to rewrite the mathematical formulas of a model to directly incorporate the conversion Equation 6 wherever the quantity appears.

All of these approaches provide general solutions to the problem of supporting any units requiring offsets in the unit system of SBML Level 2 Versions 2–4. It can be used for other temperature units requiring an offset (e.g., degrees Rankine, degrees Réaumur), although the likelihood of a real-life model requiring such other temperature units seems exceedingly small.

In summary, the removal of `offset` does not impede the creation of models using alternative units. If conversions are needed, then converting between temperature in degrees Celsius and thermodynamic temperature can be handled rather easily by the simple substitution described above. For the rarer case of Fahrenheit and other units requiring combinations of multipliers and offsets, users are encouraged to employ the power of **FunctionDefinition**, **AssignmentRule**, or other constructs in SBML.

Examples

The following example illustrates the definition of an abbreviation "`mmls`" for the units $mmol\ l^{-1}\ s^{-1}$:

```
<listOfUnitDefinitions>
    <unitDefinition id="mmls">
        <listOfUnits>
            <unit kind="mole"   scale="-3"/>
            <unit kind="litre"  exponent="-1"/>
            <unit kind="second" exponent="-1"/>
        </listOfUnits>
    </unitDefinition>
</listOfUnitDefinitions>
```

4.4.3 Predefined units

There are five special unit identifiers in SBML, listed in Table 2 on the following page, corresponding to the five types of quantities that can play roles in SBML reactions: substance, volume, area, length and time. All SBML mathematical entities apart from parameters have default units drawn from these predefined values. Table 2 lists the default values; all of the defaults have `multiplier`="`1`" and `scale`="`0`".

Redefinition of predefined units

Table 2 also lists alternative base units that are allowed as the basis of redefined values. For example, a redefinition of the predefined unit of time must be based on units of seconds. Within certain limits, a model

Identifier	Possible Scalable Units	Default Units
`substance`	mole, item, gram, kilogram, dimensionless	mole
`volume`	litre, cubic metre, dimensionless	litre
`area`	square metre, dimensionless	square metre
`length`	metre, dimensionless	metre
`time`	second, dimensionless	second

Table 2: *SBML's predefined units. The identifiers in the left-most column are values of* `UnitSId` *(Section 3.1.8).*

may change the predefined units by reassigning the keywords **`substance`**, **`length`**, **`area`**, **`time`**, and **`volume`** in a **UnitDefinition**. The limitations on redefinitions of base units are the following:

1. The **UnitDefinition** involving a predefined unit can only contain a single **Unit** object within it.
2. The value of the **`kind`** attribute in a **Unit** instance must be drawn from one of the values in the second column of the appropriate row of Table 2.

Certain choices of base units as values of the **`kind`** attribute require specific choices of the **`exponent`** attribute in the unit definition. Specifically, if volume is involved and metre is chosen as the **`kind`**, the **`exponent`** must be "`3`" to get cubic metres; if area is involved and metre is again used for **`kind`**, the **`exponent`** must be "`2`". Conversely, if **`dimensionless`** is used as the value of **`kind`**, then the **`exponent`** value can be any integer.

Examples

The following example illustrates how to change a model's global default units of volume to be millilitres. If this definition appeared in a model, the units of volume on all components that did not explicitly specify different units would be changed to millilitres.

```
<model>
    ...
    <listOfUnitDefinitions>
        <unitDefinition id="volume">
            <listOfUnits>
                <unit kind="litre" scale="-3"/>
            </listOfUnits>
        </unitDefinition>
    </listOfUnitDefinitions>
    ...
</model>
```

4.4.4 References to units

An attribute that defines the units of a mathematical entity (e.g., the attribute **`units`** on **Parameter**) can refer to a defined unit whose identifier is chosen from among the following:

- A new unit identifier defined by a **UnitDefinition** as described at the start of Section 4.4;
- The base units listed in Table 1 on page 38; and
- The predefined units defined in Section 4.4.3 and listed in Table 2. (These are "**`substance`**", "**`volume`**", "**`area`**", "**`length`**", and "**`time`**".)

Software developers are asked to pay special attention to the units used in an SBML model. Different users and developers sometimes are accustomed to making different assumptions about units, and these assumptions may not correspond to what is actually defined in SBML. The numerical values in a model become meaningless if the corresponding units are not those being assumed. Sections 3.4.3, 4.8.5 and 4.13.5 have particularly important notes about the usage of units in SBML.

4.5 Compartment types

A *compartment type* in SBML is a grouping construct used to establish a relationship between multiple *compartments* (Section 4.7). A compartment type is represented by the **CompartmentType** object class, defined in Figure 13.

In SBML Level 2 Version 3 and beyond, a compartment type only has an identity, and this identity can only be used to indicate that particular compartments belong to this type. This may be useful for conveying a modeling intention, such as when a model contains many similar compartments, either by their biological function or the reactions they carry; without a compartment type construct, it would be impossible in the language of SBML to indicate that all of the compartments share an underlying conceptual relationship because each SBML compartment must be given a unique and separate identity. Compartment types have no mathematical meaning in SBML Level 2 Version 5—they have no effect on a model's mathematical interpretation. Simulators and other numerical analysis software may ignore **CompartmentType** objects and references to them in a model.

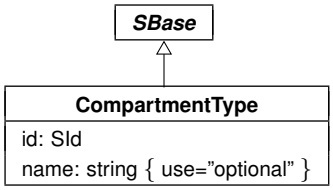


Figure 13: *The definition of class **CompartmentType**. A sequence of one or more instances of **CompartmentType** objects can be located in an instance of **ListOfCompartmentTypes** in **Model**, as shown in Figure 10.*

There is no mechanism in SBML for representing hierarchies of compartment types. One **CompartmentType** object instance cannot be the subtype of another **CompartmentType** object; SBML provides no means of defining such relationships.

4.5.1 The `id` and `name` attributes

As with other major class of objects in SBML, **CompartmentType** has a mandatory attribute, `id`, used to give the compartment type an identifier. The identifier must be a text string conforming to the syntax permitted by the `SId` data type described in Section 3.1.7. **CompartmentType** also has an optional `name` attribute, of type `string`. The `name` and `id` attributes must be used as described in Section 3.3.

4.5.2 The `sboTerm` attribute

CompartmentType inherits an optional `sboTerm` attribute of type `SBOTerm` from its parent class ***SBase*** (see Sections 3.1.9 and 5). When a value is given to this attribute in a **CompartmentType** instance, the value should be an SBO identifier referring to a material entity (i.e., terms derived from `SBO:0000240`, "material entity"). The relationship is of the form "the compartment type *is a* X", where X is the SBO term. The term chosen should be the most precise (narrow) one that captures the role of the compartment type in the model.

4.5.3 Examples

The following partial SBML example illustrates a compartment type used to relate together many individual compartments in a hypothetical model.

```
<model>
    ...
    <listOfCompartmentTypes>
        <compartmentType id="mitochondria"/>
    </listOfCompartmentTypes>
    <listOfCompartments>
        <compartment id="m1" size="0.013" compartmentType="mitochondria" outside="cell"/>
        <compartment id="m2" size="0.013" compartmentType="mitochondria" outside="cell"/>
        <compartment id="m3" size="0.013" compartmentType="mitochondria" outside="cell"/>
        <compartment id="cell" size="190.0"/>
    </listOfCompartments>
    ...
</model>
```

4.6 Species types

The term *species type* refers to reacting entities independent of location. These include simple ions (e.g., protons, calcium), simple molecules (e.g., glucose, ATP), large molecules (e.g., RNA, polysaccharides, and proteins), and others. The **SpeciesType** object class is intended to represent these entities. Its definition is shown in Figure 14.

SpeciesType objects are included in SBML to enable **Species** (Section 4.8) of the same type to be related together. It is a conceptual construct; the existence of **SpeciesType** objects in a model has no effect on the model's numerical interpretation. Except for the requirement for uniqueness of species/species type combinations located in compartments (described in Section 4.8.2), simulators and other numerical analysis software may ignore **SpeciesType** structures and references to them in a model.

There is no mechanism in SBML for representing hierarchies of species types. One **SpeciesType** object instance cannot be the subtype of another **SpeciesType**; SBML provides no means of defining such relationships.

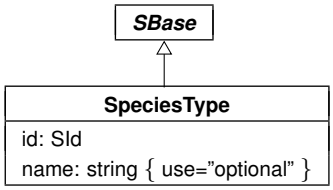


Figure 14: *The definition of class **SpeciesType**. A sequence of one or more instances of **SpeciesType** objects can be located in an instance of **ListOfSpeciesTypes** in **Model**, as shown in Figure 10.*

4.6.1 The `id` and `name` attributes

As with other major object classes in SBML, **SpeciesType** has a mandatory attribute, `id`, used to give the species type an identifier. The identifier must be a text string conforming to the syntax permitted by the `SId` data type described in Section 3.1.7. **SpeciesType** also has an optional `name` attribute, of type `string`. The `name` and `id` attributes must be used as described in Section 3.3.

4.6.2 The `sboTerm` attribute

SpeciesType inherits an optional `sboTerm` attribute of type `SBOTerm` from its parent class ***SBase*** (see Sections 3.1.9 and 5). When a value is given to this attribute in a **SpeciesType** instance, the value should be an SBO identifier referring to a material entity (i.e., terms derived from `SBO:0000240`, "material entity"). The relationship is of the form "the species type *is a* X", where X is the SBO term. The term chosen should be the most precise (narrow) one that captures the role of the species type in the model.

4.6.3 Example

The following XML fragment is an example of two **SpeciesType** objects embedded in an SBML model.

```
<model>
    ...
     <listOfSpeciesTypes>
         <speciesType id="ATP"/>
     </listOfSpeciesTypes>
     <listOfCompartments>
         <compartment id="cytosol"/>
         <compartment id="mitochon"/>
     </listOfCopartments>
     <listOfSpecies>
         <species id="ATPc" speciesType="ATP" compartment="cytosol"  initialConcentration="1"/>
         <species id="ATPm" speciesType="ATP" compartment="mitochon" initialConcentration="2"/>
     </listOfSpecies>
    ...
</model>
```

4.7 Compartments

A *compartment* in SBML represents a bounded space in which species are located. Compartments do not necessarily have to correspond to actual structures inside or outside of a biological cell, although models are often designed that way. The definition of **Compartment** is shown in Figure 15.

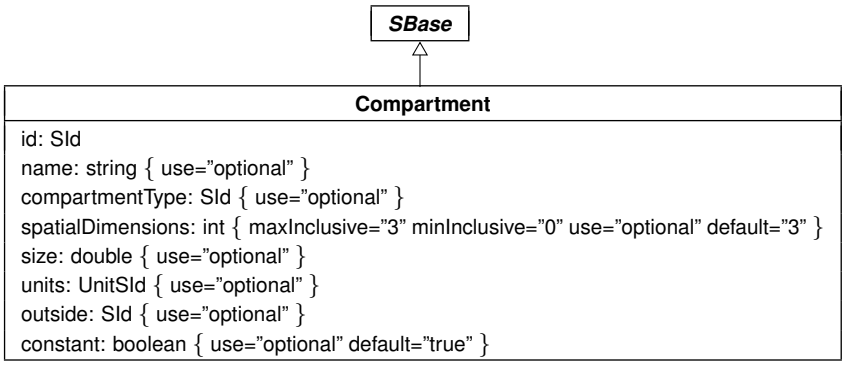


Figure 15: *The definition of class **Compartment**. A sequence of one or more instances of **Compartment** objects can be located in an instance of **ListOfCompartments** in **Model**, as shown in Figure 10.*

It is important to note that although compartments are optional in the overall definition of **Model** (see Section 4.2), every species in an SBML model must be located in a compartment. This in turn means that if a model defines any species, the model must also define at least one compartment. The reason is simply that species represent physical things, and therefore must exist *somewhere*. Compartments represent the *somewhere*.

4.7.1 *The* `id` *and* `name` *attributes*

Compartment has one required attribute, `id`, of type `SId`, to give the compartment a unique identifier by which other parts of an SBML model definition can refer to it. A compartment can also have an optional `name` attribute of type `string`. Identifiers and names must be used according to the guidelines described in Section 3.3.

4.7.2 *The* `compartmentType` *attribute*

Each compartment in a model may optionally be designated as belonging to a particular compartment type. The optional attribute `compartmentType` of type `SId` is used identify the compartment type represented by the **Compartment** object. The `compartmentType` attribute's value must be the identifier of a **CompartmentType** instance defined in the model. If the `compartmentType` attribute is not present on a particular compartment definition, a unique virtual compartment type is assumed for that compartment, and no other compartment can belong to that compartment type.

The values of `compartmentType` attributes on compartments have no effect on the numerical interpretation of a model. Simulators and other numerical analysis software may ignore `compartmentType` attributes.

4.7.3 *The* `spatialDimensions` *attribute*

A **Compartment** object has an optional attribute `spatialDimensions`, whose value must be a positive integer indicating the number of spatial dimensions possessed by the compartment. The maximum value is "`3`", meaning a three-dimensional structure (a volume). Other permissible values are "`2`" (for a two-dimensional area), "`1`" (for a one-dimensional curve), and "`0`" (for a point). The default value is "`3`". Note that the number of spatial dimensions possessed by a compartment affects certain aspects of the compartment's size and units-of-size; see the following two subsections.

4.7.4 *The* `size` *attribute*

Each compartment has an optional floating-point attribute named `size`, representing the initial total size of the compartment. The size may be a volume (if the compartment is a three-dimensional one), or it may be an area (if the compartment is two-dimensional), or a length (if the compartment is one-dimensional).

It is important to note that in SBML Level 2, a missing `size` value *does not imply that the compartment size is 1*. There is no default value of compartment size. (This is unlike the definition of compartment `volume` in SBML Level 1.) When the `spatialDimensions` attribute does not have a value of "`0`", a missing value for `size` for a given compartment signifies that the value either is unknown, or to be obtained from an external source, or determined by an initial assignment (Section 4.10) or a rule (Section 4.11) elsewhere in the model. The `size` attribute must not be present if the `spatialDimensions` attribute has a value of "`0`"; otherwise, a logical inconsistency would exist because a zero-dimensional object cannot have a physical size.

A compartment's size is set by its `size` attribute exactly once. If the compartment's `constant` attribute value is "`true`" (the default), then the size is fixed and cannot be changed except by an **InitialAssignment** in the model (and if `spatialDimensions`="`0`", it cannot be changed by any **InitialAssignment** either). These methods of setting the size differ in that the `size` attribute can only be used to set the compartment size to a literal scalar value, whereas **InitialAssignment** allows the value to be set using an arbitrary mathematical expression. If the compartment's `constant` attribute is "`false`", the size value may be overridden by an **InitialAssignment** or changed by an **AssignmentRule** or **AlgebraicRule**, and in addition, for simulation time $t > 0$, it may also be changed by a **RateRule** or **Event**s. (However, some constructs are mutually exclusive; see Sections 4.11 and 4.14.) It is not an error to set the value of `size` on a compartment and also redefine the value using an **InitialAssignment**, but the original `size` value in that case is ignored. Section 3.4.8 provides additional information about the semantics of assignments, rules and values for simulation time $t \leq 0$.

For the reasons given above, the `size` attribute on a compartment must be defined as optional; however, *it is extremely good practice to specify values for compartment sizes* when such values are available. There are three major technical reasons for this. First, if the model contains any species whose initial amounts are given in terms of concentrations, and there is at least one reaction in the model referencing such a species, then the model is numerically incomplete if it lacks a value for the size of the compartment in which the species is located. The reason is simply that SBML **Reaction**s are defined in units of *substance*/*time* (see Section 4.13.5), not concentration per time, and thus the compartment size must at some point be used to convert from species concentration to substance units. Second, models ideally should be instantiable in a variety of simulation frameworks. A commonly-used one is the discrete stochastic framework (Gillespie, 1977; Wilkinson, 2006) in which species are represented as item counts (e.g., molecule counts). If species' initial quantities are given in terms of concentrations or densities, it is impossible to convert the values to item counts without knowing compartment sizes. Third, if a model contains multiple compartments whose sizes are not all identical to each other, it is impossible to quantify the reaction rate expressions without knowing the compartment volumes. The reason for the latter is again that reaction rates in SBML are defined in terms of *substance*/*time*, and when species quantities are given in terms of concentrations or densities, the compartment sizes become factors in the reaction rate expressions.

A final question to address is, what are the relationships between compartment sizes when compartment positioning is expressed using the `outside` attribute (Section 4.7.7)? The answer is: none. The size of a given compartment does not in any sense include the sizes of other compartments having it as the value of their `outside` attributes. In other words, if a compartment B has the identifier of compartment A as its `outside` attribute value, the size of A does not include the size of B. The compartment sizes are separate.

4.7.5 *The* `units` *attribute*

The units associated with the compartment's `size` value may be set using the optional attribute `units`. The default units, and the kinds of units allowed as values of the attribute `units`, interact with the number of spatial dimensions of the compartment. The value of the `units` attribute of a **Compartment** object must be one of the base units from Table 1, or the predefined unit identifiers "`volume`", "`area`", "`length`" or "`dimensionless`", or a new unit defined by a unit definition in the enclosing model, subject to the restrictions detailed in Table 3.

Value of attribute spatialDimensions	size allowed?	units allowed?	Allowable kinds of units	Default value of attribute units
"3"	yes	yes	units of volume, or dimensionless	"volume"
"2"	yes	yes	units of area, or dimensionless	"area"
"1"	yes	yes	units of length, or dimensionless	"length"
"0"	no	no	(no units allowed)	

Table 3: *The units permitted for compartment sizes. If* spatialDimensions="0"*, the compartment's* units *attribute must be left unset.* Units of volume *means litres, cubic metres, or units derived from them;* units of area *means square metres or units derived from square metres; and* units of length *means metres or units derived from metres. (See also Table 2 on page 42 and Table 1 on page 38.)*

The units of the compartment size, as defined by the **units** attribute or (if **units** is not set) the default value listed in Table 3, are used in the following ways when the compartment has a **spatialDimensions** value greater than "0":

- The value of the **units** attribute is used as the units of the compartment identifier when the identifier appears as a numerical quantity in a mathematical formula expressed in MathML (discussed in Section 3.4.3).
- The **math** element of an **AssignmentRule** or **InitialAssignment** referring to this compartment should have identical units (see Sections 4.11.3 and 4.10).
- In **RateRule** objects that set the rate of change of the compartment's size (Section 4.11.4), the units of the rule's **math** element should be identical to the compartment's **units** attribute divided by the default *time* units. (In other words, the units for the rate of change of compartment size are *compartment size*/*time* units.)
- When a **Species** is to be treated in terms of concentrations or density, the units of the spatial size portion of the concentration value (i.e., the denominator in the units formula *substance*/*size*) are those indicated by the value of the **units** attribute on the compartment in which the species is located.

Compartments with **spatialDimensions**="0" require special treatment in this framework. If a compartment has no size or dimensional units, how should such a compartment's identifier be interpreted when it appears in mathematical formulas? The answer is that such a compartment's identifier must not appear in mathematical formulas in the first place—it has no value, and its value cannot change (Section 4.7.6). Note also that a zero-dimensional compartment is a point, and species located at points can only be described in terms of amounts, not spatially-dependent measures such as concentration. Since SBML **KineticLaw** formulas are already in terms of *substance*/*time* and not (say) *concentration*/*time*, volume or other factors in principle are not needed for species located in zero-dimensional compartments.

4.7.6 *The* constant *attribute*

A **Compartment** also has an optional boolean attribute called **constant** that indicates whether the compartment's size stays constant or can vary during a simulation. A value of "false" indicates the compartment's **size** can be changed by other constructs in SBML. A value of "true" indicates the compartment's **size** cannot be changed by any other construct except **InitialAssignment**. In the special case of **spatialDimensions**="0", the value cannot be changed by **InitialAssignment** either. The default value for the **constant** attribute is "true" because in the most common modeling scenarios at the time of this writing, compartment sizes remain constant. The **constant** attribute must default to or be set to "true" if the value of the **spatialDimensions** attribute is "0", because a zero-dimensional compartment cannot ever have a size.

4.7.7 *The* outside *attribute*

The optional attribute **outside** of type **SId** can be used to express one type of positioning relationship between compartments. If present, the value of **outside** for a given compartment must be the **id** attribute

value of another compartment defined in the model. Doing so means that the other compartment surrounds it or is "outside" of it. This enables the representation of simple topological relationships between compartments, for those simulation systems that can make use of the information (e.g., for drawing simple diagrams of compartments).

There are two restrictions on the inside/outside relationships in SBML. First, because a compartment with `spatialDimensions` of "`0`" has no size, such a compartment cannot act as the "outside" of any other compartment *except* compartments that *also* have `spatialDimensions` values of "`0`". Second, the directed graph formed by representing **Compartment** objects as vertexes and the `outside` attribute values as edges must be acyclic. The latter condition is imposed to prevent a compartment from being located inside itself.

Although inside/outside relationships are partly taken into account by the compartmental localization of reactants and products, it is not always possible to determine purely from the reaction equations whether one compartment is meant to be located within another. In the absence of a value for `outside`, compartment definitions in SBML Level 2 do not have any implied spatial relationships between each other. For many modeling applications, the transfer of substances described by the reactions in a model sufficiently express the relationships between the compartments. (As discussed in Section 8.1, SBML Level 3 is expected introduce the ability to define geometries and spatial qualities.)

Finally, as mentioned in Section 4.7.4 above, the presence of `outside` attributes in compartment definitions has no implications whatsoever about the sizes (or any other attributes) of the compartments involved. The size of a compartment does not include the sizes of any other compartments having it as the value of their `outside` attributes. The `outside` attribute only provides semantic information and has no impact on mathematical analysis and simulation.

4.7.8 *The* `sboTerm` *attribute*

Compartment inherits an optional `sboTerm` attribute of type `SBOTerm` from its parent class ***SBase*** (see Sections 3.1.9 and 5). When a value is given to this attribute in a **Compartment** instance, the value should be an SBO identifier referring to a material entity (i.e., terms derived from `SBO:0000240`, "material entity"). The relationship is of the form "the compartment *is a* X", where X is the SBO term. The term chosen should be the most precise (narrow) one that captures the role of the compartment in the model.

4.7.9 *Examples*

The following example illustrates two compartments in an abbreviated SBML example of a model definition:

```
<model>
    ...
    <listOfCompartments>
        <compartment id="cytosol" size="2.5"/>
        <compartment id="mitochondria" size="0.3"/>
    </listOfCompartments>
    ...
</model>
```

The following is an example of using `outside` to model a cell membrane. To express that a compartment with identifier "`B`" has a membrane that is modeled as another compartment "`M`", which in turn is located within another compartment "`A`", one would write:

```
<model>
    ...
    <listOfCompartments>
        <compartment id="A"/>
        <compartment id="M" spatialDimensions="2" outside="A"/>
        <compartment id="B" outside="M"/>
    </listOfCompartments>
    ...
</model>
```

4.8 Species

A *species* refers to a pool of reacting entities of a specific *species type* that take part in reactions and are located in a specific *compartment*. The **Species** object class is intended to represent these pools. Its definition is shown in Figure 16. Although the exact definition of **Species** given here has changed from the definition in the specification of SBML Level 2 Version 1 (i.e., through the introduction of **SpeciesType**), the concept represented by **Species** remains the same.

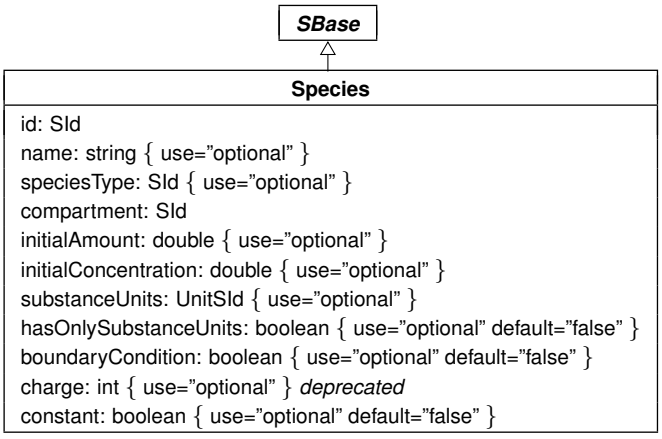


Figure 16: *The definition of class **Species**. A sequence of one or more instances of **Species** objects can be located in an instance of **ListOfSpecies** in **Model**, as shown in Figure 10.*

In previous versions of SBML Level 2, the class **Species** included an attribute called `spatialSizeUnits`, which allowed explicitly setting the units of *size* for initial concentration. SBML Level 2 Version 3 removed this attribute for two reasons. First, its semantics were confusing and introduced an implicit unit conversion. **Compartment** has its own size-setting attribute, and a separate attribute on **Species** meant that a species' initial concentration could involve spatial size units that are different from the units of the compartment in which the species was located. Since (a) the `spatialSizeUnits` attribute determined the size units of a species' quantity when that species appeared in reaction rate formulas, and (b) the compartment may have its own different spatial units, when both the species and the compartment appeared in reaction rate formulas, one of the quantities would have had to be converted into the same spatial units as the other. In other words, modelers and software tools would have had to insert explicit conversion factors into kinetic rate formulas. Such conversions can be be difficult to achieve and prone to errors. Second, although the `spatialSizeUnits` attribute provided some new functionality, it could also be argued to be redundant: a compartment's definition logically *should* be the entity controlling its own spatial size information, in the same way that a species' definition controls its quantity. For these reasons, and because few software packages seemed to take account of the implicit unit conversion, it was deemed better to remove the `spatialSizeUnits` attribute from **Species**.

4.8.1 *The* `id` *and* `name` *attributes*

As with other major objects in SBML, **Species** has a mandatory attribute, `id`, used to give the species an identifier. The identifier must be a text string conforming to the syntax permitted by the `SId` data type described in Section 3.1.7. **Species** also has an optional `name` attribute, of type `string`. The `name` and `id` attributes must be used as described in Section 3.3.

4.8.2 *The* `speciesType` *attribute*

Each species in a model may optionally be designated as belonging to a particular species type. The optional attribute `speciesType` of type `SId` is used to identify the species type of the chemical entities that make up the pool represented by the **Species** object. The attribute's value must be the identifier of an existing **SpeciesType** object. If the `speciesType` attribute is not present on a particular species definition, it means

the pool contains chemical entities of a type unique to that pool; in effect, a virtual species type is assumed for that species, and no other species can belong to that species type.

There can be only one species of a given species type in any given compartment of a model. More specifically, for all **Species** objects having a value for the `speciesType` attribute, the pair

(`speciesType` attribute value, `compartment` attribute value)

must be unique across the set of all **Species** objects in a model.

The value of `speciesType` attributes on species have no effect on the numerical interpretation of a model. Simulators and other numerical analysis software may ignore `speciesType` attributes.

4.8.3 *The* `compartment` *attribute*

The required attribute `compartment`, also of type `SId`, is used to identify the compartment in which the species is located. The attribute's value must be the identifier of an existing **Compartment** object. It is important to note that there is no default value for the `compartment` attribute on **Species**; every species in an SBML model must be assigned a compartment, and consequently, a model must define at least one compartment if that model contains any species.

4.8.4 *The* `initialAmount` *and* `initialConcentration` *attributes*

The optional attributes `initialAmount` and `initialConcentration`, both having a data type of `double`, are used to set the initial quantity of the species in the compartment where the species is located. These attributes are mutually exclusive; i.e., *only one* can have a value on any given instance of a **Species** object.

Missing `initialAmount` and `initialConcentration` values implies that their values either are unknown, or to be obtained from an external source, or determined by an initial assignment (Section 4.10) or rule (Section 4.11) elsewhere in the model. In the case where a species' compartment has a `spatialDimensions` value of "`0`", the species cannot have a value for `initialConcentration` because the concepts of "concentration" and "density" break down when a container has zero dimensions.

A species' initial quantity is set by the `initialAmount` or `initialConcentration` attributes exactly once. If the `constant` attribute is "`true`", then the value of the species' quantity is fixed and cannot be changed except by an **InitialAssignment**. These methods differ in that the `initialAmount` and `initialConcentration` attributes can only be used to set the species quantity to a literal scalar value, whereas **InitialAssignment** allows the value to be set using an arbitrary mathematical expression. If the species' `constant` attribute is "`false`", the species' quantity value may be overridden by an **InitialAssignment** or changed by **AssignmentRule** or **AlgebraicRule**, and in addition, for $t > 0$, it may also be changed by a **RateRule** or **Event**s. (However, some constructs are mutually exclusive; see Sections 4.11 and 4.14.) It is not an error to define `initialAmount` or `initialConcentration` on a species and also redefine the value using an **InitialAssignment**, but the `initialAmount` or `initialConcentration` setting in that case is ignored. Section 3.4.8 provides additional information about the semantics of assignments, rules and values for simulation time $t \leq 0$.

The units of the value in the `initialAmount` attribute are set by the `substanceUnits` attribute on **Species**. The units of the value in the `initialConcentration` attribute are *substance*/*size* units. The units of *substance* are those defined in the `substanceUnits`, and the *size* units are those given in the definition of the size of the **Compartment** in which the species is located.

4.8.5 *The* `substanceUnits` *and* `hasOnlySubstanceUnits` *attributes*

The units associated with a species' quantity, referred to as the *units of the species*, are determined via the optional attributes `substanceUnits` and `hasOnlySubstanceUnits`, in combination with the units of the size defined for the compartment object in which the species are located.

The attribute `hasOnlySubstanceUnits` takes on boolean values and defaults to "`false`". This attribute's role is to indicate whether the units of the species, when the species identifier appears in mathematical formulas, are intended to be concentration or amount. Although it may seem as though this intention could

be determined based on whether `initialConcentration` or `initialAmount` is set, the fact that these two attributes are optional means that a separate flag is needed. (Consider the situation where neither is set, and instead the species' quantity is established by an **InitialAssignment** or **AssignmentRule**.)

The possible values of *units of the species* are summarized in Table 4. The *units of the species* are of the form *substance*/*size* units (i.e., *concentration* units, using a broad definition of concentration) if the compartment's `spatialDimensions` is non-zero and `hasOnlySubstanceUnits` has the value "`false`". The *units of the species* are of the form *substance* if `hasOnlySubstanceUnits` has the value "`true`" *or* `spatialDimensions` is zero. (This dependence is due to the fact that a zero-dimensional compartment cannot support concentrations or densities.) The units of *substance* are those defined by the `substanceUnits` attribute, and the *size* units are those of the size of the compartment in which the species is located. This compartment is the one identified by the species' `compartment` attribute.

value of `hasOnlySubstanceUnits`	*units of the species* when `spatialDimensions` is greater than 0	*units of the species* when `spatialDimensions` is 0
`false` (default)	*substance*/*size*	*substance*
`true`	*substance*	*substance*

Table 4: *How to interpret the value of the* ***Species*** `hasOnlySubstanceUnits` *attribute.*

As an aside, we note that treating species in terms of *substance* units (i.e., discrete quantities such as molecule counts) rather than concentrations is common when using discrete stochastic simulation frameworks (Gillespie, 1977; Wilkinson, 2006). The appropriate way of accomplishing this in SBML is to set `hasOnlySubstanceUnits`="`true`" in the species' definitions.

The value chosen for `substanceUnits` must be either a base unit from Table 1 on page 38, a predefined unit from Table 2 on page 42, or a new unit defined by a unit definition in the enclosing model. The chosen units for `substanceUnits` must be be `dimensionless`, `mole`, `item`, `kilogram`, `gram`, or units derived from these. The `substanceUnits` attribute defaults to the the predefined unit "`substance`" shown in Table 2 on page 42.

The *units of the species* are used in the following ways:

- The species identifier has these units when the identifier appears as a numerical quantity in a mathematical formula expressed in MathML (discussed in Section 3.4.3).
- The `math` element of an **AssignmentRule** or **InitialAssignment** referring to this species should have identical units (see Sections 4.11.3 and 4.10).
- In **RateRule** objects that set the rate of change of the species' quantity (Section 4.11.4), the units of the rule's `math` element should be identical to the *units of the species* divided by the model's *time* units.

4.8.6 *The* `constant` *and* `boundaryCondition` *attributes*

The **Species** object has two optional boolean attributes named `constant` and `boundaryCondition`, used to indicate whether and how the amount of that species can vary during a simulation. Table 5 shows how to interpret the combined values of the `boundaryCondition` and `constant` attributes.

Note that while these restrict whether and how the species *amount* changes, the species *concentration* is, in SBML, a derived quantity of the species amount and the size of the **Compartment** in which it resides. That **Compartment** size, and therefore the *concentration* of a **Species** may therefore change irrespective of the `constant` attribute of the **Species**.

By default, when a species is a product or reactant of one or more reactions, its amount is determined by those reactions. In SBML, it is possible to indicate that a given species' quantity is *not* affected by the set of reactions even when that species occurs as a product or reactant; i.e., the species is on the *boundary* of the reaction system, and its quantity is not determined by the reactions. The boolean attribute `boundaryCondition` can be used to indicate this. The value of the attribute defaults to "`false`", indicating the species *is* part of the reaction system.

constant value	boundaryCondition value	Can have assignment or rate rule?	Can be reactant or product?	What can change the species' amount?
true	true	no	yes	(never changes)
false	true	yes	yes	rules and events
true	false	no	no	(never changes)
false	false	yes	yes	reactions *or* rules (but not both), and events

Table 5: *How to interpret the values of the* `constant` *and* `boundaryCondition` *attributes on* **Species**. *Note that column four is specifically about reactants and products and* not *also about species acting as modifiers; the latter are by definition unchanged by reactions.*

The `constant` attribute indicates whether the species' amount can be changed at all, regardless of whether by reactions, rules, or constructs other than **InitialAssignment**. The default value is "`false`", indicating that the species' amount can be changed, since the purpose of most simulations is precisely to calculate changes in species quantities. Note that the initial quantity of a species can be set by an **InitialAssignment** irrespective of the value of the `constant` attribute.

Note that even if a **Species** `constant` attribute is "`true`", it is the *amount* that cannot change, not necessarily the *concentration*. If the size of the **Compartment** that contains this **Species** changes, its concentration will change even as its *amount* remains constant, and it is still valid to set its `constant` attribute to "`true`".

In practice, a `boundaryCondition` value of "`true`" means a differential equation derived from the reaction definitions should not be generated for the species. However, the species' quantity may still be changed by **AssignmentRule**, **RateRule**, **AlgebraicRule**, **Event**, and **InitialAssignment** constructs if its `constant` attribute is "`false`". Conversely, if the species' `constant` attribute is "`true`", then its amount cannot be changed by anything except **InitialAssignment**.

A species having `boundaryCondition`="`false`" and `constant`="`false`" can appear as a product and/or reactant of one or more reactions in the model. If the species is a reactant or product of a reaction, it must not also appear as the target of any **AssignmentRule** or **RateRule** object in the model. If instead the species has `boundaryCondition`="`false`" and `constant`="`true`", then it cannot appear as a reactant or product, or as the target of any **AssignmentRule**, **RateRule** or **EventAssignment** object in the model.

The example model in section 7.6 contains all four possible combinations of the `boundaryCondition` and `constant` attributes on `species` elements. Section 7.7 gives an example of how one can translate into ODEs a model that uses `boundaryCondition` and `constant` attributes.

4.8.7 *The* `charge` *attribute*

The optional attribute `charge` takes an integer indicating the charge on the species (in terms of electrons, not the SI unit coulombs). This may be useful when the species is a charged ion such as calcium (Ca^{2+}). The `charge` attribute in SBML has been deprecated since Level 2 Version 2.

4.8.8 *The* `sboTerm` *attribute*

Species inherits an optional `sboTerm` attribute of type `SBOTerm` from its parent class ***SBase*** (see Sections 3.1.9 and 5). When a value is given to this attribute in a **Species** instance, the value should be an SBO identifier referring to a material entity (i.e., terms derived from `SBO:0000240`, "material entity"). The relationship is of the form "the species *is a* X", where X is the SBO term. The term chosen should be the most precise (narrow) one that captures the role of the species in the model.

4.8.9 *Example*

The following example shows two species definitions within an abbreviated SBML model definition. The example shows that species are listed under the heading `listOfSpecies` in the model:

```
<model>
    ...
    <listOfSpecies>
        <species id="Glucose" compartment="cell" initialConcentration="4"/>
        <species id="Glucose_6_P" compartment="cell" initialConcentration="0.75"/>
    </listOfSpecies>
    ...
</model>
```

4.9 Parameters

A **Parameter** is used in SBML to define a symbol associated with a value; this symbol can then be used in mathematical formulas in a model. By default, parameters have constant value for the duration of a simulation, and for this reason are called "parameters" instead of "variables" in SBML, although in truth, SBML parameters can be either. The definition of **Parameter** is shown in Figure 17.

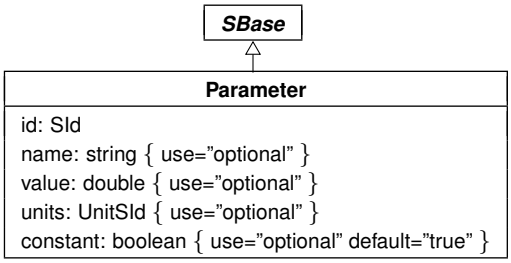


Figure 17: *The definition of class **Parameter**. A sequence of one or more instances of **Parameter** objects can be located in an instance of **ListOfParameters** in **Model**, as shown in Figure 10.*

Parameters can be defined in two places in SBML: in lists of parameters defined at the top level in a **Model** instance, and within individual reaction definitions (as described in Section 4.13). Parameters defined at the top level are *global* to the whole model; parameters that are defined within a reaction are local to the particular reaction and (within that reaction) *override* any global parameters having the same identifiers (See Section 3.3.1 for further details).

The use of the term *parameter* in SBML sometimes leads to confusion among readers who have a particular notion of what something called "parameter" should be. It has been the source of heated debate, but despite this, no one has yet found an adequate replacement term that does not have different connotations to different people and hence leads to confusion among *some* subset of users. Perhaps it would have been better to have two constructs, one called "constants" and the other called "variables". The current approach in SBML is simply more parsimonious, using a single **Parameter** construct with the boolean flag `constant` indicating which flavor it is. In any case, readers are implored to look past their particular definition of a "parameter" and simply view SBML's **Parameter** as a single mechanism for defining both constants and (additional) variables in a model. (We write *additional* because the species in a model are usually considered to be the central variables.) After all, software tools are not required to expose to users the actual names of particular SBML constructs, and thus tools can present to their users whatever terms their designers feel best matches their target audience.

4.9.1 The `id` and `name` attributes

Parameter has one required attribute, `id`, of type `SId`, to give the parameter a unique identifier by which other parts of an SBML model definition can refer to it. A parameter can also have an optional `name` attribute of type `string`. Identifiers and names must be used according to the guidelines described in Section 3.3.

4.9.2 *The* `value` *attribute*

The optional attribute `value` determines the value (of type `double`) assigned to the identifier. A missing `value` implies that the `value` either is unknown, or to be obtained from an external source, or determined by an initial assignment (Section 4.10) or a rule (Section 4.11) elsewhere in the model.

A parameter's value is set by its `value` attribute exactly once. If the parameter's `constant` attribute has the value "`true`" (the default), then the value is fixed and cannot be changed except by an **InitialAssignment**. These two methods of setting the parameter's value differ in that the `value` attribute can only be used to set it to a literal scalar value, whereas **InitialAssignment** allows the value to be set using an arbitrary mathematical expression. If the parameter's `constant` attribute has the value "`false`", the parameter's value may be overridden by an **InitialAssignment** or changed by **AssignmentRule** or **AlgebraicRule**, and in addition, for simulation time $t > 0$, it may also be changed by a **RateRule** or **Event**s. (However, some of these constructs are mutually exclusive; see Sections 4.11 and 4.14.) It is not an error to define `value` on a parameter and also redefine the value using an **InitialAssignment**, but the `value` in that case is ignored. Section 3.4.8 provides additional information about the semantics of assignments, rules and values for simulation time $t \leq 0$.

4.9.3 *The* `units` *attribute*

The units associated with the value of the parameter are specified by the attribute `units`. The value assigned to the parameter's `units` attribute must be chosen from one of the following possibilities: one of the base unit identifiers from Table 1 on page 38; one of the predefined unit identifiers appearing in first column of Table 2 on page 42; or the identifier of a new unit defined in the list of unit definitions in the enclosing **Model** definition. There are no constraints on the units that can be chosen from these sets. There are no default units for parameters.

The units of the parameter are used in the following ways:

- When the parameter identifier appears in mathematical formulas expressed in MathML in a model, the units associated with the value are those declared by the parameter's `units` attribute.
- The units of the `math` element of an **AssignmentRule**, **InitialAssignment** or **EventAssignment** setting the value of the parameter should be identical to the units declared by the parameter's `units` attribute.
- The units of the `math` element of a **RateRule** that references the parameter should be identical to *parameter units*/*time*, where *parameter units* are the units declared for the parameter using the `units` attribute and *time* is the model-wide `time` units.

The fact that parameter units are optional, and that no defaults are defined, means that models can define parameters with undeclared units. If such parameters appear in mathematical expressions elsewhere in a model, it may not be possible for a software tool to verify the consistency of units used in the expressions. Modelers and software tools should therefore assign units to parameters whenever possible.

4.9.4 *The* `constant` *attribute*

The **Parameter** object has an optional boolean attribute named `constant` which indicates whether the parameter's value can vary during a simulation. The attribute's default value is "`true`". A value of "`false`" indicates the parameter's value can be changed by rules (see Section 4.11) and that the `value` is actually intended to be the initial value of the parameter.

Parameters local to a reaction (i.e., those defined within a **Reaction**'s **KineticLaw** object, as described in Section 4.13.5) cannot be changed by rules and therefore are implicitly always constant; thus, parameter definitions within **Reaction** objects must *not* have their `constant` attribute set to "`false`".

What if a global parameter has its `constant` attribute set to "`false`", but the model does not contain any rules, events or other constructs that ever change its value over time? Although the model may be suspect, this situation is not strictly an error. A value of "`false`" for `constant` only indicates that a parameter *can* change value, not that it *must*.

4.9.5 *The* `sboTerm` *attribute*

The **Parameter** object inherits from ***SBase*** the optional `sboTerm` attribute of type `SBOTerm` (see Sections 3.1.9 and 5). When a value is given to this attribute in a parameter definition, the value should be an SBO identifier referring to a quantitative parameter defined in SBO (i.e., terms derived from `SBO:0000002`, "quantitative parameter"). The relationship is of the form "the SBML parameter *is a* X", where X is the SBO term. The term chosen should be the most precise (narrow) one that captures the role of the parameter in the model.

As discussed in Section 5, SBO labels are optional information on a model. Applications are free to ignore `sboTerm` values. A model must be interpretable without the benefit of SBO labels.

4.9.6 *Example*

The following is an example of parameters defined at the **Model** level:

```
<model>
    ...
    <listOfParameters>
        <parameter id="tau2" value="3e-2" units="second"/>
        <parameter id="Km1" value="10.7" units="moleperlitre"/>
    </listOfParameters>
    ...
</model>
```

4.10 Initial assignments

SBML Level 2 Version 5 provides two ways of assigning initial values to entities in a model. The simplest and most basic is to set the values of the appropriate attributes in the relevant components; for example, the initial value of a model parameter (whether it is a constant or a variable) can be assigned by setting its `value` attribute directly in the model definition (Section 4.9). However, this approach is not suitable when the value must be calculated, because the initial value attributes on different components such as species, compartments, and parameters are single values and not mathematical expressions. This is the reason for the introduction of **InitialAssignment**: to permit the calculation of the value of a constant or the initial value of a variable from the values of *other* quantities in a model. The definition of **InitialAssignment** is shown in Figure 18.

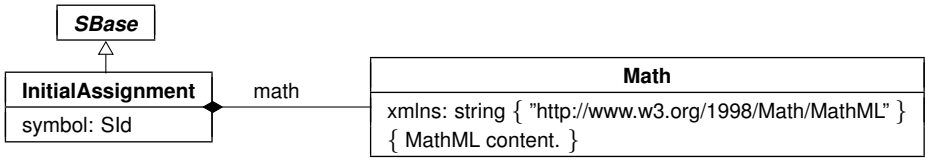


Figure 18: *The definition of class **InitialAssignment**. The contents of the **Math** class can be any MathML permitted in SBML; see Section 3.4.1. A sequence of one or more instances of **InitialAssignment** objects can be located in an instance of **ListOfInitialAssignments** in **Model**, as shown in Figure 10.*

As explained below, the provision of **InitialAssignment** does not mean that models necessarily must use this construct when defining initial values of quantities. If a value can be set using the relevant attribute of a component in a model, then that approach may be more efficient and more portable to other software tools. **InitialAssignment** should be used when the other mechanism is insufficient for the needs of a particular model.

