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

Andrew Finney, Michael Hucka
{afinney,mhucka}@cds.caltech.edu
Systems Biology Workbench Development Group
ERATO Kitano Symbiotic Systems Project
Control and Dynamical Systems, MC 107-81
California Institute of Technology, Pasadena, CA 91125, USA
http://www.sbml.org

Principal Investigators: John Doyle and Hiroaki Kitano

SBML Level 2, Version 1 (Final)
June 28, 2003

Contents

1 Introduction

We present the **S**ystems **B**iology **M**arkup **L**anguage (SBML) Level 2, a model representation formalism 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 an XML Schema (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000) that defines SBML Level 2. A downloadable copy of the XML Schema and other related documents and software are also available from the SBML project web site, `http://www.sbml.org/`.

Major releases of SBML are termed *levels*. SBML Level 2 evolved out of SBML Level 1 (Hucka et al., 2001, 2003). All of the structures of Level 1 can be mapped in a straightforward fashion to Level 2. In addition, a large subset of the structures in Level 2 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. Appendix B lists the differences between Level 1 and Level 2.

SBML Level 2 was created in collaboration with the authors of the following systems: *BASIS* (Kirkwood et al., 2003), *Bio Skektch Pad* (Belta et al., 2003), *BioSpreadsheet* (McCollum and Lancaster, 2003), *BioSpice* (Arkin, 2001), *CellDesigner* (Funahashi and Kitano, 2003), *Cellerator* (Shapiro et al., 2001, 2003), *COPASI* (Mendes, 2000), *DBsolve* (Goryanin, 2001; Goryanin et al., 1999), *E-CELL* (Tomita et al., 1999, 2001), *ESS* (Peterson and Drager, 2003), *Gepasi* (Mendes, 1997, 2001), *Jarnac* (Sauro, 2000; Sauro and Fell, 1991), *JDesigner* (Sauro, 2001), *JigCell* (Vass et al., 2003), *MCell* (Bartol and Stiles, 2002), *NetBuilder* (Schilstra and Bolouri, 2002), *PathScout* (Minch et al., 2003), *ProMoT/DIVA* (Stelling et al., 2001), *StochSim* (Bray et al., 2001; Morton-Firth and Bray, 1998), and *Virtual Cell* (Schaff et al., 2000, 2001). SBML Level 2 was developed with the help of these packages' authors, as well as help and collaboration from the creators of CellML (Hedley et al., 2001) and many other individuals listed in the Acknowledgments (Section 6.2).

1.1 Scope and Limitations

SBML Level 2 is meant to support basic biochemical network models and the kinds of operations that are possible in existing analysis/simulation tools. Future software tools will undoubtedly require further evolution of SBML, and we expect that higher SBML levels will add structures and facilities on top of Level 2 after the simulation community has had time to gain experience with the current language definition. In Section 6.1, we discuss extensions that will likely be included in SBML Level 3.

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.

1.2 Notational Conventions

We define SBML using a graphical notation based upon UML, the Unified Modeling Language (Eriksson and Penker, 1998; Oestereich, 1999). This UML-based definition in turn is used to define an XML Schema (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000) for SBML. There are three main advantages to using UML as a basis for defining SBML data structures. 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 visual notation is implementation-neutral: the defined structures 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 sources. Readers are therefore more likely to be familiar with it than other notations.

Our notation and our approach for mapping it to XML Schema is explained in a separate document (Hucka, 2000). Appendix A presents a summary, and examples throughout this document illustrate the approach. All data types in SBML follow XML Schema datatype definitions and conventions.

We follow certain naming and typographical conventions throughout this document. Specifically, the names of data structure attributes or fields begin with a lowercase letter, and the names of data structures and types begin with an uppercase letter. Keywords (names of types, XML elements, etc.) are written in a typewriter-style font; for example, `Compartment` is a type name and `compartment` is a field name. Likewise, literal XML examples are also written in a typewriter-style font.

2 Overview of SBML

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

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

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

Broken down into its constituents, this model contains a number of components: reactant species, product species, reactions, rate laws, and parameters in the rate laws. To analyze or simulate this network, additional components must be made explicit, including compartments for the species, and units on the various quantities. The top level of an SBML model definition simply consists of lists of these components:

beginning of model definition
- *list of function definitions (optional)*
- *list of unit definitions (optional)*
- *list of compartments (optional)*
- *list of species (optional)*
- *list of parameters (optional)*
- *list of rules (optional)*
- *list of reactions (optional)*
- *list of events (optional)*

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 name for a unit used in the expression of quantities in a model.

Compartment: A container of finite size for substances.

Species: A substance or entity that takes part in a reaction. Some example species are ions such as Ca^{2+} and molecules such as glucose or ATP.

Parameter: A quantity with a symbolic name. 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.

Rule: A mathematical expression used in combination with the differential equations constructed based on the set of reactions in a model; it can be used to establish constraints between variables, define how a variable can be calculated from other variables, or used to define the rate of change of a variable.

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 concentration, 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.

SBML allows models of arbitrary complexity to be represented. Each type of component in a model is described using a specific type of data structure that organizes the relevant information. The data structures determine how the resulting model is encoded in XML.

In the sections that follow, we describe in detail SBML's various constructs and their uses. Section 3 first introduces a few basic structures which are used throughout SBML Level 2, then Section 4 provides details on each of the main components. Section 5 provides a number of complete examples of models encoded in XML using SBML Level 2. Section 6 contains a list of anticipated enhancements that will be made in Level 3 and a discussion of other efforts related to SBML. Appendix A summarizes the UML-based notation used in this document, Appendix B describes the differences between SBML Level 1 Version 2 and SBML Level 2 as described in this document, and finally, Appendix C provides the complete XML Schema for SBML Level 2.

3 Preliminary Definitions

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

3.1 Type `SBase` and the SBML Type Inheritance Hierarchy

The base types in SBML (e.g., `integer`, `double`, and others) are taken directly from XML Schema (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000). SBML defines additional data types and structures beyond this. Every structure 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`. This base type is designed to allow a modeler or a software package to attach arbitrary information to each major structure or list in an SBML model. The definition of `SBase` is presented in Figure 1.

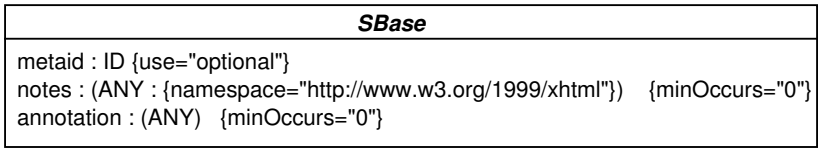


Figure 1: *The definition of `SBase`. Text enclosed in braces next to field types (e.g., {`minOccurs="0"`}) indicates constraints on the possible field values. We use the XML Schema language to express constraints because we are primarily interested in the XML encoding of SBML. The constraint expression `use="optional"` means that the indicated field is optional and may be omitted in a particular instance in a model. The constraint expression `minOccurs="0"` likewise means the indicated field is optional; this alternate form of expression must be used for those fields that are containers (i.e., fields encoded as subelements in XML).*

`SBase` contains three fields, all of which are optional: `metaid`, `notes` and `annotation`. These fields are discussed separately in the following subsections.

3.1.1 The `metaid` Field

The `metaid` field is present for supporting metadata annotations using RDF (Resource Description Format; Lassila and Swick, 1999). It has a data type of `ID` (the XML identifier type), and serves as anchors for metadata references. Metadata expressed using RDF can be placed anywhere within an `sbml` element and its subelements, *except* within MathML elements. The metadata elements can include RDF `description` elements in which the RDF `describes` attributes contain the values of the `metaid` fields of SBML elements in the model. The form of the RDF element content in SBML should follow the form described in the CellML Metadata Specification (Cuellar et al., 2002).

3.1.2 The `notes` Field

The field `notes` in `SBase` is a container for XHTML content. It is intended to serve as a place for storing optional information intended to be seen by humans. Typically, the `notes` field will contain user comments about the structure in which the `notes` field is enclosed. Every data 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. Section 5 provides examples of using `notes` in different models.

3.1.3 The `annotation` Field

`SBase` includes the field called `annotation` to provide a container for software-generated annotations that are *not* intended to be seen by humans. This field is a container for arbitrary data (XML type `any`). As with the user-visible `notes` field, every data object can have its own value for `annotation`. Section 3.2 provides guidelines for using this field.

3.1.4 The SBML Type Inheritance Hierarchy

The overall SBML inheritance hierarchy is depicted in Figure 2. In addition to the relationships shown, all substructures such as `trigger` on `Event` and the `listOf`_____ lists are also derived from `SBase`. (However, the `notes` and `annotation` elements contained inside `SBase` are not derived from `SBase`.)

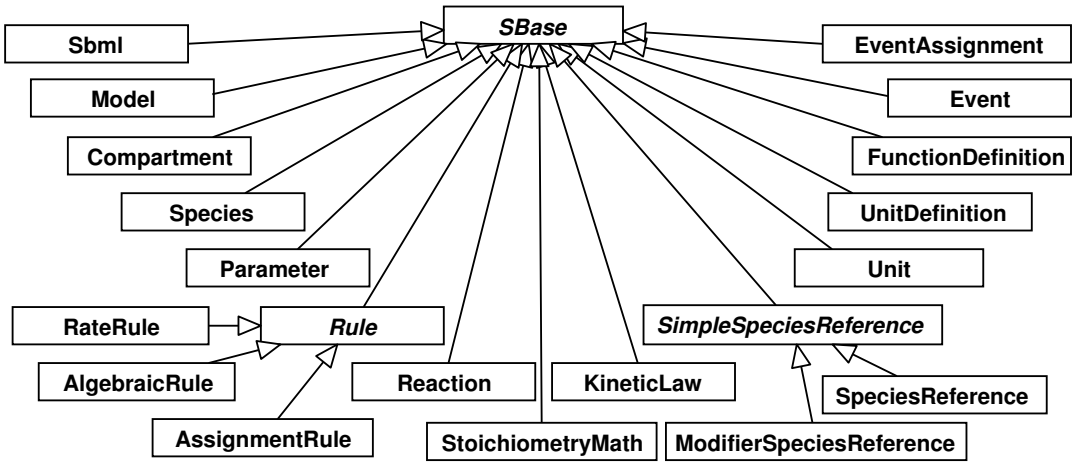


Figure 2: *A UML diagram of the inheritance hierarchy of major data types in SBML. Open arrows indicate inheritance, pointing from inheritors to their parents (Eriksson and Penker, 1998; Oestereich, 1999). In addition to these types, all substructures in SBML (including, for example, all the `listOf` lists) are also derived from `SBase`. See text for details.*

In other type definitions presented below, we follow the UML convention of hiding the attributes derived from a parent type such as `SBase`. It should be kept in mind that these attributes are always available.

3.2 Guidelines for the Use of the `annotation` Field in `SBase`

The `annotation` field in the definition of `SBase` is formally unconstrained in order that software developers may attach any information they need to the structures in an SBML model. However, it is important that this facility not be misused. In particular, it is critical that information essential to a model definition is *not* stored in `annotation`. Parameter values, functional dependencies between model structures, etc., should not be recorded as annotations.

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 essence of a model; (c) identification information for cross-referencing components in a model with items in a database.

Different applications may use XML Namespaces (Bray et al., 1999) to specify the intended vocabulary of a particular annotation. Here is an example. Suppose a particular application needs to annotate data structures in an SBML model definition with screen layout information and a time stamp. The application's developers should choose a URI (*Universal Resource Identifier*; Harold and Means 2001; W3C 2000a) reference that uniquely identifies the vocabulary the application will use for such annotations, and a prefix string for the annotations. For illustration purposes, let us say the URI reference is "`http://www.mysim.org/ns`" and the chosen prefix is `mysim`. An example of an annotation might then be as follows:

```
...
<annotation xmlns:mysim="http://www.mysim.org/ns">
    <mysim:nodecolors mysim:bgcolor="green" mysim:fgcolor="white"/>
    <mysim:timestamp>2000-12-18 18:31 PST</mysim:timestamp>
</annotation>
...
```

The namespace prefix `mysim` is used to qualify the XML elements `mysim:nodecolors` and `mysim:timestamp`; presumably these symbols have meaning to the application. This example places the XML Namespace information on `annotation` itself rather than on a higher-level enclosing construct or the enclosing document level, but other placements would be valid as well (Bray et al., 1999).

The use of XML Namespaces permits multiple applications to place annotations on XML elements of a model without risking interference or element name collisions. Annotations stored by different simulation packages can thus coexist in the same model definition. Although XML Namespace names must be URI references, an XML Namespace name is *not required* to be directly usable in the sense of identifying an actual, retrieval document or resource on the Internet (Bray et al., 1999). "`http://www.mysim.org/`" is a namespace name or URI in the example above. The name is simply intended to enable unique identification of constructs, and using URIs is a common and simple way of creating a unique name string. For the convenience of developers of simulation and analysis tools, we reserve certain namespace names for use with annotations in SBML. These reserved names are listed in Table 1.

`http://www.sbml.org/2001/ns/basis`	`http://www.sbml.org/2001/ns/jdesigner`
`http://www.sbml.org/2001/ns/biocharon`	`http://www.sbml.org/2001/ns/jigcell`
`http://www.sbml.org/2001/ns/bioreactor`	`http://www.sbml.org/2001/ns/jsim`
`http://www.sbml.org/2001/ns/biosketchpad`	`http://www.sbml.org/2001/ns/libsbml`
`http://www.sbml.org/2001/ns/biospreadsheet`	`http://www.sbml.org/2001/ns/mathsbml`
`http://www.sbml.org/2001/ns/biospice`	`http://www.sbml.org/2001/ns/mcell`
`http://www.sbml.org/2001/ns/celldesigner`	`http://www.sbml.org/2001/ns/monod`
`http://www.sbml.org/2001/ns/cellerator`	`http://www.sbml.org/2001/ns/netbuilder`
`http://www.sbml.org/2001/ns/copasi`	`http://www.sbml.org/2001/ns/pathdb`
`http://www.sbml.org/2001/ns/cytoscape`	`http://www.sbml.org/2001/ns/promot`
`http://www.sbml.org/2001/ns/dbsolve`	`http://www.sbml.org/2001/ns/sbedit`
`http://www.sbml.org/2001/ns/ecell`	`http://www.sbml.org/2001/ns/sigpath`
`http://www.sbml.org/2001/ns/gepasi`	`http://www.sbml.org/2001/ns/stochsim`
`http://www.sbml.org/2001/ns/isys`	`http://www.sbml.org/2001/ns/vcell`
`http://www.sbml.org/2001/ns/jarnac`	`http://www.sbml.org/2001/ns/winscamp`

Table 1: *Reserved XML Namespace names in SBML Level 2.*

Note that the namespaces being referred to here are XML Namespaces specifically in the context of the `annotation` field on `SBase`. The namespace issue here is unrelated to the namespaces discussed in Section 3.5 in the context of `SId` and symbols in SBML.

