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.cds.caltech.edu/erato

Principal Investigators: John Doyle and Hiroaki Kitano

SBML Level 2, Version 1

Working Draft Revision 3

April 17, 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 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.

Major releases of SBML are termed *levels*. SBML Level 2 evolved out of SBML Level 1 (Hucka et al., 2001). 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, a valid SBML Level 1 document is not a valid SBML Level 2 document or vice versa.

SBML Level 2 was created in part studying the modeling facilities provided by the following software systems: *BioSpice* (Arkin, 2001), *Cellerator* (Shapiro et al., 2000, 2001, 2003), *COPASI* (Mendes, 2000), *DBSolve* (Goryanin, 2001; Goryanin et al., 1999), *E-Cell* (Tomita et al., 1999, 2001), *Gepasi* (Mendes, 1997, 2001), *Jarnac* (Sauro, 2000; Sauro and Fell, 1991), *JDesigner* (Sauro, 2001), *JigCell* (Vass et al., 2003), *NetBuilder* (Schilstra and Bolouri, 2002), *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 was developed with the help of the authors of these packages and with help and collaboration from the authors of CellML (Hedley et al., 2001).

This SBML Level 2 specification document, the XML Schema corresponding to SBML Level 2, and other related documents are openly available from the SBML project web site, `http://www.sbml.org/`.

1.1 Scope and Limitations

SBML Level 2 is meant to support non-spatial biochemical models and the kinds of operations that are possible in existing analysis/simulation tools. Future software tools will undoubtedly require further evolution of SBML; we expect that subsequent levels will add additional structures and facilities currently missing from Level 2, once the simulation community gains 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 Differences between Level 1 Version 1 and Level 2

Compared to SBML Level 1 Version 1, 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. See Section 3.1.
- A new field called `id` replaces the `name` field previously defined for most SBML structures to identify components in a model. The new `id` field has a type of `SId`, whose definition is similar to `SName` in Level 1; see Section 3.3. In SBML Level 2, the `name` field has become optional and has been redefined to allow any Unicode characters allowed by the `string` type of XML Schema (Biron and Malhotra, 2000). The combination of `id` and `name` allows software tools to use meaningful names for components in a model, while simultaneously preserving the ability of software tools that cannot display special characters to display and manipulate *some* sort of labels for those components.

- Formulas in Level 2 are expressed using MathML (W3C, 2000b). The field named `formula` previously available on the `KineticLaw` and `Rule` structures has been replaced by a MathML element named `math` containing MathML content; see Sections 3.6, 4.8 and 4.9.3. The subset of MathML used in SBML includes logical operators enabling the expression of discontinuous functions, something that was not possible in SBML Level 1.

- All attributes which contain initial conditions or parameter values, including compartment volumes and species concentrations, are optional in Level 2 rather than, as in Level 1, either being mandatory or having default values. A missing attribute value for one of these attributes implies that the value is either unknown, not required for analysis, or should be obtained from an external source; see Sections 4.5, 4.6 and 4.7.

- The top-level `Model` structure can contain an optional list of global function definitions expressed in MathML and organized in new structures of type `FunctionDefinition`. See Sections 3.6 and 4.3.

- The top-level `Model` structure can contain an optional list of events organized in structures of type `Event` which specify the a discrete event can occur during a simulation. See Section 4.10.

- 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 uses using the new `FunctionDefinition` mechanism. Although SBML Level 2 uses a feature of MathML to allow models to refer to two particular built-in entities (a symbol representing time and another symbol representing delay functions), even these entities are not in the same namespace as the model's identifiers. See Section 3.6.2.

- Species, compartments and reactions are each optional in a model; thus, a model does not need to contain species, compartments or reactions to be valid. See section 4.2.

- 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. See Sections 4.5, 4.6 and 4.7.

- SBML Level 2 specifies certain constraints on the form and use of scalar rules in order to prevent such things as algebraic loops. See Section 4.8.

- `Rule` structures have been changed and simplified. Instances of `AssignmentRule` replace instances of `ParameterRule`, `SpeciesConcentrationRule` and `CompartmentVolumeRule`. See Section 4.8.

- A rule is not a substitute for a component definition. For example, a `Compartment` structure for a given identifier must precede an `AssignmentRule` structure for the same identifier.

- A new `listOfModifiers` element has been added to the `Reaction` structure. This list enumerates species that affect a reaction but are neither created nor destroyed by the reaction. Each species occurring in a kinetic law must appear in the list of reactants, list of products or list of modifiers. See Section 4.9.

- A reaction may have no products or no reactants but must have at least one reactant or product. See Section 4.9.

- The word *specie* has been replaced in all instances by *species*.

- 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.

- All elements, including `Sbml` and `listOf_____` elements, are derived from the base type `SBase`. The elements contained inside `SBase` are not derived from `SBase`.

1.3 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 or Java 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 Schemas is explained in a separate document (Hucka, 2000). A summary of the essential points is presented in Appendix A, and examples throughout this document illustrate the approach. We also 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, hypothetical network of biochemical reactions that can be represented in SBML:

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

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

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 function that may be used throughout the rest of the model.

Unit definition: A name for a unit used in the expression of quantities in a model. Units may be supplied in a number of contexts in an SBML model, and it is convenient to have a facility for both setting default units and for allowing combinations of units to be given abbreviated names.

Compartment: A container of finite volume for substances. In SBML Level 2, a compartment is primarily a topological structure with a volume but no geometric qualities.

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. The primary qualities associated with a chemical species in SBML Level 2 are its initial amount and the compartment in which it is located.

Parameter: A quantity that has 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: In SBML, a mathematical expression that is used in combination with the differential equations constructed based on the set of reactions; it can be used to establish constraints between variables in a model, 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 rate laws describing how quickly they take place.

Event: A statement describing an instantaneous, discontinuous transformation of a set of variables of any type (species concentration, compartment volume or parameter value) when some 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 performing 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, the various constructs in SBML and their uses are described in detail. Section 3 first introduces a few basic structures that 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 SBML Level 3 and a discussion of other efforts related to SBML. Appendix B 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 and are useful to discuss before diving into the details of the components provided in SBML Level 2.

3.1 Type `SBase`

Each of the main structures 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 inheritance hierarchy is depicted in Figure 1 on the following page. In addition to this, all elements, including `sbml`, elements used to form fields (e.g. `trigger` on `Event`) and `listOf`_____ elements, are derived from `SBase`. The elements contained inside `SBase` are not derived from `SBase`

The type `SBase` is designed to allow a modeler or a software package to attach arbitrary information to each major component in an SBML model. The definition of `SBase` is presented in Figure 2 on page 7. `SBase` contains three fields, all of which are optional: `metaid`, `notes` and `annotation`.