Initial assignments have some similarities to assignment rules (Section 4.11.3). The main differences are (a) an **InitialAssignment** can set the value of a constant whereas an **AssignmentRule** cannot, and (b) unlike **AssignmentRule**, an **InitialAssignment** definition only applies up to and including the beginning of simulation time, i.e., $t \leq 0$, while an **AssignmentRule** applies at all times.

4.10.1 *The* `symbol` *attribute*

InitialAssignment contains the attribute `symbol`, of type `SId`. The value of this attribute in an **InitialAssignment** object can be the identifier (i.e., the value of the `id` attribute) of a **Compartment**, **Species** or global **Parameter** elsewhere in the model. The purpose of the **InitialAssignment** is to define the initial value of the constant or variable referred to by the `symbol` attribute. (The attribute's name is `symbol` rather than `variable` because it may assign values to constants as well as variables in a model; see Section 4.10.4 below.)

An initial assignment cannot be made to reaction identifiers, that is, the `symbol` attribute value of an **InitialAssignment** cannot be an identifier that is the `id` attribute value of a **Reaction** object in the model. This is identical to a restriction placed on rules (see Section 4.11.5).

4.10.2 *The* `math` *element*

The `math` element contains a MathML expression that is used to calculate the value of the constant or the initial value of the variable. The units of the value computed by the formula in the `math` element should be identical to be the units associated with the identifier given in the `symbol` attribute. (That is, the units should be the units of the species, compartment, or parameter, as appropriate for the kind of object identified by the value of `symbol`.)

4.10.3 *The* `sboTerm` *attribute*

InitialAssignment inherits from ***SBase*** an optional `sboTerm` attribute of type `SBOTerm` (see Sections 3.1.9 and 5). When a value is given to this attribute in an initial assignment definition, the value should be a valid SBO identifier referring to a mathematical expression (i.e., terms derived from `SBO:0000064`, "mathematical expression"). The **InitialAssignment** object should have a "is a" relationship with the SBO term, and the term should be the most precise (narrow) term that captures the role of the **InitialAssignment** in the model.

As discussed in Section 5, SBO labels are optional information on a model. Applications are free to ignore `sboTerm` values. A model must be interpretable without the benefit of SBO labels.

4.10.4 *Semantics of initial assignments*

The value calculated by an **InitialAssignment** object overrides the value assigned to the given symbol by the object defining that symbol. For example, if a **Compartment**'s `size` is set in its definition, and the model also contains an **InitialAssignment** having that compartment's `id` as its `symbol` value, then the interpretation is that the `size` assigned in the **Compartment** definition should be ignored and the value assigned based on the computation defined in the **InitialAssignment**. Initial assignments can take place for **Compartment**, **Species** and global **Parameter** objects regardless of the value of their `constant` attribute.

This does not mean that a definition of a symbol can be omitted if there is an **InitialAssignment** object for that symbol; the symbols must always be defined even if they are assigned a value separately. For example, there must be a **Parameter** definition for a given parameter if there is an **InitialAssignment** for that parameter.

The actions of all **InitialAssignment** objects are in general terms the same, but differ in the precise details depending on the type of variable being set:

- *In the case of a species*, an **InitialAssignment** sets the referenced species' initial quantity (*concentration* or *amount of substance*) to the value determined by the formula in `math`. (See Section 4.8.5 for an explanation of how the units of the species' quantity are determined.)
- *In the case of a compartment*, an **InitialAssignment** sets the referenced compartment's initial size to the size determined by the formula in `math`. The overall units of the formula should be the same as the units specified for the size of the compartment. (See Section 4.7.5 for an explanation of how the units of the compartment's size are determined.)
- *In the case of a parameter*, an **InitialAssignment** sets the referenced parameter's initial value to that determined by the formula in `math`. The overall units of the formula should be the same as the units defined for the parameter. (See Section 4.9.3 for an explanation of how the units of the parameter are

determined.)

In the context of a simulation, initial assignments establish values that are in effect prior to and including the start of simulation time, i.e., $t \leq 0$. Section 3.4.8 provides information about the interpretation of assignments, rules, and entity values for simulation time up to and including the start time $t = 0$; this is important for establishing the initial conditions of a simulation if the model involves expressions containing the *delay* `csymbol` (Section 3.4.6).

There cannot be two initial assignments for the same symbol in a model; that is, a model must not contain two or more **InitialAssignment** objects that both have the same identifier as their `symbol` attribute value. A model must also not define initial assignments *and* assignment rules for the same entity. That is, there cannot be *both* an **InitialAssignment** and an **AssignmentRule** for the same symbol in a model, because both kinds of constructs apply prior to and at the start of simulated time—allowing both to exist for a given symbol would result in indeterminism). (See also Section 4.11.5.)

The ordering of **InitialAssignment** objects is not significant. The combined set of **InitialAssignment**, **AssignmentRule** and **KineticLaw** objects form a set of assignment statements that must be considered as a whole. The combined set of assignment statements should not contain algebraic loops: a chain of dependency between these statements should terminate. (More formally, consider the directed graph of assignment statements where nodes are a model's assignment statements and directed arcs exist for each occurrence of a symbol in an assignment statement `math` attribute. The directed arcs in this graph start from the statement assigning the symbol and end at the statement that contains the symbol in their math elements. Such a graph must be acyclic.) Examples of valid and invalid set of assignment statements are given in Section 4.11.5.

Finally, it is worth being explicit about the expected behavior in the following situation. Suppose (1) a given symbol has a value x assigned to it in its definition, and (2) there is an initial assignment having the identifier as its `symbol` value and reassigning the value to y, *and* (3) the identifier is also used in the mathematical formula of a second initial assignment. What value should the second initial assignment use? It is y, the value assigned to the symbol by the first initial assignment, not whatever value was given in the symbol's definition. This follows directly from the behavior at the defined at the beginning of this section and in Section 3.4.8: if an **InitialAssignment** object exists for a given symbol, then the symbol's value is overridden by that initial assignment.

4.10.5 Example

The following example shows how the species "`x`" can assigned the initial value $2 \times y$, where "`y`" is an identifier defined elsewhere in the model:

```
<model>
    ...
    <listOfSpecies>
        <species id="x" initialConcentration="5"/>
    </listOfSpecies>
    ...
    <listOfInitialAssignments>
        <initialAssignment symbol="x">
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <times/>
                    <ci> y </ci>
                    <cn> 2 </cn>
                </apply>
            </math>
        </initialAssignment>
    </listOfInitialAssignments>
    ...
</model>
```

The next example illustrates the more complex behavior discussed above, when a symbol has a value assigned in its definition but there also exists an **InitialAssignment** for it *and* another **InitialAssignment** uses its value

in its mathematical formula.

```
<model>
    ...
    <listOfSpecies>
        <species id="x" initialConcentration="5"/>
    </listOfSpecies>
    ...
    <listOfInitialAssignments>
        <initialAssignment symbol="x">
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <cn> 2 </cn>
            </math>
        </initialAssignment>
        <initialAssignment symbol="othersymbol">
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <times/>
                    <ci> x </ci>
                    <cn> 2 </cn>
                </apply>
            </math>
        </initialAssignment>
    </listOfInitialAssignments>
    ...
</model>
```

The value of "`othersymbol`" in the SBML excerpt above will be "`4`". The case illustrates the rule of thumb that if there is an initial assignment for a symbol, the value assigned to the symbol in its definition must be ignored and the value created by the initial assignment used instead.

4.11 Rules

In SBML, *Rules* provide additional ways to define the values of variables in a model, their relationships, and the dynamical behaviors of those variables. *Rules* enable the encoding of relationships that cannot be expressed using reactions alone (Section 4.13) nor by the assignment of an initial value to a variable in a model (Section 4.10).

SBML separates rules into three subclasses for the benefit of model analysis software. The three subclasses are based on the following three different possible functional forms (where x is a variable, f is some arbitrary function returning a numerical result, $\mathbf{V}$ is a vector of variables that does not include x, and $\mathbf{W}$ is a vector of variables that may include x):

Algebraic	left-hand side is zero:	$0 = f(\mathbf{W})$
Assignment	left-hand side is a scalar:	$x = f(\mathbf{V})$
Rate	left-hand side is a rate-of-change:	$dx/dt = f(\mathbf{W})$

In their general form given above, there is little to distinguish between *assignment* and *algebraic* rules. They are treated as separate cases for the following reasons:

- *Assignment* rules can simply be evaluated to calculate intermediate values for use in numerical methods;
- SBML needs to place restrictions on assignment rules, for example the restriction that assignment rules cannot contain algebraic loops (discussed further in Section 4.11.5);
- Some simulators do not contain numerical solvers capable of solving unconstrained algebraic equations, and providing more direct forms such as assignment rules may enable those simulators to process models they could not process if the same assignments were put in the form of general algebraic equations;
- Those simulators that *can* solve these algebraic equations make a distinction between the different categories listed above; and
- Some specialized numerical analyses of models may only be applicable to models that do not contain *algebraic* rules.

The approach taken to covering these cases in SBML is to define an abstract ***Rule*** object class containing an element, `math`, to hold the right-hand side expression, then to derive subtypes of ***Rule*** that add attributes to distinguish the cases of algebraic, assignment and rate rules. Figure 19 gives the definitions of ***Rule*** and the subtypes derived from it. The figure shows there are three subtypes, **AlgebraicRule**, **AssignmentRule** and **RateRule** derived directly from ***Rule***. These correspond to the cases *Algebraic*, *Assignment*, and *Rate* described above respectively.

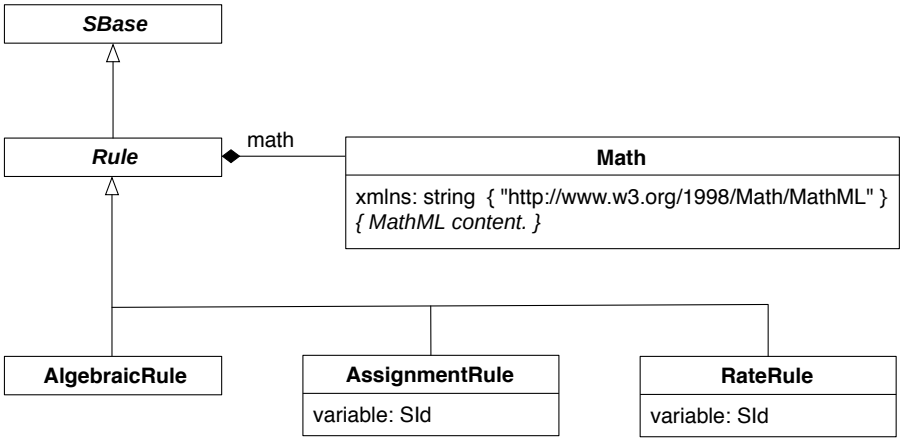


Figure 19: *The definition of **Rule** and derived types **AlgebraicRule**, **AssignmentRule** and **RateRule**.*

4.11.1 Common attributes in Rule

The classes derived from ***Rule*** inherit `math` and the attributes and elements from ***SBase***, including `sboTerm`.

The `math` *element*

A ***Rule*** object has a required element called `math`, containing a MathML expression defining the mathematical formula of the rule. This MathML formula must return a numerical value. The formula can be an arbitrary expression referencing the variables and other entities in an SBML model. The interpretation of `math` and the units of the formula are described in more detail in Sections 4.11.2, 4.11.3 and 4.11.4 below.

The `sboTerm` *attribute*

The ***Rule*** object class inherits from ***SBase*** the optional `sboTerm` attribute of type `SBOTerm` (see Sections 3.1.9 and 5). When a value is given to this attribute, it should be a valid SBO identifier referring to a mathematical expression defined in SBO (i.e., terms derived from `SBO:0000064`, "mathematical expression"). The **AlgebraicRule**, **AssignmentRule**, or **RateRule** object should have a "is a" relationship with the SBO term, and the term should be the most precise (narrow) term that captures the role of that rule in the model.

As discussed in Section 5, SBO labels are optional information on a model. Applications are free to ignore `sboTerm` values. A model must be interpretable without the benefit of SBO labels.

4.11.2 AlgebraicRule

The rule type **AlgebraicRule** is used to express equations that are neither assignments of model variables nor rates of change. **AlgebraicRule** does not add any attributes to the basic ***Rule***; its role is simply to distinguish this case from the other cases. An example of the use of **AlgebraicRule** is given in Section 7.5.

In the context of a simulation, algebraic rules are in effect at all times, $t \geq 0$. For purposes of evaluating expressions that involve the *delay* `csymbol` (Section 3.4.6), algebraic rules are considered to apply also at

$t \leq 0$. Section 3.4.8 provides additional information about the semantics of assignments, rules, and entity values for simulation time $t \leq 0$.

The ability to define arbitrary algebraic expressions in an SBML model introduces the possibility that a model is mathematically overdetermined by the overall system of equations constructed from its rules and reactions. An SBML model must not be overdetermined; this is discussed in Section 4.11.5 below.

4.11.3 AssignmentRule

The rule type **AssignmentRule** is used to express equations that set the values of variables. The left-hand side (the `variable` attribute) of an assignment rule can refer to the identifier of a **Species**, **Compartment**, or **Parameter** object in the model (but not a reaction). The entity identified must not have its `constant` attribute set to "`true`". The effects of an **AssignmentRule** are in general terms the same, but differ in the precise details depending on the type of variable being set:

- *In the case of a species*, an **AssignmentRule** sets the referenced species' quantity (*concentration* or *amount of substance*) to the value determined by the formula in `math`. The units of the formula in `math` should be the same as the *units of the species* (Section 4.8.5) for the species identified by the `variable` attribute of the **AssignmentRule**.

  Restrictions: There must not be both an **AssignmentRule** `variable` attribute and a **SpeciesReference** `species` attribute having the same value, unless that species has its `boundaryCondition` attribute set to "`true`". In other words, an assignment rule cannot be defined for a species that is created or destroyed in a reaction unless that species is defined as a boundary condition in the model.

- *In the case of a compartment*, an **AssignmentRule** sets the referenced compartment's size to the value determined by the formula in `math`. The overall units of the formula in `math` should be the same as the units of the size of the compartment (Section 4.7.5).

- *In the case of a parameter*, an **AssignmentRule** sets the referenced parameter's value to that determined by the formula in `math`. The overall units of the formula in `math` should be the same as the units defined for the parameter (Section 4.9.3).

In the context of a simulation, assignment rules are in effect at all times, $t \geq 0$. For purposes of evaluating expressions that involve the *delay* `csymbol` (Section 3.4.6), assignment rules are considered to apply also at $t \leq 0$. Section 3.4.8 provides additional information about the semantics of assignments, rules, and entity values for simulation time $t \leq 0$.

A model must not contain more than one **AssignmentRule** or **RateRule** object having the same value of `variable`; in other words, in the set of all assignment rules and rate rules in an SBML model, each variable appearing in the left-hand sides can only appear once. This simply follows from the fact that an indeterminate system would result if a model contained more than one assignment rule for the same variable or both an assignment rule and a rate rule for the same variable.

Similarly, a model must also not contain *both* an **AssignmentRule** and an **InitialAssignment** for the same variable, because both kinds of constructs apply prior to and at the start of simulation time, i.e., $t \leq 0$. If a model contained both an initial assignment and an assignment rule for the same variable, an indeterminate system would result. (See also Section 4.10.4.)

The value calculated by an **AssignmentRule** object overrides the value assigned to the given symbol by the object defining that symbol. For example, if a **Compartment**'s `size` is set in its definition, and the model also contains an **AssignmentRule** having that compartment's `id` as its `variable` value, then the `size` assigned in the **Compartment** definition is ignored and the value assigned based on the computation defined in the **AssignmentRule**. This does *not* mean that a definition for a given symbol can be omitted if there is an **AssignmentRule** object for it. For example, there must be a **Parameter** definition for a given parameter if there is an **AssignmentRule** for that parameter.

4.11.4 RateRule

The rule type **RateRule** is used to express equations that determine the rates of change of variables. The left-hand side (the `variable` attribute) can refer to the identifier of a species, compartment, or parameter (but not a reaction). The entity identified must have its `constant` attribute set to "`false`". The effects of a **RateRule** are in general terms the same, but differ in the precise details depending on which variable is being set:

- *In the case of a species*, a **RateRule** sets the rate of change of the species' quantity (*concentration* or *amount of substance*) to the value determined by the formula in `math`. The overall units of the formula in `math` should be *species quantity*/*time*, where the *time* units are the predefined units of time described in Section 4.4 and the *species quantity* units are the *units of the species* as defined in Section 4.8.5.

  Restrictions: There must not be both a **RateRule** `variable` attribute and a **SpeciesReference** `species` attribute having the same value, unless that species has its `boundaryCondition` attribute is set to "`true`". This means a rate rule cannot be defined for a species that is created or destroyed in a reaction, unless that species is defined as a boundary condition in the model.

- *In the case of a compartment*, a **RateRule** sets the rate of change of the compartment's size to the value determined by the formula in `math`. The overall units of the formula should be *size*/*time*, where the *time* units are the predefined units of time described in Section 4.4 and the *size* units are the units of size on the compartment (Section 4.7.5).

- *In the case of a parameter*, a **RateRule** sets the rate of change of the parameter's value to that determined by the formula in `math`. The overall units of the formula should be $x/time$, where x are the units of the parameter (Section 4.9.3).

In the context of a simulation, rate rules are in effect for simulation time $t > 0$. Other types of rules and initial assignments are in effect at different times; Section 3.4.8 describes these conditions.

As mentioned in Section 4.11.3 for **AssignmentRule**, a model must not contain more than one **RateRule** or **AssignmentRule** object having the same value of `variable`; in other words, in the set of all assignment rules and rate rules in an SBML model, each variable appearing in the left-hand sides can only appear once. This simply follows from the fact that an indeterminate system would result if a model contained more than one assignment rule for the same variable or both an assignment rule and a rate rule for the same variable.

4.11.5 Additional restrictions on rules

An important design goal of SBML rule semantics is to ensure that a model's simulation and analysis results will not be dependent on when or how often rules are evaluated. To achieve this, SBML needs to place two additional restrictions on rule use in addition to the conditions described above regarding the use of **AlgebraicRule**, **AssignmentRule** and **RateRule**. The first concerns algebraic loops in the system of assignments in a model, and the second concerns overdetermined systems.

The model must not contain algebraic loops

The combined set of **InitialAssignment**, **AssignmentRule** and **KineticLaw** objects constitute a set of assignment statements that should be considered as a whole. (A **KineticLaw** object is counted as an assignment because it assigns a value to the symbol contained in the `id` attribute of the **Reaction** object in which it is defined.) This combined set of assignment statements must not contain algebraic loops—dependency chains between these statements must terminate. To put this more formally, consider a directed graph in which nodes are assignment statements and directed arcs exist for each occurrence of an SBML species, compartment or parameter symbol in an assignment statement's `math` element. Let the directed arcs point from the statement assigning the symbol to the statements that contain the symbol in their `math` element expressions. This graph must be acyclic.

SBML does not specify when or how often rules should be evaluated. Eliminating algebraic loops ensures that assignment statements can be evaluated any number of times without the result of those evaluations

changing. As an example, consider the following equations:

$$x = x + 1, \quad y = z + 200, \quad z = y + 100$$

If this set of equations were interpreted as a set of assignment statements, it would be invalid because the rule for x refers to x (exhibiting one type of loop), and the rule for y refers to z while the rule for z refers back to y (exhibiting another type of loop).

Conversely, the following set of equations would constitute a valid set of assignment statements:

$$x = 10, \quad y = z + 200, \quad z = x + 100$$

The model must not be overdetermined

An SBML model must not be overdetermined; that is, a model must not define more equations than there are unknowns in a model. An SBML model that does not contain **AlgebraicRule** objects cannot be overdetermined.

Assessing whether a given continuous, deterministic, mathematical model is overdetermined does not require dynamic analysis; it can be done by analyzing the system of equations created from the model. One approach is to construct a bipartite graph in which one set of vertices represents the variables and the other the set of vertices represents the equations. Place edges between vertices such that variables in the system are linked to the equations that determine them. For algebraic equations, there will be edges between the equation and each variable occurring in the equation. For ordinary differential equations (such as those defined by rate rules or implied by the reaction rate definitions), there will be a single edge between the equation and the variable determined by that differential equation. A mathematical model is overdetermined if the maximal matchings (Chartrand, 1977) of the bipartite graph contain disconnected vertexes representing equations. (If one maximal matching has this property, then all the maximal matchings will have this property; i.e., it is only necessary to find one maximal matching.) Appendix D describes a method of applying this procedure to specific SBML data objects.

*The model must not change the value of a zero-dimensional **Compartment***

As described in Section 4.7, a zero-dimensional **Compartment** (i.e. one with a `spatialDimensions` value of "`0`") has no value, and its value may not change. It follows that no **Compartment** identifier of this type may appear as the `variable` of a **RateRule** or an **AssignmentRule**, nor may it appear in the **Math** of an **AlgebraicRule**.

4.11.6 Example of rule use

This section contains an example set of rules. Consider the following set of equations:

$$k = \frac{k_3}{k_2}, \quad s_2 = \frac{k \cdot x}{1 + k_2}, \quad A = 0.10 \cdot x$$

This can be encoded by the following scalar rule set (where the definitions of `x`, `s`, `k`, `k2`, `k3` and `A` are assumed to be located elsewhere in the model and not shown in this abbreviated example):

```
<listOfRules>
    <assignmentRule variable="k">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply> <divide/> <ci> k3 </ci> <ci> k2 </ci> </apply>
        </math>
    </assignmentRule>
    <assignmentRule variable="s2">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
                <divide/>
                    <apply> <times/> <ci> k </ci> <ci> x </ci> </apply>
                    <apply> <plus/> <cn> 1 </cn> <ci> k2 </ci> </apply>
            </apply>
```

```
            </math>
        </assignmentRule>
        <assignmentRule variable="A">
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply> <times/> <cn> 0.10 </cn> <ci> x </ci> </apply>
            </math>
        </assignmentRule>
    </listOfRules>
```

4.12 Constraints

The **Constraint** object is a mechanism for stating the assumptions under which a model is designed to operate. The *constraints* are statements about permissible values of different quantities in a model. Figure 20 shows the definition of the **Constraint** object class.

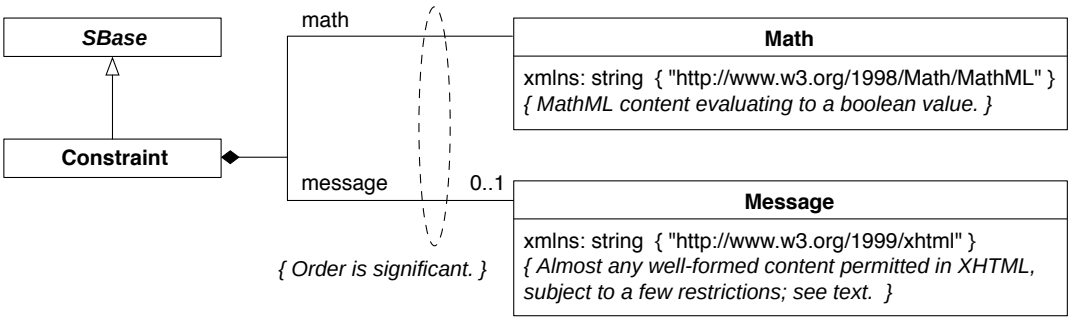


Figure 20: *The definition of class* ***Constraint****. The contents of the* ***Math*** *class can be any MathML permitted in SBML, but it must return a boolean value. As shown above, an instance of* ***Constraint*** *can also contain zero or one instances of* ***Message****; this element is simply a wrapper (in the XML form, within* `<message> ... </message>` *tags) for XHTML content. The same guidelines for XHTML content as explained in Section 3.2.3 for notes on* ***SBase*** *also apply to the XHTML within messages in a* ***Constraint****. A sequence of one or more instances of* ***Constraint*** *objects can be located in an instance of* ***ListOfConstraints*** *in* ***Model****, as shown in Figure 10.*

The essential meaning of a constraint is this: if a dynamical analysis of a model (such as a simulation) reaches a state in which a constraint is no longer satisfied, the results of the analysis are deemed invalid beginning with that point in time. The exact behavior of a software tool, upon encountering a constraint violation, is left up to the software; *however*, a software tool must somehow indicate to the user when a model's constraints are no longer satisfied. (Otherwise, a user may not realize that the analysis has reached an invalid state and is potentially producing nonsense results.) If a software tool does not have support for constraints, it should indicate this to the user when encountering a model containing constraints.

4.12.1 *The* `math` *element*

Constraint has one required subelement, `math`, containing a MathML formula defining the condition of the constraint. This formula must return a boolean value of "`true`" when the model is in a *valid* state. The formula can be an arbitrary expression referencing the variables and other entities in an SBML model. The evaluation of `math` and behavior of constraints are described in more detail in Section 4.12.4 below.

4.12.2 *The* `message` *element*

A **Constraint** object has an optional element called `message`. This can contain a message in XHTML format that may be displayed to the user when the condition of the constraint in `math` evaluates to a value of "`false`". Software tools are not required to display the message, but it is recommended that they do so as a matter of best practice.

The XHTML content within a `message` element must follow the same restrictions as for the `notes` element on ***SBase*** described in Section 3.2.3. For example, `message` must not contain an XML declaration or a DOCTYPE declaration, and the permitted content can only take one of the following general forms: (1) a

complete XHTML document beginning with the element <html> and ending with </html>; (2) the "body" portion of a document beginning with the element <body> and ending with </body>; or (3) XHTML content that is permitted within a <body> ... </body> elements. Appendix F describes one approach to reading the message content.

4.12.3 The sboTerm attribute

The Constraint object inherits from *SBase* the optional sboTerm attribute of type SBOTerm (see Sections 3.1.9 and 5). When a value is given to this attribute in a constraint definition, the value should be a valid SBO identifier referring to a mathematical expression (i.e., terms derived from SBO:0000064, "mathematical expression"). The Constraint should have an "is a" relationship with the SBO term, and the term should be the most precise (narrow) term that captures the role of the Constraint in the model.

As discussed in Section 5, SBO labels are optional information on a model. Applications are free to ignore sboTerm values. A model must be interpretable without the benefit of SBO labels.

4.12.4 Semantics of constraints

In the context of a simulation, a Constraint has effect at all times $t \geq 0$. Each Constraint's math element is first evaluated after any InitialAssignment definitions in a model at $t = 0$ and can conceivably trigger at that point. (In other words, a simulation could fail a constraint immediately.)

Constraint definitions *cannot and should not* be used to compute the dynamical behavior of a model as part of, for example, simulation. Constraints may be used as input to non-dynamical analysis, for instance by expressing flux constraints for flux balance analysis.

The results of a simulation of a model containing a constraint are invalid from any simulation time at and after a point when the function given by the math returns a value of "false". Invalid simulation results do not make a prediction of the behavior of the biochemical reaction network represented by the model. The precise behavior of simulation tools is left undefined with respect to constraints. If invalid results are detected with respect to a given constraint, the message element (Section 4.12.2) may optionally be displayed to the user. The simulation tool may also halt the simulation or clearly delimit in output data the simulation time point at which the simulation results become invalid.

SBML does not impose restrictions on duplicate Constraint definitions or the order of evaluation of Constraint objects in a model. It is possible for a model to define multiple constraints all with the same math element. Since the failure of any constraint indicates that the model simulation has entered an invalid state, a system is not required to attempt to detect whether other constraints in the model have failed once any one constraint has failed.

4.12.5 Example

As an example, the following SBML fragment demonstrates the constraint that species S_1 should only have values between 1 and 100:

```
<model>
    ...
    <listOfConstraints>
        <constraint>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <and/>
                        <apply> <lt/> <cn> 1 </cn> <ci> S1 </ci> </apply>
                        <apply> <lt/> <ci> S1 </ci> <cn> 100 </cn> </apply>
                </apply>
            </math>
            <message>
                <p xmlns="http://www.w3.org/1999/xhtml"> Species S1 is out of range. </p>
            </message>
        </constraint>
    </listOfConstraints>
    ...
```

```
</model>
```

4.13 Reactions

A *reaction* represents any transformation, transport or binding process, typically a chemical reaction, that can change the quantity of one or more species. In SBML, a reaction is defined primarily in terms of the participating reactants and products (and their corresponding stoichiometries), along with optional modifier species, an optional rate at which the reaction takes place, and optional parameters. These various parts of a reaction are recorded in the SBML **Reaction** object class and other supporting data classes, defined in Figure 21 on the following page.

4.13.1 Reaction

Each reaction in an SBML model is defined using an instance of a **Reaction** object. As shown in Figure 21 on the next page, it contains several scalar attributes and several lists of objects.

The `id` *and* `name` *attributes*

As with most other main kinds of objects in SBML, the **Reaction** object class includes a mandatory attribute called `id`, of type `SId`, and an optional attribute `name`, of type `string`. The `id` attribute is used to give the reaction a unique identifier in the model. This identifier can be used in mathematical formulas elsewhere in an SBML model to represent the rate of that reaction; this usage is explained in detail in Section 4.13.7 below. The `name` attribute can be used to give the reaction a more free-form, descriptive name. The `name` and `id` attributes must be used as described in Section 3.3.

The lists of reactants, products and modifiers

The species participating as reactants, products, and/or modifiers in a reaction are declared using lists of **SpeciesReference** and/or **ModifierSpeciesReference** instances stored in `listOfReactants`, `listOfProducts` and `listOfModifiers`. **SpeciesReference** and **ModifierSpeciesReference** are described in more detail in Sections 4.13.3 and 4.13.4 below.

Certain restrictions are placed on the appearance of species in reaction definitions:

- The ability of a species to appear as a reactant or product of any reaction in a model is governed by certain flags in that species' definition; see Section 4.8.6 for more information.
- Any species appearing in the mathematical formula of the `kineticLaw` of a **Reaction** instance must be declared in at least one of that **Reaction**'s lists of reactants, products, and/or modifiers. Put another way, it is an error for a reaction's kinetic law formula to refer to species that have not been declared for that reaction.
- A reaction definition can contain an empty list of reactants *or* an empty list of products, but it must have at least one reactant or product; in other words, a reaction without any reactant or product species is not permitted. (This restriction does not apply to modifier species, which remain optional in all cases.)

The `kineticLaw` *element*

A reaction can contain up to one **KineticLaw** object in the `kineticLaw` element of the **Reaction**. This "kinetic law" defines the speed at which the process defined by the reaction takes place. A detailed description of **KineticLaw** is left to Section 4.13.5 below.

Note that the inclusion of a **KineticLaw** object in an instance of a **Reaction** component is optional; however, in general there is no useful default that can be substituted in place of a missing rate expression in a reaction. Moreover, a reaction's rate cannot be defined in any other way in SBML—**InitialAssignment**, **AssignmentRule**, **RateRule**, **AlgebraicRule**, **Event**, and other constructs in SBML cannot be used to set the reaction rate separately. Nevertheless, for some modeling applications, reactions without any defined rate

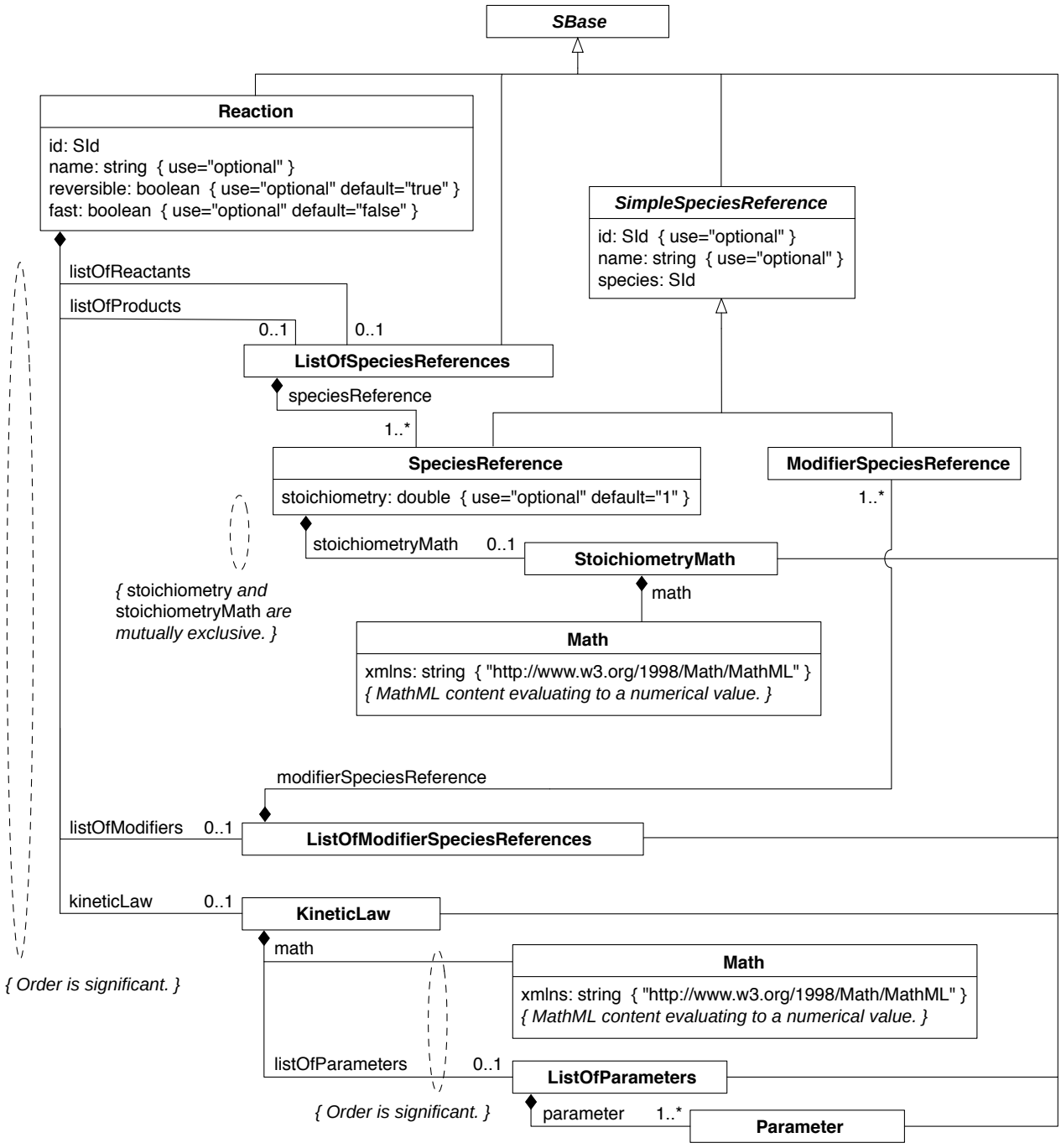


Figure 21: *The definitions of classes* ***Reaction****,* ***KineticLaw****,* ***SpeciesReference****,* ***ModifierSpeciesReference****, as well as the container classes* ***ListOfReactions****,* ***ListOfReactions****,* ***ListOfReactions****, and* ***ListOfParameters****. Note that* ***SimpleSpeciesReference*** *is an abstract class used only to provide some common attributes to its derived classes. The class* ***Parameter*** *is defined in Section 4.9.*

can be perfectly acceptable.

The `reversible` *attribute*

The optional boolean attribute **`reversible`** indicates whether the reaction is reversible. The default is "`true`".

To say that a reaction is *reversible* is to say it can proceed in either the forward or the reverse direction. Although the reversibility of a reaction can sometimes be deduced by inspecting its rate expression, this is not always the case, especially for complicated expressions. Having a separate attribute supports the ability to perform some kinds of model analyses in the absence of performing a time-course simulation of the model. Moreover, the need in SBML to allow rate expressions (i.e., **KineticLaw**) to be optional leads to the need for a separate flag indicating reversibility. Information about reversibility in the absence of a **KineticLaw** in a **Reaction** is useful in certain kinds of structural analyses such as elementary mode analysis.

Mathematically, the **`reversible`** attribute on **Reaction** has no impact on the construction of the equations giving the overall rates of change of each species quantity in a model. A concrete explanation may help illustrate this. Suppose a model consists of multiple reactions, of which two particular irreversible reactions R_f and R_r are actually the forward and reverse processes of the same underlying reaction. The product species of R_f necessarily will be the reactants of R_r, and the reactants of R_f will be the products of R_r. Let $f_f(\mathbf{X})$ and $f_r(\mathbf{X})$ be the SBML kinetic rate formulas of R_f and R_r, respectively, with $\mathbf{X}$ representing the species, parameters and compartments in the model. For the sake of this example, suppose we are using a continuous differential equation framework to simulate the system of reactions. Then for each species, we need to construct an expression representing the overall rate of change of that species' amount in the model. This overall expression will be a sum of the contributions of all the relevant rate formulas,

$$\frac{dS}{dt} = \ldots - n \cdot f_f(\mathbf{X}) + n \cdot f_r(\mathbf{X}) + \ldots$$

where S is a reactant species of R_f and a product of R_r, n is the effective stoichiometry of S in R_f (which by implication must be the same as its stoichiometry in R_r), and "$\ldots$" indicates other rate formulas in the model involving the particular species S. Now, contrast this to the case of an identical second SBML model, except that instead of having separate **Reaction** definitions for the forward and reverse reactions, this model has a single **Reaction** R_c labeled as reversible and whose reactants and products are the same as those of R_f in the first model. The rate of this reaction will be a formula $f_c = f_f(\mathbf{X}) - f_r(\mathbf{X})$. In constructing an expression representing the overall rate of change for the species S involved in that reaction, we will have

$$\begin{aligned} \frac{dS}{dt} &= \ldots - n \cdot f_c(\mathbf{X}) + \ldots \\ &= \ldots - n \cdot f_f(\mathbf{X}) + n \cdot f_r(\mathbf{X}) + \ldots \end{aligned}$$

In other words, the result is the same final expression for the rate of change of a species. Although in this simple example we used an expression for f_c that had clearly separated terms, in the general case the expression may have a more complicated form.

Note that labeling a reaction as irreversible is an assertion that the reaction always proceeds in the given forward direction. (Why else would it be flagged as irreversible?) This implies the rate expression in the **KineticLaw** always has a non-negative value during simulations. Software tools could provide a means of optionally testing that this condition holds. The presence of reversibility information in two places (i.e., the rate expression and the **`reversible`** flag) leaves open the possibility that a model could contain contradictory information, but the creation of such a model would be an error on the part of the software generating it.

The `fast` *attribute*

The optional boolean attribute **`fast`** is another optional boolean attribute of **Reaction**. The attribute's default value is "`false`".

Previous definitions of SBML indicated that software tools could ignore this attribute if they did not implement support for the corresponding concept; however, further research has revealed that this is incorrect and **`fast`** *cannot be ignored* if it is set to "`true`". SBML Level 2 Version 3 and Version 5therefore stipulate that

if a model has any reactions with `fast` set to "`true`", a software tool must be able to respect the attribute or else indicate to the user that it does not have the capacity to do so. Analysis software cannot ignore the value of the `fast` attribute because doing so may lead to different results as compared to a software system that *does* make use of `fast`.

When a model contains a true value for `fast` on any of its reactions, it indicates that the creator of the model is distinguishing different time scales of reactions in the system. The model's reaction definitions are divided into two sets by the values of the `fast` attributes. The set of reactions having `fast`="`true`" (known as *fast reactions*) should be assumed to be operating on a time scale significantly faster than the other reactions (the *slow reactions*). Fast reactions are considered to be instantaneous relative to the slow reactions.

Software tools must use a pseudo steady-state approximation for the set of fast reactions when constructing the system of equations for the model. More specifically, the set of reactions that have the `fast` attribute set to "`true`" forms a subsystem that should be described by a pseudo steady-state approximation in relationship to all other reactions in the model. Under this description, relaxation from any initial condition or perturbation from any intermediate state of this subsystem would be infinitely fast. Appendix E provides a technical explanation of an approach to solving systems with fast reactions.

The correctness of the approximation requires a significant separation of time scales between the fast reactions and other processes. This is not trivial to estimate a priori, and may even change over the course of a simulation, but can reasonably be assessed a posteriori in most cases.

The `sboTerm` *attribute on* ***Reaction***

The **Reaction** object class inherits from ***SBase*** the optional `sboTerm` attribute of type `SBOTerm` (see Sections 3.1.9 and 5). When a value is given to this attribute in a reaction definition, the value should be a valid SBO identifier referring to an interaction (i.e., terms derived from `SBO:0000231`, "interaction"). The **Reaction** object should have an "is a" relationship with the SBO term. The SBO term chosen should be the most precise (narrow) term that defines the interaction represented by the reaction as a whole. For example, a given reaction in a model might represent *non-covalent binding*, which has a term in SBO (identifier `SBO:0000177`, definition "Interaction between several biochemical entities that results in the formation of a non-covalent complex "). The corresponding **Reaction** instance in SBML would have `sboTerm="SBO:0000177"`.

As mentioned elsewhere, SBO labels are optional information on a model. Applications are free to ignore `sboTerm` values, and a model must be interpretable without the benefit of SBO labels. Section 5 gives more information about this principle and the use of SBO.

4.13.2 The SimpleSpeciesReference abstract type

As mentioned above, every species that enters into a given reaction must appear in that reaction's lists of reactants, products and/or modifiers. In an SBML model, all species that may participate in any reaction are listed in the `listOfSpecies` element of the top-level **Model** instance (see Section 4.2). Lists of products, reactants and modifiers in **Reaction** objects do not introduce new species, but rather, they refer back to those listed in the model's top-level `listOfSpecies`. For reactants and products, the connection is made using a **SpeciesReference** object; for modifiers, it is made using a **ModifierSpeciesReference** object. ***SimpleSpeciesReference***, defined in Figure 21 on page 66, is an abstract type that serves as the parent class of both **SpeciesReference** and **ModifierSpeciesReference**. It is used simply to hold the attributes and elements that are common to the latter two objects.

The `id` *and* `name` *attributes*

The optional identifier stored in the `id` attribute allows **SpeciesReference** and **ModifierSpeciesReference** instances to be referenced from other object. No SBML object classes currently do this; however, such classes are anticipated in future SBML Levels. The value of `id` must be a text string conforming to the syntax permitted by the `SId` data type described in Section 3.1.7. The `id` value (whether it is in a **SpeciesReference** or **ModifierSpeciesReference** object) exists in the global namespace of the model, as described in Section 3.3. The `id` and `name` attributes must be used as described in Section 3.3.

The `species` *attribute*

The ***SimpleSpeciesReference*** object class has a required attribute, `species`, of type `SId`. As with the other attributes, it is inherited by **SpeciesReference** and **ModifierSpeciesReference**. The value of `species` must be the identifier of a species defined in the enclosing **Model**. The species is thereby declared as participating in the reaction being defined. The precise role of that species as a reactant, product, or modifier in the reaction is determined by the subtype of ***SimpleSpeciesReference*** (i.e., either **SpeciesReference** or **ModifierSpeciesReference**) in which the identifier appears.

The `sboTerm` *attribute*

The class ***SimpleSpeciesReference*** inherits from ***SBase*** the optional `sboTerm` attribute of type `SBOTerm` (see Sections 3.1.9 and 5). This means that the object classes derived from ***SimpleSpeciesReference***, namely **SpeciesReference** and **ModifierSpeciesReference**, all have `sboTerm` attributes. When a value is given to this attribute, it should be a valid SBO identifier referring to a participant role. The appropriate SBO term depends on whether the object is a reactant, product or modifier. If a reactant, then it should be a term in the `SBO:0000010`, "reactant" sub-branch; if a product, then it should be a term in the `SBO:0000011`,

"product" sub-branch; and if a modifier, then it should be a term in the SBO:0000019, "modifier" sub-branch. The **SpeciesReference** and **ModifierSpeciesReference** instances should have an "is a" relationship to the term identified by the SBO identifier.

The SBO terms chosen should be the most precise (narrow) one that defines the role of the species in the reaction. An SBO reactant term can only be assigned to the `sboTerm` attribute of a **SpeciesReference** instance when that instance is contained in the list of reactants in the containing **Reaction** instance. Similarly, an SBO product term can only be assigned to the `sboTerm` attribute of a **SpeciesReference** instance when that instance is contained in the list of products in the containing **Reaction** instance.

4.13.3 SpeciesReference

The **Reaction** object class provides a way to express which species act as reactants and which species act as products in a reaction. In a given reaction, references to those species acting as reactants and/or products are made using instances of **SpeciesReference** objects in **Reaction**'s lists of reactants and products. The **SpeciesReference** structure inherits the mandatory attribute `species` and optional attributes `id`, `name`, and `sboTerm`, from the parent type ***SimpleSpeciesReference***; see Section 4.13.2 for their definitions. It also defines attribute `stoichiometry` and element `stoichiometryMath`, described below.