3.3 The `id` and `name` Fields on SBML Components

As will become apparent below, most structures in SBML include two common fields: `id` and `name`. The `id` field is usually required for most structures and is used to identify a component within the model definition.

Other SBML structures can refer to the component using this identifier. Section 3.4 provides a definition of the data type SId used for the id field, and Section 3.5 describes the scoping and namespace rules for these identifiers.

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 including the identifiers of unit definitions.

In contrast to the id field, the name field 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 the name field is the type string defined in XML Schema (Biron and Malhotra, 2000; Thompson et al., 2000). This type includes 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, Section 2.4) must be obeyed:

- The ampersand (&) character must be escaped using the entity &.
- The apostrophe (") and single-quote character (') must be escaped using the entities ' and ", 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. SBML imposes no restrictions as to the content of name fields 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 fields, it should display this content to the user as a component's label instead of the component's id field. 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 field is missing on a given component, then the user interface should display the value of the id field instead. (Script language interpreters are especially likely to display id fields instead of name fields.)

As a consequence of the above, authors of systems that automatically generate the values of id fields 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 reasonably easy for humans to type and read.

An additional point worth mentioning is although there are restrictions on the uniqueness of id values (see Section 3.5 below), there are no restrictions on the uniqueness of name values in a model. This allows a software package more leeway in assigning component identifiers. For example, a species in an SBML model must be located in a compartment, which means that if the same species appears in multiple compartments (e.g., in the context of a transport reaction), they must be given different identifiers. It is currently the case that users and software differ sharply in philosophy about how to treat this situation: some treat these as different species, and others treat them as the same species located in different places. Those in the latter group often want to use the same name but have different id values for the differently-localized "instances" of the species. The freedom from restrictions on name values enables SBML to accommodate both philosophies.

3.4 Type SId

The type SId is the type of the id field found on the majority of SBML components. SId is a data type derived from the basic XML type string, but with restrictions about the types of characters permitted and the sequence in which they may appear. Its definition is shown in Figure 3 on the next page.

The SId is purposefully not derived from the XML ID type. 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 extensions for supporting model/submodel composition. Further, the use of the ID type for SBML identifiers would have limited utility because MathML ci elements are not of the type IDREF (see Section 3.6). Since the IDREF-ID linkage cannot be exploited in MathML constructs, the utility of the XML ID type is greatly reduced.

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

Figure 3: *The definition of the type `SId` expressed in the variant of BNF used by the XML 1.0 specification (Bray et al., 2000). The characters (and) are used for grouping, and the character * indicates "zero or more times".*

3.5 Component Identifiers and Namespaces in SBML

A biochemical network model can contain a large number of components representing different parts of a model. This leads to a problem in deciding the scope of an identifer: 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 namespace, so that an identifier X represents the same thing wherever it appears in a given model definition. The *local* approach places symbols in different 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 identifer 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 identifers and namespaces.

The namespace 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 identifiers (i.e., the values of the field `id`) of functions, compartments, species, reactions, events and model-level parameters reside in the same global namespace. This means, for example, that a reaction and a species definition cannot both have the same identifier.
- Each reaction definition (see Section 4.9) establishes a private local namespace for local parameter identifiers. Within the definition of a given reaction, local parameter identifiers introduced in that reaction override (shadow) identical identifers in the global namespace.
- Unit identifiers (the values of the field `id` in the `UnitDefinition` structure) exist in a separate global namespace distinct from other identifiers.

The set of rules above can enable software packages using either local or global namespaces for parameters to exchange SBML model definitions. In particular, software environments using local namespaces for parameters internally should be able to accept SBML model definitions without needing to change component identifiers. Environments using a global namespace for parameters internally can perform a simple manipulation of the identifiers of local parameter elements within reaction definitions to avoid name collisions. (An example approach for the latter would be the following: when receiving an SBML-encoded model, prefix each parameter identifier inside each reaction with a string constructed from the reaction's identifier; when writing an SBML-encoded model, strip off the prefix.)

The namespace rules described here will hopefully provide a clean transition path to future levels of SBML, when submodels are introduced (Section 6.1). 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 namespaces within a submodel can simply be the Level 2 namespace rule described here, with each submodel having its own (to itself, global) namespace.

3.6 Mathematical Formulas in SBML Level 2

Mathematical expressions in SBML Level 2 are represented using MathML 2.0 (W3C, 2000b), the XML standard for describing mathematics in machine-readable format. It is used in the definitions of functions (Section 4.3), rules (Section 4.8), reaction kinetics (Section 4.9.7), stoichiometries (Section 4.9.5) and events (Section 4.10). The `KineticLaw`, `StoichiometryMath`, `EventAssignment` and `Rule` structures each have a single MathML `math` subelement. A function definition has a single `lambda` subelement. The `Event` structure has two math fields, `trigger` and `delay` each containing a single MathML `math` element.

The XML namespace URI for all MathML elements is "`http://www.w3.org/1998/Math/MathML`". [See the W3C document by Bray et al. (1999) for more information about using XML namespaces.] The examples in Section 5 illustrate the use of this namespace and MathML in SBML.

3.6.1 Subset of MathML Used in SBML Level 2

The subset of MathML elements used in SBML Level 2 is similar to that used by CellML and is itemized below:

- *token*: `cn`, `ci`, `csymbol`, `sep`
- *basic content*: `apply`, `piecewise`, `piece`, `otherwise`
- *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. Elements for representing partial differential calculus are not included. We anticipate that the requirements for partial differential calculus will be addressed in proposals for SBML Level 3 geometry representations (see Section 6.1).

The following are the only attributes permitted on MathML elements in SBML:

- `style`, `class` and `id` on any element;
- `encoding` and `definitionURL` on `csymbol` 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.

3.6.2 Use of `cn` Elements in MathML Expressions in SBML

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`".

3.6.3 Use of `ci` Elements in MathML Expressions in SBML

The content of a `ci` element must obey MathML whitespace rules and contain an identifier that is declared elsewhere in the model. The set of possible identifiers that can appear in a `ci` element depends on the containing structure in which the `ci` is used:

- If `ci` appears in the body of a function definition, the referenced identifier must be either (i) one of the declared arguments to the function, or (ii) the identifier of a previously defined function.
- In all other situations, the referenced identifier must be the identifier of a species, compartment, parameter or function declared 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 (*amount of substance* or *concentration*) of that species. The units associated with a species identifier are *the units of the species*, defined in Section 4.6.4.
  - *Compartment identifier*: When a compartment identifier occurs in a `ci` element, it represents the size of the compartment. The units associated with the size of the compartment are those given on the `Compartment` structure that declares the identifier; see Section 4.5.4.
  - *Parameter identifier*: When a parameter identifier occurs in a `ci` element, it represents the value assigned to that parameter. The units associated with the parameter value are the units assigned in its instance of a `Parameter` structure; see Section 4.7.3.
  - *Function identifier*: When a function 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.

3.6.4 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` field of `csymbol` should be set to `text`. The `definitionURL` should be set to one of the following predefined SBML symbol URLs:

- `http://www.sbml.org/sbml/symbols/time`. This represents the current simulation time. The units of the current time entity are determined from the built-in `time` of Table 3 on page 15.
- `http://www.sbml.org/sbml/symbols/delay`. This represents a delay function. The delay function has the form $delay(x, d)$, taking two arguments. Its value is the value of argument x at d time units before the current time. The units of the d parameter are determined from the built-in `time`. 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.

The following examples demonstrate these concepts. The XML fragment below encodes the formula $x + t$, where t is the built-in symbol 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>
```

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 it is not necessary for a parser to access the resource pointed to by the "`definitionURL:`" in this context, the URL should be interpreted as a URI. Also, the content of the `csymbol` element is for rendering purposes only and can be ignored by the parser.

4 SBML Components

In this section, we define each of the major data structures in SBML. To provide illustrations of their use, we give partial model definitions in XML. Section 5 provides many full examples of SBML in XML.

4.1 The SBML Container

The outermost portion of an SBML Level 2 model definition consists of a single `Sbml` structure enclosing a single `Model` structure (see next Section). The definition of `Sbml` is shown in Figure 4.

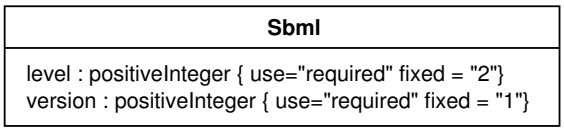


Figure 4: *The definition of `Sbml` for SBML Level 2 Version 1. Following UML notation, additional fields that are inherited from a base class, in this case `SBase`, are not shown.*

The XML namespace URI for SBML Level 2 is "`http://www.sbml.org/sbml/level2`". All SBML Level 2 elements should be encoded using this URI by assigning this URI to either the default namespace or a tag prefix. The character encoding for SBML is UTF-8. SBML documents should include the `encoding` attribute with the value `UTF-8` in the XML prologue.

In the transformation of UML to XML used in this document, the `Sbml` structure is turned into an element named `sbml`. The element has two required attributes: `level` and `version`. For SBML Level 2 Version 1, these attributes must be set to "2" and "1", respectively. (The `version` attribute is present in case SBML Level 2 must be revised in the future to correct errors.)

The following is an abbreviated example of the outermost content of an SBML model definition in XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  ...
</sbml>
```

4.2 Models

The `Model` structure is the highest-level construct in an SBML data stream or document. Its definition is shown in Figure 5 on the next page. Only one component of type `Model` is allowed per instance of an SBML document or data stream, although it does not necessarily need to represent a single biological entity.

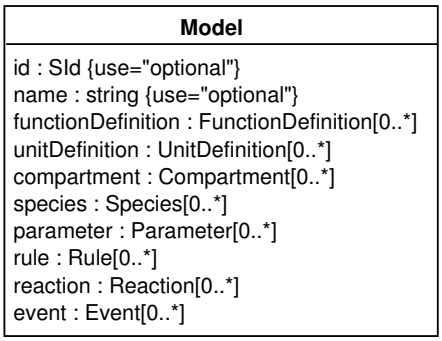


Figure 5: *The definition of* `Model`*. Following UML notation, additional fields that are inherited from a base class, in this case* `SBase`*, are not shown.*

`Model` serves as a container for `FunctionDefinition`, `UnitDefinition`, `Compartment`, `Species`, `Parameter`, `Rule`, `Reaction` and `Event` components. All of these components are optional; that is, the lists in each of the respective fields are permitted to have zero length. (However, there are dependencies between components, such that defining some requires defining others. For example, as explained in other sections below, defining a species requires defining a compartment, and defining a reaction requires defining a species.)

The `Model` structure has an optional field, `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 3.4. `Model` also has an optional `name` field, of type `string`. The `name` and `id` fields should be used as described in Section 3.3.

In the XML encoding of an SBML model, the lists of species, compartments, unit definitions, parameters, reactions, function definitions, rules and events are translated into lists of XML elements enclosed within elements of the form `listOf_____s`, where the blank is replaced by the name of the component type (e.g., "`Reaction`"). The resulting XML data object has the form illustrated by the following skeletal model:

```
<model id="My_Model">
    <listOfFunctionDefinitions>
        ...
    </listOfFunctionDefintions>
    <listOfUnitDefinitions>
        ...
    </listOfUnitDefinitions>
    <listOfCompartments>
        ...
    </listOfCompartments>
    <listOfSpecies>
        ...
    </listOfSpecies>
    <listOfParameters>
        ...
    </listOfParameters>
    <listOfRules>
        ...
    </listOfRules>
    <listOfReactions>
        ...
    </listOfReactions>
    <listOfEvents>
        ...
    </listOfEvents>
</model>
```

Readers may wonder about the motivations for the `listOf_____s` notation. A simpler approach to creating the lists of components would be to place them all directly at the top level under `<model> ... </model>`. We chose instead to group them within XML elements named after `listOf_____s`, because we believe this helps organize the components and makes visual reading of model definitions easier. These `listOf_____s` elements are derived from `SBase` which enables each list to contain its own `metaid`, `notes` and `annotation` fields. Further details of how `listOf_____s` elements implement UML lists is described in Appendix A.

4.3 Function Definitions

The `FunctionDefinition` structure associates an identifier with a function definition. The identifier can then be used in any subsequent MathML `apply` elements. `FunctionDefinition` is shown in Figure 6.

Figure 6: *The definition of `FunctionDefinition`. Following UML notation, additional fields that are inherited from a base class, in this case `SBase`, are not shown.*

The `FunctionDefinition` structure has three fields: `id`, `name` and `math`. Their purposes are explained in the following subsections.

4.3.1 The `id` and `name` Fields

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

4.3.2 The `math` Field

The `math` field is a container for MathML content that defines the function. The content of this field can only be a MathML `lambda` element. This is the only place in SBML where a `lambda` element can be used. The function is only available for use in other MathML elements that follow the `FunctionDefinition` structure in an SBML model. (These restrictions prevent recursive and mutually-recursive functions from being expressed.) The `lambda` element can contain any of the elements in the MathML subset listed in Section 3.6.1 but not any further `lamdba` elements.

The following abbreviated SBML example shows a `FunctionDefinition` structure defining $pow3(x)$ as representing x^3:

```
<model>
    ...
    <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>
    ...
</model>
```

4.4 Unit Definitions

Units may be supplied in a number of contexts in an SBML model. The units of the following mathematical entities can be specified explicitly: constants, initial conditions, symbols in formulae and the results of formulae. Rather than having to give a complete unit definition on every structure, SBML provides a facility for defining identified units which can be reused throughout a model. In addition, by default, SBML mathematical entities have units composed from built-in units in a consistent fashion (see Sections 4.4.3,

4.5.4, 4.6.4 and 4.9.7). By redefining the built-in units it is possible to change the units used throughout a model in a simple and consistent manner.