The `metaid` field is present for supporting metadata annotations using RDF. 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

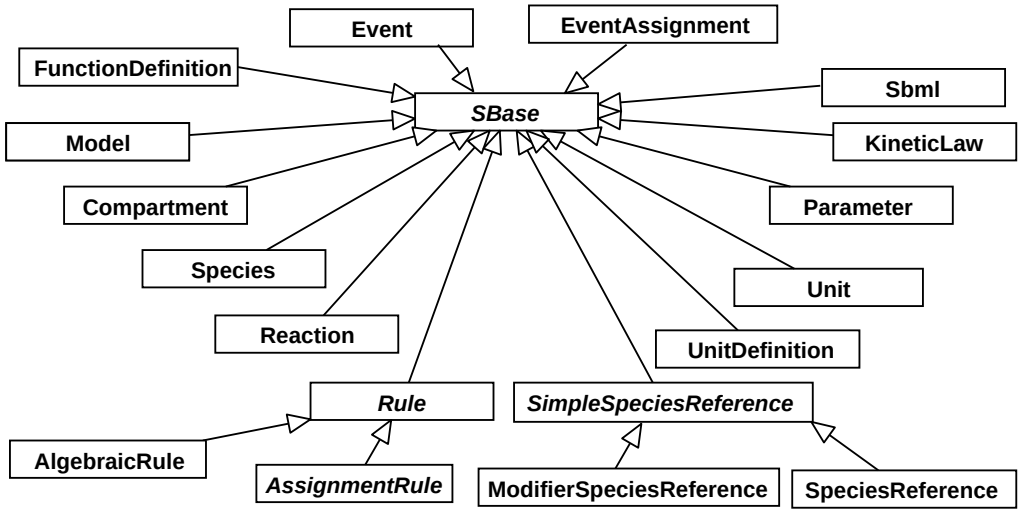

Figure 1: *A UML diagram of the inheritance hierarchy of major data types in SBML. Open arrows indicate inheritance, pointing from inheritors to their parents (Eriksson and Penker, 1998; Oestereich, 1999).*

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).

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. An example of the kind of information that would be appropriate to place inside `notes` is user comments about a particular component of the model. Every data object derived directly or indirectly from type `SBase` can have a separate value for `notes`, allowing users considerable freedom for annotating their models. A software tool that reads and manipulates SBML is expected to provide some mechanism for displaying the contents of `notes` fields in a model.

Finally, `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`. Guidelines for using this field are given in the next section.

In the other SBML 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 different components in an SBML model. However, it is important that this facility not be misused accidentally. In particular, it is critical that information essential to a model definition is *not* stored in `annotation`. Parameter values, functional dependencies between model components, 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 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

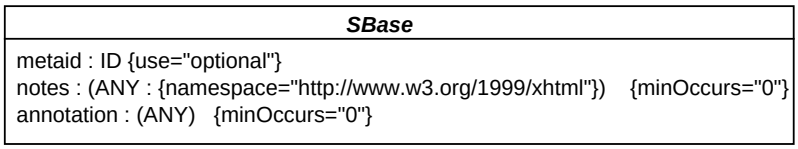

Figure 2: *The definition of* `SBase`*. Text enclosed in braces next to attribute types (e.g.,* {`minOccurs="0"`}*) indicates constraints on the possible attribute values. We use XML Schema language to express constraints since we are primarily interested in the XML encoding of SBML. The constraint expression* `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 that the indicated field is optional; the alternate form must be used in XML Schema for those fields that are containers (i.e., fields that are encoded as subelements in XML).*

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 that the application will use for such annotations, and a prefix string to be used in the annotations. For illustration purposes, let us say the URI reference is "`http://www.mysim.org/ns`" and the prefix is `mysim`. An example of an annotation might then be:

```
...
<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 SBML elements 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 ("`http://www.mysim.org/`" in the example above) 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). 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 the simulation community, we reserve certain namespace names for use with annotations in SBML. These reserved names are listed in Table 1.

Note that the namespaces being referred to here are XML Namespaces specifically in the context of the

`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/mcell`
`http://www.sbml.org/2001/ns/biospice`	`http://www.sbml.org/2001/ns/moma`
`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/stochsim`
`http://www.sbml.org/2001/ns/gepasi`	`http://www.sbml.org/2001/ns/vcell`
`http://www.sbml.org/2001/ns/isys`	`http://www.sbml.org/2001/ns/winscamp`
`http://www.sbml.org/2001/ns/jarnac`	

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

`annotation` field on `SBase`. The namespace issue here is unrelated to the namespaces discussed in Section 3.5 in the context of `SName` and symbols in SBML.

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

As will become apparent below, nearly all structures in SBML include two particular fields: `id` and `name`. The `id` field is required and is used to identify a component within the model definition. Other SBML structures refer to the component using this identifier. The next section defines the data type `SId` used for the `id` field, and Section 3.5 describes the scoping and namespace rules for these identifiers.

In contrast to the `id` field, the `name` field is optional. Its data type is the type `string` defined in XML Schema (Biron and Malhotra, 2000; Thompson et al., 2000), which includes all Unicode characters (Unicode Consortium, 1996) except for two delimiter characters, 0xFFFE and 0xFFFF (Biron and Malhotra, 2000). The purpose of the `name` field is to provide a human-readable label for the component. No restrictions as to its contents are imposed by SBML beyond those defined by the `string` type of 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 special characters), or if the `name` field is missing on a given component, then the user interface should display the value of the `id` field. (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 may wish to take some care to have their software create `id` values that are easy for humans to type and read.

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.

```
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 XML 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".*

The `SId` is purposefully not derived from the XML `ID` type. Doing so 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). If the `IDREF-ID` linkage cannot be exploited in MathML constructs, the utility of the XML `ID` type is greatly reduced.

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 that 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 rate laws. The latter approach means that a user may use the same identifer X in different rate laws 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 identifer 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 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 names exist in a separate global namespace 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 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

Math in SBML Level 2 is expressed using MathML (W3C, 2000b). MathML is used in the definitions of functions (Section 4.3), kinetic laws (Section 4.9.3), and rules (Section 4.8). The `KineticLaw` and `Rule` structures each have a single MathML `math` subelement, and a function definition has a single `lambda` subelement as well. The XML namespace for all of these elements is the URI "`http://www.w3.org/1998/Math/MathML`". [See the W3C document by Bray et al. (1999) for more information about using XML namespaces.]

3.6.1 Subset of MathML Used in SBML Level 2

Only the elements contained in the CellML subset of MathML, with the addition of `csymbol`, can be used within the MathML `math` and `lambda` elements. The subset of MathML used in SBML Level 2 is as follows:

- *token*: `cn`, `ci`, `csymbol`
- *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`, `arccosh`, `arccot`, `arccoth`, `arccsc`, `arccsc`, `arcsec`, `arcsech`, `arcsinh`, `arctanh`
- *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. Its is anticipated that the requirement for partial differential calculus will be addressed in proposals for SBML Level 3 geometry representations (see Section 6.1).