The `species` attribute value must be the identifier of an existing species defined in the enclosing **Model**; the species is thereby designated as a reactant or product in the reaction. Which one it is (i.e., reactant or product) is indicated by whether the **SpeciesReference** appears in the **Reaction**'s `reactant` or `product` lists.

The `stoichiometry` *attribute and* `stoichiometryMath` *element*

Product and reactant stoichiometries can be specified using *either* `stoichiometry` or `stoichiometryMath` in a **SpeciesReference** object. The `stoichiometry` attribute is of type double. The `stoichiometryMath` element is implemented as an element containing a MathML expression. These two are mutually exclusive; only one of `stoichiometry` or `stoichiometryMath` should be defined in a given **SpeciesReference** instance. When neither the attribute nor the element is present, the value of `stoichiometry` in the **SpeciesReference** instance defaults to "1".

For maximum interoperability, **SpeciesReference**'s `stoichiometry` attribute should be used in preference to `stoichiometryMath` when a species' stoichiometry is a simple scalar number (integer or decimal). When the stoichiometry is a rational number, or when it is a more complicated formula, `stoichiometryMath` must be used. The MathML expression in `stoichiometryMath` may also refer to identifiers of entities in a model (except reaction identifiers), as discussed in Section 3.4.3. However, the only species identifiers that can be used in `stoichiometryMath` are those referenced in the **Reaction** list of reactants, products and modifiers.

The `stoichiometry` attribute and the `stoichiometryMath` element, when either is used, is each interpreted as a factor applied to the reaction rate to give the rate of change of the species identified by the `species` attribute. This is the normal interpretation of a stoichiometry, but in SBML, one additional consideration has to be taken into account. The reaction rate, which is the result of the **KineticLaw** `math` element, is always in the model's *substance* per *time* units. However, the rate of change of the species will involve the species' *substance* units (i.e., the units identified by the **Species** object's `substanceUnits` attribute), and these units may be different from the model's default *substance* units. If the units are different, the stoichiometry should incorporate a conversion factor for converting the model's *substance* units to the species' *substance* units. The conversion factor is assumed to be included in the scalar value of the `stoichiometry` attribute if `stoichiometry` is used. If instead `stoichiometryMath` is used, then the product of the model's *substance* units times the `stoichiometryMath` units should match the *substance* units of the species. Note that in either case, if the species' units and the model's default *substance* units are the same, the stoichiometry ends up being a dimensionless number and equivalent to the standard chemical stoichiometry found in textbooks. Examples and more explanations of this are given in Section 4.13.6.

The following is a simple example of a species reference for species "X0", with stoichiometry "2", in a list of reactants within a reaction having the identifier "J1":

```
<model>
    ...
    <listOfReactions>
        <reaction id="J1">
            <listOfReactants>
                <speciesReference species="X0" stoichiometry="2">
            </listOfReactants>
            ...
        </reaction>
        ...
    </listOfReactions>
    ...
</model>
```

The following is a more complex example of a species reference for species "X0", with a stoichiometry formula consisting of a rational number:

```
<model>
    ...
    <listOfReactions>
        <reaction id="J1">
            <listOfReactants>
                <speciesReference species="X0">
                    <stoichiometryMath>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <cn type="rational"> 3 <sep/> 2 </cn>
                        </math>
                    </stoichiometryMath>
                </speciesReference>
            </listOfReactants>
            ...
        </reaction>
        ...
    </listOfReactions>
    ...
</model>
```

A species can occur more than once in the lists of reactants and products of a given **Reaction** instance. The effective stoichiometry for a species in a reaction is the sum of the stoichiometry values given in the **SpeciesReference** objects in the list of products minus the sum of stoichiometry values given in the **SpeciesReference** objects in the list of reactants. A positive value indicates the species is effectively a product and a negative value indicates the species is effectively a reactant. SBML places no restrictions on the effective stoichiometry of a species in a reaction; for example, it can be zero. In the following SBML fragment, the two reactions have the same effective stoichiometry for all their species:

```
<reaction id="x">
    <listOfReactants>
        <speciesReference species="a"/>
        <speciesReference species="a"/>
        <speciesReference species="b"/>
    </listOfReactants>
    <listOfProducts>
        <speciesReference species="c"/>
        <speciesReference species="b"/>
    </listOfProducts>
</reaction>
<reaction id="y">
    <listOfReactants>
        <speciesReference species="a" stoichiometry="2"/>
    </listOfReactants>
    <listOfProducts>
        <speciesReference species="c"/>
    </listOfProducts>
</reaction>
```

4.13.4 ModifierSpeciesReference

Sometimes a species appears in the kinetic rate formula of a reaction but is itself neither created nor destroyed in that reaction (for example, because it acts as a catalyst or inhibitor). In SBML, all such species are simply called *modifiers* without regard to the detailed role of those species in the model. The **Reaction** object class provides a way to express which species act as modifiers in a given reaction. This is the purpose of the list of modifiers available in **Reaction**. The list contains instances of **ModifierSpeciesReference** object.

As shown in Figure 21 on page 66, the **ModifierSpeciesReference** class inherits the mandatory attribute `species` and optional attributes `id` and `name` from the parent class ***SimpleSpeciesReference***; see Section 4.13.2 for their precise definitions.

The value of the `species` attribute must be the identifier of a species defined in the enclosing **Model**; this species is designated as a modifier for the current reaction. A reaction may have any number of modifiers. It is permissible for a modifier species to appear simultaneously in the list of reactants and products of the same reaction where it is designated as a modifier, as well as to appear in the list of reactants, products and modifiers of other reactions in the model.

4.13.5 KineticLaw

The **KineticLaw** object class is used to describe the rate at which the process defined by the **Reaction** takes place. As shown in Figure 21 on page 66, **KineticLaw** has elements called `math` and `listOfParameters`, in addition to the attributes and elements it inherits from ***SBase***.

Previous definitions of SBML included two additional attributes called `substanceUnits` and `timeUnits`, which allowed the *substance*/*time* units of the reaction rate expression to be defined on a per-reaction basis. These attributes were removed in SBML Level 2 Version 2 for several reasons. First, the introduction in SBML Level 2 Version 2 of mass and dimensionless units as possible units of *substance*, coupled with the previous facility for defining the units of each reaction separately and the ability to use non-integer stoichiometries, lead to the possibility of creating a valid model whose reactions nevertheless could not be integrated into a system of equations without outside knowledge for converting the quantities used. (As a simple example, consider if one reaction is defined to be in grams per second and another in moles per second, and species are given in moles: converting from mass to moles would require knowing the molecular mass of the species.) Second, the ability to change the units of a reaction provided the potential of creating unintuitive and difficult-to-reconcile systems of equations, yet the feature added little functionality to SBML. The *absence* of `substanceUnits` does not prevent the definition of any reactions; it only results in requiring the generator of the model to be explicit about any necessary conversion factors. Third, few if any software tools have ever correctly implemented support for `substanceUnits`, which made the use of this attribute in a model an impediment to interoperability. Fourth, examination of real-life models revealed that a frequent reason for using `substanceUnits` was to set the units of all reactions to the same set of substance units, which is better achieved by setting the model-wide values of "`substance`".

The `math` *element*

As shown in Figure 21 on page 66, **KineticLaw** has a element called `math` for holding a MathML formula defining the rate of the reaction. The expression in `math` may refer to species identifiers, as discussed in Section 3.4.3. The only **Species** identifiers that can be used in `math` are those declared in the lists of reactants, products and modifiers in the **Reaction** object (see Sections 4.13.2, 4.13.3 and 4.13.4). **Parameter** identifiers may be taken from the **KineticLaw**'s list of local parameters (see below) or the parameters defined globally on the **Model** instance.

Section 4.13.6 provides important discussions about the meaning and interpretation of SBML "kinetic laws".

The list of parameters

An instance of **KineticLaw** can contain a list of one or more **Parameter** objects (Section 4.9) which define new parameters whose identifiers can be used in the `math` formula. As discussed in Section 3.3.1, reactions introduce local namespaces for parameter identifiers, and within a **KineticLaw** object, a local parameter whose

identifier is identical to a global identifier defined in the model takes precedence over the value associated with the global identifier. Note that this introduces the potential for a local parameter definition to shadow a global identifier *other* than a parameter. In SBML's simple symbol system, there is no separation of symbols by class of object; consequently, inside the kinetic law mathematical formula, the value of a local parameter having the same identifier as any other global model entity (**Compartment**, **CompartmentType**, **Event**, **FunctionDefinition**, **Model**, **ModifierSpeciesReference**, **Parameter**, **Reaction**, **Species**, **SpeciesReference**, or **SpeciesType**) will override the global value, or will provide a value for an identifier that otherwise had no mathematical meaning. Modelers and software developers may wish to take precautions to avoid this happening accidentally.

The type of object used to define a parameter inside **KineticLaw** is the same **Parameter** object class used to define global parameters (Section 4.9). This simplifies the SBML language and reduces the number of unique types of data objects. However, there is a difference between local and global parameters: in the case of parameters defined locally to a **KineticLaw**, there is no means by which the parameter values can be changed. Consequently, such parameters' values are always constant, and the `constant` attribute in their definitions must always have a value of "`true`" (either explicitly or left to its default value).

The `sboTerm` *attribute*

KineticLaw inherits from ***SBase*** the optional attribute called `sboTerm` of type `SBOTerm` (see Section 5). When a value is given to this attribute, the value should be an SBO identifier referring to a term from the `SBO:0000001`, "rate law" vocabulary defined in SBO. The relationship is of the form "the kinetic law *is a* X", where X is the SBO term. The SBO term chosen should be the most precise (narrow) term that defines the type of reaction rate expression encoded by the **KineticLaw** instance.

Example

The following is an example of a **Reaction** object that defines a reaction with identifier J_1, in which $X_0 \rightarrow S_1$ at a rate given by $k \cdot [X_0] \cdot [S_2]$, where S_2 is a catalyst and k is a parameter, and the square brackets symbolizes that the species quantities have units of concentration. The example demonstrates the use of species references and **KineticLaw** objects. The units on the species here are the defaults of *substance*/*volume* (see Section 4.8), and so the rate expression $k \cdot [X_0] \cdot [S_2]$ needs to be multiplied by the compartment volume (represented by its identifier, "`c1`") to produce the final units of *substance*/*time* for the rate expression.

```
<model>
    ...
    <listOfUnitDefinitions>
        <unitDefinition id="per_concent_per_time">
            <listOfUnits>
                <unit kind="litre"/>
                <unit kind="mole"   exponent="-1"/>
                <unit kind="second" exponent="-1"/>
            </listOfUnits>
        </unitDefinition>
    </listOfUnitDefinitions>
    ...
    <listOfSpecies>
        <species id="S1" compartment="c1" initialConcentration="2.0"/>
        <species id="S2" compartment="c1" initialConcentration="0.5"/>
        <species id="X0" compartment="c1" initialConcentration="1.0"/>
    </listOfSpecies>
    ...
    <listOfReactions>
        <reaction id="J1">
            <listOfReactants>
                <speciesReference species="X0"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="S1"/>
            </listOfProducts>
            <listOfModifiers>
                <modifierSpeciesReference species="S2"/>
```

```
                </listOfModifiers>
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <times/> <ci> k </ci> <ci> S2 </ci> <ci> X0 </ci> <ci> c1 </ci>
                        </apply>
                    </math>
                    <listOfParameters>
                        <parameter id="k" value="0.1" units="per_concent_per_time"/>
                    </listOfParameters>
                </kineticLaw>
            </reaction>
        </listOfReactions>
        ...
    </model>
```

4.13.6 Traditional rate laws versus SBML "kinetic laws"

It is important to make clear that a "kinetic law" in SBML is *not* identical to a traditional rate law. The reason is that SBML must support multicompartment models, and the units normally used in traditional rate laws as well as some conventional single-compartment modeling packages are problematic when used for defining reactions between multiple compartments.

When modeling species as continuous amounts (e.g., concentrations), the rate laws used are traditionally expressed in terms of *amount of substance concentration per time*, embodying a tacit assumption that reactants and products are all located in a single, constant volume. Attempting to describe reactions between multiple volumes using *concentration/time* (which is to say, *substance/volume/time*) quickly leads to difficulties. Here is an illustration of this. Suppose we have two species pools S_1 and S_2, with S_1 located in a compartment having volume V_1, and S_2 located in a compartment having volume V_2. Let the volume $V_2 = 3V_1$. Now consider a transport reaction $S_1 \rightarrow S_2$ in which the species S_1 is moved from the first compartment to the second. Assume the simplest type of chemical kinetics, in which the rate of the transport reaction is controlled by the activity of S_1 and this rate is equal to some constant k times the activity of S_1. For the sake of simplicity, assume S_1 is in a diluted solution and thus that the activity of S_1 can be taken to be equal to its concentration $[S_1]$. The rate expression will therefore be $k \cdot [S_1]$, with the units of k being $1/time$. Then:

$$\frac{d[S_2]}{dt} = -\frac{d[S_1]}{dt} = k \cdot [S_1]$$

So far, this looks normal—until we consider the number of molecules of S_1 that disappear from the compartment of volume V_1 and appear in the compartment of volume V_2. The number of molecules of S_1 (call this n_{S_1}) is given by $[S_1] \cdot V_1$ and the number of molecules of S_2 (call this n_{S_2}) is given by $[S_2] \cdot V_2$. Since our volumes have the relationship $V_2/V_1 = 3$, the relationship above implies that $n_{S_1} = k \cdot [S_1] \cdot V_1$ molecules disappear from the first compartment per unit of time and $n_{S_2} = 3 \cdot k \cdot [S_1] \cdot V_1$ molecules appear in the second compartment. In other words, we have created matter out of nothing!

The problem lies in the use of concentrations as the measure of what is transfered by the reaction, because concentrations depend on volumes and the scenario involves multiple unequal volumes. The problem is not limited to using concentrations or volumes; the same problem also exists when using density, i.e., *mass/volume*, and dependency on other spatial distributions (i.e., areas or lengths). What must be done instead is to consider the number of "items" being acted upon by a reaction process irrespective of their distribution in space (volume, area or length). An "item" in this context may be a molecule, particle, mass, or other "thing", as long as the substance measurement is independent of the size of the space in which the items are located.

For the current example, the expressions in terms of n_{S_1} and n_{S_2} are straightforward:

$$\frac{dn_{S_2}}{dt} = -\frac{dn_{S_1}}{dt} = k \cdot [S_1] \cdot V_1$$

Given numbers of items, it is then easy to recover concentrations by dividing the item counts of each species by the volume of the compartment in which the species is located: $[S_1] = n_{S_1}/V_1$ and $[S_2] = n_{S_2}/V_2$.

The need to support multicompartment models requires that the reaction rates in SBML to be expressed in terms of *substance/time*, rather than the more typical *substance/size/time*. As a result, modelers and software tools in general cannot insert textbook rate laws unmodified as the rate expression in the `math` element of a **KineticLaw**. The unusual term "kinetic law" was chosen to alert users to this difference. We explain the general principles of converting rate laws in the following paragraphs.

Basic cases

Let us expand the simple example above by adding a second reaction, to create the system

$$S_1 \rightarrow S_2 \rightarrow S_3$$

with the left-hand reaction's rate (call this r_1) being given as $k_1 \cdot [S_1]$ and the rate of the right-hand reaction (call it r_2) as $k_2 \cdot [S_2]$. Also assume each species is located in a different compartment:

$$\begin{array}{lll} S_1 & \text{located in compartment } C_1 & \text{with volume } V_1 \\ S_2 & \text{located in compartment } C_2 & \text{with volume } V_2 \\ S_3 & \text{located in compartment } C_3 & \text{with volume } V_3 \end{array}$$

As before, converting the rate of the first reaction ($S_1 \rightarrow S_2$) to units of *substance/time* in this case is a simple matter of multiplying by the volume of the compartment where the reactants are located, leading to the following SBML rate formula:

$$R_1 = r_1 \cdot V_1 = k_1 \cdot [S_1] \cdot V_1$$

The second rate expression becomes

$$R_2 = r_2 \cdot V_2 = k_2 \cdot [S_2] \cdot V_2$$

The units of k_1 and k_2 are $1/time$ (often $1/sec$, but not necessarily), as is typical for reactions that are first-order in one reactant. The expressions R_1 and R_2 are what would be written in **KineticLaw** `math` definitions for the two reactions in this system. The formulas give the speed of each reaction in terms of the substance change over time. The reader of the SBML model needs to combine the individual contributions of each reaction to construct equations for the *overall* rates of change of each species in the model using these expressions. In terms of differential equations, these are:

$$\begin{aligned} \frac{dn_{S_1}}{dt} &= -R_1 &&= -k_1 \cdot [S_1] \cdot V_1 \\ \frac{dn_{S_2}}{dt} &= +R_1 - R_2 &&= +k_1 \cdot [S_1] \cdot V_1 - k_2 \cdot [S_2] \cdot V_2 \\ \frac{dn_{S_3}}{dt} &= +R_2 &&= +k_2 \cdot [S_2] \cdot V_2 \end{aligned}$$

To recover the concentration values, we add the following to the system of equations:

$$\begin{aligned} [S_1] &= n_{S_1}/V_1 \\ [S_2] &= n_{S_2}/V_2 \\ [S_3] &= n_{S_3}/V_3 \end{aligned}$$

Note that this formulation works properly even if the compartment sizes V_1, V_2 and V_3 vary during simulation.

Extrapolating from this example, we can now provide a general approach to translating a system of reactions involving species located in multiple compartments, for the restricted case where all reactants of any given reaction are in the same compartment (but where the compartments involved may be different for each reaction). For a species S_i involved in m reactions whose rates are given (in "textbook" form, without

volume adjustments) as $r_1, r_2, \ldots, r_m$, where the reactants of r_j are located in the compartment of size V_j,

$$\begin{aligned}\frac{dn_{S_i}}{dt} = \quad & \text{sign}_1 \cdot \text{stoich}_1 \cdot r_1 \cdot V_1 \\ & + \text{sign}_2 \cdot \text{stoich}_2 \cdot r_2 \cdot V_2 \\ & + \ldots \\ & + \text{sign}_m \cdot \text{stoich}_m \cdot r_m \cdot V_m \end{aligned} \tag{7}$$

$$[S_i] = n_{S_i}/V_i$$

In Equation (7), each term sign_j is "−" if S_i is a reactant in r_j and "+" if it is a product, and each term stoich_j is the stoichiometry of S_i in reaction r_j. Letting $R_j = r_j \cdot V_j$ represent the form of the rate expressions as they would be written in the **KineticLaw** `math` elements, then we can equivalently write

$$\begin{aligned}\frac{dn_{S_i}}{dt} = \quad & \text{sign}_1 \cdot \text{stoich}_1 \cdot R_1 \\ & + \text{sign}_2 \cdot \text{stoich}_2 \cdot R_2 \\ & + \ldots \\ & + \text{sign}_m \cdot \text{stoich}_m \cdot R_m \end{aligned} \tag{8}$$

$$[S_i] = n_{S_i}/V_i$$

This approach preserves the use of concentration terms within the reaction rate expressions so that the core of those rate expressions can be ordinary rate laws. This is important when modeling species as continuous quantities, because most textbook rate expressions are measured in terms of concentrations, and most rate constants have units involving concentration rather than item counts. For example, the second-order rate constant in a mass-action rate law has units of $1/(M \cdot s)$, which is to say, $volume/(substance \cdot time)$; this constant is then multiplied by two concentration terms. Reaction definitions in SBML models can be constructed by taking such expressions and multiplying them by the volume of the compartment in which the reactants are located. By contrast, if we were to simply replace concentrations of species by item counts in such rate laws, it would in most cases be incorrect. At the very least, the constants in the equations would need to be converted in some way to make such expressions valid.

The preceding discussion of problems involving rate laws concerns modeling approaches that use continuous quantities. There is an alternative approach to modeling that instead treats species as discrete populations (Wilkinson, 2006). In those cases, the rate expressions already use substance or item counts rather than concentrations and there is no need to convert them.

A full SBML example of translating a complete multicompartmental model into ODEs is given in Section 7.7. An example of translating a discrete model is given in Section 7.3.

Advanced cases

The explanation above applies to reactions where all of the reactants are in the same compartment. What about cases where two or more reactant species are in separate compartments?

This is a more difficult situation, and the guidelines described above for Equation (7) cannot always be applied because there will be more than one compartment size term by which the core rate expression needs to be multiplied. Unfortunately, there is often no straightforward way to mechanically convert such models without requiring a more significant change to the reaction rate expression. An example will help illustrate the difficulty. Suppose we have a simple reaction system consisting of only

$$S_1 + S_2 \rightarrow S_3$$

where S_1, S_2 and S_3 are each located in separate compartments with volumes V_1, V_2 and V_3, and the rate expression is given as $k \cdot [S_1] \cdot [S_2]$. (In reality, one would not use such a rate law in this case, but for the

sake of this example, let us ignore the fact that a mass-action rate law would actually involve an assumption that all reactants are in a well-mixed solution.) A straightforward examination of the possibilities eventually leads to the conclusion that in order to take account of the multiple volumes, the rate expressions in terms of *substance/time* have to be written as

$$\frac{dn_{S_i}}{dt} = -k'(V_1, V_2) \cdot \big([S_1] \cdot V_1\big) \cdot \big([S_2] \cdot V_2\big)$$

The crux of the problem is that the new factor $k'(V_1, V_2)$ is not the original k; to make the overall units of the expression work out, $k'(V_1, V_2)$ must be a function of the volumes, and its value must change if V_1 or V_2 changes. It is no longer a standard rate constant. In an SBML model, it is easy to define an **AssignmentRule** to compute the value of k' based on k, V_1, V_2, and possibly other variables in the system as needed, but only the modeler can determine the proper formula for their particular modeling situation. (For example, the modeler may know that in their hypothesized physical system, the reaction actually takes place completely in one or the other compartment and therefore the factor should be designed accordingly, or perhaps the reaction takes place on a membrane between compartments and a scaling factor based on the area should be used.)

Thus, although these models can be represented in SBML, constructing the correct rate expression in terms of *substance/time* units depends on problem-specific knowledge, and we cannot provide a general recipe.

Mixed species units

The discussion so far has assumed that all of the species appearing in a given reaction's rate expression had the same units, whether they be concentration or amounts or other. However, **Species** objects can each declare units separately. What happens then?

It is important to realize that implicit conversions in this situation are not defined in SBML. A species identifier appearing in a mathematical expression has the units attributed to that species (see Section 4.8.5 for a definition of the *the units of the species*). If a reaction contains species each having different units, the creator of the SBML model must explicitly incorporate the necessary conversions to make the units of the rate expression consistent. The most appropriate way is to include the conversion factor as part of the value of `stoichiometry` or `stoichiometryMath` in the **SpeciesReference** for that species.

An example may help illustrate this. Suppose we have a system of two biochemical reactions with mass-action kinetics, written in the traditional form

$$\begin{aligned} A + 2\,B &\xrightarrow{k_1} C \\ C &\xrightarrow{k_2} 3\,A \end{aligned} \tag{9}$$

with the units of k_1 being $liter^2 s^2 mol^{-2}$ and k_2's units being s^{-1}. Assume the reactions take place in a single compartment of volume V, but now let us throw a wrench into the problem: suppose that the species in the model are defined with mixed units as follows:

A	is in millimoles per litre
B	is in grams per litre
C	is in items per litre

When biochemical reaction equations of the form (9) above are written, the units of species' quantities usually are assumed to be the same, and therefore the stoichiometries in the reaction equations (9) represent simple ratios between the quantities of the species in those units. (*"One mole of* this *and two moles of* that *react to produce one mole of* that other*."*) If instead the quantities of the species are given in mixed units, as in the present example, the quantities not only need to be in proper stoichiometric relationships, the units also have to be made consistent. In SBML, this is done by appropriately setting the `stoichiometry` attribute value in the species references of the lists of reactants, products and modifiers in a **Reaction**. This then permits a properly balanced system of equations to be constructed for each species' rate of change of quantity.

In the present example, the SBML "kinetic law" formulas for the reaction rates will be written as conventional mass-action reaction rates adjusted for volume as described previously,

$$R_1 = {k_1}' \cdot [A] \cdot [B]^2 \cdot V$$
$$R_2 = {k_2}' \cdot [C] \cdot V$$

where ${k_2}' = k_2$ for this particular example but ${k_1}'$ will depend on other units as described in the paragraphs below. When these formulas are combined into overall expressions for the rates of change of A, B, and C, the result is

$$dA/dt = -a_1 \cdot R_1 + a_2 \cdot R_2$$
$$dB/dt = -b_1 \cdot R_1$$
$$dC/dt = +c_1 \cdot R_1 - c_2 \cdot R_2$$

where

a_1	is the SBML stoichiometry of A in reaction 1
a_2	is the SBML stoichiometry of A in reaction 2
b_1	is the SBML stoichiometry of B in reaction 1
c_1	is the SBML stoichiometry of C in reaction 1
c_2	is the SBML stoichiometry of C in reaction 2

We use the term *SBML stoichiometries* to highlight the fact that in this example involving mixed-units species, the values may not be identical to the biochemical stoichiometries in the reaction equations (9). And just what are the *SBML stoichiometries*? In the kind of mixed-units situation faced in this example, they must encompass both the biochemical stoichiometries and any necessary unit conversions. Thus, letting m_B stand for the molecular mass of B:

$a_1 = 1000$	(in each reaction event, 1 mole of A is consumed, with A expressed in millimoles)
$a_2 = 3000$	(in each reaction event, 3 moles of A are produced, with A expressed in millimoles)
$b_1 = 2 \cdot m_B$	(in each reaction event, 2 moles of B are consumed, with B expressed in grams)
$c_1 = 6.02 \cdot 10^{23}$	(in each reaction event, 1 mole of C is produced, expressed as item counts)
$c_2 = 6.02 \cdot 10^{23}$	(in each reaction event, 1 mole of C is consumed, expressed as item counts)

and ${k_1}' = k_1 \cdot 10^{-3} \cdot 1/{m_B}^2$.

What happens if the definition of the SBML predefined unit "`substance`" is changed in the model to be millimoles? Then the stoichiometries must be changed to the following:

$a_1 = 1$	(in each reaction event, 1 millimole of A is consumed, expressed in millimoles)
$a_2 = 3$	(in each reaction event, 3 millimoles of A are produced, expressed in millimoles)
$b_1 = 2 \cdot 10^{-3} \cdot m_B$	(in each reaction event, 2 millimoles of B are consumed, expressed in grams)
$c_1 = 6.02 \cdot 10^{20}$	(in each reaction event, 1 millimole of C is produced, expressed as item counts)
$c_2 = 6.02 \cdot 10^{20}$	(in each reaction event, 1 millimole of C is consumed, expressed as item counts)

and ${k_1}' = k_1 \cdot (10^3/m_B)^2$.

What happens if instead the definition of the SBML predefined unit "`substance`" is changed in the model to be "`item`"? Then the stoichiometries must be changed to the following:

$a_1 = 1/(6.02 \cdot 10^{20})$	(in each reaction event, 1 item of A is consumed, expressed in millimoles)
$a_2 = 3/(6.02 \cdot 10^{20})$	(in each reaction event, 3 items of A are produced, expressed in millimoles)
$b_1 = 2 \cdot 1/(6.02 \cdot 10^{23}) \cdot m_B$	(in each reaction event, 2 items of B are consumed, expressed in grams)
$c_1 = 1$	(in each reaction event, 1 item of C is produced, expressed as item counts)
$c_2 = 1$	(in each reaction event, 1 item of C is consumed, expressed as item counts)

and ${k_1}' = k_1 \cdot 6.02 \cdot 10^{20} \cdot \left(\dfrac{6.02 \cdot 10^{23}}{m_B}\right)^2$.

And finally, what happens if the definition of the SBML predefined unit "`substance`" is changed in the model to be "`gram`"? Then the stoichiometries must be changed again, to the following:

$a_1 = 1000/m_A$	(in each reaction event, 1 gram of A is consumed, expressed in millimoles)
$a_2 = 3000/m_A$	(in each reaction event, 3 grams of A are produced, expressed in millimoles)
$b_1 = 2$	(in each reaction event, 2 grams of B are consumed, expressed in grams)
$c_1 = 6.02 \cdot 10^{23} \cdot m_C$	(in each reaction event, 1 gram of C is produced, expressed as item counts)
$c_2 = 6.02 \cdot 10^{23} \cdot m_C$	(in each reaction event, 1 gram of C is consumed, expressed as item counts)

and ${k_1}' = k_1 \cdot 10^{-3} \cdot m_A$, where m_A, m_B, and m_C are the molecular masses of species A, B, and C, respectively.

In all of these cases, one straightforward approach to computing the value of ${k_1}'$ is to use an **InitialAssignment** construct for setting the value of ${k_1}'$ to the result of a formula such as one of those given in the examples above. And finally, note that if the species units *were* the same throughout (and in most models they are), the unit conversion aspects of the SBML stoichiometries would become unity, leaving only the expected biochemical stoichiometry values. Isn't that nice?

4.13.7 Use of reaction identifiers in mathematical expressions

The value of the `id` attribute of a **Reaction** can be used as the content of a `ci` element in MathML formulas elsewhere in the model. Such a `ci` element or symbol represents the rate of the given reaction as given by the reaction's **KineticLaw** object. The symbol has the units of *substance/time*.

A **KineticLaw** object in effect forms an assignment statement assigning the evaluated value of the `math` element to the symbol value contained in the **Reaction** `id` attribute. No other object can assign a value to such a reaction symbol; i.e., the `variable` attributes of **InitialAssignment**, **RateRule**, **AssignmentRule** and **EventAssignment** objects cannot contain the value of a **Reaction** `id` attribute.

The combined set of **InitialAssignment**, **AssignmentRule** and **KineticLaw** objects form a set of assignment statements that should be considered as a whole. The combined set of assignment rules should not contain algebraic loops: a chain of dependency between these statements should terminate. (More formally, consider the directed graph of assignment statements where nodes are statements and directed arcs exist for each occurrence of a symbol in a assignment statement `math` element. The directed arcs start from the statement defining the symbol to the statements that contain the symbol in their math elements. Such a graph must be acyclic.) Examples of valid and invalid set of assignment statements are given in Section 4.11.5.

4.14 Events

Model has an optional list of **Event** objects that describe the time and form of explicit instantaneous discontinuous state changes in the model. For example, an event may describe that one species quantity is halved when another species quantity exceeds a given threshold value.

An **Event** object defines when the event can occur, the variables that are affected by the event, and how the variables are affected. The effect of the event can optionally be delayed after the occurrence of the condition which invokes it. The operation of an event is divided into two phases (even when the event is not delayed): one when the event is *fired* and the other when the event is *executed*. The **Event** type is defined in Figure 22 on the following page. The object classes **Event**, **Trigger**, **Delay** and **EventAssignment** are derived from ***SBase*** (see Section 3.2). An example of a model which uses events is given below.

4.14.1 Event

An **Event** definition has two required parts: a trigger condition and at least one **EventAssignment**. In addition, an event can include an optional delay. These features of **Event** are described below.

Previous definitions of **Event** in SBML Level 2 included an additional attribute called `timeUnits`, which allowed the time units of the **Delay** to be set explicitly. This attribute was removed in SBML Level 2 Version 3 for several reasons. First, the ability to change the time units of the delay time of an **Event** to be different from the units of time for the whole model meant that computing an **Event**'s time of triggering and

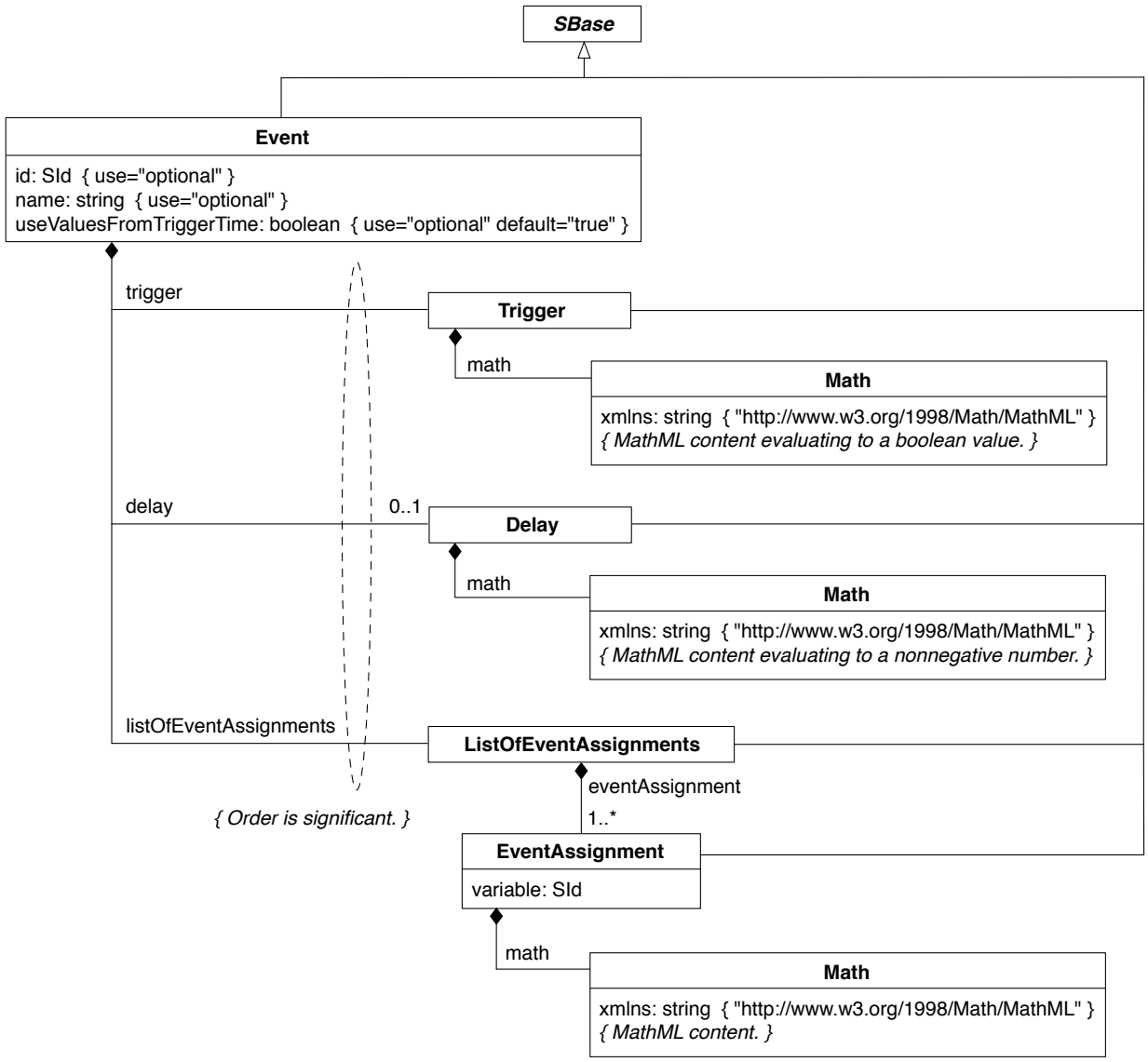


Figure 22: *The definitions of* ***Event****,* ***Trigger****,* ***Delay*** *and* ***EventAssignment****, and the container class* ***ListOfEventAssignment****.*

its delay might have to be done using two different sets of units—a potential source of overlooked errors. Second, the ability to redefine the units of time for the delay of an **Event** became inconsistent with the lack of such an attribute on other SBML Level 2 Version 5 components involving an element of time, such as **RateRule** and **KineticLaw**. On balance, the `timeUnits` feature was judged to add needless complexity and inconsistency for little gain in functionality.

The `id` *and* `name` *attributes*

As with most components in SBML, an **Event** has `id` and `name` attributes, but in the case of **Event**, both are optional. These attributes operate in the manner described in Section 3.3.

The optional `sboTerm` *attribute on* ***Event***

As with all SBML components derived from ***SBase***, an **Event** has an optional attribute `sboTerm` of type `SBOTerm` (see Sections 3.1.9 and 5). When a value is given to this attribute, it should be a valid term derived

from SBO:0000231, "interaction" in SBO. The **Event** should have an "is a" relationship with the SBO term, and the term should be the most precise (narrow) term that captures the meaning of the event in the model.

As discussed in Section 5, SBO labels are optional information on a model. Applications are free to ignore sboTerm values. A model must be interpretable without the benefit of SBO labels.

The optional useValuesFromTriggerTime *attribute*

The optional **Delay** on **Event** means there are two times to consider when computing the results of an event: the time at which the event *fires*, and the time at which assignments are *executed*. It is also possible to distinguish between the time at which the **EventAssignment**'s expression is calculated, and the time at which the assignment is made: the expression could be evaluated at the same time the assignments are performed, i.e., when the event is *executed*, but it could also be defined to be evaluated at the time the event *fired*.

In SBML Level 2 versions prior to Version 4, the semantics of **Event** time delays were defined such that the expressions in the event's assignments were always evaluated at the time the event was *fired*. This definition made it difficult to define an event whose assignment formulas were meant to be evaluated at the time the event was *executed* (i.e., after the time period defined by the value of the **Delay** element). As of SBML Level 2 Version 4, the useValuesFromTriggerTime attribute on **Event** allows a model to indicate the time at which the event's assignments are intended to be evaluated. The default value is "true", which corresponds to the interpretation of event assignments prior to SBML Level 2 Version 4: the values of the assignment formulas are computed at the moment the event fired, not after the delay. If useValuesFromTriggerTime="false", it means that the formulas in the event's assignments are to be computed after the delay, at the time the event is executed.

4.14.2 Trigger

As shown in Figure 22, the trigger element of an **Event** must contain exactly one object of class **Trigger**. This object contains one math element containing a MathML expression. The expression must evaluate to a value of type boolean. The exact moment at which the expression evaluates to "true" is the time point when the **Event** is *fired*.

An event only fires when its **Trigger** expression makes the transition in value from "false" to "true". The event will also fire at any future time points when the trigger make this transition; in other words, an event can fire multiple times during a simulation if its trigger condition makes the transition from "false" to "true" more than once.

An important question is whether an event can fire prior to, or at, initial simulation time, i.e., $t \leq 0$. The answer is no: an event can only be triggered immediately after initial simulation time i.e., $t > 0$.

The optional sboTerm *attribute on* ***Trigger***

As with all SBML components derived from ***SBase***, **Trigger** inherits the optional attribute sboTerm of type SBOTerm (see Sections 3.1.9 and 5). When a value is given to this attribute, it should be a valid term representing a mathematical expression (i.e., a term chosen from the SBO:0000064, "mathematical expression" hierarchy). The formula in the **Trigger**'s math expression should have an "is a" relationship with the SBO term, and the term should capture most precise (narrow) meaning of the mathematical formula of the trigger.

4.14.3 Delay

As shown in Figure 22, an **Event** object can contain an optional delay element of class **Delay**. The **Delay** is derived from ***SBase*** and contains a mathematical formula stored in math. The formula is used to compute the length of time between when the event has *fired* and when the event's assignments (see below) are actually *executed*. If no delay is present on a given **Event**, a time delay of zero is assumed.

The expression in the **Delay** object's math element must be evaluated at the time the event is *fired*. The expression must always evaluate to a nonnegative number (otherwise, a nonsensical situation could arise where an event is defined to fire before it is triggered!).

Units of delay expressions

The units of the numerical value computed by a **Delay** instance's `math` expression should match the model's units of *time* (meaning the definition of the "`time`" units in the model; see Section 4.4.3). Note that, as in other cases of MathML expressions in SBML, units are *not* predefined or assumed. As discussed in Section 3.4.11, literal numbers (i.e., numbers enclosed in MathML `cn` elements) or expressions containing only literal numbers and/or **Parameter** objects without declared units, are considered to have unspecified units. In such cases, the correspondence between the needed units and the (unknown) units of the **Delay** `math` expression cannot be proven, and while such expressions are not considered inconsistent, all that can be assumed by model interpreters (whether software or human) is that the units *may* be consistent.

The following **Event** example fragment helps illustrate this:

```
<model>
    ...
    <listOfEvents>
        <event>
            ...
            <delay>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <cn> 10 </cn>
                </math>
            </delay>
            ...
        </event>
    </listOfEvents>
    ...
</model>
```

Note the "`<cn> 10 </cn>`" within the mathematical formula has no specified units. The model is not invalid because of this, but a recipient of the model may justifiably be concerned about what "`10`" really means. (Ten seconds? What if the global units of time on the model were changed from seconds to milliseconds? Would the modeler remember to change "`10`" to "`10 000`"?) As discussed elsewhere, leaving units unspecified may prevent software tools from performing complete validation and other useful operations such as global unit conversions. A better approach is to avoid literal numbers and instead use an approach such as defining a parameter with declared units, as in the following modified version of the example fragment:

```
<model>
    ...
    <listOfParameters>
        <parameter id="transcriptionDelay" value="10" units="time"/>
    </listOfParameters>
    ...
    <listOfEvents>
        <event>
            ...
            <delay>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <ci> transcriptionDelay </ci>
                </math>
            </delay>
            ...
        </event>
    </listOfEvents>
    ...
</model>
```

The optional `sboTerm` *attribute on* ***Delay***

As with all SBML components derived from ***SBase***, **Delay** inherits an optional `sboTerm` attribute of type `SBOTerm` (see Sections 3.1.9 and 5). When a value is given to this attribute, it should be a valid term derived from the `SBO:0000064`, "mathematical expression" hierarchy in SBO. The **Delay** formula should have an "is a" relationship with the chosen SBO term, and the term should be the most precise (narrow) term that captures the meaning of the delay expression.

4.14.4 *EventAssignment*

Event contains a mandatory element called `listOfEventAssignments`, of class **ListOfEventAssignment**. In every instance of an event definition in a model, the object's `listOfEventAssignments` element must have a non-empty list of one or more `eventAssignment` elements of class **EventAssignment**. The object class **EventAssignment** has one required attribute, `variable`, and a required element, `math`. Being derived from ***SBase***, it also has all the usual attributes and elements of its parent class.

An "event assignment" has effect when the event is *executed*; that is, at the end of any given delay period (if given) following the moment that the **Event** is triggered. See Section 4.14.6 below for more information about events and event assignments in SBML.

The `variable` *attribute*

The `variable` attribute is of type `SId` and can contain the identifier of a **Compartment**, **Species** or **Parameter** instance defined in the model. When the event is executed, the value of the model component identified by `variable` is changed by the **EventAssignment** to the value computed by the `math` element; that is, a species' quantity, compartment's size or parameter's value are reset to the value computed by `math`.

Certain restrictions are placed on what can appear in `variable`:

- The object identified by the value of the `variable` attribute must not have its `constant` attribute set to or default to "`true`". (Constants cannot be affected by events.)
- The `variable` attribute must not contain the identifier of a reaction; only species, compartment and parameter values may be set by an **Event**.
- The value of every `variable` attribute must be unique among the set of **EventAssignment** objects within a given **Event** instance. In other words, a single event cannot have multiple **EventAssignment**s assigning the same variable. (All of them would be performed at the same time, when that particular **Event** triggers, resulting in indeterminacy.) Separate **Event** instances can refer to the same variable.
- A variable cannot be assigned a value in an **EventAssignment** object instance and also be assigned a value by an **AssignmentRule**, i.e., the value of the `variable` attribute in an **EventAssignment** instance cannot be the same as the value of a `variable` attribute in a **AssignmentRule** instance. (Assignment rules hold at all times, therefore it would be inconsistent to also define an event that reassigns the value of the same variable.)

Note that the time of assignment of the object identified by the value of `variable` is always the time at which the **Event** is *executed*, not when it is *fired*. The timing is controlled by the optional **Delay** in an **Event**. The time of assignment is not affected by the `useValuesFromTriggerTime` attribute on **Event**—that attribute affects the time at which the **EventAssignment**'s `math` expression is evaluated. In other words, SBML allows decoupling the time at which the `variable` is assigned from the time at which its value expression is calculated.

The optional `sboTerm` *attribute on* ***EventAssignment***

EventAssignment has an optional `sboTerm` attribute of type `SBOTerm` (see Sections 3.1.9 and 5). When a value is given to this attribute, it should be a valid SBO term identifier referring to a mathematical expression (i.e., terms derived from `SBO:0000064`, "mathematical expression"). The **EventAssignment** should have an "is a" relationship with the SBO term, and the term should be the most precise (narrow) term that captures the form of the assignment formula in the model.