The SBML `UnitDefinition` and `Unit` structures enable combinations of units to be given abbreviated names, and enable built-in units to be redefined. The definitions of `UnitDefinition` and `Unit` are shown in Figure 7.

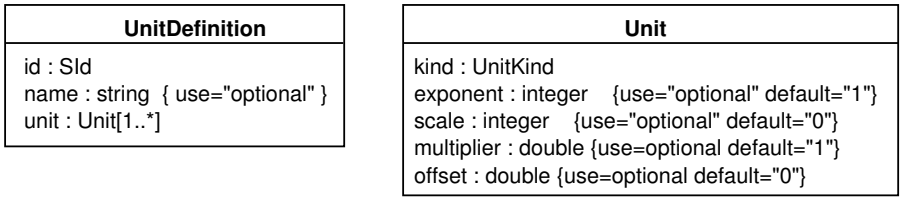


Figure 7: *The definition of `UnitDefinition` and `Unit`. Following UML notation, additional fields that are inherited from a base class, in this case `SBase`, are not shown.*

4.4.1 The `UnitDefinition` Structure

An instance of a `UnitDefinition` consists of an `id` field of type `SId`, an optional string field `name` and a list of structures of type `Unit`. As mentioned in Section 3.5, unit identifiers defined by the `id` field are considered to be in a separate global namespace distinct from the namespace of other identifiers in a model; thus, unit identifiers cannot collide with the identifiers of species, compartments, reactions, etc.

The approach to defining units in SBML is compositional; for example, $meter\ second^{-2}$ is constructed by combining, within the same `UnitDefinition` `Unit` list, a `Unit` structure representing *meter* with a `Unit` structure representing $second^{-2}$. The `Unit` structure is described in the next subsection.

4.4.2 The `Unit` Structure

The `Unit` data structure has one required field, `kind`, whose value must be taken from `UnitKind`, an enumeration of base units. The possible values of `UnitKind` are given in Table 2.

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

Table 2: *The possible values of `kind` in a `UnitKind` structure. 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 does not have units, 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 so commonly used in SBML's areas of application that they are included as predefined unit names. (The standard SI unit of mass is in fact the kilogram, and volume is defined in terms of cubic meters.)*

Note that the set of acceptable values for the field `kind` does not include units defined by `UnitDefinition` structures. This means that the units definition feature in SBML is not hierarchial—user-defined units cannot be built on top of other user-defined units, only on top of base units. (SBML differs from CellML in this respect; CellML allows the construction of hierarchial unit definitions.)

A `Unit` structure represents a (possibly transformed) reference to a base unit chosen from `UnitKind`. The formula for a single transformation represented by a `Unit` structure is as follows (where u is the original base unit and u_{new} is the new unit):

$$u_{new} = (multiplier \times 10^{scale} \times u^{exponent}) + offset$$

The optional `exponent` field on `Unit` represents an exponent on the unit. Its default value is "`1`" (one). For the example mentioned at the beginning of this section, $second^{-2}$ would be obtained by using `kind="second"` and `exponent="-2"`. A `Unit` structure also has an optional `scale` field; 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 $10^{-3} * gram$, or milligrams. The default value of `scale` is "`0`" (zero), because $10^0 = 1$.

The optional `multiplier` field 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` field has a default value of "`1`" (one). Finally, the `offset` field is used to represent the addition of a constant in the transformation of the `kind` unit. For example, an `offset` value of "`32.0`" would be needed to define Fahrenheit in terms of degrees Celsius. The `offset` field has a default value of "`0`" (zero).

The composition of n `Unit` structures within a `UnitDefinition` to create more complex units involves a linear product according to the following formula:

$$u_{new} = m_1 \times \ldots \times m_n \times 10^{s_1} \times \ldots \times 10^{s_n} \times u^{e_1} \times \ldots \times u^{e_n}$$

The following example illustrates the definition of an abbreviation named "`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>
```

The following example defines Fahrenheit:

```
<unitDefinition id="Fahrenheit">
     <listOfUnits>
         <unit kind="Celsius" multiplier="1.8" offset="32"/>
     </listOfUnits>
</unitDefinition>
```

4.4.3 Built-in Units

There are five special unit names in SBML, listed in Table 3, corresponding to the five types of quantities or *built-in units* that play roles in biochemical reactions: amount of substance, volume, area, length and time. All SBML mathematical entities apart from parameters have default units. These default units are composed from the set of *built-in units*. Further SBML defines defaults for the built-in units, listed in the third column of Table 3, all with default `scale` and `offset` values of zero and a default `multiplier` value of one.

Name	Possible Scalable Units	Default Units
`substance`	mole, item	mole
`volume`	litre, cubic metre	litre
`area`	square metre	square metre
`length`	metre	metre
`time`	second	second

Table 3: *SBML's built-in units.*

A field that defines the units for a mathematical entity (e.g., the field `units` on `Parameter`) can refer to a named unit chosen from among the following:

- The predefined units from Table 2 on page 14,
- New units defined in unit definitions, and
- The five predefined units "substance", "volume", "area", "length", and "time" from Table 3.

Within certain limits, a model may change the built-in units by reassigning the keywords "substance", "length", "area", "time", and "volume" in a UnitDefinition. The second column in Table 3 lists the set of units that should be used in redefining a given built-in unit.

The following example illustrates how to change the built-in units of volume to be milliliters. 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 milliliters.

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

Software developers are asked to pay special attention to the units used in an SBML model. Different users and developers sometimes make different assumptions about units, and these assumptions may not correspond to what is defined in SBML. Sections 3.6.3, 4.6.4 and 4.9.7 have particularly important notes about the usage of units in SBML.

4.5 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 cell, although models are often designed that way. The definition of Compartment is shown in Figure 8.

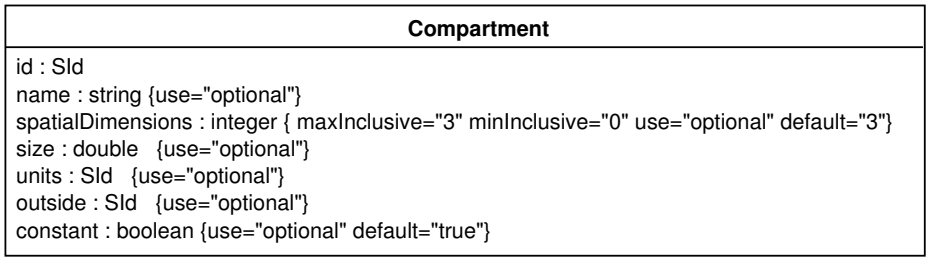


Figure 8: *The definition of Compartment. Following UML notation, additional fields that are inherited from a base class, in this case SBase, are not shown.*

It is worth pointing out 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 declares any species, the model must also declare at least one compartment.

4.5.1 The id and name Fields

Compartment has one required field, 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 field of type string. Identifiers and names should be used according to the guidelines described in Section 3.3.

4.5.2 The `spatialDimensions` Field

A `Compartment` structure has an optional field `spatialDimensions`, whose value must be a positive integer indicating the number of spatial dimensions possessed by the compartment. The maximum value of `spatialDimensions` 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`".

4.5.3 The `size` Field

Each compartment has an optional floating-point field named `size`, representing the total size of the compartment. The `size` field enables concentrations of species to be calculated in the absence of geometry information. Note in particular that in SBML Level 2, a missing `size` value does *not* imply that the compartment size is 1. (This is unlike the definition of compartment `volume` in SBML Level 1.) The `size` field must not be present if the `spatialDimensions` field is has a value of "`0`". When the `spatialDimensions` field does not have a value of "`0`", a missing value for `size` for a given compartment signifies that the value is either unknown, is determined by an assignment rule, not required for analysis or available from an external data source.

4.5.4 The `units` Field

The units associated with the compartment's `size` value may be explicitly set using the optional field `units`. The value chosen for this field must be either one of the base units from Table 2 on page 14, or the built-in units "`volume`", "`area`", "`length`" or "`dimensionless`", or a new unit defined by a unit definition in the enclosing model. The type of units assigned to the `units` field must also agree with the number of spatial dimensions of the compartment; that is, they must be units of volume if the value of `spatialDimensions` is "`3`"; they must be units of area if the value of `spatialDimensions` is "`2`"; they must be units of length if the value of `spatialDimensions` is "`1`"; and they must be "`dimensionless`" if the value of `spatialDimensions` is "`0`". The default units depends on the value of the compartment's `spatialDimensions` field according to the following rule: for spatial dimensions of 3, 2, 1 or 0, the compartment has the default units of `volume`, `area`, `length` and `dimensionless`, respectively. (See Table 3 on page 15 and Table 2 on page 14.)

The units of the compartment size, as defined by the `units` field, are used in the following ways:

- The value of the `units` field is used as the units of the `size` field of the compartment structure (see Section 4.5.3).
- The value of the `units` field is used as the units of the compartment identifier when it appears as a numerical quantity in a mathematical formula expressed in MathML (discussed in Section 3.6.3).
- The value of the `units` field is used as the units of the `math` field of an `AssignmentRule` structure determining the compartment's size (see Section 4.8.2).
- In a `RateRule` structure that sets the rate of change of the compartment's size (Section 4.8.3), the units on the rule's `math` field are those in the compartment's `units` field divided by the default *time* units. (In other words, the units for the rate of change of compartment size are *compartment size*/*time* units.)

4.5.5 The `constant` Field

A `Compartment` also has an optional boolean field 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 determined by rules (see Section 4.8), and the value of the `size` field should be taken as being the initial size of the compartment. The default value for the `constant` field is "`true`" because in the most common modeling scenarios at the time of this writing, compartment sizes remain constant. The `constant` field must default to or be set to "`true`" if the `spatialDimensions` field is 0.

4.5.6 The `outside` Field

The optional field `outside` of type `SId` can be used to express containment relationships between compartments. If present, the value of `outside` for a given compartment must be the name of another compartment enclosing it, or in other words, the compartment that 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).

Although containment 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 6.1, we expect that SBML Level 3 will introduce the ability to define geometries and spatial qualities.)

4.5.7 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 named 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>
```

4.6 Species

The term *species* refers to chemical entities that take part in reactions. 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 `Species` data structure is intended to represent these entities. Its definition is shown in Figure 9 on the following page.

4.6.1 The `id` and `name` Fields

As with other major structures in SBML, `Species` has a mandatory field, `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 3.4. `Species` also has an optional `name` field, of type `string`. The `name` and `id` fields should be used as described in Section 3.3.

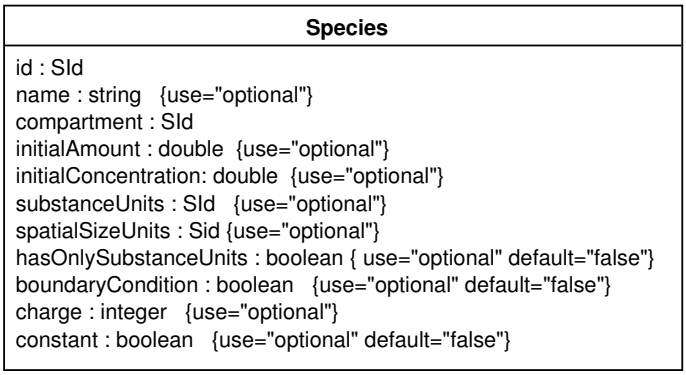


Figure 9: *The definition of* `Species`*. Following UML notation, additional fields that are inherited from a base class, in this case* `SBase`*, are not shown.*

4.6.2 The `compartment` Field

The required field `compartment`, also of type `SId`, is used to identify the compartment in which the species is located. The field's value must be the identifier of an existing `Compartment` structure. It is important to note that there is no default value for the `compartment` field 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.

4.6.3 The `initialAmount` and `initialConcentration` Fields

The optional fields `initialAmount` and `initialConcentration`, both having a data type of `double`, are used to set the initial quantity of the species in the named compartment. These fields are mutually exclusive; i.e., *only one* can have a value on any given instance of a `Species` structure. Also, `initialConcentration` must not have a value if the species' compartment has a `spatialDimensions` value of "0" or if the value of the species' `hasOnlySubstanceUnits` field is "`true`". Missing `initialAmount` or `initialConcentration` values implies that their values are either unknown, set by an assignment rule, not required for analysis or available from an external data source.

The units of the value in the `initialAmount` field is that given by the `substanceUnits` field of the species structure. The units of the value in the `initialConcentration` field are the *units of the species* as described in the next subsection.

4.6.4 The `substanceUnits`, `spatialSizeUnits` and `hasOnlySubstanceUnits` Fields

The units associated with a species' quantity, referred to as the *units of the species*, are determined via the optional fields `substanceUnits`, `spatialSizeUnits` and `hasOnlySubstanceUnits`.

`hasOnlySubstanceUnits` is a boolean field which defaults to "`false`". 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 `spatialDimensions` is zero or `hasOnlySubstanceUnits` has the value "`true`". The units of *substance* are those defined in the `substanceUnits`, and the *size* units are those given in the `spatialSizeUnits` field.

For both `substanceUnits` and `spatialSizeUnits`, the value chosen must be either a base unit from Table 2 on page 14, a built-in unit from Table 3 on page 15, or a new unit defined by a unit definition in the enclosing model. The chosen units for `substanceUnits` must be a variant of `mole` or `item` units. The `substanceUnits` field defaults to the the built-in unit "`substance`" shown in Table 3 on page 15.

The type of units assigned to the `spatialSizeUnits` field must agree with the number of spatial dimensions of the species' compartment. Specifically, they must be units of volume if the value of the com-

partment's `spatialDimensions` is "3"; they must be units of area if the value of `spatialDimensions` is "2"; and they must be units of length if the value of `spatialDimensions` is "1". The `spatialSizeUnits` must not have a value if `spatialDimensions` on the compartment has a value of "0", or if the species' `hasOnlySubstanceUnits` field has a value of "`true`". The default value of the `spatialSizeUnits` is the value of the `units` field of the species' compartment.