3.6.2 Use of Token Elements in MathML

MathML whitespace rules apply to the content of `ci` elements. The content of `ci` should always be a declared identifier. The set of possible identifiers depends on the containing structure. In the case of math function definitions, the content of `ci` elements is restricted to the declared arguments and previously declared functions. In all other cases, the content of `ci` elements can be identifers of math functions, parameters, compartments or species; i.e., the content should match the value of an `id` field of a component. When a species identifier occurs in a `ci` element, it represents the concentration (i.e., *substance/volume*) of the species. When a compartment identifier occurs in a `ci` element, it represents the volume of the compartment. The units of substance and volume are determined from the built-in `substance` and `volume` of Table 3 on page 14.

SBML Level 2 uses the MathML `csymbol` element to denote standardized math entities without introducing reserved names into the component identifier namespace. The `encoding` field of `csymbol` should be set to `SBML`. 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 14.
- `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.

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

The following examples demonstrates 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="SBML" 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="SBML" definitionURL="http://www.sbml.org/sbml/symbols/delay">
                delay
            </csymbol>
            <ci> x </ci>
            <cn> 0.1 </cn>
        </apply>
    </apply>
</math>
```

Section 5.7 contains a complete model which uses a delay function.

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

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.

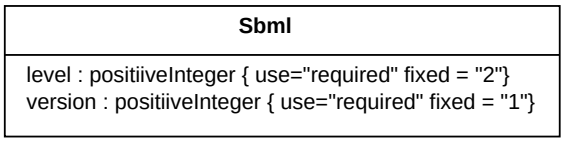

Figure 4: *The definition of `Sbml`. Additional fields are inherited from `SBase` but are not shown here.*

The XML namespace URI for SBML Level 2 is `http://www.sbml.org/sbml/level2`. 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: `version` and `level`. For SBML Level 2 Version 1, these attributes must be set to "1" and "2", 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"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2">
  ...
</sbml>
```

4.2 Models

The `Model` structure is the highest-level construct in an SBML data stream or document. It defines a grouping of components—the list of function definitions, compartments, species, reactions, parameters, rules and unit definitions that define a given model. Only one component of type `Model` is allowed per instance of an SBML Level 2 document or data stream, although it does not necessarily need to represent a single biological entity. The UML definition of the `Model` structure is shown in Figure 5 on the next page.

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.

`Model` serves as a container for `Species`, `Compartment`, `FunctionDefinition`, `UnitDefinition`, `Parameter`,

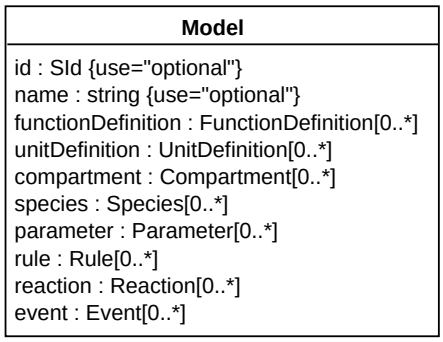

Figure 5: *The definition of* `Model`*. Additional fields are inherited from* `SBase`*.*

`Reaction`, `Rule` and `Event` components. All of these components are optional (the lists in each of the respective fields are permitted to have zero length). 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 that each have headings 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.

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 on the following page.

The `FunctionDefinition` structure has three fields, `id`, `name` and `math`. 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

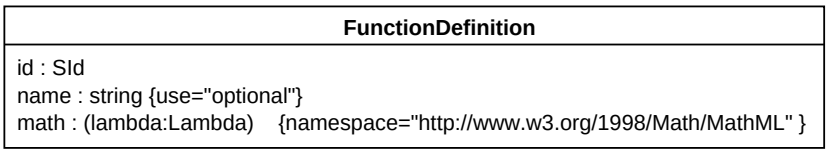

Figure 6: *The definition of* `FunctionDefinition`*. Fields inherited from* `SBase` *are omitted here but are assumed.*

refer to the function defined by a `FunctionDefinition` using the value of its `id` 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. The function is only available for use in other MathML elements that follow the place of its definition in an SBML model. (This restriction is to prevent recursive and mutually-recursive functions from being expressed.)

The following is an abbreviated SBML example. It 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. A facility for defining units is convenient to have so that combinations of units can be given abbreviated names. This is the motivation behind the `UnitDefinition` data structure, whose definition is shown in Figure 7.

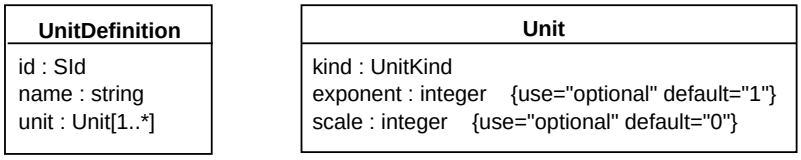

Figure 7: *The definition of* `UnitDefinition`*.*

A unit definition consists of a `id` field of type `SId`, an optional string field `name` and an optional list of structures of type `Unit`. The identifiers defined in the `id` field are in a separate global namespace from identifiers for species, compartments, reactions, events, etc.

The approach to defining units in SBML is compositional; for example, *meter second*$^{-2}$ is constructed by combining a `Unit`-type element representing *meter* with a `Unit`-type element representing *second*$^{-2}$. The `Unit` data structure has a required field, `kind`, whose value must be taken from `UnitKind`, an enumeration of units. The possible values of `UnitKind` are listed in Table 2 on the following page. The `exponent` field on `Unit` represents an exponent on the unit. Its default value is "`1`" (one). In the example just mentioned, *second*$^{-2}$ is obtained by using `kind="second"` and `exponent="-2"`. Finally, the `scale` field in `Unit` is an

integer attribute that scales the unit. For example, a unit that has a `kind` value of "`gram`" and a `scale` value of "−3" signifies $10^{-3} * gram$, or milligrams.