EventAssignment*'s* `math`

The `math` element contains a MathML expression that defines the new value of the object identified by the `variable`.

The time at which this expression is evaluated is determined by **Event**'s `useValuesFromTriggerTime` attribute. If the attribute value is "`true`" (the default), the expression must be evaluated when the event is *fired*;

more precisely, the values of identifiers occurring in MathML `ci` attributes in the **EventAssignment**'s `math` expression are the values they have at the point when the event *fired*. If, instead, `useValuesFromTriggerTime`'s value is "`false`", it means the values at *execution* time should be used; that is, the values of identifiers occurring in MathML `ci` attributes in the **EventAssignment**'s `math` expression are the values they have at the point when the event *executed*.

Units of the `math` *formula in* **EventAssignment**

In all cases, as would be expected, the units of the formula contained in the `math` element of **EventAssignment** should be consistent with the units of the object identified by the `variable` attribute. More precisely:

- *In the case of a species*, an **EventAssignment** sets the referenced species' quantity (*concentration* or *amount of substance*) to the value determined by the formula in `math`. The units of the `math` formula should be identical to the *units of the species* as defined in Section 4.8.5.
- *In the case of a compartment*, an **EventAssignment** sets the referenced compartment's size to the size determined by the formula in `math`. The overall units of the formula should be identical to the units specified for the size of the compartment identified by the value of the **EventAssignment**'s `variable` attribute. (See Section 4.7.5 for an explanation of how the units of the compartment's size are determined.)
- *In the case of a parameter*, an **EventAssignment** sets the referenced parameter's value to that determined by the formula in `math`. The overall units of the formula should be identical to the units defined for the parameter identified by the value of the **EventAssignment**'s `variable` attribute. (See Section 4.9.3 for an explanation of how the units of the parameter are determined.)

Note that the formula placed in the `math` element has no assumed units. The consistency of the units of the formula, and the units of the entity which the assignment affects, should be explicitly established just as in the case of the value of `delay`.

4.14.5 Example Event definitions

A example of an **Event** object follows. This structure makes the assignment $k_2 = 0$ at the point when $P_1 \leq P_2$:

```
<event>
    ...
    <listOfUnitDefinitions>
        <unitDefinition id="per_second">
            <listOfUnits>
                <unit kind="second" exponent="-1" multiplier="1" offset="0"/>
            </listOfUnits>
        </unitDefinition>
    </listOfUnitDefinitions>
    ...
    <listOfParameters>
        ...
        <parameter id="k2" value="0.05" units="per_second"/>
        <parameter id="k2reset" value="0.0" units="per_second"/>
        ...
    </listOfParameters>
    ...
    <listOfEvents>
        <event>
            <trigger>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <leq/>
                        <ci> P_1 </ci>
                        <ci> P_2 </ci>
                    </apply>
```

```
                    </math>
                </trigger>
                <listOfEventAssignments>
                    <eventAssignment variable="k2">
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <ci> k2reset </ci>
                        </math>
                    </eventAssignment>
                <listOfEventAssignments>
            </event>
        </listOfEvents>
        ...
    </model>
```

A complete example of a model using events is given in Section 7.10.

4.14.6 Detailed semantics of events

The description of events above describes the action of events in isolation from each other. This section describes how events interact.

Events whose `trigger` expression is true at the start of a simulation do not *fire* at the start of the simulation ($t = 0$). Events *fire* only when the trigger *becomes* true, i.e., the trigger expression transitions from false to true, which cannot happen at $t = 0$ but can happen at $t > 0$.

Any transition of a `trigger` expression from "`false`" to "`true`" will cause an `event` to *fire*. Consider an `event` E with delay d where the `trigger` expression makes a transition from false to true at times t_1 and t_2. The **EventAssignment** object will have effect at $t_1 + d$ and $t_2 + d$ irrespective of the relative times of t_1 and t_2. For example events can "overlap" so that $t_1 < t_2 < t_1 + d$ still causes an event assignments to occur at $t_1 + d$ and $t_2 + d$.

It is possible for events to *fire* other events, i.e., an event assignment can cause an event to *fire*, therefore it is possible for a model to be entirely encoded in **Event** objects.

It is entirely possible for two events to be *executed* simultaneously in simulated time. It is assumed that, although the precise time at which these events are *executed* is not resolved beyond the given point in simulated time, the order in which the events occur is resolved. This order can be significant in determining the overall outcome of a given simulation. SBML Level 2 does not define the algorithm for determining this order (the tie-breaking algorithm). As a result, the outcomes of simulations involving events may vary when simultaneous events occur during simulation. SBML Level 3 defines a specific set of tie-breaking algorithms and a mechanism for models to indicate which algorithm should be applied during simulation, if this is critical for the system being modeled. However, it must be emphasized that even for SBML Level 2, all triggered simultaneous events must fire, and the order in which they are executed is not defined. They may be executed randomly, alphabetically, arbitrarily, or in any other order determined by the simulation software.

Despite the absence of a specific tie-breaking algorithm, SBML event simulation is constrained as follows. When an event X *fires* another event Y and event Y has zero delay then event Y is added to the existing set of simultaneous events that are pending *execution*. Events such as Y do not have a special priority or ordering within the tie-breaking algorithm. Events X and Y form a cascade of events at the same point in simulation time. All events in a model are open to being in a cascade. The position of an event in the event list does not affect whether it can be in the cascade: Y can be triggered whether it is before or after X in the list of events. A cascade of events can be infinite (never terminate). When this occurs a simulator should indicate this has occurred; i.e., it is incorrect for the simulator to arbitrarily break the cascade and continue the simulation without at least indicating the infinite cascade occurred. A variable can change more than once when processing simultaneous events at simulation time t. The model behavior (output) for such a variable is the value of the variable at the end of processing all the simultaneous events at time t.

5 The Systems Biology Ontology and the sboTerm attribute

The values of `id` attributes on SBML components allow the components to be cross-referenced within a model. The values of `name` attributes on SBML components provide the opportunity to assign them meaningful labels suitable for display to humans (Section 3.3). The specific identifiers and labels used in a model necessarily must be unrestricted by SBML, so that software and users are free to pick whatever they need. However, this freedom makes it more difficult for software tools to determine, without additional human intervention, the semantics of models more precisely than the semantics provided by the SBML object classes defined in other sections of this document. For example, there is nothing inherent in a parameter with identifier "`k`" that would indicate to a software tool it is a first-order rate constant (if that's what "`k`" happened to be in some given model). However, one may need to convert a model between different representations (e.g., Henri-Michaelis-Menten vs. elementary steps), or to use it with different modelling approaches (discrete or continuous). One may also need to relate the model components with other description formats, using a deeper semantics, such as SBGN (`http://www.sbgn.org/`). Although an advanced software tool *might* be able to deduce the semantics of some model components through detailed analysis of the kinetic rate expressions and other parts of the model, this quickly becomes infeasible for any but the simplest of models.

An approach to solving this problem is to associate model components with terms from carefully curated controlled vocabularies (CVs). This is the purpose of the optional `sboTerm` attribute provided on the SBML class ***SBase***. The `sboTerm` attribute always refers to terms belonging to the Systems Biology Ontology (SBO, `http://biomodels.net/SBO/`). In this section, we discuss the `sboTerm` attribute, SBO, the motivations and theory behind their introduction, and guidelines for their use.

SBO is not part of SBML; it is being developed separately, to allow the modeling community to evolve the ontology independently of SBML. However, the terms in the ontology are being designed keeping SBML components in mind, and are classified into subsets that can be directly related with SBML components such as reaction rate expressions, parameters, and a few others, see below. The use of `sboTerm` attributes is optional, and the presence of `sboTerm` on an element does not change the way the model is *interpreted*. Annotating SBML elements with SBO terms adds additional semantic information that may be used to *convert* the model into another model, or another format. Although SBO support provides an important source of information to understand the meaning of a model, software does not need to support `sboTerm` to be considered SBML-compliant.

5.1 Principles

Labeling model components with terms from shared controlled vocabularies allows a software tool to identify each component using identifiers that are not tool-specific. An example of where this is useful is the desire by many software developers to provide users with meaningful names for reaction rate equations. Software tools with editing interfaces frequently provide these names in menus or lists of choices for users. However, without a standardized set of names or identifiers shared between developers, a given software package cannot reliably interpret the names or identifiers of reactions used in models written by other tools.

The first solution that might come to mind is to stipulate that certain common reactions always have the same name (e.g., "Michaelis-Menten"), but this is simply impossible to do: not only do humans often disagree on the names themselves, but it would not allow for correction of errors or updates to the list of predefined names except by issuing new releases of the SBML specification—to say nothing of many other limitations with this approach. Moreover, the parameters and variables that appear in rate expressions also need to be identified in a way that software tools can interpret mechanically, implying that the names of these entities would also need to be regulated.

The Systems Biology Ontology (SBO) provides terms for identifying most elements of SBML. The relationship implied by an `sboTerm` on an SBML model component is "is a" between the characteristic of the component meant to be described by SBO on this element and the SBO term identified by the value of the `sboTerm`. By adding SBO term references on the components of a model, a software tool can provide additional details using independent, shared vocabularies that can enable *other* software tools to recognize precisely what the component is meant to be. Those tools can then act on that information. For example,

if the SBO identifier `SBO:0000049` is assigned to the concept of "first-order irreversible mass-action kinetics, continuous framework", and a given **KineticLaw** object in a model has an `sboTerm` attribute with this value, then regardless of the identifier and name given to the reaction itself, a software tool could use this to inform users that the reaction is a first-order irreversible mass-action reaction. This kind of reverse engineering of the meaning of reactions in a model would be difficult to do otherwise, especially for more complex reaction types.

The presence of an SBO label on a compartment, species, or reaction, can help map SBML elements to equivalent concepts in other standards, such as (but not limited to) BioPAX (http://www.biopax.org/), PSI-MI (http://www.psidev.info/index.php?q=node/60), or the Systems Biology Graphical Notation (SBGN, http://www.sbgn.org/). Such mappings can be used in conversion procedures, or to build interfaces, with SBO becoming a kind of "glue" between standards of representation.

The presence of the label on a kinetic expression can also allow software tools to make more intelligent decisions about reaction rate expressions. For example, an application could recognize certain types of reaction formulas as being ones it knows how to solve with optimized procedures. The application could then use internal, optimized code implementing the rate formula indexed by identifiers such as `SBO:0000049` appearing in SBML models.

Finally, SBO labels may be a very valuable tool when it comes to model integration, by helping identify interfaces, convert mathematical expressions and parameters etc.

Although the use of SBO can be beneficial, it is critical to keep in mind that the presence of an `sboTerm` value on an object *must not change the fundamental mathematical meaning* of the model. An SBML model must be defined such that it stands on its own and does not depend on additional information added by SBO terms for a correct mathematical interpretation. SBO term definitions will not imply any alternative mathematical semantics for any SBML object labeled with that term. Two important reasons motivate this principle. First, it would be too limiting to require all software tools to be able to understand the SBO vocabularies in addition to understanding SBML. Supporting SBO is not only additional work for the software developer; for some kinds of applications, it may not make sense. If SBO terms on a model are optional, it follows that the SBML model *must* remain unambiguous and fully interpretable without them, because an application reading the model may ignore the terms. Second, we believe allowing the use of `sboTerm` to alter the mathematical meaning of a model would allow too much leeway to shoehorn inconsistent concepts into SBML objects, ultimately reducing the interoperability of the models.

5.2 Using SBO and sboTerm

The `sboTerm` attribute data type is always `SBOTerm`, defined in Section 3.1.9. When present in a given model object instance, the attribute's value must be an identifier taken from the Systems Biology Ontology (SBO; `http://biomodels.net/SBO/`). This identifier must refer to a single SBO term that best defines the entity encoded by the SBML object in question. An example of the type of relationship intended is: *the KineticLaw in reaction R1 is a first-order irreversible mass action rate law.*

Note the careful use of the words "defines" and "entity encoded by the SBML object" in the paragraph above. As mentioned, the relationship between the SBML object and the URI is:

> The "thing" encoded by this SBML object has a characteristic that *is an* instance of the "thing" represented by the referenced SBO term.

The characteristic relevant for each SBML object is described in the second column of Table 6.

5.2.1 The structure of the Systems Biology Ontology

The goal of SBO labeling for SBML is to clarify to the fullest extent possible the nature of each element in a model. The approach taken in SBO begins with a hierarchically-structured set of controlled vocabularies with seven main divisions: (1) physical entity representation, (2) participant role, (3) systems description parameter, (4) modeling framework, (5) mathematical expression, (6) occurring entity representation, and (7) metadata representation. Figure 23 on the next page illustrates the highest level of SBO.

Each of the seven branches of Figure 23 have a hierarchy of terms underneath them. At this time, we can only begin to list some initial concepts and terms in SBO; what follows is not meant to be complete, comprehensive or even necessarily consistent with future versions of SBO. The web site for SBO (`http://biomodels.net/SBO/`) should be consulted for the current version of the ontology. Section 5.4.1 describes how the impact of SBO changes on software applications is minimized.

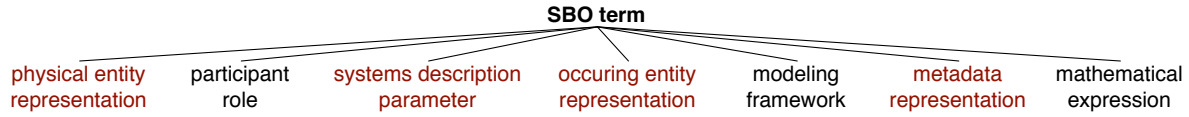


Figure 23: *The seven controlled vocabularies (CVs) that make up the main branches of SBO.*

Figure 24 shows the structure for the *physical entity representation* branch, which reflects the hierarchical groupings of the types of entities that can be represented by a `compartmentType`, a `compartment`, a `speciesType` or a `species`. Note that the values taken by the `sboTerm` attribute on those elements should refer to SBO terms belonging to the *material entity* branch, so as to distinguish whether the element represents a macromolecule, a simple chemical, etc. Indeed, this information remains valid for the whole model. The term should not belong to the *material entity* branch, representing the function of the entity within a certain functional context. If one wants to use this information, one should refer to the SBO terms using a controlled RDF annotation instead (Section 6), carefully choosing the qualifiers (Section 6.5) to reflect the fact that a given `species`, for instance, can fulfill different functions within a given model (e.g., EGF receptor is a receptor and an enzyme).

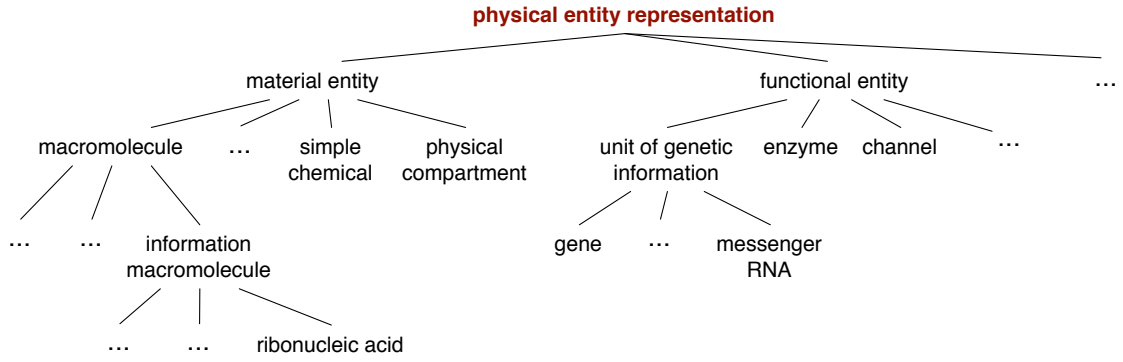


Figure 24: *Partial expansion of some of the terms in the* entity *branch of SBO.*

Figure 25 shows the structure for the *participant role* branch, also grouping the concepts in a hierarchical manner. For example, in reaction rate expressions, there are a variety of possible modifiers. Some classes of modifiers can be further subdivided and grouped. All of this is easy to capture in the ontology. As more agreement is reached in the modeling community about how to define and name modifiers for different cases, the ontology can grow to accommodate it.

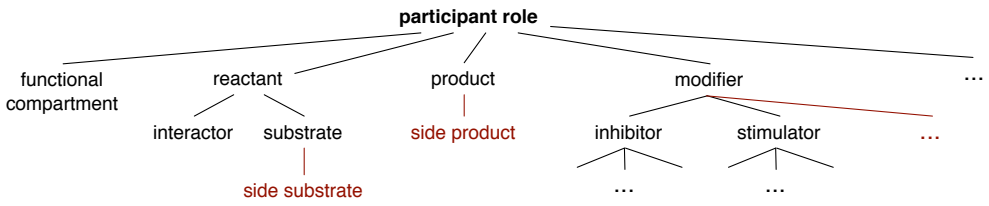


Figure 25: *Partial expansion of some of the terms in the* participant role *branch of SBO.*

The controlled vocabulary for quantitative parameters is illustrated in Figure 26. Note the separation of *kinetic constant* into separate terms for unimolecular, bimolecular, etc. reactions, as well as for forward and reverse reactions. The need to have separate terms for forward and reverse rate constants arises in reversible mass-action reactions. This distinction is not always necessary for all quantitative parameters; for example, there is no comparable concept for the Michaelis constant. Another distinction for some quantitative parameters is a decomposition into different versions based on the modeling framework being assumed. For example, different terms for continuous and discrete formulations of kinetic constants represent specializations of the constants for particular simulation frameworks. Not all quantitative parameters will need to be distinguished along this dimension.

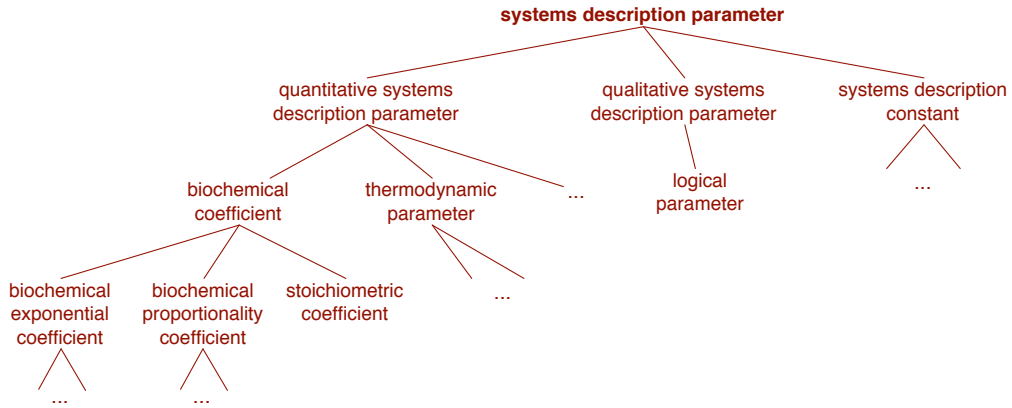


Figure 26: *Partial expansion of some of the terms in the* quantitative parameter *branch.*

The terms of the SBO quantitative systems description parameter branch contain mathematical formulas encoded using MathML 2.0 expressing the parameter using other SBO parameters. The main use of that approach is to avoid listing all the variants of a mathematical expression, escaping a combinatorial explosion.

The *modeling framework* controlled vocabulary is needed to elucidate how to simulate a mathematical expression used in models. Figure 27 illustrates the structure of this branch, which is at this point fairly simple, but we expect that more terms will evolve in the future.

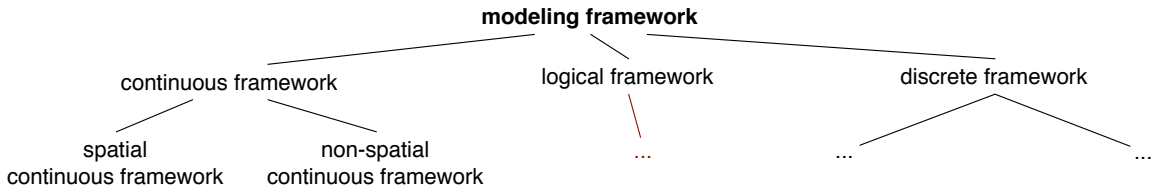


Figure 27: *Partial expansion of some of the terms in the* modeling framework *branch.*

The *mathematical expression* vocabulary encompasses the various mathematical expressions that constitute a model. Figure 28 on the following page illustrates a portion of the hierarchy. Rate law or conservation law formulas are part of the mathematical expression hierarchy, and subdivided by successively more refined distinctions until the leaf terms represent precise statements of common reaction or rule types. Other types of mathematical expressions may be included in the future in order to be able to further characterize mathematical components of a model, such as initial assignments, assignment rules, rate rules, algebraic rules, constraints, and event triggers and assignments.

The leaf terms of the mathematical expression branch contain the mathematical formulas encoded using MathML 2.0. There are many potential uses for this. One is to allow a software application to obtain the formula corresponding to a term and insert it into a model. In effect, the formulas given in the CV act

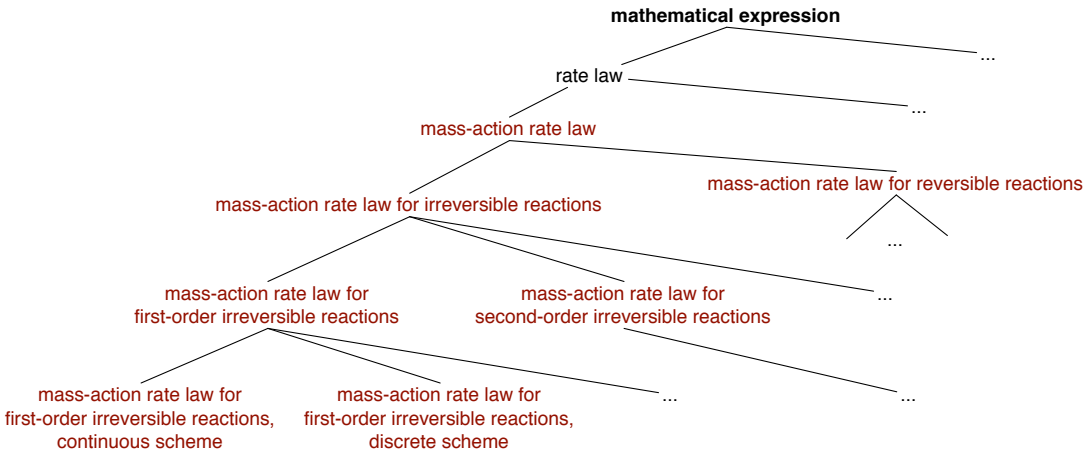


Figure 28: *Partial expansion of some of the terms in the* mathematical expression *branch.*

as templates for what to put into an SBML construct such as **KineticLaw** or ***Rule***. The MathML definition also acts as a precise statement about the rate law in question. In particular, it carries information about the modeling framework to use in order to interpret the formula. Some of the non-leaf terms also contain formulas encoded using MathML 2.0. In that case, the formulas contained in the children terms are specific versions of the formula contained in the parent term. Those formulas may be generic, containing MathML constructs not yet supported by SBML, and need to be expanded into the MathML subset allowed in SBML before they can be used in conjunction with SBML models.

To make this discussion concrete, here is an example definition of an entry in the SBO rate law hierarchy at the time of this writing. This term represents second-order, irreversible, mass-action rate laws with one reactant, formulated for use in a continuous modeling framework:

ID: SBO:0000052

Name: second-order irreversible mass action rate law, one reactant, continuous scheme.

Definition: Reaction scheme where the products are created from the reactants and the change of a product quantity is proportional to the product of reactant activities. The reaction scheme does not include any reverse process that creates the reactants from the products. The change of a product quantity is proportional to the square of one reactant quantity. It is to be used in a reaction modelled using a continuous framework.

Parent(s):

- SBO:0000050 second order irreversible mass action rate law, one reactant (is a).
- SBO:0000163 irreversible mass action rate law, continuous scheme (is a).

MathML:

```
<math xmlns="http://www.w3.org/1998/Math/MathML">
   <semantics definitionURL="http://biomodels.net/SBO/#SBO:0000062">
      <lambda>
         <bvar><ci definitionURL="http://biomodels.net/SBO/#SBO:0000036">k</ci></bvar>
         <bvar><ci definitionURL="http://biomodels.net/SBO/#SBO:0000509">R</ci></bvar>
         <apply>
            <times/>
            <ci>k</ci>
            <ci>R</ci>
            <ci>R</ci>
         </apply>
      </lambda>
   </semantics>
</math>
```

In the MathML definition of the term shown above, the bound variables in the `lambda` expression are tagged with references to terms in the SBO *systems description parameter* branch (for `k` and `R`). This makes it possible for software applications to interpret the intended meanings of the parameters in the expression. This also permits to convert an expression into another, by using the MathML 2.0 formula contained in the SBO terms associated with the parameters.

The *occurring entity representation* branch of SBO defines types of biological processes, events or relationship involving entities. It lists the types of biochemical reactions, such as binding, conformational transition, or cleavage, and also the different controls that modify a biochemical reaction, such as inhibition, catalysis, etc.

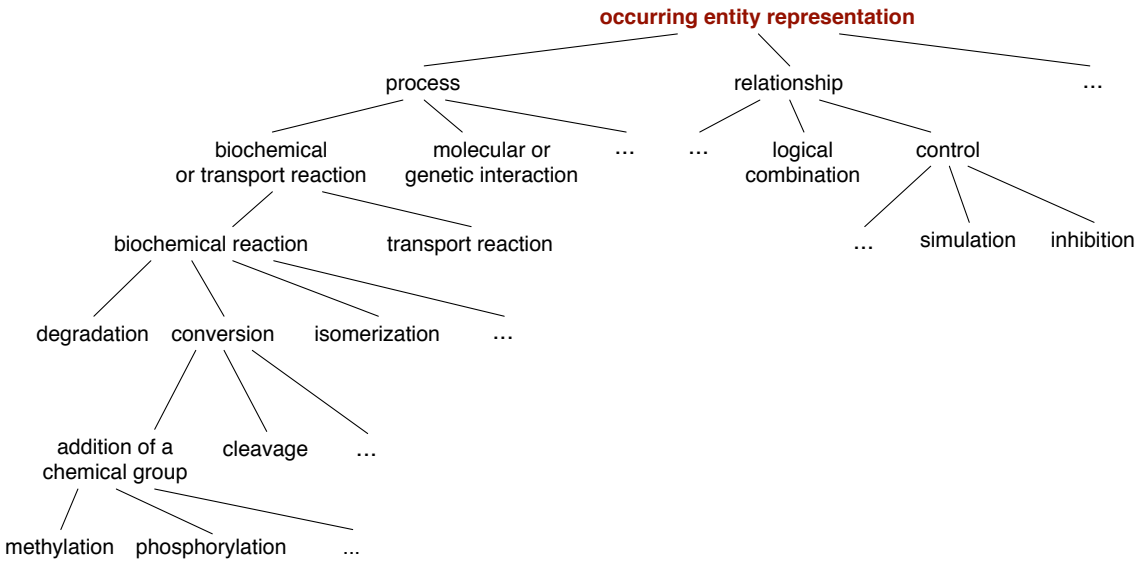


Figure 29: *Partial expansion of some of the terms in the* occurring entity representation *branch.*

One of the goals of SBO is to permit a tool to traverse up and down the hierarchy in order to find equivalent terms in different frameworks. The hope is that when a software tool encounters a given rate formula in a model, the formula will be a specific form (say, "mass-action rate law, second order, one reactant, for discrete simulation"), but by virtue of the consistent organization of the reaction rate CV into framework-specific definitions, and the declaration of every parameters involved in each expression, the tool should in principle be able to determine the definitions for other frameworks (say, "mass-action rate law, second order, one reactant for *continuous* simulation"). If the software tool is designed for continuous simulation and it encounters an SBML model with rate laws formulated for discrete simulation, it could in principle look up the rate laws' identifiers in the CV and search for alternative definitions intended for discrete simulation. And of course, the converse is true, for when a tool designed for discrete simulation encounters a model with rate laws formulated for continuous simulation.

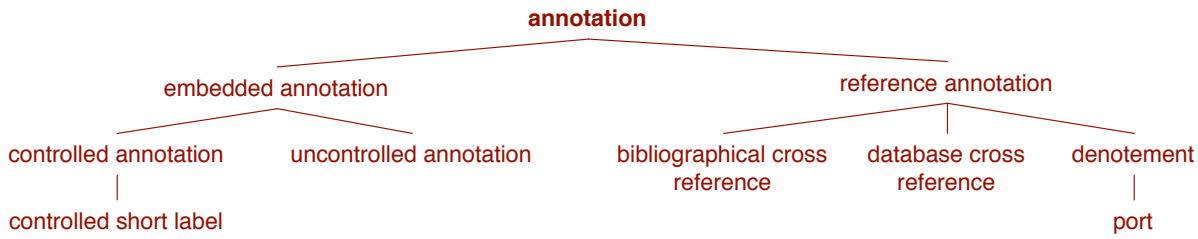


Figure 30: *Current expansion of the terms in the* annotation *branch of SBO.*

The controlled vocabulary for annotations is illustrated in Figure 30 on the previous page, the single child of the 'metadata representation' branch of SBO. As this branch is for annotating annotations themselves, its branches cannot usually be used for ***SBase***-derived elements, as those generally depict basic model information, and not annotations for that model.

5.2.2 Relationships between individual SBML components and SBO terms

The `sboTerm` attribute is defined on the abstract class ***SBase*** and can be used in all derived elements. However, not all SBO terms should be used to annotate all SBML elements. Table 6 summarizes the relationships between SBML components and the branches within SBO that apply to that component (There are currently no specific SBO term that correspond to the **Sbml**, **UnitDefinition**, **Unit**, and various **ListOf**_____ list classes.).

SBML Component	SBO Branch	Branch Identifier
Model	occurring entity representation	SBO:0000231
FunctionDefinition	mathematical expression	SBO:0000064
CompartmentType	material entity	SBO:0000240
SpeciesType	material entity	SBO:0000240
Compartment	material entity	SBO:0000240
Species	material entity	SBO:0000240
Reaction	occurring entity representation	SBO:0000231
Parameter	quantitative systems description parameter	SBO:0000002
SpeciesReference	participant role	SBO:0000003
ModifierSpeciesReference	participant role	SBO:0000003
KineticLaw	rate law	SBO:0000001
InitialAssignment	mathematical expression	SBO:0000064
AlgebraicRule	mathematical expression	SBO:0000064
AssignmentRule	mathematical expression	SBO:0000064
RateRule	mathematical expression	SBO:0000064
Constraint	mathematical expression	SBO:0000064
Event	occurring entity representation	SBO:0000231
Trigger	mathematical expression	SBO:0000064
Delay	mathematical expression	SBO:0000064
EventAssignment	mathematical expression	SBO:0000064

Table 6: *SBML components and the main types of SBO terms that may be assigned to them. The identifiers of the highest-level SBO terms in each branch are provided for guidance, but actual values used for* `sboTerm` *attributes should be more specific child terms within these branches. Note that the important aspect here is the set of specific SBO identifiers, not the SBO term names, because the names may change as SBO continues to evolve. See text for further explanations.*

The parent identifiers shown in Table 6 are provided for reference. They are the highest-level terms in their respective branch; however, these are *not* the terms that would be used to annotate an element in SBML, because there are more specific terms underneath the parents shown here. A software tool should use the most specific SBO term available for a given concept rather than using the top-level identifier acting as the root of that particular vocabulary.

5.2.3 Tradeoffs in using SBO terms

The SBO-based approach to annotating SBML components with controlled terms has the following strengths:

1. The syntax is minimally intrusive and maximally simple, requiring only one string-valued attribute.
2. It supports a significant fraction of what SBML users have wanted to do with controlled vocabularies.
3. It does not interfere with any other scheme. The more general annotation-based approach described in Section 6 can still be used simultaneously in the same model.

The scheme has the following weaknesses:

1. An object can only have one `sboTerm` attribute; therefore, it can only be related to a single term in SBO. (This also impacts the design of SBO: it must be structured such that a class of SBML elements can logically only be associated with one class of terms in the ontology.)

2. The only relationship that can be expressed by `sboTerm` is "is a". It is not possible to represent different relationships (known as *verbs* in ontology-speak). This limits what can be expressed using SBO.

The weaknesses are not shared by the annotation scheme described in Section 6.

5.3 Relationships to the SBML `annotation` element

Another means of providing this kind of information would be to place SBO terms inside the `annotation` element defined on ***SBase*** (Sections 3.2 and 6). If an application's needs cannot be met using SBO terms, software developers might wish to examine the approach described in Section 6. However, in the interest of making the use of SBO in SBML as interoperable as possible between software tools, the best-practice recommendation is to place SBO references in the `sboTerm` attribute rather than inside the `annotation` element of an object. If instead the approach of using `annotation` is taken, the qualifiers (Section 6.5) linking the SBML element and SBO term should be chosen extremely carefully, since it will no longer be possible to assume an "instance to class" relationship.

Although `sboTerm` is just another kind of optional annotation in SBML, SBO references are separated into their own attribute on SBML components, both to simplify their use for software tools and because doing so asserts a stronger and more focused connection in a more regimented fashion. SBO references are intended to allow a modeler to make a statement of the form "this object is identical in meaning and intention to the object defined in the term X of SBO", and do so in a way that a *software tool can interpret unambiguously*.

Some software applications may have their own vocabulary of terms similar in purpose to SBO. For maximal software interoperability, the best-practice recommendation in SBML is nonetheless to use SBO terms in preference to using application-specific annotation schemes. Software applications should therefore attempt to translate their private terms to and from SBO terms when writing and reading SBML, respectively.

5.4 Discussion

Here we discuss some additional points about the SBO-based approach.

5.4.1 Frequency of change in the ontology

The SBO development approach follows conventional ontology development approaches in bioinformatics. One of the principles being followed is that identifiers and meanings of terms in the CVs never change and the terms are never deleted. Where some terms are deemed obsolete, the introduction of new terms refine or supersede existing terms, but the existing identifiers are left in the CV. Thus, references never end up pointing to nonexistent entries. In the case where synonymous terms are merged after agreement that multiple terms are identical, the term identifiers are again left in the CV and they still refer to the same concept as before. Out-of-date terms cached or hard-coded by an application remain usable in all cases. (Moreover, machine-readable CV encodings and appropriate software design should render possible the development of API libraries that automatically map older terms to newer terms as the CVs evolve.)

Therefore, a model is never in danger of ending up with SBO identifiers that cannot be dereferenced. If an application finds an old model with a term `SBO:0000065`, it can be assured that it will be able to find this term in SBO, even if it has been superseded by other, more preferred terms.

5.4.2 Consistency of information

If you have a means of linking (say) a reaction rate formula to a term in a CV, it is possible to have an inconsistency between the formula in the SBML model and the one defined for the CV term. However, this is not a new problem; it arises in other situations involving SBML models already. The guideline for these situations is that the model must be self-contained and stand on its own. Therefore, in cases where they

differ, the definitions in the SBML model take precedence over the definitions referenced by the CV. In other words, the model (and its MathML) is authoritative.

5.4.3 Implications for network access

A software tool does not need to have the ability to access the network or read the CV every time it encounters a model or otherwise works with SBML. Since the SBO will likely stabilize and change infrequently once a core set of terms is defined, applications can cache the controlled vocabulary, and not make network accesses to the master SBO copy unless something forces them to (e.g., detecting a reference in a model to an SBO term that the application does not recognize). Applications could have user preference settings indicating how often the CV definitions should be refreshed (similar to how modern applications provide a setting dictating how often they should check for new versions of themselves). Simple applications may go further and hard code references to terms in SBO that have reached stability and community consensus. SBO is available for download under different formats (`http://biomodels.net/SBO/`). Web services are also available to provide programmatic access to the ontology.

5.4.4 Implications for software tools

If a software tool does not pay attention to the SBO annotations described here, one is faced with exactly the situation that exists today: the SBML model must be interpreted as-is, without benefit of the information added by the SBO terms. The purpose of introducing an ontology scheme and guidelines for its use is to give tools enough information that they *could* perform added processing, if they were designed to take advantage of that information.

6 A standard format for the annotation element

This section describes the recommended non-proprietary format for the content of annotation elements in SBML when (a) referring to controlled vocabulary terms and database identifiers which define and describe biological and biochemical entities, and (b) describing the creator of a model and its modification history. Such a structured format should facilitate the generation of models compliant with the MIRIAM guidelines for model curation (Le Novère et al., 2005).

The format described in this section is intended to be the form of one of the top-level elements that could reside in an annotation element attached to an SBML object derived from ***SBase***. The element is named rdf:RDF. The format described here is compliant with the constraints placed on the form of annotation elements described in Section 3.2.4. We refer readers to Section 3.2.4 for important information on the structure and organization of application-specific annotations; these are not described here.

6.1 Motivation

The SBML structures described elsewhere in this document do not have any biochemical or biological semantics. The format described in this section provides a scheme for linking SBML structures to external resources so that those structures can have such semantics. The motivation for the introduction of this scheme is similar to that given for the introduction of sboTerm; however, this scheme is significantly more flexible.

It is generally not recommended that this format be used to refer to SBO terms. In most cases, the SBO terms should be referred to using the attribute sboTerm part of SBase (Section 5). However in certain situations, for instance to be able to add further information about the functional role of a species, it is necessary to add this additional information using the annotation format described here.

6.2 XML namespaces in the standard annotation

This format uses a restricted form of Dublin Core (Dublin Core Metadata Initiative, 2005) and BioModels qualifier elements (see http://sbml.org/miriam/qualifiers) embedded in RDF (W3C, 2004b). It uses a number of external XML standards and associated XML namespaces. Table 7 lists these namespaces and relevant documentation on those namespaces. The format constrains the order of elements in these namespaces beyond the constraints defined in the standard definitions for those namespaces. For each standard listed, the format only uses a subset of the possible syntax defined by the given standard. Thus it is possible for an annotation element to include XML that is compliant with those external standards but is not compliant with the format described here. Parsers wishing to support this format should be aware that a valid annotation element may contain an rdf:RDF element which is not compliant with the format described here. A parser should check that all aspects of the syntax defined here before assuming that the contained data is encoded in the format.

	Namespace URI	Definition Document
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	W3C (2004a)
dcterms	http://purl.org/dc/terms/	Kokkelink and Schwänzl (2002) DCMI Usage Board (2005)
vcard	http://www.w3.org/2001/vcard-rdf/3.0#	Iannella (2001)
bqbiol	http://biomodels.net/biology-qualifiers/	
bqmodel	http://biomodels.net/model-qualifiers/	

Table 7: *The XML standards used in the SBML standard format for annotation. The namespace prefix are shown to indicate only the prefix used in the main text.*

6.3 General syntax for the standard annotation

An outline of the format syntax is shown below.

```
<SBML_ELEMENT +++ metaid="SBML_META_ID" +++ >
  +++
  <annotation>
    +++
    <rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:dcterms="http://purl.org/dc/terms/"
      xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
      xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
      xmlns:bqmodel="http://biomodels.net/model-qualifiers/"
    >
      <rdf:Description rdf:about="#SBML_META_ID">
        [MODEL_HISTORY]
        <RELATION_ELEMENT>
          <rdf:Bag>
            <rdf:li rdf:resource="URI" />
            ...
            [NESTED CONTENT]
          </rdf:Bag>
        </RELATION_ELEMENT>
        ...
      </rdf:Description>
      +++
    </rdf:RDF>
    +++
  </annotation>
  +++
</SBML_ELEMENT>
```

The above outline shows the order of the elements. The capitalized identifiers refer to generic strings of a particular type: `SBML_ELEMENT` refers to any SBML element name that can contain an `annotation` element; `SBML_META_ID` is a XML `ID` string; `RELATION_ELEMENT` refers to element names in either the namespace `http://biomodels.net/biology-qualifiers/` or `http://biomodels.net/model-qualifiers/`; and `URI` is a URI. `[MODEL_HISTORY]` refers to an optional section described in Section 6.6 which can only be present within SBML model elements. `NESTED_CONTENT` refers to other optional RDF elements such as `bqbiol:isDescribedBy` that describe a clarification or another annotation about the `RELATION_ELEMENT` in which it appears. Nested content allows one to, for example, describe protein modifications on species, or to add evidence codes for an annotation. As with all annotations, these nested annotations only add additional information, and never change existing information. As such, they can be ignored without changing the (broader) meaning of the model. '+++' is a placeholder for either no content or valid XML syntax that is not defined by the standard annotation scheme but is consistent with the relevant standards for the enclosing elements. '`...`' is a placeholder for zero or more elements of the same form as the immediately preceding element. The precise form of whitespace and the XML namespace prefix definitions is not constrained; however, the elements and attributes must be in the namespaces shown. The rest of this section describes the format formally in English.

In this format, the annotation of an element is located in a single `rdf:RDF` element contained within an SBML `annotation` element. The annotation element can contain other elements in any order as described in Section 3.2.4. The format described in this section only defines the form of the `rdf:RDF` element. The containing SBML ***SBase*** element must have a `metaid` attribute value. (As this attribute is of the type `ID` its value must unique to the entire SBML document.)

The first element of the `rdf:RDF` element must be an `rdf:Description` element with an `rdf:about` attribute. The value of the `rdf:about` attribute must be of the form `#<string>` where the string component is equal to the value of the `metaid` attribute of the containing SBML element. This format doesn't define the form of subsequent subelements of the `rdf:RDF` element. In particular, the unique `rdf:RDF` element contained in the annotation can contain other `rdf:Description`, which content can be any valid RDF.

The `rdf:Description` element can contain only an optional model history section (see Section 6.6) followed by a sequence of zero or more BioModels relation elements. The optional model history section can only be present within an SBML **Model** element. The specific type of the relation elements will vary depending on the relationship between the SBML component and referenced information or resource.

Although Section 6.5 describes the detailed semantics of each of the relation element types, the content of these elements follows the same form. The BioModels qualifiers relation elements must only contain a single `rdf:Bag` element which in turn must only contain one or more `rdf:li` elements, and may additionally contain nested content that provides additional annotations about the contents of the `rdf:Bag`. The `rdf:li` elements must only have a `rdf:resource` attribute containing a URI referring to an information resource (See Section 6.4).

Note that the various namespaces (`xmlns:rdf`, `xmlns:dcterms`, etc.) may be declared in any order, and that only the namespaces that are actually used need be declared. If no `vcard` terms are used in a particular annotation, for example, the line `xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"` is optional.

Annotations in this format can be located at different depths within a model component.

6.4 Use of URIs

The format represents a set of relationships between the SBML element and the resources referred to by the contained `rdf:resource` attribute values. The BioModels relation elements simply define the type of the relationship.

For example, a **Species** element representing a protein could be annotated with a reference to the database UniProt by the `http://identifiers.org/uniprot/P12999` resource identifier, identifying exactly the protein described by the **Species** element. This identifier maps to a unique entry in UniProt which is never deleted from the database. In the case of UniProt, this is the "accession" of the entry. When the entry is merged with another one, both "accession" are conserved. Similarly in a controlled vocabulary resource, each term is associated with a perennial identifier. The UniProt entry also possess an "entry name" (the Swiss-Prot "identifier"), a "protein name", "synonyms" etc. Only the "accession" is perennial and should be used.

The value of a `rdf:resource` attribute is a URI that both uniquely identifies the resource, and the data in the resource. The value of the `rdf:resource` attribute is a URI, not a URL; as such, a URI does not have to reference a physical web object but simply identifies a controlled vocabulary term or database object (a URI is a label). For instance, a true URL for an Internet resource such as `http://www.uniprot.org/entry/P12999` might correspond to the URI `http://identifiers.org/uniprot/P12999`.

SBML does not specify how a parser is to interpret a URI. In the case of a transformation into a physical URL, there could be several solutions. For example, the URI `http://identifiers.org/go/GO:0007268` can be translated into:

```
http://www.ebi.ac.uk/ego/DisplayGoTerm?selected=GO:0007268
http://www.godatabase.org/cgi-bin/amigo/go.cgi?view=details&query=GO:0007268
http://www.informatics.jax.org/searches/GO.cgi?id=GO:0007268
```

Similarly the URI `http://identifiers.org/ec-code:3.5.4.4` can refer to:

```
http://www.ebi.ac.uk/intenz/query?cmd=SearchEC&ec=3.5.4.4
http://www.expasy.org/cgi-bin/nicezyme.pl?3.5.4.4
http://www.chem.qmul.ac.uk/iubmb/enzyme/EC3/5/4/4.html
http://www.genome.jp/dbget-bin/www_bget?ec:3.5.4.4
```

etc.