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

- The species `initialConcentration` field has these units (see Section 4.6.3).
- The species identifier has these units when it appears as a numerical quantity in a mathematical formula expressed in MathML (discussed in Section 3.6.3).
- The `math` field of an `AssignmentRule` structure determining the species' quantity (see Section 4.8.2) has these units.
- In `RateRule` structures that set the rate of change of the species' quantity (Section 4.8.3), the units on the rule's `math` field are the *units of the species* divided by the built-in *time* units.

4.6.5 The `constant` and `boundaryCondition` Fields

The `Species` structure has an optional boolean field named `constant` used to indicate whether the concentration of that species can vary during a simulation. The default value is "`false`", indicating that the species' concentration can be determined by reactions and rules.

Another optional field defined for `Species` is `boundaryCondition`. By default, when a species is a product or reactant of one or more reactions, its concentration is determined by those reactions. In SBML, it is possible to indicate that a given species' concentration is *not* determined 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 but is a component of the rest of the model. The boolean field `boundaryCondition` can be used to indicate this. The value of the field defaults to "`false`", indicating the species *is* part of the reaction system. Table 4 shows how to interpret the combined values of the `boundaryCondition` and `constant` fields. In practice, a `boundaryCondition` value of "`true`" means a differential equation derived from the reaction definitions should not be generated for the species. The example model in section 5.5 contains all four possible combinations of the `boundaryCondition` and `constant` fields on `species` elements. Section 5.6 contains a translation into ODEs of a model which uses `boundaryCondition` and `constant` fields.

`constant` value	`boundaryCondition` value	can have assignment or rate rule	can be reactant or product	concentration is changed by
true	true	no	yes	never changes
false	true	yes	yes	rule
true	false	no	no	never changes
false	false	yes	yes	reactions or rule but not both

Table 4: *How to interpret the values of the `constant` and `boundaryCondition` fields of the `Species` structure.*

4.6.6 The `charge` Field

The optional field `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+}).

4.6.7 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>
```

4.7 Parameters

A `Parameter` structure is used to declare a variable for use in mathematical formulae in an SBML model definition. By default, parameters have constant value for the duration of a simulation and for this reason are called "parameters" instead of variables in SBML. The definition of `Parameter` is shown in Figure 10.

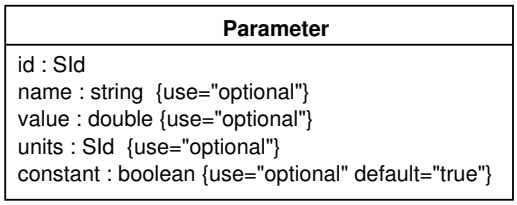


Figure 10: *The definition of* `Parameter`*. Following UML notation, additional fields that are inherited from a base class, in this case* `SBase`*, are not shown.*

4.7.1 The `id` and `name` Fields

`Parameter` has one required field, `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` field of type `string`. Identifiers and names should be used according to the guidelines described in Section 3.3.

4.7.2 The `value` Field

The optional field `value` determines the value (of type `double`) assigned to the identifer. A missing `value` implies that the `value` is either unknown, determined by an assignment rule, not required for analysis or available from an external data source. If the parameter is not constant then the `value` field contains the initial value.

4.7.3 The `units` Field

The units associated with the value of the parameter are specified by the field `units`. These units are used when the parameter identifier appears in MathML expressions and in `AssignmentRule` structures setting the value of the parameter. A `RateRule` structure that may determine the value of the parameter has units *parameter units*/*time*, where *parameter units* are the units assigned to the parameter and *time* is the built-in `time` units. The value assigned to the parameter's `units` field must be chosen from one of the following possibilities: one of the base unit names from Table 2 on page 14; one of the built-in unit names appearing in first column of Table 3 on page 15; or the name of a new unit defined in the list of unit definitions in the enclosing `Model` structure. There are no constraints on which units can be chosen from these sets. There are no default units for parameters.

4.7.4 The `constant` Field

The `Parameter` structure has an optional boolean field named `constant` which indicates whether the parameter's value can vary during a simulation. The field's default value is "`true`"; a value of "`false`" indicates the parameter's value can be changed by rules (see Section 4.8) and the `value` is actually intended to be the initial value of the parameter.

4.7.5 Local and Global Parameters

Parameters can be defined in two places in SBML: in lists of parameters defined at the top level in a `Model` structure and within individual reaction definitions (as described in Section 4.9). 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 names (See Section 3.5 for further details). Parameters local to a reaction cannot be changed by rules and therefore are implicitly always constant; thus, parameter definitions within `Reaction` structures should not have their `constant` field set.

4.7.6 Example

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

```
<model>
    ...
    <listOfParameters>
        <parameter id="tau1" value="2.3" units="second"/>
        <parameter id="Km1" value="10.7" units="moleperliter"/>
    </listOfParameters>
    ...
</model>
```

4.8 Rules

Rules provide a way to create constraints on variables for cases in which the constraints cannot be expressed using reactions (Section 4.9) nor the assignment of an initial value to a component in a model. There are two orthogonal dimensions by which rules can be described. First, there are three different possible functional forms, corresponding to the following three general cases (where x is a variable, f is some arbitrary function, V is a vector of variables that does not include x, and W is a vector of variables that may include x):

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

The second dimension concerns the role of variable x in the equations above: x can be the identifier of a compartment (to set its size), a species (to set its concentration), or a parameter (to set its value).

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;
- Some simulators do not contain numerical solvers capable of solving unconstrained *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` structure containing only one field, `math`, to hold the right-hand side expression, then to derive subtypes of `Rule` that add fields to distinguish the cases of algebraic, assignment and rate rules. Figure 11 on the next page 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.

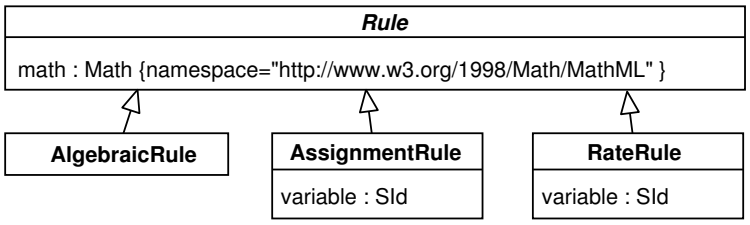


Figure 11: *The definition of* `Rule` *and derived types. Following UML notation fields that are inherited from a base class are not shown.*

4.8.1 `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 fields to the basic `Rule`; its role is simply to distinguish this case from the other cases. An example of the use of `AlgebraicRule` structures is given in Section 5.4.

4.8.2 `AssignmentRule`

The rule type `AssignmentRule` is used to express equations that set the values of variables. The left-hand side (the `variable` field) of an assignment rule can refer to the identifier of a species, compartment, or parameter. Two or more `RateRule` or `AssignmentRule` structures cannot have the same left-hand side or `variable` field value in an SBML model definition. In all cases, as would be expected, the units of the formula representing the right hand side, the `math` field, are identical to the units associated with the left hand side, the `variable` field, when that variable appears in other formulae.

The effects of an `AssignmentRule` structure 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 are the *units of the species* as defined in Section 4.6.4.

  Restrictions: In a given SBML Level 2 model, there cannot be both a `AssignmentRule` `variable` field and a `SpeciesReference` `species` field having the same value. (See Section 4.9 for the definition of `SpeciesReference`.) This means an assignment rule cannot be defined for a species that is created or destroyed in a reaction. The only exception is when the given species is a boundary condition; i.e., on the `Species` structure the `boundaryCondition` field is set to "`true`".

- *In the case of a compartment*, an `AssignmentRule` sets the referenced compartment's size to the size determined by the formula in `math`. The overall units of the formula are the units specified for the size of the compartment identified by the value of the `AssignmentRule`'s `variable` field. (See Section 4.5.4 for an explanation of how the units of the compartment's size are determined.)

- *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 are the units defined for the parameter identified by the value of the `AssignmentRule`'s `variable` field. (See Section 4.7.3 for an explanation of how the units of the parameter are determined.)

4.8.3 `RateRule`

The rule type `RateRule` is used to express equations that determine the rates of change of variables. The left-hand side (the `variable` of a rate rule) can refer to the identifier of a species, compartment, or parameter. Two or more `RateRule` or `AssignmentRule` structures cannot have the same left-hand side or `variable` field value in an SBML model definition. In all cases, as would be expected, the units of the formula representing the right hand side, in the `math` field, are of the form $x/time$ where x are the same units as associated with the symbol in the `variable` field, when that variable appears in other formulae. *time* is a built-in unit (see Section 4.4). 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 to the value determined by the formula in `math`. The overall units of the formula must be *species quantity*/*time*, where the *time* units are the built-in units of time described in Section 4.4 and the *species quantity* units are the *units of the species* as defined in Section 4.6.4.

  Restrictions: In a given model, there cannot be both a `SpeciesReference species` field and a `RateRule variable` field having the same value. (See Section 4.9 for the definition of `SpeciesReference`.) This means an assignment rule cannot be defined for a species that is created or destroyed in a reaction. The only exception is when the given species is a boundary condition; i.e., on the `Species` structure that defines the species the `boundaryCondition` field is set to "`true`".

- *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 are *size*/*time*, where the *time* units are the built-in units of time described in Section 4.4 and the *size* units are the units of size on the compartment identified by the value of the `RateRule`'s `variable` field. (See Section 4.5.4 for an explanation of how the units of the compartment's size are determined.)

- *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 are of the form $x/time$ where x are the units defined for the parameter identified by the value of the `AssignmentRule`'s `variable` field and the *time* units are the built-in units of time described in Section 4.4. (See Section 4.7.3 for an explanation of how the units of the parameter are determined.)

4.8.4 Constraints on rules

SBML specifically does not stipulate the form of the algorithms that can be applied to rules and reactions. For example, SBML does not specify when or how often rules should be evaluated. The constraints described by rules and kinetic rate laws are meant to apply collectively to the set of variable values for a specific instant in time.

The ordering of assignment rules is significant: they are always evaluated in the order given in SBML.

No more than one assignment or rate rule can be defined for a given identifier. No assignment or rate rule can be defined for an identifier whose corresponding structure has the field `constant` set to `true`.

An assignment rule for a given identifier overrides the initial value assigned to that identifier; i.e., the initial value should be ignored. This does not mean that a structure declaring an identifier can be omitted if there is an assignment rule for that identifier. For example, there must be a `Parameter` structure for a given parameter if there is a rule for that parameter.

The `math` field of an assignment rule structure can contain any identifier in a MathML `ci` element except for the following: (a) identifiers for which there exists a subsequent assignment rule, and (b) the identifier for which the rule is defined. These constraints are designed to eliminate algebraic loops among the scalar rules; eliminating algebraic loops ensures that assignment rules can be evaluated any number of times without the result of those evaluations changing. As an example, consider the following equations, in the order shown:

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

If this set of equations were interpreted as a set of assignment rules, it would be invalid because the rule for x refers to x and the rule for y refers to z before z is defined.

4.8.5 Example of Rule Use

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

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

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):