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

Table 2: *The possible values of `kind` in a `UnitKind` structure. All are names of base or derived SI units, except for "`dimensionless`" and "`item`", which are SBML additions important for handling certain common cases. "`Dimensionless`" is intended for cases where a quantity does not have units, and "`item`" is needed in certain contexts to express such things as "N items" (e.g., "100 molecules"). Although "`Celsius`" is capitalized, for simplicity, SBML requires that these names be treated in a case-insensitive manner. Also, note that the gram and liter/litre are not strictly part of SI (Bureau International des Poids et Mesures, 2000); 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.)*

Unit combinations are constructed by listing several `Unit` structures inside a `UnitDefinition`-type structure. 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>
```

There are three special unit names in SBML, listed in Table 3, corresponding to the three types of quantities that play roles in biochemical reactions: amount of substance, volume and time. SBML defines default units for these quantities, all with a default `scale` value of `0`. The various components of a model, such as parameters, can use only the predefined units from Table 2, new units defined in unit definitions, or the three predefined names "`substance`", "`time`", and "`volume`" from Table 3. The latter usage signifies that the units to be used should be the designated defaults.

Name	**Allowable Units**	**Default Units**
`substance`	moles *or* no. of molecules	moles
`volume`	liters	liters
`time`	seconds	seconds

Table 3: *SBML's built-in quantities. Each of these units has a default `scale` value of `0`.*

A model may change the default scales by reassigning the keywords "`substance`", "`time`", and "`volume`" in a unit definition. This takes advantage of the `UnitDefinition` structure's facility for defining scales on units. The following example changes the default units of volume to be milliliters:

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

```
        </unitDefinition>
    </listOfUnitDefinitions>
    ...
</model>
```

If the definition above appeared in a model, the volume scale on all components that did not explicitly use different units would be changed to milliliters.

4.5 Compartments

A `Compartment` represents a bounded container in which species are located. The definition of `Compartment` is shown in Figure 8.

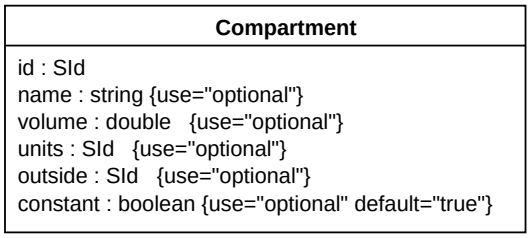

Figure 8: *The definition of `Compartment`. Fields inherited from `SBase` are omitted here but are assumed.*

A `Compartment` data object has an `id` field of type `SId` and an optional `name` field of type `string`. A compartment also has a floating-point field called `volume`, representing the total volume of the compartment in the default units of volume. (See Table 3 on the preceding page.) This enables concentrations of species to be calculated in the absence of geometry information. The `volume` field is optional. A missing `volume` value implies that the value is either unknown, not required for analysis, or available from an external data source. A missing `volume` value does not imply that the compartment volume is 1.

A `Compartment` structure has an optional boolean field called `constant` which indicates whether the compartment's volume stays constant or can vary during a simulation. A value of `false` indicates that the compartment's volume can be determined by rules, and the value of the `volume` field should be taken to be the initial volume of the compartment. The default value for the `constant` field is `true` because in typical modeling scenarios, compartment volumes do not change.

The units of volume may be explicitly set using the optional field `units` in `Compartment`; the named units must be either one of the base units from Table 2 on the page before, the built-in default named `volume`, or a new unit defined by a unit definition in the enclosing model. If absent, the units default to the value set by the built-in `volume` of Table 3.

In an XML data stream containing an SBML model, compartments are listed inside an XML element called `listOfCompartments` within a `Model`-type data structure. (See the discussion of `Model` in Section 4.2.) The following example illustrates two compartments in an abbreviated SBML example of a model definition:

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

On the `Compartment` structure, 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 should be the identifier of the compartment enclosing it, or in other words, the compartment that is "outside" of it. This facility can be used to model cell membranes. For example, to express that a compartment 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" outside="A"/>
        <compartment id="B" outside="M"/>
    </listOfCompartments>
    ...
</model>
```

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. (SBML Level 2 currently does not provide for spatial characteristics aside from compartment volume and containment. As discussed in Section 6.1, we expect that SBML Level 3 will introduce the ability to define geometries and spatial qualities.)

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), and large molecules (e.g., RNA, polysaccharides, and proteins). The `Species` data structure is intended to represent these entities. Its definition is shown in Figure 9.

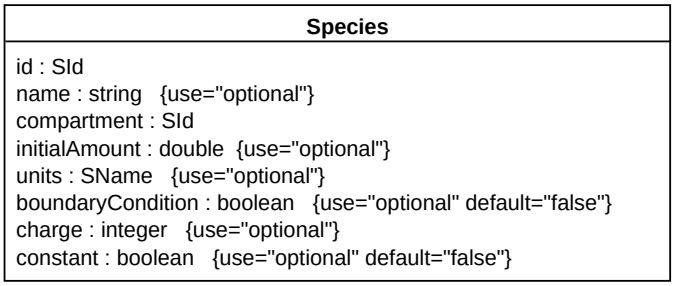

Figure 9: *The definition of `Species`. As usual, fields inherited from `SBase` are omitted here but are assumed.*

`Species` has an `id` field of type `SId` and optional `name` field of type `string`. The field `compartment`, also of type `SId`, is used to identify the compartment in which the species is located. The optional field `initialAmount`, of type `double`, is used to set the initial amount of the species in the named compartment. A missing `initialAmount` value implies that the value is either unknown, not required for analysis or available from an external data source. The units of the substance quantity may be explicitly set using the optional field `units`. The value assigned to `units` must be chosen from one of the following possibilities: one of the unit names from Table 2 on page 14, the name "`substance`", or a new unit name defined by a unit definition in the enclosing model. If absent, the units default to the value set by the built-in "`substance`" of Table 3 on page 14.

The `Species` structure has an optional boolean field named `constant` which indicates 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 rules and reactions.

`Species` also has another optional boolean field, `boundaryCondition`. By default, when a species is a product or reactant of one or more reactions, the concentration of that species 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 optional boolean field `boundaryCondition` indicates that the given species is on the boundary of the reaction system. The value of the field defaults to "`false`", indicating that by default, 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, the `boundaryCondition` attribute means that a differential equation derived from the reaction definitions should not be generated for the species. The example model in section 5.5 contains all 4 possible states of the `boundaryCondition` and `constant` attributes on `species` elements.

`constant` value	`boundaryCondition` value	can have assignment 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.*

The optional field `charge` on `Species` 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 involved is a charged ion such as calcium (Ca^{2+}).

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

4.7 Parameters

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

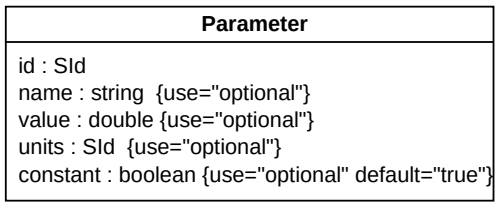

Figure 10: *The definition of `Parameter`. Fields inherited from `SBase` are omitted here but are assumed.*

`Parameter` has an `id` field of type `SId` and an optional `name` field of type `string`. The symbol in the `id` field identifies the parameter. The optional field `value` determines the value (of type `double`) assigned to the identifer. A missing `value` implies that the `value` is either unknown, not required for analysis or available from an external data source. The units of the parameter `value` are specified by the optional field `units`. The value assigned to `units` must be chosen from one of the following possibilities: one of base unit names from Table 2 on page 14; one of the three names "`substance`", "`time`", or "`volume`" (see Table 3); or the name of a new unit defined in the list of unit definitions in the enclosing `Model` structure.

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

Parameters are used in two places in SBML: in lists of parameters defined at the top level in a `Model`-type structure, and within individual reaction definitions. 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.)