To enable interoperability, the community has agreed on an initial set of standardized valid URI syntax rules which may be used within the standard annotation format. This set of rules is not part of the SBML standard but will grow independently from specific SBML Levels and Versions. As the set changes, a given URI syntax rule will not be modified, although the physical resources associated with the rule may change. These URIs will always be composed as `resource:id`. The MIRIAM Resources facility located

at http://sbml.org/miriam lists URI syntaxes and possible physical links to controlled vocabularies and databases. Each entry contains a list of SBML and relation elements in which the given URI can be appropriately embedded. To enable consistent and thus useful links to external resources, the URI syntax rule set must have a consistent view of the concepts represented by the different SBML elements for the purposes of this format. For example, as the rule set is designed to link SBML biological and biochemical resources the rule set assumes that a **Species** element represents the concept of a biochemical entity type rather than mathematical symbol. The URI rule list will evolve with the evolution of databases and resources. The annotation format described in this section does not require a simple parser of this format to access this list.

6.5 Relation elements

To enable the format to encode different types of relationships between SBML elements and resources, different BioModel qualifier elements are used to enclose a set of `rdf:li` elements. The relation elements imply a specific relationship between the enclosing SBML element and the resources referenced by the `rdf:li` elements.

The detailed semantics (i.e. from the perspective of automatic parser) of the relation elements is defined by the URI list at http://sbml.org/miriam, and thus is outside the scope of SBML. The URI list generally assumes that the biological entity represented by the element is the concept linked to the reference resource.

Several relation elements with a given tag, enclosed in the same SBML element, each represent an alternative annotation to the SBML element. For example two `bqbiol:hasPart` elements within a **Species** SBML element represent two different sets of references to the parts making up the the chemical entity represented by the species. (The species is not made up of all the entities represented by all the references combined).

The complete list of the qualifier elements in the BioModels qualifier namespaces is documented at http://sbml.org/miriam/qualifiers. The list is divided into two namespaces one for biological qualifiers `http://biomodels.net/biology-qualifiers/` (prefix used here `bqbiol`) and the other for model qualifiers `http://biomodels.net/model-qualifiers/`) (prefix used here `bqmodel`). This list will only grow i.e no element will be removed from the list. The following is the list of elements at the time of writing:

- `bqmodel:is` The modeling object encoded by the SBML component is the subject of the referenced resource. For instance, this qualifier might be used to link the model to a model database.
- `bqmodel:isDescribedBy` The modeling object encoded by the SBML component is described by the referenced resource. This relation might be used to link SBML components to the literature that describes this model or this kinetic law.
- `bqbiol:is` The biological entity represented by the SBML component is the subject of the referenced resource. This relation might be used to link a reaction to its exact counterpart in (e.g.) CHEBI or Reactome.
- `bqbiol:hasPart` The biological entity represented by the SBML component includes the subject of the referenced resource, either physically or logically. This relation might be used to link a complex to the description of its components.
- `bqbiol:isPartOf` The biological entity represented by the SBML component is a physical or logical part of the subject of the referenced resource. This relation might be used to link a component to the description of the complex it belongs to.
- `bqbiol:isVersionOf` The biological entity represented by the SBML component is a version or an instance of the subject of the referenced resource.
- `bqbiol:hasVersion` The subject of the referenced resource is a version or an instance of the biological entity represented by the SBML component.
- `bqbiol:isHomologTo` The biological entity represented by the SBML component is homolog, to the subject of the referenced resource, i.e. they share a common ancestor.

- `bqbiol:isDescribedBy` The biological entity represented by the SBML component is described by the referenced resource. This relation should be used, for example, to link a species or a parameter to the literature that describes the quantity of the species or the value of the parameter.
- `bqbiol:isEncodedBy` The biological entity represented by the SBML component is encoded, either directly or by virtue of transitivity, by the subject of the referenced resource.
- `bqbiol:encodes` The biological entity represented by the SBML component encodes, either directly or by virtue of transitivity, the subject of the referenced resource.
- `bqbiol:occursIn` The biological entity represented by the SBML component takes place in the subject of the reference resource.
- `bqbiol:hasTaxon` The biological entity represented by the SBML element is taxonomically restricted, where the restriction is the subject of the referenced resource (biological entity B). This relation may be used to ascribe a species restriction to a biochemical reaction.

6.6 Model history

When enclosed in an SBML **Model** element, the format described in previous sections can include additional elements to describe the history of the model. This history data must occur immediately before the first BioModels relation elements. These additional elements encode information on the model creator and a sequence of dates recording changes to the model. The syntax for this section is outlined below.

```
<dcterms:creator>
  <rdf:Bag>
    <rdf:li rdf:parseType="Resource">
      [[
      +++
      <vCard:N rdf:parseType="Resource">
        <vCard:Family>FAMILY_NAME</vCard:Family>
        <vCard:Given>GIVEN_NAME</vCard:Given>
      </vCard:N>
      +++
      [<vCard:EMAIL>EMAIL_ADDRESS</vCard:EMAIL>]
      +++
      [<vCard:ORG rdf:parseType="Resource" >
        <vCard:Orgname>ORGANIZATION_NAME</vCard:Orgname>
      </vCard:ORG>]
      +++
      ]]
    </rdf:li>
    ...
  </rdf:Bag>
</dcterms:creator>
<dcterms:created rdf:parseType="Resource">
  <dcterms:W3CDTF>DATE</dcterms:W3CDTF>
</dcterms:created>
<dcterms:modified rdf:parseType="Resource">
  <dcterms:W3CDTF>DATE</dcterms:W3CDTF>
</dcterms:modified>
...
```

The order of elements is as shown above except that elements of the format contained between `[[` and `]]` can occur in any order (`vCard:N`, `vCard:EMAIL`, and `vCard:ORG`). The capitalized identifiers refer to generic strings of a particular type: `FAMILY_NAME` is the family name of a person who created the model; `GIVEN_NAME` is the first name of the same person who created the model; `EMAIL_ADDRESS` is the email address of the same person who created the model; and `ORGANIZATION_NAME` is the name of the organization with which the same person who created the model is affiliated `DATE` is a date in W3C date format (Wolf and Wicksteed, 1998). `W3CDTF`, `N`, `ORG` and `EMAIL` are literal strings. The elements of the format contained between `[` and `]` (`vCard:EMAIL` and `vCard:ORG`) are optional, but everything else is required. '+++' is a placeholder for either no content or valid XML syntax that is not defined by the standard annotation scheme but is consistent with

the relevant standards for the enclosing elements. '...' is a placeholder for zero or more elements of the same form as the immediately preceding element. The precise form of whitespace and the XML namespace prefix definitions is not constrained. The remaining text in this section describes the syntax formally in English.

The additional elements of the model history sub-format consist in sequence of a `dcterms:creator` element, a `dcterms:created` element and zero or more `dcterms:modified` elements. The last two elements must have the attribute `rdf:parseType` set to `Resource`.

The `dcterms:creator` element describes the person who created the SBML encoding of the model and contains a single `rdf:Bag` element. The `rdf:Bag` element can contain any number of elements; however, the first element must be a `rdf:li` element. The `rdf:li` element can contain any number of elements in any order. The set of elements contained with the `rdf:li` element can include the following informative elements: `vCard:N`, `vCard:EMAIL` and `vCard:ORG`. The `vCard:N` contains the name of the creator and must consist of a sequence of two elements: `vCard:Family` and the `vCard:Given` whose content is the family (surname) and given (first) names of the creator respectively. The `vCard:N` must have the attribute `rdf:parseType` set to `Resource`. The content of the `vCard:EMAIL` element must be the email address of the creator. The content of the `vCard:ORG` element must contain a single `vCard:Orgname` element. The `vCard:Orgname` element must contain the name of an organization to which the creator is affiliated.

The `dcterms:created` and `dcterms:modified` elements must each contain a single `dcterms:W3CDTF` element whose content is a date in W3C date format (Wolf and Wicksteed, 1998) which is a a profile of (restricted form of) ISO 8601.

Note that `dcterms:creator` has been added to the `http://purl.org/dc/terms/` namespace relatively recently, but the same term (with the same meaning) once only lived in the `http://purl.org/dc/elements/1.1/` namespace. It is legal to continue using the old namespace (called "`dc`" in previous versions of the SBML specifications), but as all the terms once defined there are now also defined in `http://purl.org/dc/terms/`, it is now recommended to just use the latter.

6.7 Examples

The following shows the annotation of a model with model creation data and links to external resources:

```
<model metaid="_180340" id="GMO" name="Goldbeter1991_MinMitOscil">
    <annotation>
        <rdf:RDF
                xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:dcterms="http://purl.org/dc/terms/"
                xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#"
                xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
                xmlns:bqmodel="http://biomodels.net/model-qualifiers/"
        >
            <rdf:Description rdf:about="#_180340">
                <dcterms:creator>
                    <rdf:Bag>
                        <rdf:li rdf:parseType="Resource">
                            <vCard:N rdf:parseType="Resource">
                                <vCard:Family>Shapiro</vCard:Family>
                                <vCard:Given>Bruce</vCard:Given>
                            </vCard:N>
                            <vCard:EMAIL>bshapiro@jpl.nasa.gov</vCard:EMAIL>
                            <vCard:ORG rdf:parseType="Resource">
                                <vCard:Orgname>NASA Jet Propulsion Laboratory</vCard:Orgname>
                            </vCard:ORG>
                        </rdf:li>
                    </rdf:Bag>
                </dcterms:creator>
                <dcterms:created rdf:parseType="Resource">
                    <dcterms:W3CDTF>2005-02-06T23:39:40+00:00</dcterms:W3CDTF>
                </dcterms:created>
                <dcterms:modified rdf:parseType="Resource">
                    <dcterms:W3CDTF>2005-09-13T13:24:56+00:00</dcterms:W3CDTF>
```

```
                    </dcterms:modified>
                    <bqmodel:is>
                        <rdf:Bag>
                            <rdf:li rdf:resource="http://identifiers.org/biomodels.db/BIOMD0000000003"/>
                        </rdf:Bag>
                    </bqmodel:is>
                    <bqmodel:isDescribedBy>
                         <rdf:Bag>
                             <rdf:li rdf:resource="http://identifiers.org/pubmed/1833774"/>
                         </rdf:Bag>
                    </bqmodel:isDescribedBy>
                    <bqbiol:isVersionOf>
                        <rdf:Bag>
                            <rdf:li rdf:resource="http://identifiers.org/kegg.pathway/hsa04110"/>
                            <rdf:li rdf:resource="http://identifiers.org/reactome/REACT_152"/>
                        </rdf:Bag>
                    </bqbiol:isVersionOf>
            </rdf:Description>
        </rdf:RDF>
    </annotation>
```

The following example shows a **Reaction** structure annotated with a reference to its exact Reactome counterpart.

```
<reaction id="cdc2Phospho" metaid="jb007">
  <annotation>
    <rdf:RDF
      xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    >
      <rdf:Description rdf:about="#jb007">
        <bqbiol:is>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/reactome/REACT_6327"/>
          </rdf:Bag>
        </bqbiol:is>
      </rdf:Description>
    </rdf:RDF>
  </annotation>
  <listOfReactants>
    <speciesReference species="cdc2"/>
  </listOfReactants>
  <listOfProducts>
    <speciesReference species="cdc2-Y15P"/>
  </listOfProducts>
  <listOfModifiers>
    <modifierSpeciesReference species="wee1"/>
  </listOfModifiers>
</reaction>
```

The following example describes a species that represents a complex between the protein calmodulin and calcium ions:

```
<species id="Ca_calmodulin" metaid="cacam">
  <annotation>
    <rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
    >
      <rdf:Description rdf:about="#cacam">
        <bqbiol:hasPart>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/uniprot/P62158"/>
            <rdf:li rdf:resource="http://identifiers.org/kegg.compound/C00076"/>
          </rdf:Bag>
        </bqbiol:hasPart>
      </rdf:Description>
    </rdf:RDF>
```

```
    </annotation>
  </species>
```

The following example describes a species that represents either "Calcium/calmodulin-dependent protein kinase type II alpha chain" or "Calcium/calmodulin-dependent protein kinase type II beta chain". This is the case, for example, in the somatic cytoplasm of striatal medium-size spiny neurons, where both are present but they cannot be functionally differentiated.

```
  <species id="calcium_calmodulin" metaid="cacam">
    <annotation>
      <rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
      >
        <rdf:Description rdf:about="#cacam">
          <bqbiol:hasVersion>
            <rdf:Bag>
              <rdf:li rdf:resource="http://identifiers.org/uniprot/Q9UQM7"/>
              <rdf:li rdf:resource="http://identifiers.org/uniprot/Q13554"/>
            </rdf:Bag>
          </bqbiol:hasVersion>
        </rdf:Description>
      </rdf:RDF>
    </annotation>
  </species>
```

The above approach should not be used to describe "any Calcium/calmodulin-dependent protein kinase type II chain", because such an annotation requires referencing the products of other genes such as gamma or delta. All the known proteins could be enumerated, but such an approach would almost surely lead to inaccuracies because biological knowledge continues to evolve. Instead, the annotation should refer to generic information such as Ensembl family ENSF00000000194 "CALCIUM/CALMODULIN DEPENDENT KINASE TYPE II CHAIN" or PIR superfamily PIRSF000594 "Calcium/calmodulin-dependent protein kinase type II".

The following two examples show how to use the qualifier `isVersionOf`. The first example is the relationship between a reaction and an EC code. An EC code describes an enzymatic activity and an enzymatic reaction involving a particular enzyme can be seen as an instance of this activity. For example, the following reaction represents the phosphorylation of a glutamate receptor by a complex calcium/calmodulin kinase II.

```
  <reaction id="NMDAR_phosphorylation" metaid="thx1138">
    <annotation>
      <rdf:RDF
        xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      >
        <rdf:Description rdf:about="#thx1138">
          <bqbiol:isVersionOf>
            <rdf:Bag>
              <rdf:li rdf:resource="http://identifiers.org/ec-code/2.7.1.17"/>
            </rdf:Bag>
          </bqbiol:isVersionOf>
        </rdf:Description>
      </rdf:RDF>
    </annotation>
    <listOfReactants>
      <speciesReference species="NMDAR"/>
    </listOfReactants>
    <listOfProducts>
      <speciesReference species="P-NMDAR"/>
    </listOfProducts>
    <listOfModifiers>
      <modifierSpeciesReference species="CaMKII"/>
    </listOfModifiers>
    <kineticLaw>
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <times/>
```

```
            <ci>CaMKII</ci>
            <ci>kcat</ci>
            <apply>
              <divide/>
              <ci>NMDAR</ci>
              <apply>
                </times>
                <ci>NMDAR</ci>
                <ci>Km</ci>
              </apply>
            </apply>
          </apply>
        </math>
        <listOfParameters>
          <parameter id="kcat" value="1"/>
          <parameter id="Km" value="5e-10"/>
        </listOfParameters>
      </kineticLaw>
    </reaction>
```

The second example of the use of `isVersionOf` is the complex between Calcium/calmodulin-dependent protein kinase type II alpha chain and Calcium/calmodulin, that is only one of the "calcium- and calmodulin-dependent protein kinase complexes" described by the Gene Ontology term GO:0005954.

```
    <species id="CaCaMKII" metaid="C8H10N4O2">
      <annotation>
        <rdf:RDF
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
        >
          <rdf:Description rdf:about="#C8H10N4O2">
            <bqbiol:isVersionOf>
              <rdf:Bag>
                <rdf:li rdf:resource="http://identifiers.org/go/GO:0005954"/>
              </rdf:Bag>
            </bqbiol:isVersionOf>
          </rdf:Description>
        </rdf:RDF>
      </annotation>
    </species>
```

The previous case is different form the following one, although they could seem similar at first sight. The "Calcium/calmodulin-dependent protein kinase type II alpha chain" is a part of the above mentioned "calcium- and calmodulin-dependent protein kinase complex".

```
    <species id="CaMKIIalpha" metaid="C10H14N2">
      <annotation>
        <rdf:RDF
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
        >
          <rdf:Description rdf:about="#C10H14N2">
            <bqbiol:isPartOf>
              <rdf:Bag>
                <rdf:li rdf:resource="http://identifiers.org/go/GO:0005954"/>
              </rdf:Bag>
            </bqbiol:isPartOf>
          </rdf:Description>
        </rdf:RDF>
      </annotation>
    </species>
```

It is possible describe a component with several alternative sets of qualified annotations. For example, the following species represents a pool of GMP, GDP and GTP. We annotate it with the three corresponding KEGG compound identifiers but also with the three corresponding ChEBI identifiers. The two alternative annotations are encoded in separate `hasVersion` qualifier elements.

```
<species id="GXP" metaid="GXP">
  <annotation>
    <rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
    >
      <rdf:Description rdf:about="#GXP">
        <bqbiol:hasVersion>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/chebi/CHEBI:17345"/>
            <rdf:li rdf:resource="http://identifiers.org/chebi/CHEBI:17552"/>
            <rdf:li rdf:resource="http://identifiers.org/chebi/CHEBI:17627"/>
          </rdf:Bag>
        </bqbiol:hasVersion>
        <bqbiol:hasVersion>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/kegg.compound/C00035"/>
            <rdf:li rdf:resource="http://identifiers.org/kegg.compound/C00044"/>
            <rdf:li rdf:resource="http://identifiers.org/kegg.compound/C00144"/>
          </rdf:Bag>
        </bqbiol:hasVersion>
      </rdf:Description>
    </rdf:RDF>
  </annotation>
</species>
```

The following example presents a reaction being actually the combination of three different elementary molecular reactions. We annotate it with the three corresponding KEGG reactions, but also with the three corresponding enzymatic activities. Again the two `hasPart` elements represent two alternative annotations. The process represented by the **Reaction** structure is composed of three parts, and not six parts.

```
<reaction id="adenineProd" metaid="adeprod">
  <annotation>
    <rdf:RDF
      xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    >
      <rdf:Description rdf:about="#adeprod">
        <bqbiol:hasPart>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/ec-code/2.5.1.22"/>
            <rdf:li rdf:resource="http://identifiers.org/ec-code/3.2.2.16"/>
            <rdf:li rdf:resource="http://identifiers.org/ec-code/4.1.1.50"/>
          </rdf:Bag>
        </bqbiol:hasPart>
        <bqbiol:hasPart>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/kegg.reaction/R00178"/>
            <rdf:li rdf:resource="http://identifiers.org/kegg.reaction/R01401"/>
            <rdf:li rdf:resource="http://identifiers.org/kegg.reaction/R02869"/>
          </rdf:Bag>
        </bqbiol:hasPart>
      </rdf:Description>
    </rdf:RDF>
  </annotation>
</reaction>
```

It is possible to mix different URIs in a given set. The following example presents two alternative annotations of the human hemoglobin, the first with ChEBI heme and the second with KEGG heme.

```
<species id="heme" metaid="heme">
  <annotation>
    <rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
    >
     <rdf:Description rdf:about="#heme">
       <bqbiol:hasPart>
```

```
            <rdf:Bag>
              <rdf:li rdf:resource="http://identifiers.org/uniprot/P69905"/>
              <rdf:li rdf:resource="http://identifiers.org/uniprot/P68871"/>
              <rdf:li rdf:resource="http://identifiers.org/chebi/CHEBI:17627"/>
            </rdf:Bag>
          </bqbiol:hasPart>
          <bqbiol:hasPart>
            <rdf:Bag>
             <rdf:li rdf:resource="http://identifiers.org/uniprot/P69905"/>
              <rdf:li rdf:resource="http://identifiers.org/uniprot/P68871"/>
              <rdf:li rdf:resource="http://identifiers.org/kegg.compound/C00032"/>
            </rdf:Bag>
          </bqbiol:hasPart>
        </rdf:Description>
      </rdf:RDF>
    </annotation>
  </species>
```

As formally defined above it is possible to use different qualifiers in the same annotation element. The following phosphorylation is annotated by its exact KEGG counterpart and by the generic GO term "phosphorylation".

```
  <reaction id="phosphorylation" metaid="phosphorylation">
    <annotation>
      <rdf:RDF
        xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      >
        <rdf:Description rdf:about="#phosphorylation">
          <bqbiol:is>
            <rdf:Bag>
              <rdf:li rdf:resource="http://identifiers.org/kegg.reaction/R03313"/>
            </rdf:Bag>
          </bqbiol:is>
          <bqbiol:isVersionOf>
            <rdf:Bag>
              <rdf:li rdf:resource="http://identifiers.org/go/GO:0016310"/>
            </rdf:Bag>
          </bqbiol:isVersionOf>
        </rdf:Description>
      </rdf:RDF>
    </annotation>
  </reaction>
```

The following example demonstrates the use of nested terms to describe not only that a species is in a particular compartment, but why this is believed to be true:

```
  <species id="S1" metaid="_000004" compartment="lysosome"
           hasOnlySubstanceUnits="false" boundaryCondition="false"
           constant="false">
    <annotation>
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
               xmlns:bqbiol="http://biomodels.net/biology-qualifiers/"
               xmlns:bqmodel="http://biomodels.net/model-qualifiers/">
        <rdf:Description rdf:about="#_000004">
          <bqbiol:occursIn>
            <rdf:Bag>
              <rdf:li rdf:resource="http://identifiers.org/go/GO:0005764"/>
              <bqbiol:isDescribedBy>
                <rdf:Bag>
                  <rdf:li rdf:resource="http://identifiers.org/pubmed/1111111"/>
                </rdf:Bag>
              </bqbiol:isDescribedBy>
              <bqbiol:isDescribedBy>
                <rdf:Bag>
                  <rdf:li rdf:resource="http://identifiers.org/eco/ECO:0000004"/>
                </rdf:Bag>
              </bqbiol:isDescribedBy>
```

```
                </rdf:Bag>
              </bqbiol:occursIn>
            </rdf:Description>
          </rdf:RDF>
        </annotation>
      </species>
```

In descriptive terms, the SBML species "S1" (with metaid "_000004") occurs in "go/GO:0005764" (the lysosome). This is described by the publication "pubmed/1111111", and is believed to be true because of the evidence "eco/ECO:0000004" (cell fractionation evidence).

7 Example models expressed in XML using SBML

In this section, we present several examples of complete models encoded in XML using SBML Level 2.

7.1 A simple example application of SBML

Consider the following representation of an enzymatic reaction:

$$E + S \underset{k_{\mathrm{off}}}{\overset{k_{\mathrm{on}}}{\rightleftharpoons}} ES \xrightarrow{k_{\mathrm{cat}}} E + P$$

The following is the minimal SBML document encoding the model shown above:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model name="EnzymaticReaction">
        <listOfUnitDefinitions>
            <unitDefinition id="per_second">
                <listOfUnits>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
            <unitDefinition id="litre_per_mole_per_second">
                <listOfUnits>
                    <unit kind="mole"   exponent="-1"/>
                    <unit kind="litre"  exponent="1"/>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
        </listOfUnitDefinitions>
        <listOfCompartments>
            <compartment id="cytosol" size="1e-14"/>
        </listOfCompartments>
        <listOfSpecies>
            <species compartment="cytosol" id="ES" initialAmount="0"     name="ES"/>
            <species compartment="cytosol" id="P"  initialAmount="0"     name="P"/>
            <species compartment="cytosol" id="S"  initialAmount="1e-20" name="S"/>
            <species compartment="cytosol" id="E"  initialAmount="5e-21" name="E"/>
        </listOfSpecies>
        <listOfReactions>
            <reaction id="veq">
                <listOfReactants>
                    <speciesReference species="E"/>
                    <speciesReference species="S"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="ES"/>
                </listOfProducts>
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <times/>
                            <ci>cytosol</ci>
                            <apply>
                                <minus/>
                                <apply>
                                    <times/>
                                    <ci>kon</ci>
                                    <ci>E</ci>
                                    <ci>S</ci>
                                </apply>
                                <apply>
                                    <times/>
                                    <ci>koff</ci>
                                    <ci>ES</ci>
                                </apply>
                            </apply>
                        </apply>
                    </math>
```

```
                        <listOfParameters>
                            <parameter id="kon"  value="1000000" units="litre_per_mole_per_second"/>
                            <parameter id="koff" value="0.2"     units="per_second"/>
                        </listOfParameters>
                    </kineticLaw>
                </reaction>
                <reaction id="vcat" reversible="false">
                    <listOfReactants>
                        <speciesReference species="ES"/>
                    </listOfReactants>
                    <listOfProducts>
                        <speciesReference species="E"/>
                        <speciesReference species="P"/>
                    </listOfProducts>
                    <kineticLaw>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                                <times/>
                                <ci>cytosol</ci>
                                <ci>kcat</ci>
                                <ci>ES</ci>
                            </apply>
                        </math>
                        <listOfParameters>
                            <parameter id="kcat" value="0.1" units="per_second"/>
                        </listOfParameters>
                    </kineticLaw>
                </reaction>
            </listOfReactions>
        </model>
    </sbml>
```

In this example, the model has the identifier "`EnzymaticReaction`". The model contains one compartment (with identifier "`cytosol`"), four species (with identifiers "`ES`", "`P`", "`S`", and "`E`"), and two reactions ("`veq`" and "`vcat`"). The elements in the `listOfReactants` and `listOfProducts` in each reaction refer to the names of elements listed in the `listOfSpecies`. The correspondences between the various elements is explicitly stated by the `speciesReference` elements.

The model also features local parameter definitions in each reaction. In this case, the three parameters ("`kon`", "`koff`", "`kcat`") all have unique identifiers and they could also have just as easily been declared global parameters in the model. Local parameters frequently become more useful in larger models, where it may become tedious to assign unique identifiers for all the different parameters.

7.2 Example involving units

The following model uses the units features of SBML Level 2. In this model, the default value of `substance` is changed to be mole units with a scale factor of -3, or millimoles. This sets the default substance units in the model. The `volume` and `time` built-in units are left to their defaults, meaning volume is in litres and time is in seconds. The result is that, in this model, kinetic law formulas define rates in millimoles per second and the species identifiers in them represent concentration values in millimoles per litres. All the `species` elements set the initial amount of every given species to 1 millimole. The parameters "`vm`" and "`km`" are defined to be in millimoles per litres per second, and millimoles per litres, respectively.

```
    <?xml version="1.0" encoding="UTF-8"?>
    <sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
          xmlns:xhtml="http://www.w3.org/1999/xhtml">
        <model>
            <listOfUnitDefinitions>
                <unitDefinition id="substance">
                    <listOfUnits>
                        <unit kind="mole" scale="-3"/>
                    </listOfUnits>
                </unitDefinition>
                <unitDefinition id="mmls">
                    <listOfUnits>
```

```
                <unit kind="mole" scale="-3"/>
                <unit kind="litre" exponent="-1"/>
                <unit kind="second" exponent="-1"/>
            </listOfUnits>
        </unitDefinition>
        <unitDefinition id="mml">
            <listOfUnits>
                <unit kind="mole" scale="-3"/>
                <unit kind="litre" exponent="-1"/>
            </listOfUnits>
        </unitDefinition>
    </listOfUnitDefinitions>
    <listOfCompartments>
        <compartment id="cell" size="1"/>
    </listOfCompartments>
    <listOfSpecies>
        <species id="x0" compartment="cell" initialConcentration="1"/>
        <species id="x1" compartment="cell" initialConcentration="1"/>
        <species id="s1" compartment="cell" initialConcentration="1"/>
        <species id="s2" compartment="cell" initialConcentration="1"/>
    </listOfSpecies>
    <listOfParameters>
        <parameter id="vm" value="2" units="mmls"/>
        <parameter id="km" value="2" units="mml"/>
    </listOfParameters>
    <listOfReactions>
        <reaction id="v1">
            <listOfReactants>
                <speciesReference species="x0"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="s1"/>
            </listOfProducts>
            <kineticLaw>
                <notes>
                    <xhtml:p>((vm * s1)/(km + s1))*cell</xhtml:p>
                </notes>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <apply>
                            <divide/>
                            <apply>
                                <times/>
                                <ci> vm </ci>
                                <ci> s1 </ci>
                            </apply>
                            <apply>
                                <plus/>
                                <ci> km </ci>
                                <ci> s1 </ci>
                            </apply>
                        </apply>
                        <ci> cell </ci>
                    </apply>
                </math>
            </kineticLaw>
        </reaction>
        <reaction id="v2">
            <listOfReactants>
                <speciesReference species="s1"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="s2"/>
            </listOfProducts>
            <kineticLaw>
                <notes>
                    <xhtml:p>((vm * s2)/(km + s2))*cell</xhtml:p>
                </notes>
```

```
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                                <times/>
                                <apply>
                                    <divide/>
                                    <apply>
                                        <times/>
                                        <ci> vm </ci>
                                        <ci> s2 </ci>
                                    </apply>
                                    <apply>
                                        <plus/>
                                        <ci> km </ci>
                                        <ci> s2 </ci>
                                    </apply>
                                </apply>
                                <ci> cell </ci>
                            </apply>
                        </math>
                    </kineticLaw>
                </reaction>
                <reaction id="v3">
                    <listOfReactants>
                        <speciesReference species="s2"/>
                    </listOfReactants>
                    <listOfProducts>
                        <speciesReference species="x1"/>
                    </listOfProducts>
                    <kineticLaw>
                        <notes>
                            <xhtml:p>((vm * x1)/(km + x1))*cell</xhtml:p>
                        </notes>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                                <times/>
                                <apply>
                                    <divide/>
                                    <apply>
                                        <times/>
                                        <ci> vm </ci>
                                        <ci> x1 </ci>
                                    </apply>
                                    <apply>
                                        <plus/>
                                        <ci> km </ci>
                                        <ci> x1 </ci>
                                    </apply>
                                </apply>
                                <ci> cell </ci>
                            </apply>
                        </math>
                    </kineticLaw>
                </reaction>
            </listOfReactions>
        </model>
    </sbml>
```

7.3 Example of a discrete version of a simple dimerization reaction

(Model contributed by Darren J. Wilkinson, Newcastle University, Newcastle upon Tyne, UK.)

This example illustrates subtle differences between models formulated for use in a continuous simulation framework (e.g., using differential equations) and those intended for a discrete simulation framework. The model shown here is suitable for use with a discrete stochastic simulation algorithm of the sort developed by Gillespie (1977). In such an approach, species are described in terms of molecular counts and simulation proceeds by computing the probability of the time and identity of the next reaction, then updating the species amounts appropriately.

The model involves a simple dimerization reaction for a protein named "P":

$$2P \leftrightarrow P_2$$

The SBML representation is shown below. There are several important points to note. First, the species "P" and "P2" declare they are always in discrete amounts by using the flag hasOnlySubstanceUnits="true". This indicates that when the species identifiers appear in mathematical formulas, the units are *substance*, not the default of *substance/size*. A second point is that, as a result, the corresponding "kinetic law" formulas do not need volume corrections. In Gillespie's approach, the constants in the rate expressions (here, "c1" and "c2") contain a contribution from the kinetic constants of the reaction and the size of the compartment in which the reactions take place. Finally, it is worth noting the rate expression for the forward reaction is a second-order mass-action reaction, but it is the *discrete* formulation of such a reaction rate (Gillespie, 1977).

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model id="dimerization">
        <listOfUnitDefinitions>
            <unitDefinition id="substance">
                <listOfUnits>
                    <unit kind="item" multiplier="1"/>
                </listOfUnits>
            </unitDefinition>
            <unitDefinition id="per_second">
                <listOfUnits>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
        </listOfUnitDefinitions>
        <listOfCompartments>
            <compartment id="Cell" size="1e-15"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="P"  compartment="Cell" initialAmount="301"
                             hasOnlySubstanceUnits="true"/>
            <species id="P2" compartment="Cell" initialAmount="0"
                             hasOnlySubstanceUnits="true"/>
        </listOfSpecies>
        <listOfReactions>
            <reaction id="Dimerization" reversible="false">
                <listOfReactants>
                    <speciesReference species="P" stoichiometry="2"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="P2"/>
                </listOfProducts>
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <divide/>
                            <apply>
                                <times/>
                                <ci> c1 </ci>
                                <ci> P </ci>
                                <apply>
                                    <minus/>
                                    <ci> P </ci>
                                    <cn type="integer"> 1 </cn>
                                </apply>
                            </apply>
                            <cn type="integer"> 2 </cn>
                        </apply>
                    </math>
                    <listOfParameters>
                        <parameter id="c1" value="0.00166" units="per_second"/>
                    </listOfParameters>
                </kineticLaw>
            </reaction>
```

```
                <reaction id="Dissociation" reversible="false">
                    <listOfReactants>
                        <speciesReference species="P2"/>
                    </listOfReactants>
                    <listOfProducts>
                        <speciesReference species="P" stoichiometry="2"/>
                    </listOfProducts>
                    <kineticLaw>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                                <times/>
                                <ci> c2 </ci>
                                <ci> P </ci>
                            </apply>
                        </math>
                        <listOfParameters>
                            <parameter id="c2" value="0.2" units="per_second"/>
                        </listOfParameters>
                    </kineticLaw>
                </reaction>
            </listOfReactions>
        </model>
    </sbml>
```

This example also illustrates the need to provide additional information in a model so that software tools using different mathematical frameworks can properly interpret it. In this case, a simulation tool designed for continuous ODE-based simulation would likely misinterpret the model (in particular the reaction rate formulas), unless it deduced that a discrete stochastic simulation was intended. One of the purposes of SBO annotations (Section 5) is to enable such interpretation without the need for deduction.

7.4 Example involving assignment rules

This section contains a model that simulates a system containing a fast reaction. This model uses rules to express the mathematics of the fast reaction explicitly rather than using the `fast` attribute on a reaction element. The system modeled is

$$\begin{aligned} X_0 &\xrightarrow{k_1 X_0} S_1 \\ S_1 &\xrightarrow{k_f S_1 - k_r S_2} S_2 \\ S_2 &\xrightarrow{k_2 S_2} X_1 \end{aligned}$$

$$k_1 = 0.1, \quad k_2 = 0.15, \quad k_f = K_{eq}10000, \quad k_r = 10000, \quad K_{eq} = 2.5.$$

where X_0, S_1, S_1, and S_2 are species in concentration units, and k_1, k_2, k_f, k_r, and K_{eq} are parameters. This system of reactions can be approximated with the following new system:

$$\begin{aligned} X_0 &\xrightarrow{k_1 X_0} T \\ T &\xrightarrow{k_2 S_2} X_1 \end{aligned}$$

$$S_1 = \frac{T}{1 + K_{eq}}$$

$$S_2 = K_{eq} S_1$$

where T is a new species. The following example SBML model encodes the second system.

```
    <?xml version="1.0" encoding="UTF-8"?>
    <sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
          xmlns:math="http://www.w3.org/1998/Math/MathML">
        <model>
```

```
<listOfUnitDefinitions>
    <unitDefinition id="per_second">
        <listOfUnits>
            <unit kind="second" exponent="-1"/>
        </listOfUnits>
    </unitDefinition>
</listOfUnitDefinitions>
<listOfCompartments>
    <compartment id="cell" size="1"/>
</listOfCompartments>
<listOfSpecies>
    <species id="X0" compartment="cell" initialConcentration="1"/>
    <species id="X1" compartment="cell" initialConcentration="0"/>
    <species id="T"  compartment="cell" initialConcentration="0"/>
    <species id="S1" compartment="cell" initialConcentration="0"/>
    <species id="S2" compartment="cell" initialConcentration="0"/>
</listOfSpecies>
<listOfParameters>
    <parameter id="Keq" value="2.5" units="dimensionless"/>
</listOfParameters>
<listOfRules>
    <assignmentRule variable="S1">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
                <divide/>
                <ci> T </ci>
                <apply>
                    <plus/>
                    <cn> 1 </cn>
                    <ci> Keq </ci>
                </apply>
            </apply>
        </math>
    </assignmentRule>
    <assignmentRule variable="S2">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
                <times/>
                <ci> Keq </ci>
                <ci> S1 </ci>
            </apply>
        </math>
    </assignmentRule>
</listOfRules>
<listOfReactions>
    <reaction id="in" reversible="false">
        <listOfReactants>
            <speciesReference species="X0"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="T"/>
        </listOfProducts>
        <kineticLaw>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <times/>
                    <ci> k1 </ci>
                    <ci> X0 </ci>
                    <ci> cell </ci>
                </apply>
            </math>
            <listOfParameters>
                <parameter id="k1" value="0.1" units="per_second"/>
            </listOfParameters>
        </kineticLaw>
    </reaction>
    <reaction id="out" reversible="false">
        <listOfReactants>
            <speciesReference species="T"/>
```

```
                    </listOfReactants>
                    <listOfProducts>
                        <speciesReference species="X1"/>
                    </listOfProducts>
                    <listOfModifiers>
                        <modifierSpeciesReference species="S2"/>
                    </listOfModifiers>
                    <kineticLaw>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                                <times/>
                                <ci> k2 </ci>
                                <ci> S2 </ci>
                                <ci> cell </ci>
                            </apply>
                        </math>
                        <listOfParameters>
                            <parameter id="k2" value="0.15" units="per_second"/>
                        </listOfParameters>
                    </kineticLaw>
                </reaction>
            </listOfReactions>
        </model>
    </sbml>
```

7.5 Example involving algebraic rules

This section contains an example model that contains an **AlgebraicRule** object. The model contains a different formulation of the fast reaction described in Section 7.4. The system described in Section 7.4 can be approximated with the following system:

$$X_0 \xrightarrow{k_1 X_0} T$$

$$T \xrightarrow{k_2 S_1} X_1$$

$$S_2 = K_{eq} S_1$$

with the constraint:

$$S_1 + S_2 - T = 0$$

The following example SBML model encodes this approximate form.

```
    <?xml version="1.0" encoding="UTF-8"?>
    <sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
        <model>
            <listOfUnitDefinitions>
                <unitDefinition id="per_second">
                    <listOfUnits>
                        <unit kind="second" exponent="-1"/>
                    </listOfUnits>
                </unitDefinition>
            </listOfUnitDefinitions>
            <listOfCompartments>
                <compartment id="cell" size="1"/>
            </listOfCompartments>
            <listOfSpecies>
                <species id="X0" compartment="cell" initialConcentration="1"/>
                <species id="X1" compartment="cell" initialConcentration="0"/>
                <species id="T"  compartment="cell" initialConcentration="0"/>
                <species id="S1" compartment="cell" initialConcentration="0"/>
                <species id="S2" compartment="cell" initialConcentration="0"/>
            </listOfSpecies>
            <listOfParameters>
                <parameter id="Keq" value="2.5" units="dimensionless"/>
            </listOfParameters>
```

```
<listOfRules>
    <assignmentRule variable="S2">
        <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
                <times/>
                <ci> Keq </ci>
                <ci> S1 </ci>
            </apply>
        </math>
    </assignmentRule>
    <algebraicRule>
        <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
                <minus/>
                <apply>
                    <plus/>
                    <ci> S2 </ci>
                    <ci> S1 </ci>
                </apply>
                <ci> T </ci>
            </apply>
        </math>
    </algebraicRule>
</listOfRules>
<listOfReactions>
    <reaction id="in" reversible="false">
        <listOfReactants>
            <speciesReference species="X0"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="T"/>
        </listOfProducts>
        <kineticLaw>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <times/>
                    <ci> k1 </ci>
                    <ci> X0 </ci>
                    <ci> cell </ci>
                </apply>
            </math>
            <listOfParameters>
                <parameter id="k1" value="0.1" units="per_second"/>
            </listOfParameters>
        </kineticLaw>
    </reaction>
    <reaction id="out" reversible="false">
        <listOfReactants>
            <speciesReference species="T"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="X1"/>
        </listOfProducts>
        <listOfModifiers>
                <modifierSpeciesReference species="S2"/>
        </listOfModifiers>
        <kineticLaw>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <times/>
                    <ci> k2 </ci>
                    <ci> S2 </ci>
                    <ci> cell </ci>
                </apply>
            </math>
            <listOfParameters>
                <parameter id="k2" value="0.15" units="per_second"/>
            </listOfParameters>
        </kineticLaw>
```

```
            </reaction>
        </listOfReactions>
    </model></sbml>
```

7.6 Example with combinations of boundaryCondition and constant values on Species with RateRule objects

In this section, we discuss a model that includes four species, each with a different combination of values for their boundaryCondition and constant attributes. The model represents a hypothetical system containing one reaction,

$$S_1 + S_2 \xrightarrow{k_1 S_1 S_2 S_3} S_4$$

where S_3 is a species that catalyzes the conversion of species S_1 and S_2 into S_4. S_1 and S_2 are on the boundary of the system (i.e., S_1 and S_2 are reactants but their values are not determined by a kinetic law). The value of S_1 in the system is determined over time by the rate rule:

$$\frac{dS_1}{dt} = k_2$$

The values of constant parameters in the system are:

$$S_2 = 1, \quad S_3 = 2, \quad k_1 = 0.5, \quad k_2 = 0.1$$

and the initial values of species are:

$$S_1 = 0, \quad S_4 = 0$$

The value of S_1 varies over time so in SBML S_1 has a constant attribute with a default value of "false". The values of S_2 and S_3 are fixed so in SBML they have a constant attribute values of "true". S_3 only occurs as a modifier so the value of its boundaryCondition attribute can default to "false". S_4 is a product whose value is determined by a kinetic law and therefore in the SBML representation has "false" (the default) for both its boundaryCondition and constant attributes.