```
<model>
    ...
    <listOfRules>
        <assignmentRule variable="k">
            <notes>
                <xhtml:p>
                    k = k3/k2
                </xhtml:p>
            </notes>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <divide/>
                    <ci> k3 </ci>
                    <ci> k2 </ci>
                </apply>
            </math>
        </assignmentRule>
        <assignmentRule variable="s2">
            <notes>
                <xhtml:p>
                    s2 = (k * x)/(1 + k2)
                </xhtml:p>
            </notes>
            <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">
            <notes>
                <xhtml:p>
                    A = 0.10 * x
                </xhtml:p>
            </notes>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <times/>
                    <cn> 0.10 </cn>
                    <ci> x </ci>
                </apply>
            </math>
        </assignmentRule>
    </listOfRules>
    ...
</model>
```

4.9 Reactions

A *reaction* represents any transformation, transport or binding process, typically a chemical reaction, that can change the amount 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 kinetic law describing the rate at which the reaction takes place, and optional parameters entering into the kinetic law. These various parts of a reaction are recorded in the SBML `Reaction` type defined in Figure 12 on the following page.

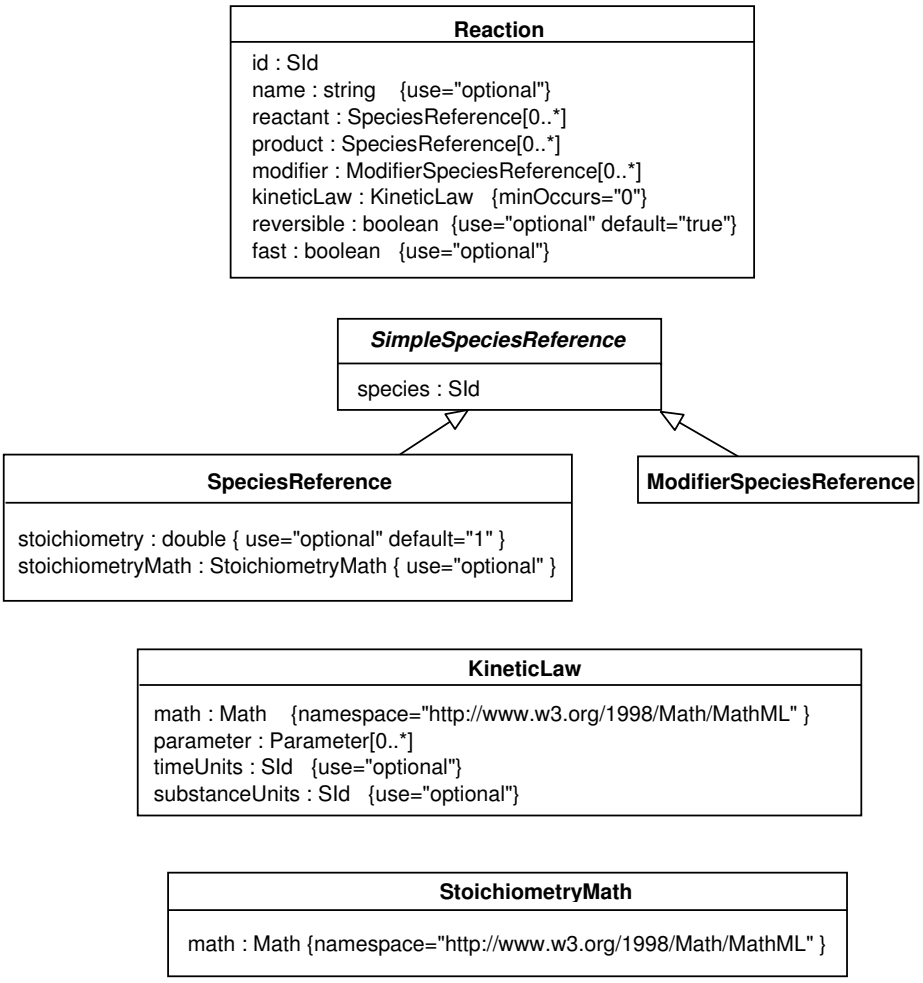


Figure 12: *The definitions of* `Reaction`*,* `KineticLaw`*,* `SpeciesReference` *and* `ModiferSpeciesReference`*. Following UML notation fields that are inherited from a base class are not shown.*

4.9.1 The `id` and `name` Fields

As with most other main structures in SBML, the `Reaction` data structure includes a required `id` and an optional `name`. These must be used according to the guidelines described in Section 3.3.

4.9.2 The `reactant`, `product` and `modifier` Fields

The reactant species, product species and modifier species in a reaction are described using the fields `reactant`, `product` and `modifiers`, respectively. These fields are optional lists of `SpeciesReference` and `ModifierSpeciesReference` structures, as shown in Figure 12. They are described in more detail in Sections 4.9.5 and 4.9.6 below. The abstract type `SimpleSpeciesReference` is shown simply to demonstrate the common field, `species`, of the `SpeciesReference` and `ModifierSpeciesReference` structures. In future levels of SBML it is anticipated that `SimpleSpeciesReference` will have additional fields.

4.9.3 The `reversible` Field

The optional boolean field `reversible` indicates whether the reaction is reversible. The field is optional, and if left unspecified in a model, it defaults to a value of "`true`". Although the reversibility of a reaction is determined by its rate law, the need to allow rate expressions in SBML to be optional leads to the need for a

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. It is true that the presence of this information in two places (i.e., the rate expression and the flag `reversible`) leaves open the possibility of a model containing contradictory information, but the creation of such a model would indicate an error on the part of the software generating it. Software developers must take care to guard against logical contradictions in the definitions of reactions.

4.9.4 The `fast` Field

The optional boolean field `fast` is another boolean field in the `Reaction` data structure; a value of "`true`" signifies that the given reaction is a "fast" one compared to others in the system being modeled. This may be relevant when computing equilibrium concentrations of rapidly equilibrating reactions. Simulation/analysis packages may chose to use this information to reduce the number of ODEs required and thereby optimize such computations. (A simulator/analysis package that has no facilities for dealing with fast reactions can ignore this field. In theory, if the choice of which reactions are fast is correctly made, then a simulation performed with them should give the same results as a simulation performed without fast reactions. However, currently there appears to be no single unambiguous method for designating which reactions should be considered fast, and some users may designate a reaction as fast when in fact it is not.) The `fast` field does not have a default value: a missing value indicates the modeller does not know or wish to specify the rate of the reaction relative to other reactions in the model.

4.9.5 `SpeciesReference`

Every species that enters into a given reaction must appear in that reaction's lists of reactants, products or modifiers. In an SBML model, all species that participate in any reaction are listed in the `listOfSpecies` field of the top-level `Model` data structure (see Section 4.2). Lists of products, reactants and modifiers in `Reaction` structures do not introduce new species, but rather, they refer back to those listed in the model's `listOfSpecies`. For reactants and products, the connection is made using the `SpeciesReference` data structure defined in Figure 12 on the preceding page.

In `SpeciesReference`, the field `species` of type `SId` must refer to the name of an existing species defined in the enclosing `Model` structure. The stoichiometry for the product or reactant can be specified using either the `stoichiometry` or `stoichiometryMath` fields of the `SpeciesReference` structure. The `stoichiometry` field is of type double and should contain values greater than 0. The `stoichiometryMath` field is implemented as an element containing a MathML math expression in dimensionless units. Only one of the `stoichiometry` and `stoichiometryMath` fields should be used on a given `SpeciesReference` structure. When neither field is present then the stoichiometry associated with the `SpeciesReference` structure is "`1`".

When generating SBML Level 2, it is *recommended* for maximal interoperability that the `stoichiometry` field be used in preference to the `stoichiometryMath` field and that the `stoichiometry` field contains integer values. Parsing software should expect and handle appropriately all possible values of the `stoichiometry` and `stoichiometryMath` fields including, for example, non-integer values for `stoichiometry`.

The following is a simple example of a species reference for species "X0", with stoichiometry 2, in a list of reactants within a reaction named "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 expression consisting of the parameter `x`:

```
<model>
    ...
    <listOfReactions>
        <reaction id="J1">
            <listOfReactants>
                <speciesReference species="X0">
                    <stoichiometryMath>
                        <math xmlns="http://www.w3.org/1998/Math/MathML">
                            <ci>x</ci>
                        </math>
                    </stoichiometryMath>
                </speciesReference>
            </listOfReactants>
            ...
        </reaction>
        ...
    </listOfReactions>
    ...
</model>
```

A reaction can contain an empty list of reactants or an empty list of products but must have at least one reactant or product. Also note that whether a given species is allowed to appear as a reactant or product is dictated by certain flags on the structure describing the species in the `Model`; see Table 4 on page 20 for more information.

A species can occur more than once in the lists of reactants and products of a given reaction. Such a `Reaction` structure represents the combined effect of all the `SpeciesReference` structures that are contained in it. The effective stoichiometry for a species in a reaction is the sum of the stoichiometry values given on the `SpeciesReference` structures in the list of products minus the sum of stoichiometry values given on the `SpeciesReference` structures 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"/>
    </listProducts>
</reaction>
<reaction id="y">
    <listOfReactants>
        <speciesReference species="a" stoichiometry="2"/>
    </listOfReactants>
    <listOfProducts>
        <speciesReference species="c"/>
    </listProducts>
</reaction>
```

4.9.6 `ModifierSpeciesReference`

In some cases, a species may act as a catalyst or inhibitor of a reaction and would not appear in the list of reactants or products because it is neither created nor destroyed in that reaction. In these cases, the species is known as a modifier. (That same species may still be a reactant or product of another reaction.)

The `Reaction` structure provides a way to express which species act as modifiers in a given reaction. This is the purpose of the `modifier` field in `Reaction`; this field is a list of `ModifierSpeciesReference` structures

defined in Figure 12 on page 26. The `ModifierSpeciesReference` structure has only one field, `species`, of type `SId`; its value must be the identifier of a species defined in the enclosing `Model`.

4.9.7 `KineticLaw`

A `kineticLaw` structure, enclosed in a `Reaction` structure, describes the rate at which the reaction takes place. The inclusion of a `KineticLaw` structure 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 law definition in a reaction. As shown in Figure 12 on page 26, the `KineticLaw` structure has a field called `math`; this is a MathML expression defining the rate of the reaction.

It is important to make clear at the outset that a "kinetic law" in SBML is *not* equivalent to a traditional rate law. The reason is that SBML must support multi-compartment models, and the units used in textbook rate laws as well as some conventional single-compartment modeling packages are problematic when used for defining reactions between multiple compartments. Rate expressions in SBML are expressed in terms of *substance/time*, rather than the more typical *concentration/time*. Converting between these two conventions is a simple matter of dividing the *substance/time* expression by the size of the compartment where a given species is located. To put this in slightly more precise terms, suppose there are two species S and P in a reaction $S \rightarrow P$, where S is located in a compartment A having volume V_A, P is located in a compartment B having volume V_B, and the kinetic law expression gives the rate of the reaction as being R. Then the rate of change in the concentration of S is given by $d[S]_A/dt = -R/V_A$ and the rate of change in the concentration of P is $d[P]_B/dt = R/V_B$. The translation of a complete multi-compartmental model into ODEs is given in Section 5.6.

The expression in `math` may refer to species identifiers; in that case, as discussed in Section 3.6.3, the units of each species reference are those of *concentration*. The only species identifiers that can be used in `math` are those listed in the `reactant`, `product` and `modifier` fields of the `Reaction` structure.

As was previously stated, the overall rate expression in the `math` field must have the units of *substance/time*. The optional fields `substanceUnits` and `timeUnits` determine the units of substance and time for the reaction respectively. The values of these fields must be chosen from one of the following possibilities: one of the base unit names from Table 2 on page 14; one of the built-in unit names appearing in the first column of Table 3 on page 15); or the name of a new unit defined in the list of unit definitions in the enclosing `Model` structure. In the case of `substanceUnits`, the value chosen must be a scaled and/or multiplied variant of `moles` or `item`. In the case of `timeUnits`, the value chosen must be a scaled and/or multiplied variant of `seconds`. If these fields are not set in a given `KineticLaw` instance, the units are taken from the defaults defined by the built-in "`substance`" and "`time`" of Table 3 on page 15.

An instance of a `KineticLaw` type structure can contain zero or more `parameter` structures (Section 4.7) which define symbols that can be used in the `math` field. As discussed in Section 3.5, reactions introduce local namespaces for parameter identifiers. Within a `KineticLaw` structure inside a reaction definition, a local parameter whose identifier is identical to a global parameter defined in the enclosing `Model`-type structure takes precedence over that global parameter.

The following is an example of a `Reaction` structure that defines a reaction named J_1, in which $X_0 \rightarrow S_1$ at a rate given by kX_0S_2, where S_2 is a catalyst and k is a parameter. It demonstrates the use of species references and the `KineticLaw` structure. The units on the species here are the defaults of *substance/volume* (see Section 4.6), and so the units of the parameter k are set appropriately to make the rate expression have final units of *substance/time*.

```
<model>
    ...
    <listOfUnitDefinitions>
        <unitDefinition id="vol_per_concent_per_time">
            <listOfUnits>
                <unit kind="litre"  exponent="2"/>
                <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>
                    </apply>
                </math>
                <listOfParameters>
                    <parameter id="k" value="0.1" units="vol_per_concent_per_time"/>
                </listOfParameters>
            </kineticLaw>
        </reaction>
    </listOfReactions>
    ...
</model>
```

4.10 Events

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

An `Event` structure 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` structure 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 13 on the next page. Both `Event` and `EventAssignment` are derived from `SBase` (see Section 3.1). An example of a model which uses events is given below.

4.10.1 The `id` and `name` Fields

These optional fields are available to support external references to event structures. These fields operate in the manner described in Section 3.3 except that the `id` field is optional.

4.10.2 The `trigger` Field

The `trigger` field defines when the `Event` structure has an effect on the model. The `trigger` field contains a MathML boolean expression. The exact instant that the expression evaluates to true is the time point when the `Event` is *fired*. The event only fires when the `trigger` makes the transition from false to true. The event will fire at any further time points when the `trigger` make this transition.

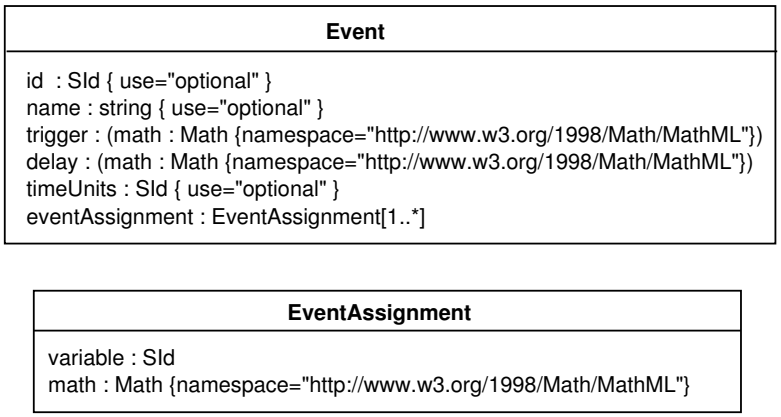


Figure 13: *The definitions of* `Event` *and* `EventAssignment`*. Following UML notation, additional fields that are inherited from a base class, in this case* `SBase`*, are not shown.*

4.10.3 The `delay` Field

The optional `delay` field defines the length of time after the event has *fired* that the event is *executed*. The `delay` field is another MathML expression. This expression should be evaluated when the rule is *fired*. The default value for the `delay` field is 0. The value of the `delay` field should always be positive. The units of the `delay` field are those given on the `timeUnits` field.

4.10.4 The `timeUnits` Field

The optional field `timeUnits` determines the units of time that apply to the `delay` field. The value of the `timeUnits` field must be either "`second`" from Table 2 on page 14, "`time`" from Table 3 on page 15, or a new unit defined by a unit definition in the enclosing model which must be a variant of "`second`" units. The default value of the `timeUnits` field is "`time`".

4.10.5 The `eventAssignment` Field

The `eventAssignment` field consists of a non-empty list of `EventAssignment` structures. This field is implemented as a `listOfEventAssignments` element containing one or more `eventAssignment` elements. The `EventAssignment` structures represent variable assignments that have effect when the event is *executed*. The `Assignment` structure is shown in Figure 13. The `variable` field is of type `SId` and contains the identifier of a variable i.e. a compartment, species or parameter. The structures referenced by the `variable` field must have their `constant` fields set to "`false`". The `math` field contains a MathML expression that defines the new value of the variable. This expression is evaluated when the `Event` is *fired* but the variable only acquires the result or new value when the `Event` is *executed*. The order of the `EventAssignment` structures is not significant (unlike assignment rules); the effect of one assignment cannot affect the result of another assignment. The identifiers occurring in the MathML `ci` fields of the `EventAssignment` structures represent the value of the identifier at the point when the `Event` is *fired*.

In all cases, as would be expected, the units of the formula are identical to the units associated with the `variable` field, when that variable appears in other formulae. However, the precise details, which are identical to those of `AssignmentRule` structures, depend on the variable that is being set:

- *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 formula are the *units of the species* as defined in Section 4.6.4.
- *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 are the units specified for the size of the compartment identified by the value of the `EventAssignment`'s `variable` field. (See Section 4.5.4 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 are the units defined for the parameter identified by the value of the `EventAssignment`'s `variable` field. (See Section 4.7.3 for an explanation of how the units of the parameter are determined.)