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

```
<model>
    ...
    <listOfParameters>
        <parameter id="Km1" value="2.3" units="second"/>
        <parameter id="Km2" value="10.7" units="second"/>
    </listOfParameters>
    ...
</model>
```

4.8 Rules

In SBML, *rules* provide a way to create constraints on variables for cases in which the constraints cannot be expressed using the reaction components (Section 4.9). There are three different possible functional forms of rules, corresponding to the following three general cases (where x is a variable, f is some arbitrary function, and X is the vector of variables that may include species, compartments and parameters):

Algebraic rules, left-hand side is zero:	$0 = f(X)$
Scalar rules, left-hand side is a scalar:	$x = f(X)$
Rate rules, left-hand side is a rate-of-change:	$dx/dt = f(X)$

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

- *Scalar* 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.

In SBML, *scalar* and *rate* rules are collectively referred to as *assignment* rules. SBML uses an abstract `Rule` structure that contains only one field, `math`, to hold the right-hand side expression, then derives subtypes of `Rule` that add fields to distinguish the case of algebraic, scalar 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 two subtypes, `AlgebraicRule` and `AssignmentRule`, defined directly from `Rule`.

The `type` field introduced in `AssignmentRule` is an enumeration of type `RuleType` that determines whether a rule falls into the *scalar* or *rate* categories. In SBML Level 2, the enumeration has two possible values: "`scalar`" and "`rate`". The former means that the expression has a scalar value on the left-hand side [i.e., $x = f(X)$]; the latter means that the expression has a rate of change differential on the left-hand side [i.e. $dx/dt = f(X)$]. Future releases of SBML may add to the possible values of `RuleType`.

4.8.1 `AlgebraicRule`

The rule type `AlgebraicRule` is used to express equations whose left-hand sides are zero. `AlgebraicRule` does not add any fields to the basic `Rule`; its role is simply to distinguish this case from the assignment rule

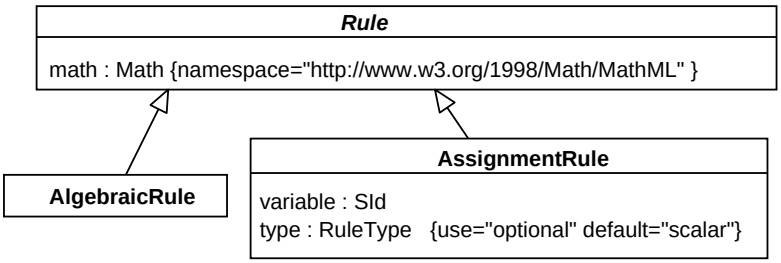

Figure 11: *The definition of* `Rule` *and derived types.*

case. An example of the use of `AlgebraicRule` structures is given in Section 5.4.

4.8.2 `AssignmentRule`

The left-hand sides of assignment rules can be the identifiers of species, compartments, or parameters. The effects are in general terms the same, but have different effects depending on what variable is being set:

- *In the case of a species*: if the value of `type` is "`scalar`", the rule sets the referenced species's concentration to the value determined by the formula in `math`; if the value is "`rate`", the rule sets the rate of change of the species's concentration to the value determined by the formula. The units are in terms of *substance*/*volume*, where the *substance* units are those that are declared on the referenced `Species` element, and the *volume* units are those declared on the `compartment` element that contains the `Species`.

  Restrictions: A rule and a `SpeciesReference` structure (see Section 4.9) cannot both have the same `species` attribute value. This means that a 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*: if the value of `type` is "`scalar`", the rule sets the referenced compartment's volume to the volume determined by the formula in `math`; if the type is "`rate`", the rule sets the rate of change of the compartment's volume to the volume determined by the formula.

  Restrictions: No more than one `AssignmentRule` can refer to a given compartment in an SBML model definition.

- *In the case of a parameter*: if the value of `type` is "`scalar`", the rule sets the referenced parameter's value to that determined by the formula in `math`; if the type is "`rate`", the rule sets the rate of change of the parameter's value to that determined by the formula.

  Restrictions: No more than one `AssignmentRule` can refer to a given parameter.

4.8.3 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 time.

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

A `scalar` rule for a given identifier overrides the initial value of that identifier; i.e., the initial value should be ignored. This does not mean that any structure declaring an identifier can be omitted if there is a `scalar` 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 ordering of `scalar` rules is significant: they are always evaluated in the order given in SBML. The `math` field of a `scalar` rule structure can contain any identifier in a MathML `ci` element except for the following:

(a) identifiers for which there exists a subsequent `scalar` rule, and (b) the identifier for which the rule is defined. These constraints are designed to eliminate algebraic loops among the scalar rules. 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 scalar 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.

Eliminating these algebraic loops ensures that scalar rules can be evaluated any number of times without the result of those evaluations changing.

4.8.4 Example of Rule Use

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

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

This math can be encoded by the following scalar rule set:

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

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

4.9 Reactions

A *reaction* represents any transformation, transport or binding process, typically a chemical reaction, that can transform or 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.