The following is the SBML rendition of the model shown above:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model id="BoundaryCondExampleModel">
        <listOfUnitDefinitions>
            <unitDefinition id="mole_per_litre_per_second">
                <listOfUnits>
                    <unit kind="mole" />
                    <unit kind="litre" exponent="-1"/>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
            <unitDefinition id="litre_sq_per_mole_sq_per_second">
                <listOfUnits>
                    <unit kind="mole" exponent="-2"/>
                    <unit kind="litre" exponent="2"/>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
        </listOfUnitDefinitions>
        <listOfCompartments>
            <compartment id="compartmentOne" size="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="S1" initialConcentration="0" compartment="compartmentOne"
                             boundaryCondition="true"/>
            <species id="S2" initialConcentration="1" compartment="compartmentOne"
                             boundaryCondition="true" constant="true"/>
            <species id="S3" initialConcentration="3" compartment="compartmentOne"
                             constant="true"/>
```

```
            <species id="S4" initialConcentration="0" compartment="compartmentOne"/>
        </listOfSpecies>
        <listOfParameters>
            <parameter id="k1" value="0.5" units="litre_sq_per_mole_sq_per_second"/>
            <parameter id="k2" value="0.1" units="mole_per_litre_per_second"/>
        </listOfParameters>
        <listOfRules>
            <rateRule variable="S1">
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <ci> k2 </ci>
                </math>
            </rateRule>
        </listOfRules>
        <listOfReactions>
            <reaction id="reaction_1" reversible="false">
                <listOfReactants>
                    <speciesReference species="S1"/>
                    <speciesReference species="S2"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="S4"/>
                </listOfProducts>
                <listOfModifiers>
                    <modifierSpeciesReference species="S3"/>
                </listOfModifiers>
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <times/>
                            <ci> k1 </ci>
                            <ci> S1 </ci>
                            <ci> S2 </ci>
                            <ci> S3 </ci>
                            <ci> compartmentOne </ci>
                        </apply>
                    </math>
                </kineticLaw>
            </reaction>
        </listOfReactions>
    </model>
</sbml>
```

7.7 Example of translation from a multi-compartmental model to ODEs

This section contains a model with 2 compartments and 4 reactions. The model is derived from Lotka-Volterra, with the addition of a reversible transport step. When observed in a time-course simulation, three of this model's species display damped oscillations.

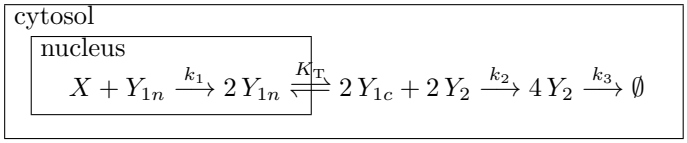


Figure 31: *A example multi-compartmental model.*

Figure 31 illustrates the arrangement of compartments and reactions in the model `LotkaVolterra_tranport`. The text of the SBML representation of the model is shown below, and it is followed by its complete translation into ordinary differential equations. In this SBML model, the reaction equations are in substance per time units. The reactions have also been simplified to reduce common stoichiometric factors. The species variables are in concentration units; their initial quantities are declared using the attribute `initialAmount` on the `species` definitions, but since the attribute `hasOnlySubstanceUnits` is *not* set to true, the identifiers of the species represent their concentrations when those identifiers appear in mathematical expressions elsewhere

in the model. Note that the species whose identifier is "X" is a boundary condition, as indicated by the attribute boundaryCondition="true" in its definition. The attribute speciesType="Y" in the definitions of "Y1n" and "Y1c" indicates that these species are pools of the same participant, but located in different compartments.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model name="LotkaVolterra_tranport">
        <listOfUnitDefinitions>
            <unitDefinition id="per_second">
                <listOfUnits>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
            <unitDefinition id="litre_per_mole_per_second">
                <listOfUnits>
                    <unit kind="mole" exponent="-1"/>
                    <unit kind="litre" exponent="1"/>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
        </listOfUnitDefinitions>
        <listOfSpeciesTypes>
            <speciesType id="Y1"/>
        </listOfSpeciesTypes>
        <listOfCompartments>
            <compartment id="cytoplasm" size="5"/>
            <compartment id="nucleus" outside="cytoplasm" size="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="X"   compartment="nucleus"   initialAmount="1" constant="true"
                              boundaryCondition="true"/>
            <species id="Y1n" compartment="nucleus"   speciesType="Y1"  initialAmount="1"/>
            <species id="Y1c" compartment="cytoplasm" speciesType="Y1"  initialAmount="0"/>
            <species id="Y2"  compartment="cytoplasm" initialAmount="1"/>
        </listOfSpecies>
        <listOfParameters>
            <parameter id="k1" value="2500"  units="litre_per_mole_per_second"/>
            <parameter id="k2" value="2500"  units="litre_per_mole_per_second"/>
            <parameter id="KT" value="25000" units="per_second"/>
            <parameter id="k3" value="2500"  units="per_second"/>
        </listOfParameters>
        <listOfReactions>
            <reaction id="production" reversible="false">
                <listOfReactants>
                    <speciesReference species="X"/>
                    <speciesReference species="Y1n"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="Y1n"/>
                    <speciesReference species="Y1n"/>
                </listOfProducts>
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <times/>
                            <ci>nucleus</ci>
                            <ci>k1</ci>
                            <ci>X</ci>
                            <ci>Y1n</ci>
                        </apply>
                    </math>
                </kineticLaw>
            </reaction>
            <reaction id="transport" reversible="true">
                <listOfReactants>
                    <speciesReference species="Y1n"/>
                </listOfReactants>
                <listOfProducts>
```

```
                        <speciesReference species="Y1c"/>
                    </listOfProducts>
                    <kineticLaw>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                                <times/>
                                <ci>cytoplasm</ci>
                                <ci>KT</ci>
                                <apply>
                                    <minus/>
                                    <ci>Y1n</ci>
                                    <ci>Y1c</ci>
                                </apply>
                            </apply>
                        </math>
                    </kineticLaw>
                </reaction>
                <reaction id="transformation" reversible="false">
                    <listOfReactants>
                        <speciesReference species="Y1c"/>
                        <speciesReference species="Y2"/>
                    </listOfReactants>
                    <listOfProducts>
                        <speciesReference species="Y2" stoichiometry="2"/>
                    </listOfProducts>
                    <kineticLaw>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                                <times/>
                                <ci>cytoplasm</ci>
                                <ci>k2</ci>
                                <ci>Y1c</ci>
                                <ci>Y2</ci>
                            </apply>
                        </math>
                    </kineticLaw>
                </reaction>
                <reaction id="degradation" reversible="false">
                    <listOfReactants>
                        <speciesReference species="Y2"/>
                    </listOfReactants>
                    <kineticLaw>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                                <times/>
                                <ci>cytoplasm</ci>
                                <ci>k3</ci>
                                <ci>Y2</ci>
                            </apply>
                        </math>
                    </kineticLaw>
                </reaction>
            </listOfReactions>
        </model>
    </sbml>
```

The ODE translation of this model is as follows. First, we give the values of the constant parameters:

$$k_1 = 2500, \quad k_2 = 2500, \quad K_3 = 25000, \quad K_T = 2500$$

Now on to the initial conditions of the variables. In the following, the symbols representing species (X, Y_{1n}, Y_{1c}, and Y_2) have values in terms of concentrations. (Readers may wonder why, when their values in the SBML model are given as initial *amounts*. The reason goes back to the **Species** defaults and the meaning of the `hasOnlySubstanceUnits` attribute: if the attribute is not set and the compartment in which the species is located has more than 0 spatial dimensions, a species' symbol in a model is interpreted as a concentration or density regardless of whether its initial value is set using `initialAmount` or `initialConcentration`.) We

use V_n to represent the size of compartment "nucleus" and V_c the size of compartment "cytoplasm":

$$V_n = 1, \quad V_c = 5, \quad X = 1, \quad Y_{1n} = 1, \quad Y_{1c} = 0, \quad Y_2 = 1/5$$

And finally, here are the differential equations:

$$\frac{dX}{dt} = 0$$

$$V_n \frac{dY_{1n}}{dt} = k_1 \cdot X \cdot Y_{1n} \cdot V_n - K_T \cdot (Y_{1n} - Y_{1c}) \cdot V_c \qquad \text{reactions production and transport}$$

$$V_c \frac{dY_{1c}}{dt} = K_T \cdot (Y_{1n} - Y_{1c}) \cdot V_c - k_2 \cdot Y_{1c} \cdot Y_2 \cdot V_c \qquad \text{reactions transport and transformation}$$

$$V_c \frac{dY_2}{dt} = k_2 \cdot Y_{1c} \cdot Y_2 \cdot V_c - k_3 \cdot Y_2 \cdot V_c \qquad \text{reactions transformation and degradation}$$

As formulated here, this example assumes constant volumes. If the sizes of the compartments "cytoplasm" or "nucleus" could change during simulation, then it would be preferable to use a different approach to constructing the differential equations. In this alternative approach, the ODEs would compute substance change rather than concentration change, and the concentration values would be computed using separate equations. This approach is used in Section 4.13.6.

7.8 Example involving function definitions

This section contains a model that uses the function definition feature of SBML. Consider the following hypothetical system:

$$S_1 \xrightarrow{f(S_1)} S_2$$

where

$$f(x) = 2 \times x$$

The following is the XML document that encodes the model shown above:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model id="Example">
        <listOfFunctionDefinitions>
            <functionDefinition id="f">
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <lambda>
                        <bvar>
                            <ci> x </ci>
                        </bvar>
                        <apply>
                            <times/>
                            <ci> x </ci>
                            <cn> 2 </cn>
                        </apply>
                    </lambda>
                </math>
            </functionDefinition>
        </listOfFunctionDefinitions>
        <listOfCompartments>
            <compartment id="compartmentOne" size="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="S1" initialConcentration="1" compartment="compartmentOne"/>
            <species id="S2" initialConcentration="0" compartment="compartmentOne"/>
        </listOfSpecies>
        <listOfParameters>
```

```
                <parameter id="t" value = "1" units="second"/>
            </listOfParameters>
            <listOfReactions>
                <reaction id="reaction_1" reversible="false">
                    <listOfReactants>
                        <speciesReference species="S1"/>
                    </listOfReactants>
                    <listOfProducts>
                        <speciesReference species="S2"/>
                    </listOfProducts>
                    <kineticLaw>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <apply>
                            <divide/>
                                <apply>
                                    <times/>
                                    <apply>
                                        <ci> f </ci>
                                        <ci> S1 </ci>
                                    </apply>
                                    <ci> compartmentOne </ci>
                                </apply>
                                <ci> t</ci>
                            </apply>
                        </math>
                    </kineticLaw>
                </reaction>
            </listOfReactions>
        </model>
    </sbml>
```

7.9 Example involving *delay* functions

The following is a simple model illustrating the use of *delay* to represent a gene that suppresses its own expression. The model can be expressed in a single rule:

$$\frac{dP}{dt} = \frac{\frac{1}{1 + m(P_{delayed})^q} - P}{\tau}$$

where

$P_{delayed}$	is $delay(P, \Delta_t)$ or P at $t - \Delta_t$
P	is protein concentration
τ	is the response time
m	is a multiplier or equilibrium constant
q	is the Hill coefficient

and the species quantities are in concentration units. The text of an SBML encoding of this model is given below:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model>
        <listOfCompartments>
            <compartment id="cell" size="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="P" compartment="cell" initialConcentration="0"/>
        </listOfSpecies>
        <listOfParameters>
            <parameter id="tau"     value="1"   units="second"/>
            <parameter id="m"       value="0.5" units="dimensionless"/>
            <parameter id="q"       value="1"   units="dimensionless"/>
            <parameter id="delta_t" value="1"   units="second"/>
        </listOfParameters>
        <listOfRules>
```

```
                <rateRule variable="P">
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                      <apply>
                        <divide/>
                        <apply>
                          <minus/>
                          <apply>
                            <divide/>
                            <cn> 1 </cn>
                            <apply>
                              <plus/>
                              <cn> 1 </cn>
                              <apply>
                                <times/>
                                <ci> m </ci>
                                <apply>
                                  <power/>
                                  <apply>
                                    <csymbol encoding="text"
                                             definitionURL="http://www.sbml.org/sbml/symbols/delay">
                                        delay
                                    </csymbol>
                                    <ci> P </ci>
                                    <ci> delta_t </ci>
                                  </apply>
                                  <ci> q </ci>
                                </apply>
                              </apply>
                            </apply>
                          </apply>
                          <ci> P </ci>
                        </apply>
                        <ci> tau </ci>
                      </apply>
                    </math>
                </rateRule>
            </listOfRules>
        </model>
    </sbml>
```

7.10 Example involving events

This section presents a simple model system that demonstrates the use of events in SBML. Consider a system with two genes, G_1 and G_2. G_1 is initially on and G_2 is initially off. When turned on, the two genes lead to the production of two products, P_1 and P_2, respectively, at a fixed rate. When P_1 reaches a given concentration, G_2 switches on. This system can be represented mathematically as follows:

$$\begin{aligned} \frac{dP_1}{dt} &= k_1(G_1 - P_1) \\ \frac{dP_2}{dt} &= k_2(G_2 - P_2) \\ G_2 &= \begin{cases} 0 & \text{when } P_1 \leq \tau, \\ 1 & \text{when } P_1 > \tau. \end{cases} \end{aligned}$$

The initial values are:

$$G_1 = 1, \quad G_2 = 0, \quad \tau = 0.25, \quad P_1 = 0, \quad P_2 = 0, \quad k_1 = k_2 = 1.$$

The SBML Level 2 representation of this as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
      xmlns:math="http://www.w3.org/1998/Math/MathML">
```

```
<model>
    <listOfUnitDefinitions>
        <unitDefinition id="per_second">
            <listOfUnits>
                <unit kind="second" exponent="-1"/>
            </listOfUnits>
        </unitDefinition>
        <unitDefinition id="concentration">
            <listOfUnits>
                <unit kind="mole"/>
                <unit kind="litre" exponent="-1"/>
            </listOfUnits>
        </unitDefinition>
    </listOfUnitDefinitions>
    <listOfCompartments>
        <compartment id="cell" size="1"/>
    </listOfCompartments>
    <listOfSpecies>
        <species id="P1" compartment="cell" initialConcentration="0"/>
        <species id="P2" compartment="cell" initialConcentration="0"/>
    </listOfSpecies>
    <listOfParameters>
        <parameter id="k1"   value="1"    units="per_second"/>
        <parameter id="k2"   value="1"    units="per_second" />
        <parameter id="tau"  value="0.25" units="concentration"/>
        <parameter id="G1"   value="1"    units="concentration" constant="false"/>
        <parameter id="G2"   value="0"    units="concentration" constant="false"/>
    </listOfParameters>
    <listOfRules>
        <rateRule variable="P1">
            <math:math>
                <math:apply>
                    <math:times/>
                    <math:ci> k1 </math:ci>
                    <math:apply>
                        <math:minus/>
                        <math:ci> G1 </math:ci>
                        <math:ci> P1 </math:ci>
                    </math:apply>
                </math:apply>
            </math:math>
        </rateRule>
        <rateRule variable="P2">
            <math:math>
                <math:apply>
                    <math:times/>
                    <math:ci> k2 </math:ci>
                    <math:apply>
                        <math:minus/>
                        <math:ci> G2 </math:ci>
                        <math:ci> P2 </math:ci>
                    </math:apply>
                </math:apply>
            </math:math>
        </rateRule>
    </listOfRules>
    <listOfEvents>
        <event>
            <trigger>
                <math:math>
                    <math:apply>
                        <math:gt/>
                        <math:ci> P1 </math:ci>
                        <math:ci> tau </math:ci>
                    </math:apply>
                </math:math>
            </trigger>
            <listOfEventAssignments>
                <eventAssignment variable="G2">
```

```
                        <math:math>
                            <math:cn> 1 </math:cn>
                        </math:math>
                    </eventAssignment>
                </listOfEventAssignments>
            </event>
            <event>
                <trigger>
                    <math:math>
                        <math:apply>
                            <math:leq/>
                            <math:ci> P1 </math:ci>
                            <math:ci> tau </math:ci>
                        </math:apply>
                    </math:math>
                </trigger>
                <listOfEventAssignments>
                    <eventAssignment variable="G2">
                        <math:math>
                            <math:cn> 0 </math:cn>
                        </math:math>
                    </eventAssignment>
                </listOfEventAssignments>
            </event>
        </listOfEvents>
    </model>
</sbml>
```

7.11 Example involving two-dimensional compartments

The following example is a model that uses a two-dimensional compartment. It is a fragment of a larger model of calcium regulation across the plasma membrane of a cell. The model includes a calcium influx channel, "Ca_channel", and a calcium-extruding PMCA pump, "Ca_Pump". It also includes two cytosolic proteins that buffer calcium via the "CalciumCalbindin_gt_BoundCytosol" and "CalciumBuffer_gt_BoundCytosol" reactions. Finally, the rate expressions in this model do not include explicit factors of the compartment volumes; instead, the various rate constants are assume to include any necessary corrections for volume.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model id="facilitated_ca_diffusion">
        <listOfUnitDefinitions>
            <unitDefinition id="substance">
                <listOfUnits>
                    <unit kind="mole" scale="-6"/>
                </listOfUnits>
            </unitDefinition>
            <unitDefinition id="area">
                <listOfUnits>
                    <unit kind="metre" scale="-6" exponent="2"/>
                </listOfUnits>
            </unitDefinition>
            <unitDefinition id="per_second">
                <listOfUnits>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
            <unitDefinition id="litre_per_mole_per_second">
                <listOfUnits>
                    <unit kind="mole" exponent="-1" scale="-6"/>
                    <unit kind="litre" exponent="1"/>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
            <unitDefinition id="subs_per_vol">
                <listOfUnits>
                    <unit kind="mole" exponent="1" scale="-6"/>
                    <unit kind="litre" exponent="-1"/>
```

```
            </listOfUnits>
        </unitDefinition>
    </listOfUnitDefinitions>
    <listOfCompartments>
        <compartment id="Extracellular"
                     spatialDimensions="3" size="1"/>
        <compartment id="PlasmaMembrane"
                     outside="Extracellular" spatialDimensions="2" size="1"/>
        <compartment id="Cytosol"
                     outside="PlasmaMembrane" spatialDimensions="3" size="1"/>
    </listOfCompartments>
    <listOfSpecies>
        <species id="CaBPB_C"
                 compartment="Cytosol" initialConcentration="47.17"/>
        <species id="B_C"
                 compartment="Cytosol" initialConcentration="396.04"/>
        <species id="CaB_C"
                 compartment="Cytosol" initialConcentration="3.96"/>
        <species id="Ca_C"
                 name="Ca" compartment="Cytosol" initialConcentration="0.1"/>
        <species id="Ca_EC"
                  name="Ca" compartment="Extracellular" initialConcentration="1000"/>
        <species id="CaCh_PM"
                 compartment="PlasmaMembrane" initialConcentration="1"/>
        <species id="CaPump_PM"
                 compartment="PlasmaMembrane" initialConcentration="1"/>
        <species id="CaBP_C"
                 compartment="Cytosol" initialConcentration="202.83"/>
    </listOfSpecies>
    <listOfReactions>
        <reaction id="CalciumCalbindin_gt_BoundCytosol" fast="true">
            <listOfReactants>
                <speciesReference species="CaBP_C"/>
                <speciesReference species="Ca_C"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="CaBPB_C"/>
            </listOfProducts>
            <kineticLaw>
                <notes>
                    <p xmlns="http://www.w3.org/1999/xhtml">
                       (((Kf_CalciumCalbindin_BoundCytosol * CaBP_C) * Ca_C) -
                          (Kr_CalciumCalbindin_BoundCytosol * CaBPB_C))
                    </p>
                </notes>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <ci> Cytosol </ci>
                        <apply>
                            <minus/>
                            <apply>
                                <times/>
                                <ci> Kf_CalciumCalbindin_BoundCytosol </ci>
                                <ci> CaBP_C </ci>
                                <ci> Ca_C </ci>
                            </apply>
                            <apply>
                                <times/>
                                <ci> Kr_CalciumCalbindin_BoundCytosol </ci>
                                <ci> CaBPB_C </ci>
                            </apply>
                        </apply>
                    </apply>
                </math>
                <listOfParameters>
                    <parameter id="Kf_CalciumCalbindin_BoundCytosol" value="20.0"
                               units="litre_per_mole_per_second"/>
                    <parameter id="Kr_CalciumCalbindin_BoundCytosol" value="8.6"
```

```
                            units="per_second"/>
            </listOfParameters>
        </kineticLaw>
    </reaction>
    <reaction id="CalciumBuffer_gt_BoundCytosol" fast="true">
        <listOfReactants>
            <speciesReference species="Ca_C"/>
            <speciesReference species="B_C"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="CaB_C"/>
        </listOfProducts>
        <kineticLaw>
            <notes>
                <p xmlns="http://www.w3.org/1999/xhtml">
                (((Kf_CalciumBuffer_BoundCytosol * Ca_C) * B_C) -
                    (Kr_CalciumBuffer_BoundCytosol * CaB_C))
                </p>
            </notes>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
                <times/>
                <ci> Cytosol</ci>
                <apply>
                    <minus/>
                    <apply>
                        <times/>
                        <ci> Kf_CalciumBuffer_BoundCytosol </ci>
                        <ci> Ca_C </ci>
                        <ci> B_C </ci>
                    </apply>
                    <apply>
                        <times/>
                        <ci> Kr_CalciumBuffer_BoundCytosol </ci>
                        <ci> CaB_C </ci>
                    </apply>
                </apply>
            </apply>
            </math>
            <listOfParameters>
                <parameter id="Kf_CalciumBuffer_BoundCytosol" value="0.1"
                           units="litre_per_mole_per_second"/>
                <parameter id="Kr_CalciumBuffer_BoundCytosol" value="1.0"
                           units="per_second"/>
            </listOfParameters>
        </kineticLaw>
    </reaction>
    <reaction id="Ca_Pump">
        <listOfReactants>
            <speciesReference species="Ca_C"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="Ca_EC"/>
        </listOfProducts>
        <listOfModifiers>
            <modifierSpeciesReference species="CaPump_PM"/>
        </listOfModifiers>
        <kineticLaw>
            <notes>
                <p xmlns="http://www.w3.org/1999/xhtml">
                ((Vmax * kP * ((Ca_C - Ca_Rest) / (Ca_C + kP)) /
                    (Ca_Rest + kP)) * CaPump_PM)
                </p>
            </notes>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
                <times/>
                <ci> PlasmaMembrane</ci>
                <apply>
```

```
                                <divide/>
                                <apply>
                                    <times/>
                                    <ci> Vmax </ci>
                                    <ci> kP </ci>
                                    <ci> CaPump_PM </ci>
                                    <apply>
                                        <minus/>
                                        <ci> Ca_C </ci>
                                        <ci> Ca_Rest </ci>
                                    </apply>
                                </apply>
                                <apply>
                                    <times/>
                                    <apply>
                                        <plus/>
                                        <ci> Ca_C </ci>
                                        <ci> kP </ci>
                                    </apply>
                                    <apply>
                                        <plus/>
                                        <ci> Ca_Rest </ci>
                                        <ci> kP </ci>
                                    </apply>
                                </apply>
                            </apply>
                        </apply>
                        </math>
                        <listOfParameters>
                            <parameter id="Vmax" value="4000" units="per_second"/>
                            <parameter id="kP" value="0.25" units="subs_per_vol"/>
                            <parameter id="Ca_Rest" value="0.1" units="subs_per_vol"/>
                        </listOfParameters>
                    </kineticLaw>
                </reaction>
                <reaction id="Ca_channel">
                    <listOfReactants>
                        <speciesReference species="Ca_EC"/>
                    </listOfReactants>
                    <listOfProducts>
                        <speciesReference species="Ca_C"/>
                    </listOfProducts>
                    <listOfModifiers>
                        <modifierSpeciesReference species="CaCh_PM"/>
                    </listOfModifiers>
                    <kineticLaw>
                        <notes>
                            <p xmlns="http://www.w3.org/1999/xhtml">
                                (J0 * Kc * (Ca_EC - Ca_C) / (Kc + Ca_C) * CaCh_PM)
                            </p>
                        </notes>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <times/>
                            <ci> PlasmaMembrane </ci>
                            <apply>
                                <divide/>
                                <apply>
                                    <times/>
                                    <ci> CaCh_PM </ci>
                                    <ci> J0 </ci>
                                    <ci> Kc </ci>
                                    <apply>
                                        <minus/>
                                        <ci> Ca_EC </ci>
                                        <ci> Ca_C </ci>
                                    </apply>
                                </apply>
                                <apply>
```

```
                                        <plus/>
                                        <ci> Kc </ci>
                                        <ci> Ca_C </ci>
                                    </apply>
                                </apply>
                            </apply>
                            </math>
                            <listOfParameters>
                                <parameter id="J0" value="0.014" units="litre_per_mole_per_second"/>
                                <parameter id="Kc" value="0.5" units="subs_per_vol"/>
                            </listOfParameters>
                        </kineticLaw>
                    </reaction>
                </listOfReactions>
            </model>
        </sbml>
```

8 Discussion

The volume of data now emerging from molecular biotechnology leave little doubt that extensive computer-based modeling, simulation and analysis will be critical to understanding and interpreting the data (Abbott, 1999; Gilman, 2000; Popel and Winslow, 1998; Smaglik, 2000). This has lead to an explosion in the development of computer tools by many research groups across the world. The explosive rate of progress is exciting, but the rapid growth of the field is accompanied by problems and pressing needs.

One problem is that simulation models and results often cannot be directly compared, shared or re-used, because the tools developed by different groups often are not compatible with each other. As the field of systems biology matures, researchers increasingly need to communicate their results as computational models rather than box-and-arrow diagrams. They also need to reuse published and curated models as library components in order to succeed with large-scale efforts (e.g., the Alliance for Cellular Signaling; Gilman, 2000; Smaglik, 2000). These needs require that models implemented in one software package be portable to other software packages, to maximize public understanding and to allow building up libraries of curated computational models.

We offer SBML to the systems biology community as a suggested format for exchanging models between simulation/analysis tools. SBML is an open model representation language oriented specifically towards representing systems of biochemical reactions.

Our vision for SBML is to create an open standard that will enable different software tools to exchange computational models. SBML is not static; we continue to develop and experiment with it, and we interact with other groups who seek to develop similar markup languages. We plan on continuing to evolve SBML with the help of the systems biology community to make SBML increasingly more powerful, flexible and useful.

8.1 Future enhancements: SBML Level 3 and beyond

Many people have expressed a desire to see additional capabilities added to SBML. The following summarizes additional features that are under consideration to be included in SBML Level 3; additional information is available in the wiki on the SBML project website (`http://sbml.org`).

- *Arrays.* This will enable the creation of arrays of components (species, reactions, compartments and submodels).
- *Connections.* This will be a mechanism for describing the connections between items in an array.
- *Geometry.* This will enable the encoding of the spatial characteristics of models including the geometry of compartments, the diffusion properties of species and the specification of different species concentrations across different regions of a cell.
- *Model Composition.* This will enable a large model to be built up out of instances of other models. It will also allow the reuse of model components and the creation of several instances of the same model.
- *Multistate and Complex Species.* This will allow the straight-forward construction of models involving species with a large number of states or species composed of subcomponents. The representation scheme would be designed to contain the combinatorial explosion of objects that often results from these types of models.
- *Diagrams.* This feature will allow components to be annotated with data to enable the display of the model in a diagram.
- *Dynamic Structure.* This will enable model structure to vary during simulation. One aspect of this allowing rules and reactions to have their effect conditional on the state of the model system. For example in SBML Level 2 it is possible to create a rule with the effect:

$$\frac{ds}{dt} = \begin{cases} 0 & \text{if } s > 0 \\ y & \text{otherwise} \end{cases}$$

Dynamic restructuring would enable the expression of the following example:

$$\text{if } s > 0 \quad \frac{ds}{dt} = y$$

where s is not determined by the rule when $s \leq 0$.

- *Tie-breaking algorithm.* This will include a controlled vocabulary and associated attributes on models to indicate the simultaneous event tie-breaking algorithm required to correctly simulate the model.
- *Distributions.* This will provide a means of specifying random variables and statistical distribution of values.

Acknowledgments

The development of SBML was originally funded entirely by the Japan Science and Technology Agency (JST) under the ERATO Kitano Symbiotic Systems Project during the years 2000–2003. From 2003 to today, general support for development of SBML and associated software such as libSBML and the SBML Test Suite has been provided by the National Institute of General Medical Sciences (USA) via grant numbers GM070923 and GM077671.

We gratefully acknowledge additional sponsorship from the following funding agencies: the National Human Genome Research Institute (USA); the International Joint Research Program of NEDO (Japan); the JST ERATO-SORST Program (Japan); the Japanese Ministry of Agriculture; the Japanese Ministry of Education, Culture, Sports, Science and Technology; the BBSRC e-Science Initiative (UK); the DARPA IPTO Bio-Computation Program (USA); and the Air Force Office of Scientific Research (USA).

Additional support has been or continues to be provided by the following institutions: the California Institute of Technology (USA), EML Research gGmbH (Germany), the European Molecular Biology Laboratory's European Bioinformatics Institute (UK), the Molecular Sciences Institute (USA), the University of Heidelberg (Germany), the University of Hertfordshire (UK), the University of Newcastle (UK), the Systems Biology Institute (Japan), and the Virginia Bioinformatics Institute (USA).

SBML was first conceived at the JST/ERATO-sponsored *First Workshop on Software Platforms for Systems Biology*, held in April, 2000, at the California Institute of Technology in Pasadena, California, USA. The participants collectively decided to begin developing a common XML-based declarative language for representing models. A draft version of the Systems Biology Markup Language was developed by the Caltech ERATO team and delivered to all collaborators in August, 2000. This draft version underwent extensive discussion over mailing lists and then again during the *Second Workshop on Software Platforms for Systems Biology* held in Tokyo, Japan, November 2000. A revised version of SBML was issued by the Caltech ERATO team in December, 2000, and after further discussions over mailing lists and in meetings, we produced a specification for SBML Level 1 (Hucka et al., 2001).

SBML Level 2 was conceived at the *5th Workshop on Software Platforms for Systems Biology*, held in July 2002, at the University of Hertfordshire, UK. The participants collectively decided to revise the form of SBML in SBML Level 2. The first draft of the Level 2 Version 1 document was released in August 2002. The final set of features in SBML Level 2 Version 1 was finalized in May 2003 at the *7th Workshop on Software Platforms for Systems Biology* in Ft. Lauderdale, Florida.

SBML Level 2 Version 2 was largely finalized after the 2005 SBML Forum meeting in Boston and a final document was issued in September 2006. SBML Level 2 Version 3 was finalized after the 2006 SBML Forum meeting in Yokohama, Japan, and the 2007 SBML Hackathon in Newcastle, UK. SBML Level 2 Version 4 was finalized after the 2008 SBML Forum in Göteborg, Sweden. They were developed with contributions from so many people constituting the worldwide *SBML Forum* that we regret it has become infeasible to list individuals by name. For discussions and help developing SBML, and for feedback about this specification, we are grateful to everyone on the sbml-discuss@caltech.edu and sbml-interoperability@caltech.edu mailing lists, and many other groups and developers at large, notably the creators of CellML (Hedley et al., 2001), the members of the DARPA Bio-SPICE project, and the authors of all of the software systems that support SBML.

A guide to software known to support SBML is provided on the SBML.org website at the following URL: http://sbml.org/SBML_Software_Guide.

A XML Schema for SBML

The following is an XML Schema definition for SBML Level 2 Version 5, using the W3C Recommendation for XML Schema version 1.0 of 2 May 2001 (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000). This Schema does not define all aspects of SBML Level 2: an SBML document validated by this schema is not necessarily a valid SBML Level 2 document. Appendix B contains a schema for the SBML MathML subset. Appendix C contains a list of the remaining checks required to validate a model in addition to making it consistent with these two schemas.

Note to implementors: the following schema is self-contained and makes reference to the official XML Schema for MathML hosted at the W3. However, for use in software systems, it is more efficient to store the MathML subset Schema of Appendix B in a file on a user's local disk, and change the `schemaLocation` value (text line 25 below) to refer to this local copy of the MathML subset Schema. Doing so will avoid requiring a network access every time this SBML Schema is used.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.sbml.org/sbml/level2/version5"
            xmlns="http://www.sbml.org/sbml/level2/version5"
            xmlns:mml="http://www.w3.org/1998/Math/MathML"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified"
            attributeFormDefault="unqualified"
            version="$ $Id: apdx-schema.tex 20433 2014-06-05 09:53:09Z sarahkeating $ $">
    <xsd:import namespace="http://www.w3.org/1998/Math/MathML"
                schemaLocation="http://www.w3.org/Math/XMLSchema/mathml2/mathml2.xsd"/>
    <xsd:annotation>
        <xsd:documentation>
 * Filename    : sbml.xsd
 * Description: XML Schema for SBML Level 2 Version 4.
 * Author(s)  : Michael Hucka
 * Revision    : $Id: apdx-schema.tex 20433 2014-06-05 09:53:09Z sarahkeating $
 * $HeadURL: https://svn.code.sourceforge.net/p/sbml/code/trunk/specifications/sbml-level-2/version-5/spec/apdx-schema.tex $
 *
 * Copyright 2007 California Institute of Technology.
 * Copyright 2003-2006 California Institute of Technology, the Japan Science
 * and Technology Corporation, and the University of Hertfordshire.
 *
 * This software is licensed according to the terms described in the file
 * named "LICENSE.txt" included with this distribution and available
 * online at http://sbml.org/xml-schemas/LICENSE.txt
        </xsd:documentation>
    </xsd:annotation>
    <!--The definition of new primitive types follows.-->
    <xsd:simpleType name="SId">
        <xsd:annotation>
            <xsd:documentation>The type SId is used throughout SBML as the
            type of the 'id' attributes on model elements.</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="(_|[a-z]|[A-Z])(_|[a-z]|[A-Z]|[0-9])*"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="UnitSId">
        <xsd:annotation>
            <xsd:documentation>The type UnitSId is used to refer to units.</xsd:documentation>
        </xsd:annotation>
        <xsd:union>
            <xsd:simpleType>
                <xsd:restriction base="SId" />
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:restriction base="SId">
                    <xsd:enumeration value="ampere"/>
                    <xsd:enumeration value="becquerel"/>
                    <xsd:enumeration value="candela"/>
                    <xsd:enumeration value="coulomb"/>
                    <xsd:enumeration value="dimensionless"/>
                    <xsd:enumeration value="farad"/>
                    <xsd:enumeration value="gram"/>
                    <xsd:enumeration value="gray"/>
                    <xsd:enumeration value="henry"/>
                    <xsd:enumeration value="hertz"/>
```

```
                    <xsd:enumeration value="item"/>
                    <xsd:enumeration value="joule"/>
                    <xsd:enumeration value="katal"/>
                    <xsd:enumeration value="kelvin"/>
                    <xsd:enumeration value="kilogram"/>
                    <xsd:enumeration value="litre"/>
                    <xsd:enumeration value="lumen"/>
                    <xsd:enumeration value="lux"/>
                    <xsd:enumeration value="metre"/>
                    <xsd:enumeration value="mole"/>
                    <xsd:enumeration value="newton"/>
                    <xsd:enumeration value="ohm"/>
                    <xsd:enumeration value="pascal"/>
                    <xsd:enumeration value="radian"/>
                    <xsd:enumeration value="second"/>
                    <xsd:enumeration value="siemens"/>
                    <xsd:enumeration value="sievert"/>
                    <xsd:enumeration value="steradian"/>
                    <xsd:enumeration value="tesla"/>
                    <xsd:enumeration value="volt"/>
                    <xsd:enumeration value="watt"/>
                    <xsd:enumeration value="weber"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:restriction base="SId">
                    <xsd:enumeration value="substance"/>
                    <xsd:enumeration value="volume"/>
                    <xsd:enumeration value="area"/>
                    <xsd:enumeration value="length"/>
                    <xsd:enumeration value="time"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>
    <xsd:simpleType name="SBOTerm">
        <xsd:annotation>
            <xsd:documentation>The data type for sboTerm attribute values.</xsd:documentation>
        </xsd:annotation>
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="(SBO:)([0-9]7)"/>
        </xsd:restriction>
    </xsd:simpleType>
    <!--The definition of SBase follows.-->
    <xsd:complexType name="SBase" abstract="true">
        <xsd:annotation>
            <xsd:documentation>The SBase type is the base type of all main
            components in SBML.  It supports attaching metadata, notes and
            annotations to components.</xsd:documentation>
        </xsd:annotation>
        <xsd:sequence>
            <xsd:element name="notes" minOccurs="0">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:any namespace="http://www.w3.org/1999/xhtml"
                                 processContents="skip"
                                 minOccurs="0" maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="annotation" minOccurs="0">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:any processContents="skip"
                                 minOccurs="0" maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="metaid" type="xsd:ID" use="optional"/>
        <xsd:attribute name="sboTerm" type="SBOTerm" use="optional"/>
    </xsd:complexType>
    <!--The definition of main SBML classes follows.-->
    <xsd:complexType name="FunctionDefinition">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element ref="mml:math"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="SId" use="required"/>
```

```
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Unit">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:attribute name="kind" type="UnitSId" use="required"/>
                <xsd:attribute name="exponent" type="xsd:int" default="1"/>
                <xsd:attribute name="scale" type="xsd:int" default="0"/>
                <xsd:attribute name="multiplier" type="xsd:double" default="1"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ListOfUnits">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element name="unit" type="Unit" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="UnitDefinition">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element name="listOfUnits" type="ListOfUnits"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="UnitSId" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="CompartmentType">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:attribute name="id" type="SId" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="SpeciesType">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:attribute name="id" type="SId" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Compartment">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:attribute name="id" type="SId" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
                <xsd:attribute name="compartmentType" type="SId" use="optional"/>
                <xsd:attribute name="spatialDimensions" use="optional" default="3">
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:int">
                            <xsd:minInclusive value="0"/>
                            <xsd:maxInclusive value="3"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                </xsd:attribute>
                <xsd:attribute name="size" type="xsd:double" use="optional"/>
                <xsd:attribute name="units" type="UnitSId" use="optional"/>
                <xsd:attribute name="outside" type="SId" use="optional"/>
                <xsd:attribute name="constant" type="xsd:boolean"
                               use="optional" default="true"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Species">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:attribute name="id" type="SId" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
                <xsd:attribute name="speciesType" type="SId"  use="optional"/>
                <xsd:attribute name="compartment" type="SId" use="required"/>
                <xsd:attribute name="initialAmount" type="xsd:double" use="optional"/>
```

```
                <xsd:attribute name="initialConcentration" type="xsd:double" use="optional"/>
                <xsd:attribute name="substanceUnits" type="UnitSId" use="optional"/>
                <xsd:attribute name="hasOnlySubstanceUnits" type="xsd:boolean"
                               use="optional" default="false"/>
                <xsd:attribute name="boundaryCondition" type="xsd:boolean"
                               use="optional" default="false"/>
                <xsd:attribute name="charge" type="xsd:int" use="optional"/>
                <xsd:attribute name="constant" type="xsd:boolean"
                               use="optional" default="false"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Parameter">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:attribute name="id" type="SId" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
                <xsd:attribute name="value" type="xsd:double" use="optional"/>
                <xsd:attribute name="units" type="UnitSId" use="optional"/>
                <xsd:attribute name="constant" type="xsd:boolean"
                               use="optional" default="true"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ListOfParameters">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element name="parameter" type="Parameter" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="InitialAssignment">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element ref="mml:math"/>
                </xsd:sequence>
                <xsd:attribute name="symbol" type="SId" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Rule" abstract="true">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element ref="mml:math"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="AlgebraicRule">
        <xsd:complexContent>
            <xsd:extension base="Rule"/>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="AssignmentRule">
        <xsd:complexContent>
            <xsd:extension base="Rule">
                <xsd:attribute name="variable" type="SId" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="RateRule">
        <xsd:complexContent>
            <xsd:extension base="Rule">
                <xsd:attribute name="variable" type="SId" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Constraint">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element ref="mml:math"/>
                    <xsd:element name="message" minOccurs="0">
                        <xsd:complexType>
                            <xsd:sequence>
                                <xsd:any namespace="http://www.w3.org/1999/xhtml"
```

```
                                        processContents="skip"
                                        minOccurs="0" maxOccurs="unbounded"/>
                                </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="KineticLaw">
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element ref="mml:math"/>
                        <xsd:element name="listOfParameters" type="ListOfParameters" minOccurs="0"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="SimpleSpeciesReference" abstract="true">
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:attribute name="id" type="SId" use="optional"/>
                    <xsd:attribute name="name" type="xsd:string" use="optional"/>
                    <xsd:attribute name="species" type="SId" use="required"/>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="ModifierSpeciesReference">
            <xsd:complexContent>
                <xsd:extension base="SimpleSpeciesReference"/>
            </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="ListOfModifierSpeciesReferences">
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element name="modifierSpeciesReference"
                                     type="ModifierSpeciesReference"
                                     maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="StoichiometryMath">
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element ref="mml:math"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="SpeciesReference">
            <xsd:complexContent>
                <xsd:extension base="SimpleSpeciesReference">
                    <xsd:sequence>
                        <xsd:element name="stoichiometryMath"
                                     type="StoichiometryMath"
                                     minOccurs="0"/>
                    </xsd:sequence>
                    <xsd:attribute name="stoichiometry" type="xsd:double"
                                   use="optional" default="1"/>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="ListOfSpeciesReferences">
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element name="speciesReference"
                                     type="SpeciesReference"
                                     maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="Reaction">
            <xsd:complexContent>
                <xsd:extension base="SBase">
```

```
                <xsd:sequence>
                    <xsd:element name="listOfReactants" type="ListOfSpeciesReferences"
                                 minOccurs="0"/>
                    <xsd:element name="listOfProducts" type="ListOfSpeciesReferences"
                                 minOccurs="0"/>
                    <xsd:element name="listOfModifiers" type="ListOfModifierSpeciesReferences"
                                 minOccurs="0"/>
                    <xsd:element name="kineticLaw" type="KineticLaw"
                                 minOccurs="0"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="SId" use="required"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
                <xsd:attribute name="reversible" type="xsd:boolean"
                               use="optional" default="true"/>
                <xsd:attribute name="fast" type="xsd:boolean"
                               use="optional" default="false"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="EventAssignment">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element ref="mml:math"/>
                </xsd:sequence>
                <xsd:attribute name="variable" type="SId" use="required"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="ListOfEventAssignments">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element name="eventAssignment"
                                 type="EventAssignment"
                                 maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Trigger">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element ref="mml:math"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Delay">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element ref="mml:math"/>
                </xsd:sequence>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Event">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element name="trigger" type="Trigger"/>
                    <xsd:element name="delay" type="Delay" minOccurs="0"/>
                    <xsd:element name="listOfEventAssignments" type="ListOfEventAssignments"/>
                </xsd:sequence>
                <xsd:attribute name="id" type="SId" use="optional"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
                <xsd:attribute name="useValuesFromTriggerTime" type="xsd:boolean"
                               use="optional" default="true"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="Model">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element name="listOfFunctionDefinitions" minOccurs="0">
                        <xsd:complexType>
                            <xsd:complexContent>
```

```
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element name="functionDefinition"
                                     type="FunctionDefinition"
                                     maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="listOfUnitDefinitions" minOccurs="0">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element name="unitDefinition"
                                     type="UnitDefinition"
                                     maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="listOfCompartmentTypes" minOccurs="0">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element name="compartmentType"
                                     type="CompartmentType"
                                     maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="listOfSpeciesTypes" minOccurs="0">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element name="speciesType"
                                     type="SpeciesType"
                                     maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="listOfCompartments" minOccurs="0">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element name="compartment"
                                     type="Compartment"
                                     maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="listOfSpecies" minOccurs="0">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="SBase">
                    <xsd:sequence>
                        <xsd:element name="species"
                                     type="Species"
                                     maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="listOfParameters" type="ListOfParameters" minOccurs="0"/>
    <xsd:element name="listOfInitialAssignments" minOccurs="0">
        <xsd:complexType>
            <xsd:complexContent>
                <xsd:extension base="SBase">
```

```
                                    <xsd:sequence>
                                        <xsd:element name="initialAssignment"
                                                     type="InitialAssignment"
                                                     maxOccurs="unbounded"/>
                                    </xsd:sequence>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="listOfRules" minOccurs="0">
                        <xsd:complexType>
                            <xsd:complexContent>
                                <xsd:extension base="SBase">
                                    <xsd:choice maxOccurs="unbounded">
                                        <xsd:element name="algebraicRule"
                                                     type="AlgebraicRule"
                                                     minOccurs="0"/>
                                        <xsd:element name="assignmentRule"
                                                     type="AssignmentRule"
                                                     minOccurs="0"/>
                                        <xsd:element name="rateRule"
                                                     type="RateRule"
                                                     minOccurs="0"/>
                                    </xsd:choice>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="listOfConstraints" minOccurs="0">
                        <xsd:complexType>
                            <xsd:complexContent>
                                <xsd:extension base="SBase">
                                    <xsd:sequence>
                                        <xsd:element name="constraint"
                                                     type="Constraint"
                                                     maxOccurs="unbounded"/>
                                    </xsd:sequence>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="listOfReactions" minOccurs="0">
                        <xsd:complexType>
                            <xsd:complexContent>
                                <xsd:extension base="SBase">
                                    <xsd:sequence>
                                        <xsd:element name="reaction"
                                                     type="Reaction"
                                                     maxOccurs="unbounded"/>
                                    </xsd:sequence>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="listOfEvents" minOccurs="0">
                        <xsd:complexType>
                            <xsd:complexContent>
                                <xsd:extension base="SBase">
                                    <xsd:sequence>
                                        <xsd:element name="event"
                                                     type="Event"
                                                     maxOccurs="unbounded"/>
                                    </xsd:sequence>
                                </xsd:extension>
                            </xsd:complexContent>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="id" type="SId" use="optional"/>
                <xsd:attribute name="name" type="xsd:string" use="optional"/>
            </xsd:extension>
        </xsd:complexContent>
    </xsd:complexType>
    <!-- The following is the type definition for the top-level element in an SBML document.-->
    <xsd:complexType name="Sbml">
        <xsd:complexContent>
            <xsd:extension base="SBase">
                <xsd:sequence>
                    <xsd:element name="model" type="Model"/>
                </xsd:sequence>
```