4.10.6 Example `Event` structure

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

```
<event>
    <trigger>
        <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply>
                <leq/>
                <ci> P1 </ci>
                <ci> t </ci>
            </apply>
        </math>
    </trigger>
    <listOfEventAssignments>
        <eventAssignment variable="k2">
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <cn> 0 </cn>
            </math>
        </eventAssignment>
    <listOfEventAssignments>
</event>
```

A complete example of a model using events is given in Section 5.9.

4.10.7 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. Events *fire* only when the trigger *becomes* true, i.e. the trigger expression transitions from false to true. 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 model to be entirely encoded in `Event` structures.

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 results of simulations involving events may vary when simultaneous events occur during simulation. It is anticipated that future versions or levels of SBML will define a specific set of tie-breaking algorithms and a mechanism for models to indicate which algorithm should be applied during simulation.

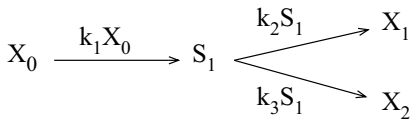
Despite the absence of a specific tie-breaking algorithm, SBML event simulation is constrained as follows. When an event X *fires* another event Y and event Y has zero delay then event Y is added to the existing set of simultaneous events that are pending *execution*. Events such as Y do not have a special 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 Example Models Expressed in XML Using SBML

In this section, we present several examples of complete models encoded in XML using SBML Level 2.

5.1 A Simple Example Application of SBML

Consider the following hypothetical branched system:



The following is an XML document encoding the model shown above:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
    <model id="Branch">
        <notes>
            <body xmlns="http://www.w3.org/1999/xhtml">
                <p>Simple branch system.</p>
                <p>The reaction looks like this:</p>
                <p>reaction-1:   X0 -> S1; k1*X0;</p>
                <p>reaction-2:   S1 -> X1; k2*S1;</p>
                <p>reaction-3:   S1 -> X2; k3*S1;</p>
            </body>
        </notes>
        <listOfCompartments>
            <compartment id="compartmentOne" size="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="S1" initialConcentration="0" compartment="compartmentOne"
                     boundaryCondition="false"/>
            <species id="X0" initialConcentration="0" compartment="compartmentOne"
                     boundaryCondition="true"/>
            <species id="X1" initialConcentration="0" compartment="compartmentOne"
                     boundaryCondition="true"/>
            <species id="X2" initialConcentration="0" compartment="compartmentOne"
                     boundaryCondition="true"/>
        </listOfSpecies>
        <listOfReactions>
            <reaction id="reaction_1" reversible="false">
                <listOfReactants>
                    <speciesReference species="X0"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="S1"/>
                </listOfProducts>
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <times/>
                            <ci> k1 </ci>
                            <ci> X0 </ci>
                        </apply>
                    </math>
                    <listOfParameters>
                        <parameter id="k1" value="0"/>
                    </listOfParameters>
                </kineticLaw>
            </reaction>
            <reaction id="reaction_2" reversible="false">
                <listOfReactants>
                    <speciesReference species="S1"/>
                </listOfReactants>
                <listOfProducts>
```

```
                <speciesReference species="X1"/>
            </listOfProducts>
            <kineticLaw>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <ci> k2 </ci>
                        <ci> S1 </ci>
                    </apply>
                </math>
                <listOfParameters>
                    <parameter id="k2" value="0"/>
                </listOfParameters>
            </kineticLaw>
        </reaction>
        <reaction id="reaction_3" reversible="false">
            <listOfReactants>
                <speciesReference species="S1"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="X2"/>
            </listOfProducts>
            <kineticLaw>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <ci> k3 </ci>
                        <ci> S1 </ci>
                    </apply>
                </math>
                <listOfParameters>
                    <parameter id="k3" value="0"/>
                </listOfParameters>
            </kineticLaw>
        </reaction>
    </listOfReactions>
</model>
</sbml>
```

In this example, the model has the identifier "Branch". The model contains one compartment, four species, and three reactions. 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 includes a `<notes>` annotation that summarizes the model in text form, with formatting encoded in XHTML. This may be useful for a software package that is able to read such annotations and, for example, render them in HTML in a graphical user interface.

5.2 Example Involving Units

The following model uses the units features of SBML Level 2. In this model, the default value of `substance` is changed in the list of unit definitions to be mole units with a scale factor of -3, or millimoles. This sets the default substance units in the model; components can override this scale locally. The `size` and `time` built-ins are left to their defaults, ensuring size is in liters and time is in seconds. The result is that, in this model, kinetic law formulas define rates in millimoles per second and the species symbols in them represent concentration values in millimoles per liter. 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 liter per second, and milliMolar, respectively.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1"
      xmlns:html="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="mm">
        <listOfUnits>
            <unit kind="mole" scale="-3"/>
        </listOfUnits>
    </unitDefinition>
</listOfUnitDefinitions>
<listOfCompartments>
    <compartment id="cell"/>
</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="mm"/>
</listOfParameters>
<listOfReactions>
    <reaction id="v1">
        <listOfReactants>
            <speciesReference species="x0"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="s1"/>
        </listOfProducts>
        <kineticLaw>
            <notes>
                <html:p>((vm * s1)/(km + s1))*cell</html: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>
                    <html:p>((vm * s2)/(km + s2))*cell</html: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>
                    <html:p>((vm * x1)/(km + x1))*cell</html: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>
```

5.3 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 implicit `fast` field on a reaction element.

The system modeled is

$$\begin{array}{ccc} X_0 & \xrightarrow{k_1 X_0} & S_1 \\ S_1 & \xrightarrow{k_f S_1 - k_r S_2} & S_2 \\ S_2 & \xrightarrow{k_2 S_2} & X_1 \end{array}$$

$$k_1 = 0.1, \quad k_2 = 0.15, \quad k_f = K_{eq}10000, \quad k_r = 10000, \quad K_{eq} = 2.5.$$

This can be approximated with the following system:

$$\begin{array}{ccc} X_0 & \xrightarrow{k_1 X_0} & T \\ T & \xrightarrow{k_2 S_2} & X_1 \end{array}$$

$$S_1 = \frac{T}{1 + K_{eq}}, \quad S_2 = K_{eq} S_1$$

The following example SBML model encodes the approximate form.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1"
      xmlns:math="http://www.w3.org/1998/Math/MathML">
    <model>
        <listOfCompartments>
            <compartment id="cell"/>
        </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"/>
        </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">
```

```
            <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>
                    </apply>
                </math>
                <listOfParameters>
                    <parameter id="k1" value="0.1"/>
                </listOfParameters>
            </kineticLaw>
        </reaction>
        <reaction id="out">
            <listOfReactants>
                <speciesReference species="T"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="X1"/>
            </listOfProducts>
            <kineticLaw>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <ci> k2 </ci>
                        <ci> S2 </ci>
                    </apply>
                </math>
                <listOfParameters>
                    <parameter id="k2" value="0.15"/>
                </listOfParameters>
            </kineticLaw>
        </reaction>
    </listOfReactions>
  </model>
</sbml>
```

5.4 Example Involving Algebraic Rules

This section contains an example model that contains an `AlgebraicRule` structure. The model contains a different formulation of the fast reaction described in Section 5.3.

The system described in Section 5.3 can be approximated with the following system:

$$X_0 \quad \xrightarrow{k_1 X_0} \quad T$$

$$T \quad \xrightarrow{k_2 S_1} \quad X_1$$

$$S_2 = K_{eq} S_1$$

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" level="2" version="1">
    <model>
```

```
<listOfCompartments>
    <compartment id="cell"/>
</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"/>
</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">
        <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>
                </apply>
            </math>
            <listOfParameters>
                <parameter id="k1" value="0.1"/>
            </listOfParameters>
        </kineticLaw>
    </reaction>
    <reaction id="out">
        <listOfReactants>
            <speciesReference species="T"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="X1"/>
        </listOfProducts>
        <kineticLaw>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <times/>
```

```
                        <ci> k2 </ci>
                        <ci> S2 </ci>
                    </apply>
                </math>
                <listOfParameters>
                    <parameter id="k2" value="0.15"/>
                </listOfParameters>
            </kineticLaw>
        </reaction>
    </listOfReactions>
</model>
</sbml>
```

5.5 Example Involving Combinations of `boundaryCondition` and `constant` Values on `Species` with `RateRule` structures

This section contains a model that includes four species each with a different combination of values of for the `boundaryCondition` and `constant` fields.

Consider the following hypothetical system:

$$S_1 + S_2 \quad \xrightarrow{k_1 S_1 S_2 S_3} \quad S_4$$

S_3 is a catalyst that catalyzes the conversion S_1 and S_2 into S_4. S_1 and S_2 are on the boundary of the system (S_1 and S_2 are reactants but their values are not determined by a kinetic law). The value of S_1 is determined over time by the rate rule:

$$\frac{dS_1}{dt} = k_2$$

Constant values are:

$$S_2 = 1$$

$$S_3 = 2$$

$$k_1 = 0.5$$

$$k_2 = 0.1$$

and initial values are:

$$S_1 = 0$$

$$S_4 = 0$$

The value of S_1 varies over time so in SBML S_1 has a `constant` field with a default value of "`false`". The values of S_2 and S_3 are fixed so in SBML they have a `constant` field values of "`true`". S_3 only occurs as a modifier so the value of its `boundaryCondition` field 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 values, the default values, for both `boundaryCondition` and `constant` fields.

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" level="2" version="1">
    <model id="BoundaryCondExampleModel">
        <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"/>
            <parameter id="k2" value="0.1"/>
        </listOfParameters>
        <listOfRules>
            <rateRule variable="S1">
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <ci> k2 </ci>
                    </apply>
                </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>
                         </apply>
                    </math>
                </kineticLaw>
            </reaction>
        </listOfReactions>
    </model>
</sbml>
```

5.6 Example of translation from a multi-compartmental model to ODEs

This section contains a model with 2 compartments, 4 reactions and 2 rate rules. The model is followed by its complete translation into ordinary differential equations. The model is shown in figure 14.

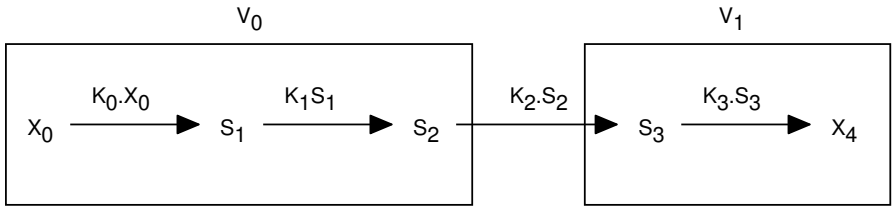


Figure 14: *A example multi-compartmental model.*

Reaction equations are in substance per time units. Species variables are in concentrations. Species X_0 and X_4 are boundary conditions. The volume of compartment V_1 and the concentration of X_0 vary according to rules.

The SBML for this model is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
    <model id="ODEExampleModel">
        <listOfCompartments>
            <compartment id="V0" size="10"/>
            <compartment id="V1" size="1" constant="false"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="X0" initialConcentration="0" compartment="V0"
                boundaryCondition="true" />
            <species id="S1" initialConcentration="0" compartment="V0"/>
            <species id="S2" initialConcentration="0" compartment="V0"/>
            <species id="S3" initialConcentration="0" compartment="V1"/>
            <species id="X4" initialConcentration="0" compartment="V1"
                boundaryCondition="true" constant="true"/>
        </listOfSpecies>
        <listOfParameters>
            <parameter id="K0" value="0.1"/>
            <parameter id="K1" value="0.5"/>
            <parameter id="K2" value="0.1"/>
            <parameter id="K3" value="0.5"/>
            <parameter id="Kv" value="0.5"/>
            <parameter id="Kin" value="0.1"/>
        </listOfParameters>
        <listOfRules>
            <rateRule variable="X0">
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <ci> Kin </ci>
                    </apply>
                </math>
            </rateRule>
            <rateRule variable="V1">
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <ci> Kv </ci>
                    </apply>
                </math>
            </rateRule>
        </listOfRules>
        <listOfReactions>
            <reaction id="reaction_1" reversible="false">
                <listOfReactants>
                    <speciesReference species="X0"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="S1"/>
                </listOfProducts>
                <kineticLaw>
                    <math xmlns="http://www.w3.org/1998/Math/MathML">
                        <apply>
                            <times/>
                            <ci> K0 </ci>
                            <ci> X0 </ci>
                         </apply>
                    </math>
                </kineticLaw>
            </reaction>
            <reaction id="reaction_2" reversible="false">
                <listOfReactants>
                    <speciesReference species="S1"/>
                </listOfReactants>
                <listOfProducts>
```

```
                <speciesReference species="S2"/>
            </listOfProducts>
            <kineticLaw>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <ci> K1 </ci>
                        <ci> S1 </ci>
                     </apply>
                </math>
            </kineticLaw>
        </reaction>
        <reaction id="reaction_3" reversible="false">
            <listOfReactants>
                <speciesReference species="S2"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="S3"/>
            </listOfProducts>
            <kineticLaw>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <ci> K2 </ci>
                        <ci> S2 </ci>
                     </apply>
                </math>
            </kineticLaw>
        </reaction>
        <reaction id="reaction_4" reversible="false">
            <listOfReactants>
                <speciesReference species="S3"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="X4"/>
            </listOfProducts>
            <kineticLaw>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <ci> K3 </ci>
                        <ci> S3 </ci>
                     </apply>
                </math>
            </kineticLaw>
        </reaction>
    </listOfReactions>
  </model>
</sbml>
```

The ODE translation of this model is as follows assuming all species variables contain the species concentration:

Parameters and Constants:

$$X_4 = 0$$

$$V_0 = 10$$

$$K_0 = 0.1$$

$$K_1 = 0.5$$

$$K_2 = 0.1$$

$$K_3 = 0.5$$

$$K_{in} = 0.1$$

$$K_v = 0.5$$

Initial Conditions of Variables:

$$V_1 = 1$$

$$X_0 = S_1 = S_2 = S_3 = 0$$

Differential Equations:

$$\frac{dV_1}{dt} = K_v \qquad \text{rule}$$

$$\frac{dX_0}{dt} = K_{in} \qquad \text{rule}$$

$$\frac{dS_1}{dt} = \frac{K_0 X_0 - K_1 S_1}{V_0} \qquad \text{reactions 1 and 2}$$

$$\frac{dS_2}{dt} = \frac{K_1 S_1 - K_2 S_2}{V_0} \qquad \text{reactions 2 and 3}$$

$$\frac{dS_3}{dt} = \frac{K_2 S_2 - K_3 S_3}{V_1} \qquad \text{reactions 3 and 4}$$