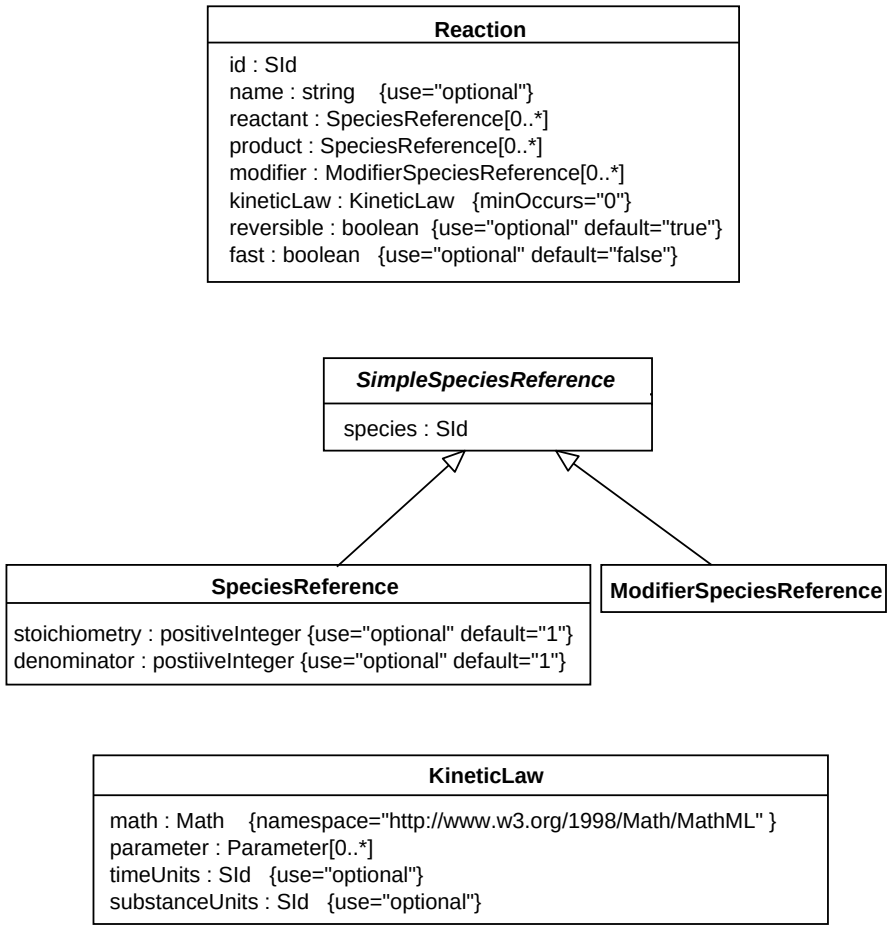

Figure 12: *The definitions of `Reaction`, `KineticLaw`, `SpeciesReference` and `ModiferSpeciesReference`.*

As with the other main structures in SBML, the `Reaction` data structure includes a required `id` field and an optional `name`. These must be used according to the guidelines described in Section 3.3. The reactant species, product species and modifier species in a reaction are described using the fields `reactant`, `product` and `modifiers`. These fields are optional lists of either `SpeciesReference` or `ModifierSpeciesReference`

structures, as shown in Figure 12 on the page before. They are described in more detail in Sections 4.9.1 and 4.9.2 below. The `kineticLaw` is an optional field of type `KineticLaw`, described in more detail in Section 4.9.3 below.

In addition to the above, the `Reaction` structure also has two optional boolean fields. The first 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 the rate law, the need to allow rate laws to be optional leads to the need for a flag indicating reversibility. Information about reversibility in the absence of a kinetic laws 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 law 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 ensure against logical contradictions in the definitions of reactions.

The field `fast` is another boolean attribute in the `Reaction` data structure; a value of "`true`" signifies that the given reaction is a "fast" one. 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. The default value of `fast` is "`false`". (A simulator/analysis package that has no facilities for dealing with fast reactions can ignore this attribute. 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.)

4.9.1 `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` type 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` type 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`-type structure. Stoichiometric numbers for the products and reactants can be specified using two optional attributes on the `speciesReference` element: `stoichiometry` and `denominator`. Both attributes take positive integers as values, and both have default values of "`1`" (one). The absolute value of the stoichiometric number is the value of `stoichiometry` divided by `denominator`, and the sign is implicit from the role of the species (i.e., negative for reactants and positive for products). The use of separate numerator and denominator terms allows a simulator to employ rational arithmetic if it is capable of it, potentially reducing round-off errors and other problems during computations.

The following is a simple example of a species reference 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>
```

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 for more information.

4.9.2 `ModifierSpeciesReference`

In some cases, a species may act as a catalyst or inhibitor of a reaction, and may not appear in the list of reactants or products because it is neither created nor destroyed in that particular reaction. In that case, 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 21. 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`.

The following is a simple example of a modifier species reference in a list of reactants within a reaction named "J1":

```
<model>
    ...
    <listOfReactions>
        <reaction id="J1">
            ...
            <listOfModifiers>
                <modifierSpeciesReference species="X0"/>
            </listOfModifiers>
            ...
        </reaction>
        ...
    </listOfReactions>
    ...
</model>
```

4.9.3 `KineticLaw`

A `kineticLaw` 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.

The field `math` is a MathML element and contains an expression that sets the rate of the reaction, in *substance/time* units. (Section 3.6 discusses the use of MathML in SBML Level 2). The optional fields `substanceUnits` and `timeUnits` determine the units of substance and time. If not set, the units are taken from the defaults defined by the built-in "`substance`" and "`time`" of Table 3 on page 14. The only species identifiers that can be used in `math` are those listed in the `reactant`, `product` and `modifier` fields of the `Reaction` structure.

An instance of a `KineticLaw` type structure can contain zero or more `parameter` structures (Section 4.7) that define symbols that can be used in the `math` element. 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 complete `Reaction` structure that defines a reaction named J_1, in which $X_0 \longrightarrow S_1$ at a rate given by $k_1 X_0 S_2$, and in which S_2 is a catalyst and k_1 is a parameter. It demonstrates the use of species references and the `KineticLaw` structure:

```
<model>
    ...
    <listOfReactions>
        <reaction id="J1">
            <listOfReactants>
                <speciesReference species="X0" stoichiometry="1"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="S1" stoichiometry="1"/>
```

```
            </listOfProducts>
            <listOfModifiers>
                <modifierSpeciesReference species="S2"/>
            </listOfModifiers>
            <kineticLaw>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <times/>
                        <ci> k1 </ci>
                        <ci> X0 </ci>
                        <ci> S2 </ci>
                    </apply>
                </math>
                <listOfParameters>
                    <parameter id="k1" value="0.1"/>
                </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. Both `Event` and `EventAssignment` are derived from `SBase` (see Section 3.1). An example of a model which uses events is given below.

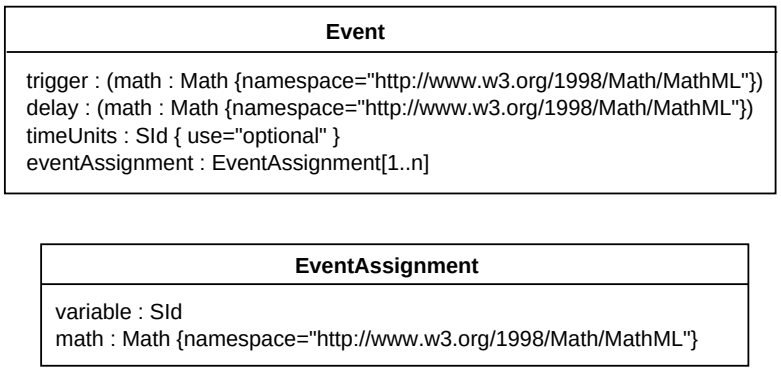

Figure 13: *The definitions of `Event` and `EventAssignment`*

The following sections describe the fields of the `Event` structure.

4.10.1 `trigger`

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.

4.10.2 `delay`

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.

4.10.3 `timeUnits`

The optional field `timeUnits` determines the units of time that apply to the `delay` field. If not set, the units are taken from the defaults defined by the built-in "`time`" of Table 3 on page 14.

4.10.4 `eventAssignment`

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 which 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 which 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 scalar 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*.