```
                    <xsd:attribute name="level" type="xsd:positiveInteger"
                                   use="required" fixed="2"/>
                    <xsd:attribute name="version" type="xsd:positiveInteger"
                                   use="required" fixed="5"/>
                </xsd:extension>
            </xsd:complexContent>
        </xsd:complexType>
        <!--The following is the (only) top-level element allowed in an SBML document.-->
        <xsd:element name="sbml" type="Sbml"/>
        <!--The end.-->
    </xsd:schema>

```

B XML Schema for MathML subset

The following XML schema defines the syntax of the MathML syntax that is used in SBML Level 2.

(Forthcoming.)

C Validation rules for SBML

This section contains a summary of all the conditions that should be true of a model, in addition to consistency with the XML Schemas given in Appendixes A and B, for that model to considered valid SBML.

General XML validation

10101. An SBML XML file must use UTF-8 as the character encoding. More precisely, the `encoding` attribute of the XML declaration at the beginning of the XML data stream cannot have a value other than "`UTF-8`". An example valid declaration is `<?xml version="1.0" encoding="UTF-8"?>`. (References: L2V2 Section 4.1; L2V3 Section 4.1; L2V4 Section 4.1; L2V5 Section 4.1.)

10102. An SBML XML document must not contain undefined elements or attributes in the SBML namespace. Documents containing unknown elements or attributes placed in the SBML namespace do not conform to the SBML Level 2 specification. (References: L2V2 Section 4.1; L2V3 Section 4.1; L2V4 Section 4.1; L2V5 Section 4.1.)

10103. An SBML XML document must conform to the XML Schema for the corresponding SBML Level, Version and Release. The XML Schema for SBML defines the basic SBML object structure, the data types used by those objects, and the order in which the objects may appear in an SBML document. (References: SBML L2V2 Section 4.1; L2V3 Section 4.1; L2V4 Section 4.1; L2V5 Section 4.1.)

General MathML validation

10201. All MathML content in SBML must appear within a `math` element, and the `math` element must be either explicitly or implicitly in the XML namespace "`http://www.w3.org/1998/Math/MathML`". (References: L2V2 Section 3.5; L2V3 Section 3.4; L2V4 Section 3.4; L2V5 Section 3.4.)

10202. The only permitted MathML 2.0 elements in SBML Level 2 are the following: `cn`, `ci`, `csymbol`, `sep`, `apply`, `piecewise`, `piece`, `otherwise`, `eq`, `neq`, `gt`, `lt`, `geq`, `leq`, `plus`, `minus`, `times`, `divide`, `power`, `root`, `abs`, `exp`, `ln`, `log`, `floor`, `ceiling`, `factorial`, `and`, `or`, `xor`, `not`, `degree`, `bvar`, `logbase`, `sin`, `cos`, `tan`, `sec`, `csc`, `cot`, `sinh`, `cosh`, `tanh`, `sech`, `csch`, `coth`, `arcsin`, `arccos`, `arctan`, `arcsec`, `arccsc`, `arccot`, `arcsinh`, `arccosh`, `arctanh`, `arcsech`, `arccsch`, `arccoth`, `true`, `false`, `notanumber`, `pi`, `infinity`, `exponentiale`, `semantics`, `annotation`, and `annotation-xml`. (References: L2V2 Section 3.5.1; L2V3 Section 3.4.1; L2V4 Section 3.4.1; L2V5 Section 3.4.1.)

10203. In the SBML subset of MathML 2.0, the MathML attribute `encoding` is only permitted on `csymbol`, `annotation` and `annotation-xml`. No other MathML elements may have the `encoding` attribute. (References: L2V2 Section 3.5.1; L2V3 Section 3.4.1; L2V4 Section 3.4.1; L2V5 Section 3.4.1.)

10204. In the SBML subset of MathML 2.0, the MathML attribute `definitionURL` is only permitted on `ci`, `csymbol`, and `semantics`. No other MathML elements may have a `definitionURL` attribute. (References: L2V2 Section 3.5.1; L2V3 Section 3.4.1; L2V4 Section 3.4.1; L2V5 Section 3.4.1.)

10205. In SBML Level 2, the only values permitted for the `definitionURL` attribute on a `csymbol` element are "`http://www.sbml.org/sbml/symbols/time`" and "`http://www.sbml.org/sbml/symbols/delay`". (References: L2V2 Section 3.5.5; L2V3 Section 3.4.6; L2V4 Section 3.4.6; L2V5 Section 3.4.6.)

10206. In the SBML subset of MathML 2.0, the MathML attribute `type` is only permitted on the `cn` construct. No other MathML elements may have a `type` attribute. (References: L2V2 Section 3.5.1; L2V3 Section 3.4.1; L2V4 Section 3.4.1; L2V5 Section 3.4.1.)

10207. The only permitted values for the `type` attribute on MathML `cn` elements are "`e-notation`", "`real`", "`integer`", and "`rational`". (References: L2V2 Section 3.5.2; L2V3 Section 3.4.2; L2V4 Section 3.4.2; L2V5 Section 3.4.2.)

10208. MathML `lambda` elements are only permitted as the first element inside the `math` element of a **FunctionDefinition** or as the first element of a `semantics` element immediately inside the `math` element of

a **FunctionDefinition**; they may not be used elsewhere in an SBML model. (References: L2V2 Sections 3.5.1 and 4.3; L2V3 Sections 3.4.1 and 4.3.2; L2V4 Section 3.4.1 and 4.3.2; L2V5 Sections 3.4.1 and 4.3.2.)

10209. The arguments of the MathML logical operators `and`, `or`, `xor`, and `not` must have boolean values. (References: L2V2 Section 3.5.8; L2v3 Section 3.4.9; L2V4 Section 3.4.9; L2V5 Section 3.4.10.)

10210. The arguments to the following MathML constructs must evaluate to be numbers (i.e., MathML real, integer, rational, or "e-notation" numbers, or the time or delay `csymbol`): `plus`, `minus`, `times`, `divide`, `power`, `root`, `abs`, `exp`, `ln`, `log`, `floor`, `ceiling`, `factorial`, `sin`, `cos`, `tan`, `sec`, `csc`, `cot`, `sinh`, `cosh`, `tanh`, `sech`, `csch`, `coth`, `arcsin`, `arccos`, `arctan`, `arcsec`, `arccsc`, `arccot`, `arcsinh`, `arccosh`, `arctanh`, `arcsech`, `arccsch`, `arccoth`. (References: L2V2 Section 3.5.8; L2V3 Section 3.4.9; L2V4 Section 3.5.8; L2V5 Section 3.4.10.)

10211. The values of all arguments to `eq` and `neq` operators must have the same type (either all boolean or all numerical). (References: L2V2 Section 3.5.8; L2V3 section 3.4.9; L2V4 Section 3.4.9; L2V5 Section 3.4.10.)

10212. The types of values within `piecewise` operators must all be consistent: the set of expressions that make up the first arguments of the `piece` and `otherwise` operators within the same `piecewise` operator must all return values of the same type. (References: L2V2 Section 3.5.8; L2V3 Section 3.4.9; L2V4 Section 3.4.9; L2V5 Section 3.4.10.)

10213. The second argument of a MathML `piece` operator must have a boolean value. (References: L2V2 Section 3.5.8; L2V3 Section 3.4.9; L2V4 Section 3.5.8; L2V5 Section 3.4.10.)

10214. Outside of a **FunctionDefinition**, if a `ci` element is the first element within a MathML `apply`, then the `ci`'s value can only be chosen from the set of identifiers of **FunctionDefinition** instances defined in the enclosing SBML **Model** instance. (References: L2V2 Section 4.3.2; L2V3 Section 4.3.2; L2V4 Section 4.3.2; L2V5 Section 4.3.2.)

10215. Outside of a **FunctionDefinition**, if a `ci` element is not the first element within a MathML `apply`, then the `ci`'s value can only be chosen from the set of identifiers of **Species**, **Compartment**, **Parameter** or **Reaction** objects defined in the SBML model. (References: L2V2 Section 4.3.2; L2V3 Section 3.4.3; L2V4 Section 3.4.3; L2V5 Section 3.4.3.)

10216. The `id` value of a **Parameter** defined within a **KineticLaw** can only be used in `ci` elements within the MathML content of that same **KineticLaw**; the identifier is not visible to other parts of the model. (References: L2V2 Sections 3.4.1, 3.5.3 and 4.13.5; L2V3 Sections 3.3.1, 3.4.3 and 4.13.5; L2V4 Sections 3.3.1, 3.4.3, and 4.13.5; L2V5 Sections 3.3.1, 3.4.3 and 4.13.5.)

10217. The MathML formulas in the following elements must yield numerical values (that is, MathML real, integer or "e-notation" numbers, or the time or delay `csymbol`): `math` in **KineticLaw**, `stoichiometryMath` in **SpeciesReference**, `math` in **InitialAssignment**, `math` in **AssignmentRule**, `math` in **RateRule**, `math` in **AlgebraicRule**, `math` in **Event Delay**, and `math` in **EventAssignment**. (References: L2V2 Sections 4.10, 4.11, 4.13 and 4.14; L3V3 Sections 4.10, 4.11, 4.13 and 4.14; L2V4 Sections 4.10, 4.11, 4.13, and 4.14; L2V5 Sections 4.10, 4.11, 4.13 and 4.14.)

10218. A MathML operator must be supplied the number of arguments appropriate for that operator. (References: L2V2 Section 3.5.1; L2V3 Section 3.4.1; L2V4 Section 3.4.1; L2V5 Section 3.4.1.)

10219. The number of arguments used in a call to a function defined by a **FunctionDefinition** must equal the number of arguments accepted by that function, or in other words, the number of `bvar` elements inside the `lambda` element of the function definition. (References: L2V4 Section 4.3.4; L2V5 Section 4.3.4.)

10222. A `ci` element may not be the identifier of a **Compartment** with a `spatialDimensions` value of "`0`". (References: L2V5 Section 4.7.)

General identifier validation

10301. The value of the `id` attribute on every instance of the following classes of objects in a model must be unique across the set of all `id` values in a model: **Model**, **FunctionDefinition**, **CompartmentType**, **SpeciesType**, **Compartment**, **Species**, **Reaction**, **SpeciesReference**, **ModifierSpeciesReference**, **Event**, and model-wide **Parameter**s. Exception: the identifiers of instances of **UnitDefinition**, and parameters defined inside a reaction, are treated separately. (References: L2V1 Section 3.5; L2V2 Section 3.4.1; L2V3 Section 3.3; L2V4 Section 3.3; L2V5 Section 3.3.)

10302. The value of the `id` attribute of every **UnitDefinition** must be unique across the set of all **UnitDefinition**s in the entire model. (References: L2V1 Section 3.5 and 4.4; L2V2 Sections 3.4.1 and 4.4; L2V3 Sections 3.3 and 4.4; L2V4 Section 3.3 and 4.4; L2V5 Sections 3.3 and 4.4.)

10303. The value of the `id` attribute of each parameter defined locally within a **KineticLaw** must be unique across the set of all such parameter definitions in that **KineticLaw**. (References: L2V1 Sections 3.4.1 and 4.13.9; L2V2 Sections 3.4.1 and 4.13.5; L2V3 Sections 3.3.1 and 4.13.5; L2V4 Section 3.3.1 and 4.13.5; L2V5 Sections 3.3.1 and 4.13.5.)

10304. The value of the `variable` attribute in all **AssignmentRule** and **RateRule** definitions must be unique across the set of all such rule definitions in a model. (References: L2V1 Section 4.8.4; L2V2 Section 4.11; L2V3 Section 4.11.3; L2V4 Section 4.11.3; L2V5 Section 4.11.3.)

10305. In each **Event**, the value of the `variable` attribute within every **EventAssignment** definition must be unique across the set of all **EventAssignment**s within that **Event**. In other words, a single **Event** cannot make more than one assignment to a given identifier. (References: L2V1 erratum 17; L2V2 Section 4.14.2; L2v3 Section 4.14.4; L2V4 Section 4.14.4; L2V5 Section 4.14.4.)

10306. An identifier used as the value of `variable` in an **EventAssignment** cannot also appear as the value of `variable` in an **AssignmentRule**. (References: L2V1 Section 4.10.5; L2V2 Section 4.14.2; L2V2 Section 4.14.4; L2V4 Section 4.14.4; L2V5 Section 4.14.4.)

10307. Every `metaid` attribute value must be unique across the set of all `metaid` values in a model. (References: L2V2 Sections 3.1.6 and 3.3.1; L2V3 Sections 3.1.6 and 3.2.1; L2V4 Section 3.2.1 and 3.1.6; L2V5 Sections 3.2.1 and 3.1.6.)

10308. The value of a `sboTerm` attribute must have the data type `SBOTerm`, which is a string consisting of the characters 'S', 'B', 'O', ':', followed by exactly seven digits. (References: L2V2 Section 3.1.9; L2V3 Section 3.1.9; L2V4 Section 3.1.9; L2V5 Section 3.1.9.)

10309. The syntax of `metaid` attribute values must conform to the syntax of the XML type `ID`. (References: L2V2 Sections 3.1.6 and 3.3.1; L2V3 Sections 3.2.1 and 3.1.6; L2V4 Section 3.2.1 and 3.1.6; L2V5 Sections 3.2.1 and 3.1.6.)

10310. The syntax of `id` attribute values must conform to the syntax of the SBML type `SId`. (References: L2V2 Section 3.1.7; L2V3 Section 3.1.7; L2V4 Section 3.1.7; L2V5 Section 3.1.7.)

10311. The syntax of unit identifiers (i.e., the values of the `id` attribute on **UnitDefinition**, the `units` attribute on **Compartment**, the `units` attribute on **Parameter**, and the `substanceUnits` attribute on **Species**) must conform to the syntax of the SBML type `UnitSId`. (References: L2V3 Section 3.1.8; L2V4 Section 3.1.8; L2V5 Section 3.1.8.)

General `annotation` validation

10401. Every top-level element within an `annotation` element must have a namespace declared. (References: L2V2 Section 3.3.3; L2V3 Section 3.2.4; L2V4 Section 3.2.4; L2V5 Section 3.2.4.)

10402. There cannot be more than one top-level element using a given namespace inside a given `annotation` element. (References: L2V2 Section 3.3.3; L2V3 Section 3.2.4; L2V4 Section 3.2.4; L2V5 Section 3.2.4.)

10403. Top-level elements within an `annotation` element cannot use any SBML namespace, whether explicitly (by declaring the namespace to be one of the URIs "`http://www.sbml.org/sbml/level1`", "`http://www.sbml.org/sbml/level2`", "`http://www.sbml.org/sbml/level2/version2`", or "`http://www.sbml.org/sbml/level2/version3`", or "`http://www.sbml.org/sbml/level2/version4`"), or implicitly (by failing to declare any namespace). (References: L2V2 Section 3.3.3; L2V3 Section 3.2.4; L2V4 Section 3.2.4; L2V5 Section 3.2.4.)

General unit validation (Warnings only)

10501. The units of the expressions used as arguments to a function call should match the units expected for the arguments of that function. (References: L2V2 Section 3.5; L2V3 Section 3.4; L2V4 Section 3.4; L2V5 Section 3.4.)

10511. When the `variable` in an **AssignmentRule** refers to a **Compartment**, the units of the rule's right-hand side should be consistent with the units of that compartment's size. (References: L2V2 Section 4.11.3; L2V3 Section 4.11.3; L2V4 Section 4.11.3; L2V5 Section 4.11.3.)

10512. When the `variable` in an **AssignmentRule** refers to a **Species**, the units of the rule's right-hand side should be consistent with the units of the species' quantity. (References: L2V2 Section 4.11.3; L2V3 Section 4.11.3; L2V4 Section 4.11.3; L2V5 Section 4.11.3.)

10513. When the `variable` in an **AssignmentRule** refers to a **Parameter**, the units of the rule's right-hand side should be consistent with the units declared for that parameter. (References: L2V2 Section 4.11.3; L2V3 Section 4.11.3; L2V4 Section 4.11.3; L2V5 Section 4.11.3.)

10521. When the `variable` in an **InitialAssignment** refers to a **Compartment**, the units of the **InitialAssignment**'s `math` expression should be consistent with the units of that compartment's size. (References: L2V2 Section 4.10.4; L2V3 Section 4.10; L2V4 Section 4.10; L2V5 Section 4.10.)

10522. When the `variable` in an **InitialAssignment** refers to a **Species**, the units of the **InitialAssignment**'s `math` expression should be consistent with the units of that species' quantity. (References: L2V2 Section 4.10.4; L2V3 Section 4.11.3; L2V4 Section 4.11.3; L2V5 Section 4.11.3.)

10523. When the `variable` in an **InitialAssignment** refers to a **Parameter**, the units of the **InitialAssignment**'s `math` expression should be consistent with the units declared for that parameter. (References: L2V2 Section 4.10.4; L2V3 Section 4.11.3; L2V4 Section 4.11.3; L2V5 Section 4.11.3.)

10531. When the `variable` in a **RateRule** definition refers to a **Compartment**, the units of the rule's right-hand side should be of the form *x per time*, where *x* is either the `units` in that **Compartment** definition, or (in the absence of explicit units declared for the compartment size) the default units for that compartment, and *time* refers to the units of time for the model. (References: L2V2 Section 4.11.4; L2V3 Section 4.11.4; L2V4 Section 4.11.4; L2V5 Section 4.11.4.)

10532. When the `variable` in a **RateRule** definition refers to a **Species**, the units of the rule's right-hand side should be of the form *x per time*, where *x* is the units of that species' quantity, and *time* refers to the units of time for the model. (References: L2V2 Section 4.11.4; L2V3 Section 4.11.4; L2V4 Section 4.11.4; L2V5 Section 4.11.4.)

10533. When the `variable` in a **RateRule** definition refers to a **Parameter**, the units of the rule's right-hand side should be of the form *x per time*, where *x* is the `units` in that **Parameter** definition, and *time* refers to the units of time for the model. (References: L2V2 Section 4.11.4; L2V3 Section 4.11.4; L2V4 Section 4.11.4; L2V5 Section 4.11.4.)

10541. The units of the `math` formula in a **KineticLaw** definition should be the equivalent of *substance per time*. (References: L2V2 Section 4.13.5; L2V3 Section 4.13.5; L2V4 Section 4.13.5; L2V5 Section 4.13.5.)

10551. The units of the mathematical formula in the `delay` of an **Event** should correspond to the model's overall units of time. (L2V3 Section 4.14.3; L2V4 Section 4.14.3; L2V5 Section 4.14.3.)

10561. When the `variable` attribute of an **EventAssignment** contains the identifier of a **Compartment** in the model, the units of the mathematical expression in the **EventAssignment**'s `math` expression should be consistent with the units of that compartment's size. (References: L2V2 Section 4.14.2; L2V3 Section 4.14.4; L2V4 Section 4.14.4; L2V5 Section 4.14.4.)

10562. When the `variable` attribute of an **EventAssignment** contains the identifier of a **Species** in the model, the units of the mathematical expression in the **EventAssignment**'s `math` expression should be consistent with the units of that species' quantity. (References: L2V2 Section 4.14.2; L2V3 Section 4.14.4; L2V4 Section 4.14.4; L2V5 Section 4.14.4.)

10563. When the `variable` attribute of an **EventAssignment** contains the identifier of a **Parameter** in the model, the units of the mathematical expression in the **EventAssignment**'s `math` expression should be consistent with the units declared for that parameter. (References: L2V2 Section 4.14.2; L2V3 Section 4.14.4; L2V4 Section 4.14.4; L2V5 Section 4.14.4.)

General model validation

10601. A system of equations created from an SBML model must not be overdetermined. (References: L2V2 Section 4.11.5; L2V3 Section 4.11.5; L2V4 Section 4.11.5; L2V5 Section 4.11.5.)

General SBO validation (Warnings only)

10701. The value of the `sboTerm` attribute on a **Model** should be an SBO identifier referring to an interaction framework defined in SBO. That is, the value should be a term derived from `SBO:0000231`, "interaction". (References: L2V2 Section 4.2.1; L2V3 Section 4.2.2; L2V4 Section 4.2.2; L2V5 Section 4.2.2.)

10702. The value of the `sboTerm` attribute on a **FunctionDefinition** should be an SBO identifier referring to a mathematical expression. That is, the value should be a term derived from `SBO:0000064`, "mathematical expression". (References: L2V2 Section 4.3.3; L2V3 Section 4.3.3; L2V4 Section 4.3.3; L2V5 Section 4.3.3.)

10703. The value of the `sboTerm` attribute on a **Parameter** should be an SBO identifier referring to a quantitative parameter. That is, the value should be a term derived from `SBO:0000002`, "quantitative parameter". (References: L2V2 Section 4.9.5; L2V3 Section 4.9.5; L2V4 Section 4.9.5; L2V5 Section 4.9.5.)

10704. The value of the `sboTerm` attribute on an **InitialAssignment** should be an SBO identifier referring to a mathematical expression. That is, the value should be a term derived from `SBO:0000064`, "mathematical expression". (References: L2V2 Section 4.10.3; L2V3 Section 4.10.3; L2V4 Section 4.10.3; L2V5 Section 4.10.3.)

10705. The value of the `sboTerm` attribute on a **AlgebraicRule**, **RateRule** or **AssignmentRule** should be an SBO identifier referring to a mathematical expression. That is, the value should be a term derived from `SBO:0000064`, "mathematical expression". (References: L2V2 Section 4.11.1; L2V3 Section 4.11.1; L2V4 Section 4.11.1; L2V5 Section 4.11.1.)

10706. The value of the `sboTerm` attribute on a **Constraint** should be an SBO identifier referring to a mathematical expression. That is, the value should be a term derived from `SBO:0000064`, "mathematical expression". (References: L2V2 Section 4.12.3; L2V3 Section 4.12.3; L2V4 Section 4.12.3; L2V5 Section 4.12.3.)

10707. The value of the `sboTerm` attribute on a **Reaction** should be an SBO identifier referring to an interaction framework. That is, the value should be a term derived from `SBO:0000231`, "interaction". (References: L2V2 Section 4.13.1; L2V3 Section 4.13.1; L2V4 Section 4.13.1; L2V5 Section 4.13.1.)

10708. The value of the `sboTerm` attribute on a **SpeciesReference** or a **ModifierSpeciesReference** should be an SBO identifier referring to a participant role. That is, the value should be a term derived

from SBO:0000003, "participant role". The appropriate term depends on whether the entity is a reactant, product or modifier. (References: L2V2 Section 4.13.2; L2V3 Sections 4.13.1 and 5; L2V4 Section 4.13.1 and 5; L2V5 Sections 4.13.1 and 5.)

10709. The value of the sboTerm attribute on a **KineticLaw** should be an SBO identifier referring to a rate law. That is, the value should be a term derived from SBO:0000001, "rate law". (References: L2V2 Section 4.13.5; L2V3 Section 4.13.1; L2V4 Section 4.13.1; L2V5 Section 4.13.1.)

10710. The value of the sboTerm attribute on an **Event** should be an SBO identifier referring to a mathematical expression. That is, the value should be a term derived from SBO:0000231, "interaction". (References: L2V2 Section 4.14.1; L2V3 Section 4.14.1; L2V4 Section 4.14.1; L2V5 Section 4.14.1.)

10711. The value of the sboTerm attribute on an **EventAssignment** should be an SBO identifier referring to a mathematical expression. That is, the value should be a term derived from SBO:0000064, "mathematical expression". (References: L2V2 Section 4.14.2; L2V3 Section 4.14.1; L2V4 Section 4.14.4; L2V5 Section 4.14.4.)

10712. The value of the sboTerm attribute on a **Compartment** should be an SBO identifier referring to a material entity. That is, the value should be a term derived from SBO:0000240, "material entity". (References: L2V3 Section 5; L2V4 Section 5; L2V5 Section 5.)

10713. The value of the sboTerm attribute on a **Species** should be an SBO identifier referring to a material entity. That is, the value should be a term derived from SBO:0000240, "material entity". (References: L2V3 Section 5; L2V4 Section 5; L2V5 Section 5.)

10714. The value of the sboTerm attribute on a **CompartmentType** should be an SBO identifier referring to a material entity. That is, the value should be a term derived from SBO:0000240, "material entity". (References: L2V3 Section 5; L2V4 Section 5; L2V5 Section 5.)

10715. The value of the sboTerm attribute on a **SpeciesType** should be an SBO identifier referring to a material entity. That is, the value should be a term derived from SBO:0000240, "material entity". (References: L2V3 Section 5; L2V4 Section 5; L2V5 Section 5.)

10716. The value of the sboTerm attribute on a **Trigger** should be an SBO identifier referring to a mathematical expression. That is, the value should be a term derived from SBO:0000064, "mathematical expression". (References: L2V3 Section 5; L2V4 Section 5; L2V5 Section 5.)

10717. The value of the sboTerm attribute on a **Delay** should be an SBO identifier referring to a mathematical expression. That is, the value should be a term derived from SBO:0000064, "mathematical expression". (References: L2V3 Section 5; L2V4 Section 5; L2V5 Section 5.)

General notes validation

10801. The contents of the notes element must be explicitly placed in the XHTML XML namespace. (References: L2V3 Section 3.2.3; L2V4 Section 3.2.3; L2V5 Section 3.2.3.)

10802. The contents of the notes element must not contain an XML declaration (i.e., a string of the form "<?xml version="1.0" encoding="UTF-8"?>" or similar). (References: L2V2 Section 3.3.2; L2V3 Section 3.2.3; L2V4 Section 3.2.3; L2V5 Section 3.2.3.)

10803. The contents of the notes element must not contain an XML DOCTYPE declaration (i.e., a string beginning with the characters "<!DOCTYPE". (References: L2V2 Section 3.3.2; L2V3 Section 3.2.3; L2V4 Section 3.2.3; L2V5 Section 3.2.3.)

10804. The XHTML content inside a notes element can only take one of the following general forms: (1) a complete XHTML document beginning with the element <html> and ending with </html>; (2) the "body" portion of a document beginning with the element <body> and ending with </body>; or (3) XHTML content that is permitted within a <body> ... </body> elements. (References: L2V2 Section 3.3.2; L2V3 Section 3.2.3; L2V4 Section 3.2.3; L2V5 Section 3.2.3.)

SBML container validation

20101. The `sbml` container element must declare the XML Namespace for SBML, and this declaration must be consistent with the values of the `level` and `version` attributes on the `sbml` element. (References: L2V2 Section 4.1; L2V3 Section 4.1; L2V4 Section 4.1; L2V5 Section 4.1.)

20102. The `sbml` container element must declare the SBML Level using the attribute `level`, and this declaration must be consistent with the XML Namespace declared for the `sbml` element. (References: L2V2 Section 4.1; L2V3 Section 4.1; L2V4 Section 4.1; L2V5 Section 4.1.)

20103. The `sbml` container element must declare the SBML Version using the attribute `version`, and this declaration must be consistent with the XML Namespace declared for the `sbml` element. (References: L2V2 Section 4.1; L2V3 Section 4.1; L2V4 Section 4.1; L2V5 Section 4.1.)

Model validation

20201. An SBML document must contain a **Model** definition. (References: L2V1, L2V2, L2V3, L2V4 and L2V5 Section 4.1).

20202. The order of subelements within a **Model** object instance must be the following, with each element optional: `listOfFunctionDefinitions`, `listOfUnitDefinitions`, `listOfCompartmentTypes`, `listOfSpeciesTypes`, `listOfCompartments`, `listOfSpecies`, `listOfParameters`, `listOfInitialAssignments`, `listOfRules`, `listOfConstraints`, `listOfReactions`, `listOfEvents`. (References: L2V2 Section 4.2; L2V3 Section 4.2; L2V4 Section 4.2; L2V5 Section 4.2.)

20203. The `listOf_` container elements in a **Model** instance are optional, but if present, the elements must not be empty. Specifically, if any of the following are present in a **Model**, they must not be empty: `listOfFunctionDefinitions`, `listOfUnitDefinitions`, `listOfCompartmentTypes`, `listOfSpeciesTypes`, `listOfCompartments`, `listOfSpecies`, `listOfParameters`, `listOfInitialAssignments`, `listOfRules`, `listOfConstraints`, `listOfReactions` and `listOfEvents`. (References: This is requirement stemming from the XML Schema used for SBML; L2V3 Section 4.2; L2V4 Section 4.2; L2V5 Section 4.2.)

20204. If a model defines any **Species**, then the model must also define at least one **Compartment**. This is an implication of the fact that the `compartment` attribute on **Species** is not optional. (References: L2V1 Section 4.5; L2V2 Section 4.8.3; L2V3 Section 4.8.3; L2V4 Section 4.8.3; L2V5 Section 4.8.3.)

FunctionDefinition validation

20301. The top-level element within `math` in a **FunctionDefinition** must be one and only one MathML `lambda` element or a MathML `semantics` element containing one and only one `lambda` element. (References: L2V1 Section 4.3.2; L2V2 Section 4.3.2; L2V3 Section 4.3.2; L2V4 Section 4.3.2; L2V5 Section 4.3.2.)

20302. (Rule does not apply in SBML Level 2 Version 5.)

20303. Inside the `lambda` of a **FunctionDefinition**, the identifier of that **FunctionDefinition** cannot appear as the value of a `ci` element. SBML functions are not permitted to be recursive. (References: L2V2 L2V2 Section 3.5.3 and 4.3.2; L2V3 Sections 3.4.3 and 4.3.2; L2V4 Sections 3.4.3 and 4.3.2; L2V5 Sections 3.4.3 and 4.3.2.)

20304. Inside the `lambda` of a **FunctionDefinition**, if a `ci` element is not the first element within a MathML `apply`, then the `ci`'s value can only be an identifier provided as the value of a `bvar` element declared in that `lambda`. This restriction also applies to the `csymbol` for *time* and *delay*. In other words, all model quantities and variables referenced inside a function definition must be passed as arguments to that function. (References: L2V2 Section 3.5.3 and 4.3.2; L2V3 Section 3.5.3 and 4.3.2; L2V4 Section 3.4.3 and 4.3.2; L2V5 Sections 3.4.3 and 4.3.2.)

20305. The value type returned by a **FunctionDefinition**'s `lambda` must be either boolean or numerical. (References: L2V2 Section 3.5.8; L2V3 Section 3.4.9; L2V4 Section 3.4.9; L2V5 Section 3.4.10.)

20308. Inside the `lambda` MathML element within a **FunctionDefinition** object, the identifier of that object (i.e., value of the **FunctionDefinition**'s `id` attribute) may not appear as an identifier within a referenced **FunctionDefinition**'s `lambda` MathML element (nor as an identifier within another **FunctionDefinition** whose identifier is referenced, etc.). Such usage entails a recursive function call, and this usage is forbidden. (References: SBML L3V2 Section 4.3.)

Unit and UnitDefinition validation

20401. The value of the `id` attribute in a **UnitDefinition** must be of type `UnitSId` and not be identical to any unit predefined in SBML. That is, the identifier must not be the same as any of the following predefined units: "`ampere`", "`gram`", "`katal`", "`metre`", "`second`", "`watt`", "`becquerel`", "`gray`", "`kelvin`", "`mole`", "`siemens`", "`weber`", "`candela`", "`henry`", "`kilogram`", "`newton`", "`sievert`", "`coulomb`", "`hertz`", "`litre`", "`ohm`", "`steradian`", "`dimensionless`", "`item`", "`lumen`", "`pascal`", "`tesla`", "`farad`", "`joule`", "`lux`", "`radian`", or "`volt`". (References: L2V1 erratum 14; L2V2 Section 4.4.2; L2V3 Section 4.4.2; L2V4 Section 4.2; L2V5 Section 4.4.2.)

20402. Redefinitions of the predefined SBML unit `substance` must be based on the unit `mole`, `item`, `gram`, `kilogram`, or `dimensionless`. More formally, a **UnitDefinition** for `substance` must simplify to a single **Unit** in which either (a) the `kind` attribute has a value of "`mole`", "`item`", "`gram`", or "`kilogram`", and the `exponent` attribute has a value of "`1`", or (b) the `kind` attribute has a value of "`dimensionless`" with any `exponent` attribute value. (References: L2V1 Section 4.4.3; L2V2 Section 4.4.3; L2V3 Section 4.4.3; L2V4 Section 4.4.3; L2V5 Section 4.4.3.)

20403. Redefinitions of the predefined SBML unit `length` must be based on the unit `metre` or `dimensionless`. More formally, a **UnitDefinition** for `length` must simplify to a single **Unit** in which either (a) the `kind` attribute has a value of "`metre`" and the `exponent` attribute has a value of "`1`", or (b) the `kind` attribute has a value of "`dimensionless`" with any `exponent` value. (References: L2V1 Section 4.4.3; L2V2 Section 4.4.3; L2V3 Section 4.4.3; L2V4 Section 4.4.3; L2V5 Section 4.4.3.)

20404. Redefinitions of the predefined SBML unit `area` must be based on squared `metre`s or `dimensionless`. More formally, a **UnitDefinition** for `area` must simplify to a single **Unit** in which either (a) the `kind` attribute has a value of "`metre`" and the `exponent` attribute has a value of "`2`", or (b) the `kind` attribute has a value of "`dimensionless`" with any `exponent` value. (References: L2V1 Section 4.4.3; L2V2 Section 4.4.3; L2V3 Section 4.4.3; L2V4 Section 4.4.3; L2V5 Section 4.4.3.)

20405. Redefinitions of the predefined SBML unit `time` must be based on the unit `second`. More formally, a **UnitDefinition** for `time` must simplify to a single **Unit** in which either (a) the `kind` attribute has a value of "`second`" and the `exponent` attribute has a value of "`1`", or (b) the `kind` attribute has a value of "`dimensionless`" with any `exponent` value. (References: L2V1 Section 4.4.3; L2V2 Section 4.4.3; L2V3 Section 4.4.3; L2V4 Section 4.4.3; L2V5 Section 4.4.3.)

20406. Redefinitions of the predefined SBML unit `volume` must be based on `litre`, `metre` or `dimensionless`. More formally, a **UnitDefinition** for `volume` must simplify to a single **Unit** in which either (a) the `kind` attribute has a value of "`metre`" and the `exponent` attribute has a value of "`3`"; (b) the `kind` attribute has a value of "`litre`" and the `exponent` attribute has a value of "`1`"; or (c) the kind attribute has a value of "`dimensionless`" with any `exponent` value. (References: L2V1 Section 4.4.3; L2V2 Section 4.4.3; L2V3 Section 4.4.3; L2V4 Section 4.4.3; L2V5 Section 4.4.3.)

20407. (Beginning in SBML Level 2 Version 4, this rule has been incorporated into 20406, to make validation rules 20402–20406 parallel and consistent.)

20408. (Beginning in SBML Level 2 Version 4, this rule has been incorporated into 20406, to make validation rules 20402–20406 parallel and consistent.)

20409. The `listOfUnits` container in a **UnitDefinition** cannot be empty. (References: L2V2 Section 4.4; L2V3 Section 4.4; L2V4 Section 4.4; L2V5 Section 4.4.)

20410. The value of the `kind` attribute of a **Unit** can only be one of the base units in SBML; that is, the SBML unit system is not hierarchical and user-defined units cannot be defined using other user-defined units. (References: L2V2 Section 4.4.1; L2V3 Section 4.4.2; L2V4 Section 4.4.2; L2V5 Section 4.4.2.)

20411. (Rule does not apply in SBML Level 2 Version 5.)

20412. (Rule does not apply in SBML Level 2 Version 5.)

Compartment validation

20501. The size of a **Compartment** must not be set if the compartment's `spatialDimensions` attribute has value `0`. (References: L2V1 Section 4.5.3; L2V2 Section 4.7.4; L2V3 Section 4.7.5; L2V4 Section 4.7.5; L2V5 Section 4.7.5.)

20502. If a **Compartment** definition has a `spatialDimensions` value of "`0`", then its `units` attribute must not be set. If the compartment has no dimensions, then no units can be associated with a non-existent size. (References: L2V1 Section 4.5.4; L2V2 Section 4.7.5; L2V3 Section 4.7.5; L2V4 Section 4.7.5; L2V5 Section 4.7.5.)

20503. If a **Compartment** definition has a `spatialDimensions` value of "`0`", then its `constant` attribute value must either default to or be set to "`true`". If the compartment has no dimensions, then its size can never change. (References: L2V1 Section 4.5.5; L2V2 Section 4.7.4; L2V3 Section 4.7.6; L2V4 Section 4.7.6; L2V5 Section 4.7.6.)

20504. The `outside` attribute value of a **Compartment** must be the identifier of another **Compartment** defined in the model. (References: L2V1 Section 4.5.6; L2V2 Section 4.7.7; L2V3 Section 4.7.7; L2V4 Section 4.7.7; L2V5 Section 4.7.7.)

20505. A **Compartment** may not enclose itself through a chain of references involving the `outside` attribute. This means that a compartment cannot have its own identifier as the value of `outside`, nor can it point to another compartment whose `outside` attribute points directly or indirectly to the compartment. (References: L2V1 erratum 11; L2V2 Section 4.7.7; L2V3 Section 4.7.7; L2V4 Section 4.7.7; L2V5 Section 4.7.7.)

20506. The `outside` attribute value of a given **Compartment** instance cannot be a compartment having a `spatialDimensions` value of "`0`", unless both compartments have `spatialDimensions`="`0`". Simply put, a zero-dimensional compartment cannot enclose compartments that have anything other than zero dimensions themselves. (References: L2V2 Section 4.7.7; L2V3 Section 4.7.7; L2V4 Section 4.7.7; L2V5 Section 4.7.7.)

20507. The value of the `units` attribute on a **Compartment** having `spatialDimensions` of "`1`" must be either "`length`", "`metre`", "`dimensionless`", or the identifier of a **UnitDefinition** based on either `metre` (with `exponent` equal to "`1`") or `dimensionless`. (References: L2V1 Section 4.5.4; L2V2 Section 4.7.5; L2V3 Section 4.7.5; L2V4 Section 4.7.5; L2V5 Section 4.7.5.)

20508. The value of the `units` attribute on a **Compartment** having `spatialDimensions` of "`2`" must be either "`area`", "`dimensionless`", or the identifier of a **UnitDefinition** based on either `metre` (with `exponent` equal to "`2`") or `dimensionless`. (References: L2V1 Section 4.5.4; L2V2 Section 4.7.5; L2V3 Section 4.7.5; L2V4 Section 4.7.5; L2V5 Section 4.7.5.)

20509. The value of the `units` attribute on a **Compartment** having `spatialDimensions` of "`3`" must be either "`volume`", "`litre`", or the identifier of a **UnitDefinition** based on either `litre`, `metre` (with `exponent` equal to "`3`"), or `dimensionless`. (References: L2V1 Section 4.5.4; L2V2 Section 4.7.5; L2V3 Section 4.7.5; L2V4 Section 4.7.5; L2V5 Section 4.7.5.)

20510. If the `compartmentType` attribute is given a value in a **Compartment** definition, it must contain the identifier of an existing **CompartmentType** defined in the model. (References: L2V2 Section 4.7.2; L2V3 Section 4.7.2; L2V4 Section 4.7.2; L2V5 Section 4.7.2.)

Species validation

20601. The value of `compartment` in a **Species** definition must be the identifier of an existing **Compartment** defined in the model. (References: L2V1 Section 4.6.2; L2V2 Section 4.8.3; L2V3 Section 4.8.3; L2V4 Section 4.8.3; L2V5 Section 4.8.3.)

20602. (Rule does not apply in SBML Level 2 Version 5.)

20603. (Rule does not apply in SBML Level 2 Version 5.)

20604. If a **Species** located in a **Compartment** whose `spatialDimensions` is set to "`0`", then that **Species** definition cannot set `initialConcentration`. (References: L2V1 Section 4.6.3; L2V2 Section 4.8.4; L2V3 Section 4.8.4; L2V4 Section 4.8.4; L2V5 Section 4.8.4.)

20605. (Rule does not apply in SBML Level 2 Version 5.)

20606. (Rule does not apply in SBML Level 2 Version 5.)

20607. (Rule does not apply in SBML Level 2 Version 5.)

20608. The value of a **Species**'s `substanceUnits` attribute can only be one of the following: "`substance`", "`mole`", "`item`", "`gram`", "`kilogram`", "`dimensionless`", or the identifier of a **UnitDefinition** derived from "`mole`" (with an `exponent` of "`1`"), "`item`" (with an `exponent` of "`1`"), "`gram`" (with an `exponent` of "`1`"), "`kilogram`" (with an `exponent` of "`1`"), or "`dimensionless`". (References: L2V1 Section 4.6.4; L2V2 Section 4.8.5; L2V3 Section 4.8.5; L2V4 Section 4.8.5; L2V5 Section 4.8.5.)

20609. A **Species** cannot set values for both `initialConcentration` and `initialAmount` because they are mutually exclusive. (References: L2V1 Section 4.6.3; L2V2 Section 4.8.4; L2V3 Section 4.8.4; L2V4 Section 4.8.4; L2V5 Section 4.8.4.)

20610. A **Species**' quantity cannot be determined simultaneously by both reactions and rules. More formally, if the identifier of a **Species** definition having `boundaryCondition`="`false`" and `constant`="`false`" is referenced by a **SpeciesReference** anywhere in a model, then this identifier cannot also appear as the value of a `variable` in an **AssignmentRule** or a **RateRule**. (References: L2V1 Section 4.6.5; L2V2 Section 4.8.6; L2V3 Section 4.8.6; L2V4 Section 4.8.6; L2V5 Section 4.8.6.)

20611. A **Species** having `boundaryCondition`="`false`" cannot appear as a reactant or product in any reaction if that **Species** also has `constant`="`true`". (References: L2V1 Section 4.6.5; L2V2 Section 4.8.6; L2V3 Section 4.8.6; L2V4 Section 4.8.6; L2V5 Section 4.8.6.)

20612. The value of `speciesType` in a **Species** definition must be the identifier of an existing **SpeciesType**. (References: L2V2 Section 4.8.2; L2V3 Section 4.8.2; L2V4 Section 4.8.2; L2V5 Section 4.8.2.)

20613. There cannot be more than one species of a given **SpeciesType** in the same compartment of a model. More formally, for any given compartment, there cannot be more than one **Species** definition in which both of the following hold simultaneously: (i) the **Species**' `compartment` value is set to that compartment's identifier and (ii) the **Species**' `speciesType` is set the same value as the `speciesType` of another **Species** that also sets its `compartment` to that compartment identifier. (References: L2V2 Section 4.8.2; L2V3 Section 4.8.2; L2V4 Section 4.8.2; L2V5 Section 4.8.2.)

20614. The `compartment` attribute in a **Species** is mandatory. A species definition in a model must include a value for this attribute. (References: L2V2 Section 4.8.3; L2V3 Section 4.8.3; L2V4 Section 4.8.3; L2V5 Section 4.8.3.)

Parameter validation

20701. The `units` in a **Parameter** definition must be a value chosen from among the following: a predefined unit (e.g., "`substance`", "`time`", etc.), the identifier of a **UnitDefinition** in the model, or one of the base units in SBML (e.g., "`litre`", "`mole`", "`metre`", etc.). (References: L2V1 Section 4.7.3; L2V2 Section 4.9.3; L2V3 Section 4.9.3; L2V4 Section 4.8.3; L2V5 Section 4.9.3.)

InitialAssignment validation

20801. The value of `symbol` in an **InitialAssignment** definition must be the identifier of an existing **Compartment**, **Species**, or **Parameter** defined in the model. (References: L2V2 Section 4.10.1; L2V3 Section 4.10; L2V4 Section 4.10; L2V5 Section 4.10.)

20802. A given identifier cannot appear as the value of more than one **InitialAssignment**'s `symbol` attribute across the set of all **InitialAssignment**s in a model. (References: L2V2 Section 4.10; L2V3 Section 4.10; L2V4 Section 4.10; L2V5 Section 4.10.)

20803. The identifier given as the value of a `symbol` attribute in any **InitialAssignment** definition cannot also appear as the value of the `variable` attribute in an **AssignmentRule**. In other words, a model cannot simultaneously define both an initial assignment and an assignment rule for the same species, compartment or parameter in a model. (References: L2V2 Section 4.10.4; L2V3 Section 4.10; L2V4 Section 4.10; L2V5 Section 4.10.)

20806. The identifier given as the value of a `symbol` attribute in any **InitialAssignment** definition cannot be the identifier of a **Compartment** with a `spatialDimensions` value of "`0`". (References: L2V5 Section 4.7.)

AssignmentRule and RateRule validation

20901. The value of an **AssignmentRule**'s `variable` must be the identifier of an existing **Compartment**, **Species**, or globally-defined **Parameter**. (References: L2V1 Section 4.8.2; L2V2 Section 4.11.3; L2V3 Section 4.11.3; L2V4 Section 4.11.3; L2V5 Section 4.11.3.)

20902. The value of a **RateRule**'s `variable` must be the identifier of an existing **Compartment**, **Species**, or globally-defined **Parameter**. (References: L2V1 Section 4.8.3; L2V2 Section 4.11.4; L2V3 Section 4.11.4; L2V4 Section 4.11.4; L2V5 Section 4.11.4.)

20903. Any **Compartment**, **Species** or **Parameter** whose identifier is the value of a `variable` attribute in an **AssignmentRule**, must have a value of "`false`" for `constant`. (References: L2V1 Section 4.8.4; L2V2 Section 4.11.3; L2V3 Section 4.11.3; L2V4 Section 4.11.3; L2V5 Section 4.11.3.)

20904. Any **Compartment**, **Species** or **Parameter** whose identifier is the value of a `variable` attribute in a **RateRule**, must have a value of "`false`" for `constant`. (References: L2V1 Section 4.8.4; L2V2 Section 4.11.4; L2V3 Section 4.11.4; L2V4 Section 4.11.4; L2V5 Section 4.11.4.)

20905. (Rule removed because it was effectively a duplicate of 10304.)

20906. There must not be circular dependencies in the combined set of **InitialAssignment**, **AssignmentRule** and **KineticLaw** definitions in a model. Each of these constructs has the effect of assigning a value to an identifier (i.e., the identifier given in the attribute `symbol` in **InitialAssignment**, the attribute `variable` in **AssignmentRule**, and the attribute `id` on the **KineticLaw**'s enclosing **Reaction**). Each of these constructs computes the value using a mathematical formula. The formula for a given identifier cannot make reference to a second identifier whose own definition depends directly or indirectly on the first identifier. (References: L2V2 Section 4.11.5; L2V3 Section 4.11.5; L2V4 Section 4.11.5; L2V5 Section 4.11.5.)

20911. The value of a **RateRule** or **AssignmentRule**'s `variable` attribute must not be the identifier of a **Compartment** with a `spatialDimensions` value of "`0`". (References: L2V5 Section 4.11.5.)

Constraint validation

21001. A **Constraint** `math` expression must evaluate to a value of type `boolean`. (References: L2V2 Section 4.12.1; L2V3 Section 4.12; L2V4 Section 4.12; L2V5 Section 4.12.)

21002. The order of subelements within **Constraint** must be the following: `math`, `message`. The `message` element is optional, but if present, must follow the `math` element. (References: L2V2 Section 4.12; L2V3 Section 4.12; L2V4 Section 4.12; L2V5 Section 4.12.)

21003. The contents of the `message` element in a **Constraint** must be explicitly placed in the XHTML XML namespace. (References: L2V3 Section 4.12.2; L2V4 Section 4.12.2; L2V5 Section 4.12.2.)

21004. The contents of the `message` element must not contain an XML declaration (i.e., a string of the form "`<?xml version="1.0" encoding="UTF-8"?>`" or similar). (References: L2V3 Section 4.12.2; L2V4 Section 4.12.2; L2V5 Section 4.12.2.)

21005. The contents of the `message` element must not contain an XML DOCTYPE declaration (i.e., a string beginning with the characters "`<!DOCTYPE`". (References: L2V3 Section 4.12.2; L2V4 Section 4.12.2; L2V5 Section 4.12.2.)

21006. The XHTML content inside a **Constraint**'s `message` element can only take one of the following general forms: (1) a complete XHTML document beginning with the element `<html>` and ending with `</xhtml>`; (2) the "body" portion of a document beginning with the element `<body>` and ending with `</body>`; or (3) XHTML content that is permitted within a `<body>` ... `</body>` elements. (References: L2V3 Section 4.12.2; L2V4 Section 4.12.2; L2V5 Section 4.12.2.)

Reaction validation

21101. A **Reaction** definition must contain at least one **SpeciesReference**, either in its `listOfReactants` or its `listOfProducts`. A reaction without any reactant or product species is not permitted, regardless of whether the reaction has any modifier species. (References: L2V2 Section 4.13.1; L2V3 Section 4.13.3; L2V4 Section 4.13.3; L2V5 Section 4.13.3.)

21102. The order of subelements within **Reaction** must be the following (where every one is optional): `listOfReactants`, `listOfProducts`, `listOfModifiers`, `kineticLaw`. (References: L2V2 Section 4.13; L2V3 Section 4.13; L2V4 Section 4.13; L2V5 Section 4.13.)

21103. The following containers are all optional in a **Reaction**, but if any is present, it must not be empty: `listOfReactants`, `listOfProducts`, `listOfModifiers`, `kineticLaw`. (References: L2V2 Section 4.13; L2V3 Section 4.13; L2V4 Section 4.13; L2V5 Section 4.13.)

21104. The list of reactants (`listOfReactants`) and list of products (`listOfProducts`) in a **Reaction** can only contain `speciesReference` elements. (References: L2V1 Section 4.9; L2V2 Section 4.13; L2V3 Section 4.13; L2V4 Section 4.13; L2V5 Section 4.13.)

21105. The list of modifiers (`listOfModifiers`) in a **Reaction** can only contain `modifierSpeciesReference` elements. (References: L2V1 Section 4.9; L2V2 Section 4.13; L2V3 Section 4.13; L2V4 Section 4.13; L2V5 Section 4.13.)

SpeciesReference and ModifierSpeciesReference validation

21111. The value of a **SpeciesReference** `species` attribute must be the identifier of an existing **Species** in the model. (References: L2V1 Section 4.9.5; L2V2 Section 4.13.2; L2V3 Section 4.13.3; L2V4 Section 4.13.3; L2V5 Section 4.13.3.)

21112. (Rule removed because it was effectively a duplicate of 20611.)

21113. A **SpeciesReference** must not have a value for both `stoichiometry` and `stoichiometryMath`; they are mutually exclusive. (References: L2V1 Section 4.9.5; L2V2 Section 4.13.3; L2V3 Section 4.13.3; L2V4 Section 4.13.3; L2V5 Section 4.13.3.)

KineticLaw validation

21121. All species referenced in the **KineticLaw** formula of a given reaction must first be declared using **SpeciesReference** or **ModifierSpeciesReference**. More formally, if a **Species** identifier appears in a `ci` element of a **Reaction**'s **KineticLaw** formula, that same identifier must also appear in at least one **SpeciesReference** or **ModifierSpeciesReference** in the **Reaction** definition. (References: L2V2 Section 4.13.5; L2V3 Section 4.13.5; L2V4 Section 4.13.5; L2V5 Section 4.13.5.)

21122. The order of subelements within **KineticLaw** must be the following: `math`, `listOfParameters`. The `listOfParameters` is optional, but if present, must follow `math`. (References: L2V2 Section 4.13.5; L2V3 Section 4.13.5; L2V4 Section 4.13.5; L2V5 Section 4.13.5.)

21123. If present, the `listOfParameters` in a **KineticLaw** must not be an empty list. (References: L2V2 Section 4.13.5; L2V3 Section 4.13; L2V4 Section 4.13; L2V5 Section 4.13.)

21124. The `constant` attribute on a **Parameter** local to a **KineticLaw** cannot have a value other than "`true`". The values of parameters local to **KineticLaw** definitions cannot be changed, and therefore they are always constant. (References: L2V2 Section 4.9.4; L2V3 Section 4.9.4; L2V4 Section 4.9.4; L2V5 Section 4.9.4.)

21125. (Rule does not apply in SBML Level 2 Version 5.)

21126. (Rule does not apply in SBML Level 2 Version 5.)

StoichiometryMath validation

21131. All species referenced in the **StoichiometryMath** formula of a given reaction must first be declared using **SpeciesReference** or **ModifierSpeciesReference**. More formally, if a **Species** identifier appears in a `ci` element of a **Reaction**'s **StoichiometryMath** formula, that same identifier must also appear in at least one **SpeciesReference** or **ModifierSpeciesReference** in the **Reaction** definition. (References: L2V2 Sections 4.13.2 and 4.13.4; L2V3 Sections 4.13.2 and 4.13.4; L2V4 Section 4.13.4; L2V5 Sections 4.13.2 and 4.13.4.)

Event validation

21201. An **Event** object must have a `trigger`. (References: L2V1 Section 4.10.2; L2V2 Section 4.14.1; L2V3 Section 4.14.2; L2V4 Section 4.14.2; L2V5 Section 4.14.2.)

21202. An **Event** `trigger` expression must evaluate to a value of type `boolean`. (References: L2V1 Section 4.10.2; L2V2 Section 4.14.1; L2V3 Section 4.14.2; L2V4 Section 4.14.2; L2V5 Section 4.14.2.)

21203. An **Event** object must have at least one **EventAssignment** object in its `listOfEventAssignments`. (References: L2V1 Section 4.10.5; L2V2 Section 4.14; L2V3 Section 4.14; L2V4 Section 4.14; L2V5 Section 4.14.)

21204. (Rule does not apply in SBML Level 2 Version 5.)

21205. The order of subelements within an **Event** object instance must be the following: `trigger`, `delay`, `listOfEventAssignments`. The `delay` element is optional, but if present, must follow `trigger`. (References: L2V2 Section 4.14; L2V3 Section 4.14; L2V4 Section 4.14; L2V5 Section 4.14.)

21206. If an **Event**'s `useValuesFromTriggerTime` attribute has the value "`false`", then the **Event** must contain a **Delay** element. The implication of `useValuesFromTriggerTime`="`false`" is that there is a delay between the time of trigger and the time of value assignments performed by the **Event**. (References: L2V4 Section 4.14; L2V5 Section 4.14.)

EventAssignment validation

21211. The value of `variable` in an **EventAssignment** can only be the identifier of a **Compartment**, **Species**, or model-wide **Parameter** definition. (References: L2V1 Section 4.10.5; L2V2 Section 4.14.2; L2V3 Section 4.14.4; L2V4 Section 4.14.4; L2V5 Section 4.14.4.)

21212. Any **Compartment**, **Species** or **Parameter** definition whose identifier is used as the value of `variable` in an **EventAssignment** must have a value of "`false`" for its `constant` attribute. (References: L2V1 Section 4.10.5; L2V2 Section 4.14.2; L2V3 Section 4.14.4; L2V4 Section 4.14.4; L2V5 Section 4.14.4.)

D A method for assessing whether an SBML model is overdetermined

As explained in Section 4.11.5, an SBML model must not be overdetermined. It is possible to use purely static analysis to assess this condition for the system of equations implied by a model, by constructing a bipartite graph of the model's variables and equations and then searching for a maximal matching (Chartrand, 1977). An efficient algorithm for finding a maximal matching is described by Hopcroft and Karp (1973). In this appendix, we provide a concrete application to SBML of the general approach described in Section 4.11.5. The approach is defined in terms of the ordinary differential equations (ODEs) implied by an SBML model; despite our use of a differential equation framework for this explanation, it should be understood that this use of ODEs has no implication about the framework actually used to simulate the model.

Definition of the method

First, we assume that an ODE is constructed for each species determined by one or more **Reaction**'s **KineticLaw** `math` expressions. We also assume that the model has already been determined to be valid in all other respects (e.g., there are no undefined variables in the equations), and what remains is to evaluate whether it is overdetermined.

We construct the bipartite graph for a given SBML model as follows:

1. For each of the following in the model, create one vertex representing an equation:
   (a) Every **Species** object having `boundaryCondition`="`false`", `constant`="`false`", and which is referenced as a reactant or product in one or more **Reaction** objects containing **KineticLaw** objects
   (b) Every **AssignmentRule** object
   (c) Every **RateRule** object
   (d) Every **AlgebraicRule** object
   (e) Every **KineticLaw** object
2. For each of the following in the model, create one vertex representing a variable:
   (a) Every **Species** object having `constant`="`false`"
   (b) Every **Compartment** object having `constant`="`false`"
   (c) Every global **Parameter** having `constant`="`false`"
   (d) Every **Reaction** object
3. For each of the following, create one edge:
   (a) Every vertex created in step 2(a) to that species' equation vertex created in step 1(a)
   (b) Every vertex created in step 1(b) to the particular vertex created in steps 2(a)–2(d) that represents the variable referenced by the `variable` attribute of the rule
   (c) Every vertex created in step 1(c) to the particular vertex created in steps 2(a)–2(d) that represents the variable referenced by the `variable` attribute of the rule
   (d) Every vertex created in step 1(e) to the particular vertex created in step 2(d) that is the **Reaction** object containing that particular **KineticLaw** object
   (e) Every vertex created in steps 2(a)–2(d) representing an identifier appearing as the content of a MathML `ci` element within an expression of an **AlgebraicRule**, to the vertex for that particular **AlgebraicRule** created in step 1(d)

Example application of the method

What follows is an example of applying the method above to the SBML model shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2/version5" level="2" version="5">
    <model id="example">
        <listOfUnitDefinitions>
            <unitDefinition id="per_second">
                <listOfUnits>
                    <unit kind="second" exponent="-1"/>
                </listOfUnits>
            </unitDefinition>
        </listOfUnitDefinitions>
        <listOfCompartments>
            <compartment id="C" size="1" constant="true"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="S1" compartment="C" initialConcentration="1"/>
            <species id="S2" compartment="C" initialConcentration="0"/>
        </listOfSpecies>
        <listOfRules>
            <algebraicRule>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <plus/>
                        <ci> S1 </ci>
                        <ci> S2 </ci>
                    </apply>
                </math>
            </algebraicRule>
        </listOfRules>
        <listOfReactions>
            <reaction id="R">
                <listOfReactants>
                    <speciesReference species="S1"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="S2"/>
                </listOfProducts>
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <times/>
                            <ci> C </ci>
                            <ci> k1 </ci>
                            <ci> S1 </ci>
                        </apply>
                    </math>
                    <listOfParameters>
                        <parameter id="k1" value="0.1" units="per_second"/>
                    </listOfParameters>
                </kineticLaw>
            </reaction>
        </listOfReactions>
    </model>
</sbml>
```