5.7 Example Involving Function Definitions

This section contains a model that uses the function definition feature of SBML. Consider the following hypothetical system:

$$S_1 \quad \xrightarrow{f(S_1)} \quad S_2$$

where

$$f(x) = x * 2$$

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" level="2" version="1">
    <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="0" compartment="compartmentOne"/>
        <species id="S2" initialConcentration="0" compartment="compartmentOne"/>
    </listOfSpecies>
    <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>
                        <ci> f </ci>
                        <ci> S1 </ci>
                     </apply>
                </math>
            </kineticLaw>
        </reaction>
    </listOfReactions>
  </model>
</sbml>
```

5.8 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

The SBML form of this model is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  <model>
    <listOfCompartments>
      <compartment id="cell" size="1"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="P" compartment="cell" initialConcentration="0"/>
    </listOfSpecies>
    <listOfParameters>
```

```
            <parameter id="tau" value="1"/>
            <parameter id="m" value="0.5"/>
            <parameter id="q" value="1"/>
            <parameter id="delta_t" value="1"/>
        </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/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>
```

5.9 Example Involving Events

This section presents a simple model system that demonstrates the use of events in SBML. Consider a system with two genes, k_1 and k_2. k_1 is initially on and k_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, k_2 switches off. This system can be represented mathematically as follows:

$$\begin{aligned} \frac{dP_1}{dt} &= k_1 - P_1 \\ \frac{dP_2}{dt} &= k_2 - P_2 \\ k_2 &= \begin{cases} 0 & \text{when } P_1 \leq \tau, \\ 1 & \text{when } P_1 > \tau. \end{cases} \end{aligned}$$

The initial values are:

$$k_1 = 1 \quad k_2 = 0 \quad \tau = 0.25 \quad P_1 = 0 \quad P_2 = 0$$

The SBML Level 2 representation of this as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1"
      xmlns:math="http://www.w3.org/1998/Math/MathML">
    <model>
        <listOfCompartments>
            <compartment id="cell"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="P1" compartment="cell" initialConcentration="0"/>
            <species id="P2" compartment="cell" initialConcentration="0"/>
        </listOfSpecies>
        <listOfParameters>
            <parameter id="k1" value="1" constant="false"/>
            <parameter id="k2" value="0" constant="false"/>
            <parameter id="tau" value="0.25"/>
        </listOfParameters>
        <listOfRules>
            <rateRule variable="P1">
                <math:math>
                    <math:apply>
                        <math:minus/>
                        <math:ci> k1 </math:ci>
                        <math:ci> P1 </math:ci>
                    </math:apply>
                </math:math>
            </rateRule>
            <rateRule variable="P2">
                <math:math>
                    <math:apply>
                        <math:minus/>
                        <math:ci> k2 </math:ci>
                        <math:ci> P2 </math:ci>
                    </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="k2">
                        <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="k2">
```

```
                    <math:math>
                        <math:cn> 0 </math:cn>
                    </math:math>
                </eventAssignment>
            </listOfEventAssignments>
        </event>
    </listOfEvents>
  </model>
</sbml>
```

5.10 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`. The model also includes two cytosolic proteins that buffer calcium via the `CalciumCalbindin_gt_BoundCytosol` and `CalciumBuffer_gt_BoundCytosol` reactions.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
    <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>
        </listOfUnitDefinitions>
        <listOfCompartments>
            <compartment id="Extracellular" spatialDimensions="3"/>
            <compartment id="PlasmaMembrane" outside="Extracellular" spatialDimensions="2"/>
            <compartment id="Cytosol" outside="PlasmaMembrane" spatialDimensions="3"/>
        </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_EC"
              compartment="Extracellular"
              initialConcentration="1000"/>
            <species
              id="Ca_C"
              compartment="Cytosol"
              initialConcentration="0.1"/>
            <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>
                    <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>
            </math>
                 <listOfParameters>
                    <parameter id="Kf_CalciumCalbindin_BoundCytosol" value="20.0"/>
                    <parameter id="Kr_CalciumCalbindin_BoundCytosol" value="8.6"/>
                 </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>
                    <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>
        </math>
        <listOfParameters>
            <parameter id="Kf_CalciumBuffer_BoundCytosol" value="0.1"/>
            <parameter id="Kr_CalciumBuffer_BoundCytosol" value="1.0"/>
        </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>
                <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>
        </math>
        <listOfParameters>
            <parameter id="Vmax" value="-4000"/>
            <parameter id="kP" value="0.25"/>
            <parameter id="Ca_Rest" value="0.1"/>
        </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>
                        <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>
                </math>
                <listOfParameters>
                    <parameter id="J0" value="0.014"/>
                    <parameter id="Kc" value="0.5"/>
                </listOfParameters>
            </kineticLaw>
        </reaction>
    </listOfReactions>
  </model>
</sbml>
```

6 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.

6.1 Future Enhancements: SBML Level 3 and Beyond

As mentioned above, SBML Level 2 is intended to provide a foundation for modeling biochemical networks. 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:

- *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.
- *Multi-state 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.
- *Controlled Vocabularies.* This will enable models and model components to be labelled with instances of controlled vocabularies. A model vocabulary will describe the SBML features used by a given model.
- *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 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 $s \leq 0$.
- *Rules for Initial Conditions.* This will enable the encoding of rules that are only executed at the start of a simulation i.e. set initial conditions. These rules will be similar to assignment rules and are evaluated in the same sequence as assignment rules but only in the very first instance of simulation.
- *Tie-breaking algorithm.* This will include a controlled vocabulary and associated fields on models to indicate the simultaneous event tie-breaking algorithm required to correctly simulate the model.
- *Composed Units.* This will enable a unit definition to be composed from one or more other unit definitions.

6.2 Relationships to Other Efforts

There are a number of ongoing efforts with similar goals as those of SBML. Many of them are oriented more specifically toward describing protein sequences, genes and related entities for database storage and search. These are generally not intended to be computational models, in the sense that they do not describe entities and behavioral rules in such a way that a simulation package could "run" the models.

The effort perhaps closest in spirit to SBML is CellML™ (Hedley et al., 2001). CellML is an XML-based markup language designed for storing and exchanging computer-based biological models. It includes facilities for representing model structure, mathematics and additional information for database storage and search. Models are described in terms of networks of connections between discrete components, where a component is a functional unit that may correspond to a physical compartment or simply a convenient modeling abstraction. Components contain variables and connections contain mappings between the variables of connected components. CellML provides facilities for grouping components and specifying the kinds of relationships that may exist between components. It also uses MathML (W3C, 2000b) for expressing mathematical relationships between components and provides the ability to use ECMAScript (formerly known as JavaScript) to define functions.

The constructs in CellML tend to be at a more abstract and general level than those in SBML Level 2, and describe the structure and underlying mathematics of cellular models in a very general way. By contrast, SBML is closer to the internal object model used in model analysis software. Because SBML Level 2 is being developed in the context of interacting with a number of existing simulation packages, it is a more concrete language than CellML and may be better suited to its purpose of enabling interoperability with existing simulation tools.

The development of SBML Level 2 has benefited from discussions with the developers of CellML. The developers of SBML and CellML are actively engaged in ensuring that the two representations can be translated between each other.

Acknowledgments

SBML was developed with funding and support from the ERATO Kitano Symbiotic Systems project, a project funded by the Japan Science and Technology Corporation and hosted in part at the California Institute of Technology.

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 description of 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 Level 2. The first draft of this document was released in August 2002. The final set of features in SBML Level 2 described in this document was finalized in May 2003 at the *7th Workshop on Software Platforms for Systems Biology* in Ft. Lauderdale, Florida.

SBML Level 2 was developed with the help of many people, especially the authors of BASIS, BioSketchPad, BioSpice, CellDesigner, Cellerator, CellML, COPASI, DBSolve, E-Cell, ESS, Gepasi, Jarnac, JDesigner, JigCell, MCell, NetBuilder, Promot/DIVA, StochSim, and Virtual Cell, and members of the `sysbio` and `sbml-discuss` mailing lists. We are particularly grateful to the following people for discussions, advice and comments: Nicolas Allen, Adam Arkin, Hamid Bolouri, Ben Bornstein, Dennis Bray, Roger Brent, Steve Burbeck, Claudine Chaouiya, Kwang Cho, Athel Cornish-Bowden, Manuel Corpas, Autumn Cuellar, John

Doyle, Serge Dronov, Drew Endy, David Fell, Carl Firth, Ed Frank, Akira Funahashi, Ralph Gauges, Martin Ginkel, Victoria Gor, Igor Goryanin, Warren Hedley, Charles Hodgman, Stefan Hoops, Nick Juty, Jay Kaserger, Sarah Keating, Hiroaki Kitano, Ben Kovitz, Andreas Kremling, Nicolas Le Novère, Fred Livingston, Les Loew, Daniel Lucio, Joanne Matthews, Mike McCollum, Pedro Mendes, Eric Minch, Eric Mjolsness, David Morley, Mineo Morohashi, Poul Nielsen, Greg Peterson, Mark Poolman, Carole Proctor, Wayne Rindone, Sven Sahle, Takeshi Sakurada, Vijay Saraswat, Herbert Sauro, James Schaff, Maria Schilstra, John Schwacke, Cliff Shaffer, Bruce Shapiro, Tom Shimizu, Herbert Sauro, Hugh Spence, Jörg Stelling, Kouichi Takahashi, Masaru Tomita, Marc Vass, John Wagner, Jonathan Webb, Jörg Weimar, Darren Wilkinson, Olaf Wolkenhauer and Tau-Mu Yi.

Appendix

A Summary of Notation

The definitive explanation for the notation used in this document can be found in the companion notation document (Hucka, 2000). Here we briefly summarize some of the main components of the notations used in describing SBML.

Within the definitions of the various object classes introduced in this document, the following types of expressions are used many times:

```
field1 : float
field2 : integer[0..*]
field3 : float {use = "optional" default = "0.0"}
math   : Math {namespace="http://www.w3.org/1998/Math/MathML"}
field4 : (math : Math {namespace="http://www.w3.org/1998/Math/MathML"})
```

The symbols `field1`, `field2`, etc., represents fields in a data structure. The colon immediately after the name separates the name of the field from the type of data that it stores.

More complex specifications use square brackets (`[]`) just after a type name. This is used to indicate that the field contains a list of elements of the same type. Specifically, the notation `[0..*]` signifies a list containing zero or more elements; the notation `[1..*]` signifies a list containing at least one element; and so on. The approach used here to translate from a list form into XML is, first, create a subelement named `listOf_____s`, where the blank indicates the capitalized name of the field, and then put a sequences of elements each named after the type of the field as the content of the `listOf_____s` element. `listOf_____s` elements representing list of the form `[0..*]` are optional: a missing `listOf_____` element indicates that the list is empty. `listOf_____` elements should always have content.

Expressions in curly braces ({}) shown after an field type indicate additional constraints placed on the field. We express constraints using XML Schema language. In the examples above, the text {`use="optional" default="0.0"`} indicates that the field `field3` is optional and that it has a default value of 0.0. A constraint of the form {`namespace="`*X*`"`} indicates that the field is not in the SBML Level 2 XML namespace but resides in the given XML namespace *X*. If a field is in a different namespace then the type of the field will not be defined by the SBML UML. In the examples above, the `math` field and its content is defined in the MathML namespace.

A field definition of the form `X : (A : B)` defines an element *X* that contains a field *A* with type *B*. If *A* is the string `ANY` then the element *X* contains an arbitrary sequence of elements. A field definition of the form `X : ( A : B ) { C }` is similar except that the field *X* and its content is constrained by constraint *C*. A field definition of the form `X : ( A : B { C } )` is similar except that the field *A* and its content is constrained by constraint *C*. In the examples above the field `field4` is an element which contains a `math` field. The `math` field is in the MathML namespace but `field4` is in the SBML namespace.

B Differences between SBML Level 1 Version 2 and Level 2

Compared to SBML Level 1 Version 2, SBML Level 2 introduces the following changes:

- SBML Level 2 supports the inclusion of metadata using the same approach as CellML (Cuellar et al., 2002). All structures in SBML can be annotated with optional content in RDF (Resource Description Format; Lassila and Swick, 1999) following the guidelines put forward by Cuellar et al.. (Section 3.1.)
- All data structures, including `Sbml` and `listOf_____` elements, are now derived from the type `SBase`. (Section 3.1.) This means all major structures in SBML can have separate annotations and metadata associated with them.
- A new field, `id`, replaces the `name` field previously defined for most SBML structures to identify each part of a model. (See Section 3.3.) The `id` field has a type of `SId`, whose definition is similar to `SName`

in Level 1. In SBML Level 2, the `name` field is optional and is defined to allow any Unicode characters allowed by the `string` type of XML Schema (Biron and Malhotra, 2000).

- Formulas in Level 2 are expressed using MathML (W3C, 2000b) 2.0. The field named `formula` previously available on the `KineticLaw` and `Rule` structures has been replaced by a MathML element named `math` containing MathML content. In addition, stoichiometry numbers may now be expressed using MathML, allowing for more flexibility in defining reactions. (Sections 3.6, 4.8 and 4.9.)

- The namespace for identifiers in a model does not contain any built-in symbols; gone, for example, are the predefined rate laws of SBML Level 1. The approach taken in SBML Level 2 is that each model must itself define whatever functions it needs using the new `FunctionDefinition` mechanism. Although SBML Level 2 *does* define two built-in entities (a symbol representing time and another symbol representing delay functions), these are referenced using a feature of MathML and are not in the same namespace as identifiers defined by a model. (Section 3.6.4.)

- SBML Level 2 makes explicit a previously unstated assumption, that the XML encoding of a model uses UTF-8. SBML documents must refer to the UTF-8 encoding in their XML declaration. (Section 4.1.)

- The top-level `Model` structure can contain an optional list of global user-defined functions expressed in MathML and organized in new structures of type `FunctionDefinition`. (Sections 4.2 and 4.3.)

- The top-level `Model` structure can contain an optional list of event definitions organized in structures of type `Event`. Events define discrete changes in model behavior at specific times during a time simulation of the model. (Section 4.2 and 4.10.)

- Unlike in SBML Level 1, unit identifiers in Level 2 are in a separate namespace from the namespace used for models, functions, species, compartments, reactions and parameters. Also, the unit names "`meter`" and "`liter`" are not defined in Level 2 because the SBML user community deemed them unnecessary. Finally, `Unit` structures now have the additional fields `multiplier` and `offset` to enable the definition of non-SI units. (Section 4.4.)

- The `Compartment` structure has a new field, `spatialDimensions`, whose value is a positive integer specifying the number of dimensions in space the compartment possesses. This enables the definition of such things as two-dimensional membranes. As a side-effect, the units of species concentration in SBML Level 2 depend on the spatial dimensions of the compartment where the species is located. To support these new capabilities, `Compartment` now uses a field named `size` instead of `volume`, and there are two new built-in units for area and length. (Sections 4.4 and 4.5.)

- All fields representing initial conditions or parameter values, including compartment sizes and species concentrations, are optional in Level 2. A missing value for one of these fields implies that the value is either unknown, not required for analysis, or should be obtained from an external source. (Sections 4.5, 4.6 and 4.7.)

- The `Compartment`, `Species` and `Parameter` structures each have a new boolean field named `constant`. This field specifies whether the variables represented by these structures can be changed by rules and reactions. (Sections 4.5, 4.6 and 4.7.)

- The `Species` structure has a new field, `initialConcentration`, for setting the initial value of a species in terms of its concentration. This is in addition to the ability, carried over from Level 1, to set the values in terms of amounts. (Section 4.6.)

- The `Species` structure has two new fields, `spatialSizeUnits` and `substanceUnits`, which replace the `units` field in Level 1. These fields are composed to form the concentration units of the species symbol. (Section 4.6.)

- The rule structures are simpler compared to SBML Level 1. There is no longer a `type` field on `Rule`, and new structures `AssignmentRule` and `RateRule` replace SBML Level 1's `ParameterRule`, `SpeciesConcentrationRule` and `CompartmentVolumeRule`. (Section 4.8.)

- The `Reaction` structure has a new list of *modifiers* in addition to the list of reactants and products. The `listOfModifiers` enumerates species that affect a reaction but are neither created nor destroyed by the reaction. (Section 4.9.)

C XML Schema for SBML

The following is an XML Schema definition for the SBML Level 2 Version 1, 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: a SBML document validated by this schema is not necessarily a valid SBML Level 2 document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
        targetNamespace="http://www.sbml.org/sbml/level2"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xlink="http://www.w3.org/1999/xlink"
        xmlns:mml="http://www.w3.org/1998/Math/MathML"
        xmlns="http://www.sbml.org/sbml/level2"
        elementFormDefault="qualified"
        attributeFormDefault="unqualified"
        version="1">
    <xsd:import
        namespace="http://www.w3.org/1998/Math/MathML"
        schemaLocation="http://www.w3.org/Math/XMLSchema/mathml2/mathml2.xsd"/>
    <xsd:annotation>
        <xsd:documentation>
      File name : sbml.xsd
      Author : M. Hucka, A. Finney, D. Lucio
      Description : XML Schema for the Systems Biology Markup Language Level 2.
                    This is designed for XML Schema version 1.0.
      Version : 1

      Copyright 2003 California Institute of Technology and Japan Science and
      Technology Corporation.

      This library is free software; you can redistribute it and/or modify it
      under the terms of the GNU Lesser General Public License as published
      by the Free Software Foundation; either version 2.1 of the License, or
      any later version.

      This file is distributed in the hope that it will be useful, but
      WITHOUT ANY WARRANTY, WITHOUT EVEN THE IMPLIED WARRANTY OF
      MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.  The software
      and documentation provided hereunder is on an "as is" basis, and the
      California Institute of Technology and Japan Science and Technology
      Corporation have no obligations to provide maintenance, support,
      updates, enhancements or modifications.  In no event shall the
      California Institute of Technology or the Japan Science and Technology
      Corporation be liable to any party for direct, indirect, special,
      incidental or consequential damages, including lost profits, arising
      out of the use of this software and its documentation, even if the
      California Institute of Technology and/or Japan Science and Technology
      Corporation have been advised of the possibility of such damage.  See
      the GNU Lesser General Public License for more details.

      You should have received a copy of the GNU Lesser General Public License
      along with this library; if not, write to the Free Software Foundation,
      Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
</xsd:documentation>
    </xsd:annotation>
    <!--The definition of SId 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>
<!--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"
                        maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="annotation" minOccurs="0">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:any processContents="skip" maxOccurs="unbounded"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="metaid" type="xsd:ID" use="optional"/>
</xsd:complexType>
<!--The definition of FunctionDefinition 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>
<!--The definition of UnitKind follows.-->
<xsd:simpleType name="UnitKind">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="ampere"/>
        <xsd:enumeration value="becquerel"/>
        <xsd:enumeration value="candela"/>
        <xsd:enumeration value="Celsius"/>
        <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>
<!--The definition of Unit follows.-->
<xsd:complexType name="Unit">
    <xsd:complexContent>
        <xsd:extension base="SBase">
            <xsd:attribute name="kind" type="UnitKind" use="required"/>
            <xsd:attribute name="exponent" type="xsd:integer" default="1"/>
            <xsd:attribute name="scale" type="xsd:integer" default="0"/>
            <xsd:attribute name="multiplier" type="xsd:double" default="1"/>
            <xsd:attribute name="offset" type="xsd:double" default="0"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!--The definition of UnitDefinition follows.-->
<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="SId" use="required"/>
            <xsd:attribute name="name" type="xsd:string" use="optional"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!--The definition of Compartment follows.-->
<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="size" type="xsd:double" use="optional"/>
            <xsd:attribute name="spatialDimensions" use="optional" default="3">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:integer">
                        <xsd:minInclusive value="0"/>
                        <xsd:maxInclusive value="3"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="units" type="SId" 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>
<!--The definition of Species follows.-->
<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="compartment" type="SId"/>
            <xsd:attribute name="initialAmount" type="xsd:double" use="optional"/>
            <xsd:attribute name="initialConcentration" type="xsd:double" use="optional"/>
            <xsd:attribute name="substanceUnits" type="SId" use="optional"/>
            <xsd:attribute name="spatialSizeUnits" type="SId" 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:integer" use="optional"/>
            <xsd:attribute name="constant" type="xsd:boolean" use="optional" default="false"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!--The definition of Parameter follows.-->
<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="SId" 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>
<!--The definition of Rule follows. -->
<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>
<!--The definition of Reaction follows.-->
<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:attribute name="timeUnits" type="SId" use="optional"/>
            <xsd:attribute name="substanceUnits" type="SId" use="optional"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="SimpleSpeciesReference" abstract="true">
    <xsd:complexContent>
        <xsd:extension base="SBase">
            <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"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!--The definition of Event follows.-->
<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="MathField">
    <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="MathField"/>
                <xsd:element name="delay" type="MathField" 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="timeUnits" type="SId" use="optional"/>
        </xsd:extension>
```

```
    </xsd:complexContent>
</xsd:complexType>
<!-- The definition of Model follows.-->
<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="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" minOccurs="0">
                    <xsd:complexType>
                        <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: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="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="1"/>
    </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>
```

References

Abbott, A. (1999). Alliance of US labs plans to build map of cell signalling pathways. *Nature*, 402:219–200.

Arkin, A. P. (2001). *Simulac* and *Deduce*. Available via the World Wide Web at `http://gobi.lbl.gov/~aparkin/Stuff/Software.html`.

Bartol, T. M. and Stiles, J. R. (2002). MCell. Available via the World Wide Web at `http://www.mcell.cnl.salk.edu/`.

Belta, C., Goulian, M., Gowen, S., Ivancic, F., Kumar, V., Owen, A., Rubin, H., Rubin, M., Schug, J., Sokolsky, O., Webb, J., and Welber, L. (2003). Bio Sketch Pad. Available via the World Wide Web at `http://bio.bbn.com/biospice/biosketchpad/`.

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, D., Firth, C., Le Novère, N., and Shimizu, T. (2001). *StochSim*. Available via the World Wide Web at `http://www.zoo.cam.ac.uk/comp-cell/StochSim.html`.

Bray, T., D. Hollander, D., and Layman, A. (1999). Namespaces in XML. World Wide Web Consortium 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`.

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`.

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

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/`.

Funahashi, A. and Kitano, H. (2003). CellDesigner. Available via the World Wide Web at `http://www.systems-biology.org/`.

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`.

Goryanin, I. (2001). *DBsolve*: Software for metabolic, enzymatic and receptor-ligand binding simulation. Available via the World Wide Web at `http://homepage.ntlworld.com/igor.goryanin/`.

Goryanin, I., Hodgman, T. C., and Selkov, E. (1999). Mathematical simulation and analysis of cellular metabolism and regulation. *Bioinformatics*, 15(9):749–758.

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 via the World Wide Web at `http://www.cellml.org/public/specification/20010810/cellml_specification.html`.

Hucka, M. (2000). SCHUCS: A notation for describing model representations intended for XML encoding. Available via the World Wide Web at `http://www.sbml.org/`.

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`.

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.

Kirkwood, T. B. L., Boys, R. J., Gillespie, C. S., Proctor, C. J., Shanley, D. P., and Wilkinson, D. J. (2003). Towards an e-biology of ageing: integrating theory and data. *Nature Reviews Molecular Cell Biology*, 4:243–249.

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/`.

McCollum, M. and Lancaster, J. (2003). Biospreadsheet.

Mendes, P. (1997). Biochemistry by numbers: Simulation of biochemical pathways with Gepasi 3. *Trends in Biochemical Sciences*, 22:361–363.

Mendes, P. (2000). New research software to simulate biochemical processes. Available via the World Wide Web at `http://www.vbi.vt.edu/pr/press_releases/press_20001218_news_new-software.htm`.

Mendes, P. (2001). Gepasi 3.21. Available via the World Wide Web at `http://www.gepasi.org`.

Minch, E., de Rinaldis, M., and Weiss, S. (2003). pathSCOUT: exploration and analysis of biochemical pathways. *Bioinformatics*, 19(3):431–432.

Morton-Firth, C. J. and Bray, D. (1998). Predicting temporal fluctuations in an intracellular signalling pathway. *Journal of Theoretical Biology*, 192:117–128.

Oestereich, B. (1999). *Developing Software with UML: Object-Oriented Analysis and Design in Practice*. Addison-Wesley Publishing Company.

Peterson, G. D. and Drager, S. L. (2003). Accelerating defense applications using high performance reconfigurable computing. In *Proceedings of 2003 Government Microcircuit Applications Conference (GOMAC)*, pages 544–547, Tampa, Florida, USA.

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`.

Sauro, H. M. (2000). Jarnac: A system for interactive metabolic analysis. In Hofmeyr, J.-H. S., Rohwer, J. M., and Snoep, J. L., editors, *Animating the Cellular Map: Proceedings of the 9th International Meeting on BioThermoKinetics*. Stellenbosch University Press.

Sauro, H. M. and Fell, D. A. (1991). SCAMP: A metabolic simulator and control analysis program. *Mathl. Comput. Modelling*, 15:15–28.

Sauro, H. S. (2001). JDesigner: A simple biochemical network designer. Available via the World Wide Web at `http://members.tripod.co.uk/sauro/biotech.htm`.

Schaff, J., Slepchenko, B., and Loew, L. M. (2000). Physiological modeling with the Virtual Cell framework. In Johnson, M. and Brand, L., editors, *Methods in Enzymology*, volume 321, pages 1–23. Academic Press, San Diego.

Schaff, J., Slepchenko, B., Morgan, F., Wagner, J., Resasco, D., Shin, D., Choi, Y. S., Loew, L., Carson, J., Cowan, A., Moraru, I., Watras, J., Teraski, M., and Fink, C. (2001). Virtual Cell. Available via the World Wide Web at `http://www.nrcam.uchc.edu`.

Schilstra, M. and Bolouri, H. (2002). Netbuilder. Available via the World Wide Web at `http://strc.herts.ac.uk/bio/maria/NetBuilder/index.html`.

Shapiro, B. E., Levchenko, A., and Mjolsness, E. (2001). Automatic model generation for signal transduction with applications to MAP-kinase pathways. In Kitano, H., editor, *Foundations of Systems Biology*, chapter 7, pages 145–161. MIT Press.

Shapiro, B. E., Levchenko, A., and Mjolsness, E. D. (2003). Cellerator web site. Available via the World Wide Web at `http://www-aig.jpl.nasa.gov/public/mls/cellerator`.

Smaglik, P. (2000). For my next trick... *Nature*, 407:828–829.

Stelling, J., Ginkel, M., Bettenbrok, K., and Gilles, E. D. (2001). Towards a virtual biological laboratory. In Kitano, H., editor, *Foundations of Systems Biology*, chapter 5, pages 189–212. MIT Press.

Thompson, H. S., Beech, D., Maloney, M., and Mendelsohn, N. (2000). XML Schema part 1: Structures (W3C candidate recommendation 24 October 2000). Available via the World Wide Web at `http://www.w3.org/TR/xmlschema-1/`.

Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J. C., and Hutchison, C. (1999). E-Cell: Software environment for whole cell simulation. *Bioinformatics*, 15(1):72–84.

Tomita, M., Nakayama, Y., Naito, Y., Shimizu, T., Hashimoto, K., Takahashi, K., Matsuzaki, Y., Yugi, K., Miyoshi, F., Saito, Y., Kuroki, A., Ishida, T., Iwata, T., Yoneda, M., Kita, M., Yamada, Y., Wang, E., Seno, S., Okayama, M., Kinoshita, A., Fujita, Y., Matsuo, R., Yanagihara, T., Watari, D., Ishinabe, S., and Miyamoto, S. (2001). E-Cell. Available via the World Wide Web at `http://www.e-cell.org/`.

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

Vass, M., Shaffer, C. A., Tyson, J. J., Ramakrishnan, N., and Watson, L. T. (2003). The jigcell model builder: A tool for modeling intra-cellular regulatory networks. *Bioinformatics*. In Press.

W3C (2000a). Naming and addressing: URIs, URLs, ... Available 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/`.