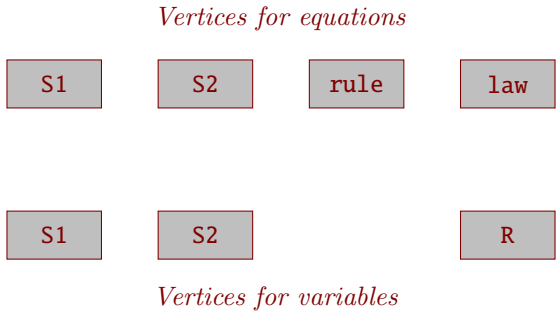
For the model above, we create *equation* vertices as follows:

1. [Corresponding to step 1(a) in Section D.] *Every* ***Species*** *object having* `boundaryCondition`="`false`", `constant`="`false`", *and which is referenced as a reactant or product in one or more* ***Reaction*** *objects containing* ***KineticLaw*** *objects.* This generates two vertices, for "`S1`" and "`S2`".
2. [Corresponding to step 1(b) in Section D.] *Every* ***AlgebraicRule*** *object.* This generates one vertex, for the model's lone algebraic rule (call it "`rule`").
3. [Corresponding to step 1(e) in Section D.] *Every* ***KineticLaw*** *object.* This generates one vertex, for the lone kinetic law in the model (call it "`law`").

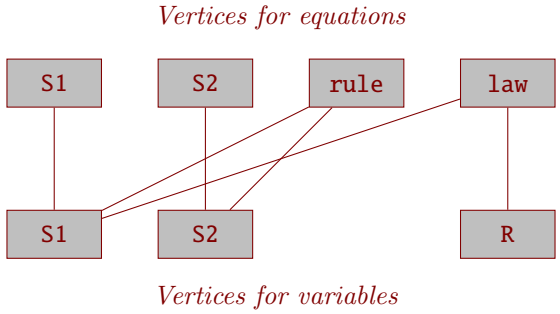
We create *variable* vertices for the following:

1. [Corresponding to step 2(a) in Section D.] *Every* **Species** *object having* `constant="false"`. This generates two vertices, for "`S1`" and "`S2`".
2. [Corresponding to step 2(d) in Section D.] *Every* **Reaction** *object.* This generates one vertex, for "`R`".

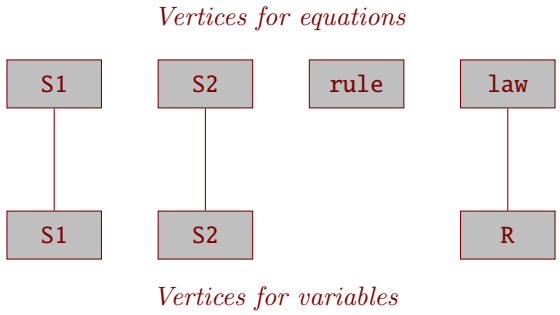
Note that it is not necessary to include parameters declared within **KineticLaw** objects because they are local to a particular reaction and cannot be affected by rules. With the steps above, we have the following set of graph nodes:



Next, we create edges following the procedure described above. Doing so results in the following graph:



The algorithm of Hopcroft and Karp (1973) can now be applied to search for a maximal matching of the bipartite graph. A maximal matching is a graph in which each vertex is connected to at most one other vertex and the maximum possible number of connections have been made. Doing so here results in the following:



If the maximal matching of the bipartite graph leaves any equation vertex unconnected, then the model is considered overdetermined. That is the case for the example shown here, because the equation vertex for "`rule`" is unconnected in the maximal matching.

E Mathematical consequences of the `fast` attribute on Reaction

(Appendix contributed by James C. Schaff, University of Connecticut Health Center, Connecticut, U.S.A.)

Section 4.13.1 described the `fast` flag available on **Reaction**. In this appendix, we discuss the principles involved in interpreting this attribute in the context of a simple biochemical reaction model. The derivation presented here is not fully rigorous and this section is not considered normative; achieving a higher level of rigor would require considerably more background exposition and a much longer appendix. Nevertheless, we hope this section is sufficient to answer unambiguously the question "How should a system of reactions be treated if some of the reactions have `fast=true`?"

Identification of "fast" reactions

First, it is worth noting that the identification of so-called *fast* reactions is actually a modeling issue, not an SBML representation issue. The notion of fast reactions is the following. A system may be decomposable into two sets of reactions, where one set may have characteristic times that are much faster than the other time scales in the system. An approximation that is sometimes useful is to assume that the fast reactions have kinetics that settle infinitely fast compared to the other reactions in the system. In other words, the fast reactions are assumed to be always in equilibrium. This is called a pseudo-steady state approximation (PSSA), and is also known as a quasi-steady state approximation (QSSA). Given a case where the time-scale separation between fast and other reactions in the system is large, an accurate and efficient approach for computing the time-course of the system behavior is to treat the fast reactions as being always in equilibrium.

The key to successful application of a PSSA is that there should be a significant separation of time scales between these fast reactions and other reactions in the system. The determination of which reactions qualify as fast is up to the creator of the model, because there is currently no known general algorithm for doing so.

Simple one-compartment biochemical system model

To explain how to solve a system containing fast reactions, we use a simple model of a biochemical reaction network located in a single compartment. Let $\mathbf{x}^*$ represent a vector of all the species in the system, $\mathbf{v}^*$ a vector of the reaction rates, and $\mathbf{A}^*$ the stoichiometry matrix, with the vector dimension being $\mathbf{n}^*$. Then the system can be described using the following matrix equation:

$$\frac{d\mathbf{x}^*}{dt} = \mathbf{A}^*\,\mathbf{v}^*(\mathbf{x}^*)$$

This system can be optionally reduced by noting that mass conservation usually implies there are linear combinations of species quantities in the system and the value of these combinations do not change over time. Identifying these combinations is the topic of structural analysis and is described in the literature (Reder, 1988; Sauro and Ingalls, 2003). Briefly, let $\mathbf{N}$ be defined as the left null space of $\mathbf{A}^*$:

$$\mathbf{N}\mathbf{A}^* = \mathbf{0}$$

Now, premultiply the previous equation by $\mathbf{N}$ to get

$$\mathbf{N}\frac{d\mathbf{x}^*}{dt} = \mathbf{N}\mathbf{A}^*\mathbf{v}^*(\mathbf{x}^*) = \mathbf{0}$$

Thus, $\mathbf{N}$ captures the space of solutions to the equation

$$\mathbf{m}^T\left(\frac{d\mathbf{x}^*}{dt}\right) = 0$$

where $\mathbf{m}$ is a vector representing the coefficients in a mass conservation relationship, that is, combinations of species that are time-invariant. Now, let

$$r = \mathrm{rank}(\mathbf{A}^*)$$
$$n = \dim(\mathbf{x}^*)$$

Then the system has $n - r$ mass conservation relationships, each of which is a linear equation. We can use these $n - r$ linear equations to solve for $n - r$ dependent variables in terms of r independent variables and the initial masses of all species. Doing that allows us to decompose $\mathbf{x}^*$ into $n - r$ dependent variables $\mathbf{x}_d$ and r independent variables $\mathbf{x}_i$ where $\mathbf{L}$ is an $(n - r) \times r$ matrix that is derived from $\mathbf{N}$ and represents $\mathbf{x}_d$ in terms of $\mathbf{x}_i$, $\mathbf{I}$ is the $r \times r$ identity matrix, and $\mathbf{T}$ is an $n \times r$ matrix:

$$\mathbf{x}^* \equiv \begin{bmatrix} \mathbf{x}_i \\ \mathbf{x}_d \end{bmatrix} = \begin{bmatrix} \mathbf{I} \\ \mathbf{L} \end{bmatrix} \mathbf{x}_i = \mathbf{T}\mathbf{x}_i$$

Using this equation, we can define a new vector of reaction velocities $\mathbf{v}$ in terms of $\mathbf{x}_i$ only:

$$\mathbf{v}(\mathbf{x}_i) \equiv \mathbf{v}^*(\mathbf{T}\mathbf{x}_i)$$

With this $\mathbf{v}$, we can now write a reduced system by substituting terms. First we define $\mathbf{A}$ as the r independent rows of $\mathbf{A}^*$corresponding to $\mathbf{x}_i$. Then:

$$\frac{d\mathbf{x}_i}{dt} = \mathbf{A}\mathbf{v}(\mathbf{x}_i)$$

This is a set of r independent differential equations in r unknowns (i.e., an r-dimensional system). To simplify the notation slightly, let

$$\mathbf{x} \equiv \mathbf{x}_i$$

and, thus,

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{v}(\mathbf{x})$$

Application of a PSSA to biochemical systems

Assume that we have eliminated redundant variables and equations using the mass conservation analysis above. Further assume that we have some external means of classifying some reactions in a given system as being *fast* as discussed earlier. We now need to apply this to the system under study. We begin by decomposing the vector of reaction velocities $\mathbf{v}$ according to fast and slow reactions:

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}_1\mathbf{v}_f(\mathbf{x}) + \mathbf{A}_2\mathbf{v}_s(\mathbf{x})$$

In the expression above, $\mathbf{A}_1$ represents the stoichiometry of the set of reactions operating on the fast time scale, and $\mathbf{A}_2$ the stoichiometry of the set of reactions operating on a slower time scale. We find the left null space of $\mathbf{A}_1$ (i.e., the space of solutions to $\mathbf{m}^T[d\mathbf{x}/dt] = 0$ on a fast time scale), and call this matrix $\mathbf{B}$:

$$\mathbf{B}\mathbf{A}_1 = \mathbf{0}$$

The matrix $\mathbf{B}$ represents the linear combination of species that do not change on a fast time scale, i.e., the slow species in the system. Now, we premultiply the equation for $d\mathbf{x}/dt$ by $\mathbf{B}$:

$$\begin{aligned} \mathbf{B}\frac{d\mathbf{x}}{dt} &= \mathbf{B}\mathbf{A}_1\mathbf{v}_f(\mathbf{x}) + \mathbf{B}\mathbf{A}_2\mathbf{v}_s(\mathbf{x}) \\ &= \mathbf{B}\mathbf{A}_2\mathbf{v}_s(\mathbf{x}) \end{aligned}$$

where the second line follows from the fact that $\mathbf{B}\mathbf{A}_1 = \mathbf{0}$. The above is an ordinary differential equation in terms of only the slow dynamics. The remaining fast dynamics are handled by applying the pseudo-steady state approximation, with fast transients assumed to have settled with respect to the slow time scale. This produces a system of nonlinear algebraic equations:

$$\mathbf{A}_1\mathbf{v}_f = \mathbf{0}$$

The last two equations form the system of equations resulting from the application of the PSSA. If $r_1 = \text{rank}(\mathbf{A}_1)$ and $r = \text{rank}(\mathbf{A})$, then there will be r_1 degrees of freedom that will be determined by solving an algebraic system (the equation $\mathbf{A}_1\mathbf{v}_f = \mathbf{0}$ above), and there will be $r - r_1$ degrees of freedom that will be determined by ordinary differential equations (the equation for $\mathbf{B}\,d\mathbf{x}/dt$).

F Processing and validating SBase `notes` and Constraint `message` content

In Section 3.2.3 and Section 4.12.2, we discussed the `notes` element on ***SBase*** and the `message` element on **Constraint**, respectively. These elements can contain a number of possible forms of XHTML content. In this appendix, we describe a general procedure for how application software can process such content. We concentrate on the common case of an SBML-reading application that needs to take the contents and pass it to an XHTML display and/or editing function obtained from a third-party API library. The content of the `notes` and `message` may not be a complete XHTML document, so the application will have to perform some processing before handing it to the XHTML editor or validator. How should this be done?

Based on the three forms of ***SBase*** `notes` content described in Section 3.2.3 and the identical forms for **Constraint** `message` described in Section 4.12.2, there are only three cases possible. Here we give an example approach for handling them, although the actual implementation details will differ depending on various factors such as the requirements of the software libraries being used. This example approach would be performed for each `notes` and `message` to be viewed or edited:

Step 1. If the XHTML viewing/editing function requires a fully compliant XML document, the SBML application could create a temporary data object containing an appropriate XML declaration and a DOCTYPE declaration; otherwise, the XML data object can be initially blank.

Step 2. The application should look at the first element inside the `notes` or `message` (or rather, the first element that is not an XML comment), and take action based on the following three possibilities:

- If the first element begins with `<html xmlns="http://www.w3.org/1999/xhtml">`, the application could assume that the content is a complete XHTML document and insert this into the temporary data object.

- Else, if the first element is `<body>`, the application should insert the following into the temporary data object,

  ```
  <html xmlns="http://www.w3.org/1999/xhtml">
      <head><title></title></head>
  ```

  then insert the content of the `notes` or `message`, and finally insert a closing `</html>`.

- Else, if the content begins with neither of the above elements, the application should insert the following into the temporary data object,

  ```
  <html xmlns="http://www.w3.org/1999/xhtml">
      <head><title></title></head>
      <body>
  ```

  then insert the content of the `notes` or `message`, and finally insert `</body></html>` to close the XHTML document.

The result of the above would be a temporary XML data object that the application could then pass to the XHTML viewing/editing API function.

G Major changes between versions of SBML Level 2 and implications for backward compatibility

In this section, we list the cumulative changes introduced in SBML Level 2 since Version 1.

G.1 Between Version 2 and Version 1

The following features were removed between SBML Level 2 Version 1 and Version 2:

- The `offset` attribute on **UnitDefinition**. (See Section 4.4.2.) The definition of offsets in SBML Level 2 Version 1 was in fact incorrect; moreover, a proper implementation would have required a complete change in the SBML unit scheme. Few models appeared to use offsets on unit definitions, so the impact of this change on models is expected to be small.
- The "`Celsius`" predefined unit. (See Section 4.4.2.) The removal of offsets on unit definitions meant an inconsistency existed if the Celsius predefined unit was left in the system. Removing Celsius removes the inconsistency. Alternative ways of using Celsius are discussed in Section 4.4.2.
- The `substanceUnit` and `timeUnits` attributes on **KineticLaw**. (See Section 4.13.5.) The ability to redefine the substance units on each reaction separately, coupled with other features in SBML Level 2 Version 1, created the opportunity for defining a valid system of reactions which potentially could not be combined into a consistent system of equations without external knowledge.

The following features were deprecated between SBML Level 2 Version 1 and Version 2:

- The `charge` attribute on **Species**. (See Section 4.8.7.) This attribute does not appear to be supported by any existing software, and moreover, since its value cannot be accessed from mathematical formulas in SBML, the impact of this change is expected to be small.

The following additional changes were made between SBML Level 2 Version 1 and Version 3:

- SBML Level 2 Version 1 did not clearly specify the value space of integer and floating-point numbers permitted in the MathML expressions in SBML; moreover, it used the XML Schema type "`integer`" instead of SBML Level 2 Version 2's "`int`". Although extremely unlikely, some previously valid SBML Level 2 Version 1 documents *may* not be valid in Version 2 and Version 3 as a result of these changes. See Sections 3.1.3, 3.1.5 and 3.4.2 for more information.
- SBML Level 2 Version 1 did not define a default value for the attribute `fast` on **Reaction**. SBML Level 2 Version 2 introduced a default value, and the value is "`false`". Further, SBML now requires that software tools *must* respect the value or indicate to the user that they do not have the capacity to do so. See Section 4.13.1.
- As of SBML Level 2 Version 2, SBML is somewhat stricter about how the content of `annotation` elements must be organized and written. Previously valid SBML Level 2 Version 1 documents *may* need changes to their `annotation` elements to comply with the specification beginning with Version 2 and Version 3. See Section 3.2.4 for more details.
- As of SBML Level 2 Version 2, SBML is slightly stricter about how the content of `notes` elements must be organized. Previously valid SBML Level 2 Version 1 documents *may* need changes to their `notes` elements to comply with the specification beginning with Version 2 and Version 3. See Section 3.2.3 for more details.
- SBML Level 2 Version 2 corrected numerous errata and ambiguities discovered in SBML Level 2 Version 1. These errata are listed on the project web site at `http://sbml.org`. As a result of changes to SBML Level 2 implied by these errata, some existing SBML Level 2 Version 1 models, even when modified as explained above, may still not be compliant with Version 2 or Version 3. The ultimate impact of the changes depends on the specific features used by a given model and the assumptions under which the model was created.

G.2 Between Version 3 and Version 2

The following features were removed between SBML Level 2 Version 2 and Version 3:

- The `spatialSizeUnits` attribute on **Species**. (See Section 4.8.5.) This attribute introduced an implicit unit conversion between the spatial units used in defining the quantity of a species and the size of the compartment in which the species was located. Moreover, the semantics of `spatialSizeUnits` were confusing and required complicated unit conversions to be written explicitly into reaction rate expressions by either the modeler or the software. Although the conversions could be worked out unambiguously, the potential for error was judged to exceed by far the utility of this feature.

- The `timeUnits` attribute **Event**. (See Section 4.14.) This attribute was judged to add needless complexity and inconsistency. For instance, the ability to change the time units of the delay of an **Event** to be different from the units of time for the whole model meant that computing an **Event**'s time of triggering and its delay might have to be done using two different sets of units. The ability to redefine the units of time for the delay of an **Event** was also inconsistent with the lack of such an attribute on other SBML components involving an element of time; for example, **RateRule**, and now **KineticLaw**, have no such attributes.

The following additional changes were made between SBML Level 2 Version 2 and Version 3:

- The definition of the XML type `ID` was incorrectly given in the SBML Level 2 Version 2 specification. This type is used as the type of the attribute `metaid` on ***SBase***. The error in the definition of `ID` was such that the type did not include the colon (`:`) character and all Unicode characters actually permitted in XML `ID`. This change is therefore entirely backward compatible: all models with valid `metaid` values valid prior to SBML Level 2 Version 3 are still valid under the new definition.

- The SBML specifications prior to SBML Level 2 Version 3 did not indicate what units are assumed for literal numbers appearing in MathML expressions (i.e., numbers inside MathML's `cn` elements). SBML Level 2 Version 3 stipulates that there are no units associated with numbers (not even "`dimensionless`"), and provides suggestions for how to associate units with numbers (Section 3.4.2).

- The SBML specifications prior to SBML Level 2 Version 3 did not make clear what units are required by the arguments to various MathML operators and other constructs. SBML Level 2 Version 3 clarifies this (Section 3.4.11).

- The `UnitKind` enumeration previously defined in the context of **Unit** and **UnitDefinition** has been eliminated in favor of simply defining the symbols as reserved words in the value space of `UnitSId`. This has no effect on written models and is completely backward compatible. It was done to resolve the problem that, previously, the values in `UnitKind` were technically inaccessible from attributes whose data type was `UnitSId`.

- The SBML specification did not point out that the value space of the data type `boolean` is different in MathML 2.0 and XML Schema 1.0. This means that the permitted values of attributes on SBML objects is different from the values permitted in MathML formulas. (Section 3.1.2 explains the difference.)

- The SBML specifications were inconsistent about the permitted number of items inside `listOf`_____ lists: the text portion of the specifications claimed the lists could have zero length, but the XML Schema definition for SBML required one or more items. As of SBML Level 2 Version 3, the specification is consistent on requiring one or more items inside these lists.

- The SBO term hierarchy (Section 5) has grown in the time intervening between SBML Level 2 Version 2 and Version 3, and the mapping of terms between SBO and SBML components was revised as a result of community discussions during the 2006 SBML Forum meeting.

- The `sboTerm` attribute, introduced on many components in SBML Level 2 Version 2, has been moved to ***SBase*** as an optional attribute and removed from the individual components. The result is that all model components may now have SBO terms associated with them. This change is completely backward compatible.

- A number of validation rules in Appendix C have been introduced; some were missing from previous specifications, and some were added to cover changes introduced in SBML Level 2 Version 3 (for example, for validation of SBO terms assigned to various SBML model components).

- The SBML specifications prior to SBML Level 2 Version 3 did not adequately explain the assumptions regarding XML namespace declarations within the `annotation` element on SBML components. SBML Level 2 Version 3 makes these assumptions more explicit, including the assumption that applications may not preserve another applications annotations unless those annotations are self-contained with the XML Namespace declaration. See Section 3.2.4 for more details.

G.3 Between Version 4 and Version 3

The following significant changes exist between SBML Level 2 Version 3 and Version 4:

- The result of an SBML community vote taken late in 2007 indicated that users and developers preferred SBML *not* to *require* consistency of units of measurement on quantities and mathematical expressions in a model. Consequently, the language and validation rules involving units in SBML Level 2 Version 4 have been changed to use the wording "should" instead of "must"—units *should* be consistent, but models that do not exhibit strict unit consistency are not invalid.

- Discussions held on the `sbml-discuss@caltech.edu` mailing list in 2007–2008 as well as the SBML Forum Meeting in 2008 (Göteborg, Sweden) made clear two points about **Event**: (1) many developers misunderstood the specification with regards to the time at which the mathematical formula in an **EventAssignment** was to be evaluated, and implemented the computation such that it was performed at the time the event was *executed* instead of (as the specification actually stipulated) the time at which the event was *fired*; and (2) once informed, most developers found the actual definition in the SBML specification counterintuitive. We believed it would have been too confusing and error-prone to change the sense of the assignments between versions of an SBML Level, so in order to address popular demand, Version 4 includes a new attribute (`useValuesFromTriggerTime`) on **Event** allowing a model to indicate which sense is intended. The default value of the `useValuesFromTriggerTime` attribute results in the same interpretation of an **Event**'s **EventAssignment**s that SBML Level 2 Version 3 specifies. There is also a new validation rule (21206) related to the new attribute, and the SBML schema presented in Appendix A includes the new `useValuesFromTriggerTime` attribute in the definition of **Event**.

- SBML Level 2 Version 3 had small syntactic errors in the RDF described in Section 6. Version 4 includes corrected RDF.

- Version 3 never made explicit the requirement that the number of arguments in a call to a user-defined function must match the number of arguments defined in the instance of the **FunctionDefinition**. SBML Level 2 Version 4 contains a new subsection (4.3.4) and a new validation rule (10219) to capture this requirement.

- The description of **Compartment**'s `outside` attribute mistakenly stated that the compartment referenced by the attribute value must be one that is already defined in the model, meaning that compartments had to be defined before being referenced. There was no validation rule to that effect, and moreover, this requirement was at odds with the general trend in later versions of SBML Level 2 to remove requirements for element ordering. In Version 4, Section 4.7.7 no longer implies a requirement for ordering.

- Prior to SBML Level 2 Version 4, it was never made clear what, if any, relationship existed between the sizes of compartments when compartments defined inside/outside relationships using the `outside` attribute. A new paragraph in Section 4.7.4 of the Version 4 specification makes clear there is no

implication on compartment sizes—the size of the outside compartment does not include the size(s) of the inside compartment(s). Moreover, additional clarifications have been added to explain that `outside` does not necessarily imply a containment relationship between compartments.

- Since the time that SBML Level 2 Version 3 Release 2 was issued, three new MIRIAM qualifiers have been defined by the MIRIAM project: `bqbiol:isEncodedBy`, `bqbiol:encodes`, and `bqbiol:occursIn`. Section 6 now includes these qualifiers.

- Since the time that SBML Level 2 Version 3 Release 2 was issued, the Systems Biology Ontology (SBO) underwent reorganization and improvement. This required numerous changes to Section 5 to be consistent with SBO now. Chief among the changes that impact SBML models are that (1) the SBO branches for **Model**, **CompartmentType**, **SpeciesType**, **Compartment** and **Species** have all changed slightly, and (2) the SBML Level 2 Version 4 specification does not *require* specific choices for `sboTerm` attribute values, only *recommends* them.

- There were some cut-and-paste errors in the text of the descriptions of the `sboTerm` attribute on several SBML components such as **Species**, **SpeciesType**, and others. The text of the Version 4 specification includes corrections for this.

- Prior Versions of SBML Level 2 used three validation rules, 20406, 20407, and 20408, to encode requirements about the predefined unit `volume`. This was inconsistent and not parallel with the way validation rules for the other predefined units such as `substance` were defined. In Version 4, the rules 20407 and 20408 are no longer defined and their content has been moved into rule 20406.

- In Section 4.1, SBML Level 3 specifications stated that "well-formed XML documents must begin with an XML declaration". In fact, this is false; the requirement is stipulated in XML 1.1, not in XML 1.0—the version of XML that SBML actually uses. However, as a concession to helping greater software portability, SBML Level 4 nonetheless requires the XML declaration to be present.

- Since elsewhere in SBML, the requirements on ordering of elements have been eliminated, the SBML Editors decided there was no point in maintaining the ordering requirement on function definitions. Therefore, in SBML Level 2 Version 4, forward references to other user-defined functions are permitted. (However, recursive or mutually recursive functions are still prohibited.) This also causes the removal of validation rule number 20302.

- Validation rules 10211, 10212, and 10708 used vague language about requirements; these have been corrected to be specific.

References

Abbott, A. (1999). Alliance of US labs plans to build map of cell signalling pathways. *Nature*, 402:219–200.

Abramowitz, M. and Stegun, I. A., editors (1977). *Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*. Dover Publications Inc.

Ausbrooks, R., Buswell, S., Carlisle, D., Dalmas, S., Devitt, S., Diaz, A., Froumentin, M., Hunter, R., Ion, P., Kohlhase, M., Miner, R., Poppelier, N., Smith, B., Soiffer, N., Sutor, R., and Watt, S. (2003). Mathematical Markup Language (MathML) Version 2.0 (second edition): W3C Recommendation 21 October 2003. Available via the World Wide Web at http://www.w3.org/TR/2003/REC-MathML2-20031021/.

Biron, P. V. and Malhotra, A. (2000). XML Schema part 2: Datatypes (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at http://www.w3.org/TR/xmlschema-2/.

Bosak, J. and Bray, T. (1999). XML and the second-generation Web. *Scientific American*, 280(5):89–93.

Bray, T., D. Hollander, D., and Layman, A. (1999). Namespaces in XML. W3C 14-January-1999. Available via the World Wide Web at http://www.w3.org/TR/1999/REC-xml-names-19990114/.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., and Maler, E. (2000). Extensible markup language (XML) 1.0 (second edition), W3C recommendation 6-October-2000. Available via the World Wide Web at http://www.w3.org/TR/1998/REC-xml-19980210.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2004). Extensible markup language (XML) 1.0 (third edition), W3C recommendation 4-February-2004. Available via the World Wide Web at http://www.w3.org/TR/2004/REC-xml-20040204.

Bureau International des Poids et Mesures (2000). The International System of Units (SI) supplement 2000: addenda and corrigenda to the 7th edition (1998). Available via the World Wide Web at http://www.bipm.fr/pdf/si-supplement2000.pdf.

Chartrand, G. (1977). *Introductory Graph Theory*. Dover Publishing, Inc., New York.

DCMI Usage Board (2005). DCMI Metadata Terms. Available online via the World Wide Web at the address http://www.dublincore.org/documents/dcmi-terms/.

Dublin Core Metadata Initiative (2005). Dublin Core metadata initiative. Available via the World Wide Web at http://dublincore.org/.

Eriksson, H.-E. and Penker, M. (1998). *UML Toolkit*. John Wiley & Sons, New York.

Fallside, D. C. (2000). XML Schema part 0: Primer (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at http://www.w3.org/TR/xmlschema-0/.

Gillespie, D. (1977). Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81:2340–2361.

Gilman, A. (2000). A letter to the signaling community. Alliance for Cellular Signaling, The University of Texas Southwestern Medical Center. Available via the World Wide Web at http://afcs.swmed.edu/afcs/Letter_to_community.htm.

Harold, E. R. and Means, E. S. (2001). *XML in a Nutshell*. O'Reilly & Associates.

Hedley, W. J., Nelson, M. R., Bullivant, D., Cuellar, A., Ge, Y., Grehlinger, M., Jim, K., Lett, S., Nickerson, D., Nielsen, P., and Yu, H. (2001). CellML specification. Available online via the World Wide Web at http://www.cellml.org/specification.

Hopcroft, J. E. and Karp, R. M. (1973). An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231.

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/Documents/Specifications.

Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C., Kitano, H., Arkin, A. P., Bornstein, B. J., Bray, D., Cornish-Bowden, A., Cuellar, A. A., Dronov, S., Gilles, E. D., Ginkel, M., Gor, V., Goryanin, I. I., Hedley, W. J., Hodgman, T. C., Hofmeyr, J.-H., Hunter, P. J., Juty, N. S., Kasberger, J. L., Kremling, A., Kummer, U., Le Novère, N., Loew, L. M., Lucio, D., Mendes, P., Minch, E., Mjolsness, E. D., Nakayama, Y., Nelson, M. R., Nielsen, P. F., Sakurada, T., Schaff, J. C., Shapiro, B. E., Shimizu, T. S., Spence, H. D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., and Wang, J. (2003). The Systems Biology Markup Language (SBML): A medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531.

Iannella, R. (2001). Representing vCard objects in RDF/XML. Available via the World Wide Web at http://www.w3.org/TR/vcard-rdf.

Jacobs, I. (2004). World Wide Web Consortium process document. Available via the World Wide Web at http://www.w3.org/2004/02/Process-20040205/.

Kokkelink, S. and Schwänzl, R. (2002). Expressing qualified Dublin Core in RDF/XML. Available via the World Wide Web at http://dublincore.org/documents/dcq-rdf-xml/index.shtml.

Lassila, O. and Swick, R. (1999). Resource description framework (RDF) model and syntax specification. Available via the World Wide Web at http://www.w3.org/TR/REC-rdf-syntax/.

Le Novère, N., Finney, A., Hucka, M., Bhalla, U., Campagne, F., Collado-Vides, J., Crampin, E. J., Halstead, M., Klipp, E., Mendes, P., Nielsen, P., Sauro, H., Shapiro, B., Snoep, J. L., Spence, H. D., and Wanner, B. L. (2005). Minimum information requested in the annotation of biochemical models (MIRIAM). *Nature Biotechnology*, 23:1509–1515.

Oestereich, B. (1999). *Developing Software with UML: Object-Oriented Analysis and Design in Practice*. Addison-Wesley Publishing Company.

Pemberton, S., Austin, D., Axelsson, J., Celik, T., Dominiak, D., Elenbaas, H., Epperson, B., Ishikawa, M., Matsui, S., McCarron, S., Navarro, Peruvemba, S., Relyea, R., Schnitzenbaumer, S., and Stark, P. (2002). XHTML™ 1.0 the Extensible HyperText Markup Language (second edition): W3C Recommendation 26 January 2000, revised 1 August 2002. Available via the World Wide Web at http://www.w3.org/TR/xhtml1/.

Popel, A. and Winslow, R. L. (1998). A letter from the directors.... Center for Computational Medicine & Biology, Johns Hopkins School of Medicine, Johns Hopkins University. Available via the World Wide Web at http://www.bme.jhu.edu/ccmb/ccmbletter.html.

Powell, A. and Johnston, P. (2003). Guidelines for implementing Dublin Core in XML. Available via the World Wide Web at http://dublincore.org/documents/dc-xml-guidelines/index.shtml.

Reder, C. (1988). Metabolic Control Theory: a structural approach. *Journal of Theoretical Biology*, 135:175–201.

Sauro, H. M. and Ingalls, B. (2003). Conservation analysis in biochemical networks: Computational issues for software writers. Available at http://www.math.uwaterloo.ca/~bingalls/Pubs/conservation.pdf.

Smaglik, P. (2000). For my next trick.... *Nature*, 407:828–829.

Thompson, H. S., Beech, D., Maloney, M., and Mendelsohn, N. (2000). XML Schema part 1: Structures (W3C candidate recommendation 24 October 2000). Available online via the World Wide Web at the address http://www.w3.org/TR/xmlschema-1/.

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

W3C (2000a). Naming and addressing: URIs, URLs, ... Available online via the World Wide Web at http://www.w3.org/Addressing/.

W3C (2000b). W3C's math home page. Available via the World Wide Web at http://www.w3.org/Math/.

W3C (2004a). RDF/XML syntax specification (revised). Available online via the World Wide Web at http://www.w3.org/TR/rdf-syntax-grammar/.

W3C (2004b). Resource description framework (RDF). Available online via the World Wide Web at the address http://www.w3.org/RDF/.

Wilkinson, D. J. (2006). *Stochastic Modelling for Systems Biology*. Chapman & Hall/CRC.

Wolf, M. and Wicksteed, C. (1998). Date and time formats. Available online via the World Wide Web at http://www.w3.org/TR/NOTE-datetime.

Zwillinger, D., editor (1996). *Standard Mathematical Tables and Formulae*. CRC Press LLC, 30th edition.