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 model that uses events is given in Section 5.8

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:

$$X_0 \xrightarrow{k_1 X_0} S_1$$

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

$$S_1 \xrightarrow{k_3 S_1} X_2$$

The following is an XML document that encodes the model shown above:

```
<?xml version="1.0"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2">
    <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" volume="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="S1" initialAmount="0" compartment="compartmentOne"
                     boundaryCondition="false"/>
            <species id="X0" initialAmount="0" compartment="compartmentOne"
                     boundaryCondition="true"/>
            <species id="X1" initialAmount="0" compartment="compartmentOne"
                     boundaryCondition="true"/>
            <species id="X2" initialAmount="0" compartment="compartmentOne"
                     boundaryCondition="true"/>
        </listOfSpecies>
        <listOfReactions>
            <reaction id="reaction_1" reversible="false">
                <listOfReactants>
                    <speciesReference species="X0" stoichiometry="1"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="S1" stoichiometry="1"/>
                </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" stoichiometry="1"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="X1" stoichiometry="1"/>
                </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" stoichiometry="1"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="X2" stoichiometry="1"/>
                </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 Simple Use of Units Feature in a Model

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 `volume` and `time` built-ins are left to their defaults, ensuring that volume 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"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2"
      xmlns:html="http://www.w3.org/1999/xhtml">
    <model>
        <listOfUnitDefinitions>
            <unitDefinition id="substance">
                <listOfUnits>
                    <unit kind="mole" scale="-3"/>
                </listOfUnits>
            </unitDefinition>
```

```
    <unitDefinition id="mls">
        <listOfUnits>
            <unit kind="mole"   scale="-3"/>
            <unit kind="liter"  exponent="-1"/>
            <unit kind="second" exponent="-1"/>
        </listOfUnits>
    </unitDefinition>
</listOfUnitDefinitions>
<listOfCompartments>
    <compartment id="cell"/>
</listOfCompartments>
<listOfSpecies>
    <species id="x0" compartment="cell" initialAmount="1"/>
    <species id="x1" compartment="cell" initialAmount="1"/>
    <species id="s1" compartment="cell" initialAmount="1"/>
    <species id="s2" compartment="cell" initialAmount="1"/>
</listOfSpecies>
<listOfParameters>
    <parameter id="vm" value="2" units="mls"/>
    <parameter id="km" value="2"/>
</listOfParameters>
<listOfReactions>
    <reaction id="v1">
        <listOfReactants>
            <speciesReference species="x0"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="s1"/>
        </listOfProducts>
        <kineticLaw>
            <notes>
                <html:p>(vm * s1)/(km + s1)</html:p>
            </notes>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <divide/>
                    <apply>
                        <times/>
                        <ci> vm </ci>
                        <ci> s1 </ci>
                    </apply>
                    <apply>
                        <plus/>
                        <ci> km </ci>
                        <ci> s1 </ci>
                    </apply>
                </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)</html:p>
            </notes>
            <math xmlns="http://www.w3.org/1998/Math/MathML">
                <apply>
                    <divide/>
                    <apply>
                        <times/>
                        <ci> vm </ci>
                        <ci> s2 </ci>
                    </apply>
```

```
                        <apply>
                            <plus/>
                            <ci> km </ci>
                            <ci> s2 </ci>
                        </apply>
                    </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)</html:p>
                </notes>
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <divide/>
                        <apply>
                            <times/>
                            <ci> vm </ci>
                            <ci> x1 </ci>
                        </apply>
                        <apply>
                            <plus/>
                            <ci> km </ci>
                            <ci> x1 </ci>
                        </apply>
                    </apply>
                </math>
            </kineticLaw>
        </reaction>
    </listOfReactions>
  </model>
</sbml>
```

5.3 Use of Assignment Rules Feature in a Model

This section contains a model which 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

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

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

this can be approximated with the following system:

$$\begin{array}{ccc} X_0 & \xrightarrow{k_1 X_0} & T \\ T & \xrightarrow{k_2 S_1} & X_1 \end{array}$$

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

The following example SBML model encodes the approximate form.

```
<?xml version="1.0"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2"
      xmlns:math="http://www.w3.org/1998/Math/MathML">
    <model>
        <listOfCompartments>
            <compartment id="cell"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="X0" compartment="cell" initialAmount="1"/>
            <species id="X1" compartment="cell" initialAmount="0"/>
            <species id="T" compartment="cell" initialAmount="0"/>
            <species id="S1" compartment="cell" initialAmount="0"/>
            <species id="S2" compartment="cell" initialAmount="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 Use of Algebraic Rules Feature in a Model

This section contains an example model which 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:

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

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

$$S_2 = K_{eq} S_1$$

with the constraint:

$$S_1 + S_2 - T = 0$$

The following example SBML model encodes this approximate form.

```
<?xml version="1.0"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2">
    <model>
        <listOfCompartments>
            <compartment id="cell"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="X0" compartment="cell" initialAmount="1"/>
            <species id="X1" compartment="cell" initialAmount="0"/>
            <species id="T" compartment="cell" initialAmount="0"/>
            <species id="S1" compartment="cell" initialAmount="0"/>
            <species id="S2" compartment="cell" initialAmount="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 Use of `boundaryCondition` and `constant` attributes on `species` elements in a Model

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

Consider the following hypothetical system:

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

where values over time are determined by:

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

$$S_2 = 1$$

$$S_3 = 2$$

$$k_1 = 0.5$$

$$k_2 = 0.1$$

and initial values are:

$$S_1 = 0$$

$$S_4 = 0$$

S_1 and S_2 are reactants but their values are not determined by a kinetic law thus they are both on the boundary of the reaction system and in SBML have `boundaryCondition` attribute values of `true`. The value of S_1 varies over time so in SBML S_1 has a `constant` attribute with a default value of `false`. The values of S_2 and S_3 are fixed so in SBML they have a `constant` attribute values of `true`. S_3 only occurs as a modifier so the value of its `boundaryCondition` attribute can default to false. S_4 is a product whose value is determined by a kinetic law and therefore in the SBML representation has false values, the default values, for both `boundaryCondition` and `constant` attributes.

The following is the XML document that encodes the model shown above:

```
<?xml version="1.0"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2">
    <model id="x">
        <listOfCompartments>
            <compartment id="compartmentOne" volume="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="S1" initialAmount="0" compartment="compartmentOne"
                boundaryCondition="true" />
            <species id="S2" initialAmount="1" compartment="compartmentOne"
                boundaryCondition="true" constant="true" />
            <species id="S3" initialAmount="3" compartment="compartmentOne"
                constant="true"/>
            <species id="S4" initialAmount="0" compartment="compartmentOne"/>
        </listOfSpecies>
        <listOfParameters>
            <parameter id="k1" value="0.5"/>
            <parameter id="k2" value="0.1"/>
        </listOfParameters>
        <listOfRules>
            <assignmentRule variable="S1" type="rate">
                <math xmlns="http://www.w3.org/1998/Math/MathML">
                    <apply>
                        <ci> k2 </ci>
                    </apply>
                </math>
            </assignmentRule>
        </listOfRules>
        <listOfReactions>
            <reaction id="reaction_1" reversible="false">
                <listOfReactants>
```

```
                    <speciesReference species="S1" stoichiometry="1"/>
                    <speciesReference species="S2" stoichiometry="1"/>
                </listOfReactants>
                <listOfProducts>
                    <speciesReference species="S4" stoichiometry="1"/>
                </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 Use of Function Definition Feature in a Model

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

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

where

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

The following is the XML document that encodes the model shown above:

```
<?xml version="1.0"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2">
    <model id="Branch">
        <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" volume="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="S1" initialAmount="0" compartment="compartmentOne"/>
            <species id="S2" initialAmount="0" compartment="compartmentOne"/>
        </listOfSpecies>
        <listOfReactions>
            <reaction id="reaction_1" reversible="false">
                <listOfReactants>
                    <speciesReference species="S1" stoichiometry="1"/>
                </listOfReactants>
```

```
                <listOfProducts>
                    <speciesReference species="S2" stoichiometry="1"/>
                </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>
```

5.7 Use of the *delay* Function in a Model

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"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2">
    <model>
        <listOfCompartments>
            <compartment id="cell"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="P" compartment="cell" initialAmount="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>
            <assignmentRule variable="P" type="rate">
                <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="SBML"
                                   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>
              </assignmentRule>
          </listOfRules>
      </model>
  </sbml>
```

5.8 Use of Events Feature in a Model

This section contains a simple model system that demonstrates the use of an events. Consider a system with two genes: k_1 and k_2. k_1 is initially on and k_2 is initially off. The genes when on produce products, P_1 and P_2 respectively, at a fixed rate when switched on. When P_1 reaches a given concentration k_2 switches. This can be represented mathematically as follows:

$$\frac{dP_1}{dt} = k_1 - P_1$$

$$\frac{dP_2}{dt} = k_2 - P_2$$

when $P_1 > \tau$ then $k_2 = 1$
when $P_1 \leq \tau$ then $k_2 = 0$

initially

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

The SBML Level 2 representation of this as follows:

```
<?xml version="1.0"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2"
      xmlns:math="http://www.w3.org/1998/Math/MathML">
    <model>
        <listOfCompartments>
            <compartment id="cell"/>
        </listOfCompartments>
        <listOfSpecies>
            <species id="P1" compartment="cell" initialAmount="0"/>
            <species id="P2" compartment="cell" initialAmount="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>
```

```
            <assignmentRule variable="P1" type="rate">
                <math:math>
                    <math:apply>
                        <math:minus/>
                        <math:ci> k1 </math:ci>
                        <math:ci> P1 </math:ci>
                    </math:apply>
                </math:math>
            </assignmentRule>
            <assignmentRule variable="P2" type="rate">
                <math:math>
                    <math:apply>
                        <math:minus/>
                        <math:ci> k2 </math:ci>
                        <math:ci> P2 </math:ci>
                    </math:apply>
                </math:math>
            </assignmentRule>
        </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>
```

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 biochemical network models.

Our vision for SBML is to create an open standard that will enable simulation software to exchange 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 the foundations for modeling biochemical networks. A number of significant capabilities are lacking from Level 2 as described in the main part of this document. Features to implement these capabilities will be introduced in future levels of 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. For example, it should be possible to create a 2-D array of compartments and then a 3-D array of reactions which transport species between the compartments, where the third dimension is the connections between the compartments. Two possible ways of describing a connection scheme are: (1) sparse/explicit, simply listing the relative co-coordinates of connected objects for patterns of points; (2) algebraic, where a conditional equation describes whether two objects are connected.
- *Geometry.* We will develop a scheme for representing the 3-D structure of compartments.
- *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.
- *Component Identification.* This will enable components to be described using some stable universal identification scheme.
- *Diagrams.* This feature will allow components to be annotated with data to enable the display of the model in a diagram.
- *Conditional rules.* This will enable 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:

$$\frac{ds}{dt} = \begin{cases} 0 & \text{if } s > 0 \\ y & \text{otherwise} \end{cases}$$

Conditional rules 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 $s \leq 0$.

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 describes 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 December 2002 at the *6th Workshop on Software Platforms for Systems Biology* at ICSB 2002 in Stockholm.

SBML Level 2 was developed with the help of many people, especially the authors of BioSketchPad, BioSpice, DBSolve, Cellerator, COPASI, E-Cell, 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: Adam Arkin, Hamid Bolouri, Benjamin Bornstein, Dennis Bray, Roger Brent, Claudine Chaouiya, Kwang Cho, Athel Cornish-Bowden, Autumn Cuellar, Serge Dronov, Drew Endy, David Fell, Carl Firth, Ed Frank, Akira Funahashi, Warren Hedley, Charles Hodgman, Stefan Hoops, Martin Ginkel, Victoria Gor, Igor Goryanin, Jay Kaserger, Hiroaki Kitano, Andreas Kremling, Nick Juty, Nicolas Le Novère, Fred Livingston, Les Loew, Daniel Lucio, Joanne Matthews, Pedro Mendes, Eric Minch, Eric Mjolsness, David Morley, Mineo Morohashi, Poul Neilsen, Mark Poolman, Wayne Rindone, Sven Sahle, Takeshi Sakurada, Herbert Sauro, James Schaff, Maria Schilstra, Cliff Shaffer, Bruce Shapiro, Tom Shimizu, Herbert Sauro, Hugh Spence, Jörg Stelling, Kouichi Takahashi, Masaru Tomita, Marc Vass, John Wagner, Jonathan Webb and Olaf Wolkenhauer.

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 attribute 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. 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 list of elements named after the field as the content of the `listOf_____s` element.

Expressions in curly braces ({}) shown after an attribute type indicate additional constraints placed on the field. We express constraints using XML Schema language. In the examples above, the expression {`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 XML Schema for SBML

The following is an XML Schema definition for the Systems Biology Markup Language, using the W3C Recommendation for XML Schema version 1.0 of 2 May 2001 (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000).

```
<?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="$Version$">
    <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
    Modified : $Date: 2003/04/07 15:59:43 $
    Revision: $Id: sbml-level-2.tex,v 1.18 2003/04/07 15:59:43 afinney Exp $

    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="liter"/>
        <xsd:enumeration value="litre"/>
        <xsd:enumeration value="lumen"/>
        <xsd:enumeration value="lux"/>
        <xsd:enumeration value="meter"/>
        <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: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" minOccurs="1" 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" minOccurs="1"/>
            </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="volume" type="xsd:double"/>
            <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="units" type="SId" use="optional"/>
            <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" minOccurs="1"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
```

```
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!--The definition of Rule follows. -->
<xsd:simpleType name="RuleType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="scalar"/>
        <xsd:enumeration value="rate"/>
    </xsd:restriction>
</xsd:simpleType>
<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:attribute name="type" type="RuleType" default="scalar"/>
        </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"
                    minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="SpeciesReference">
    <xsd:complexContent>
```

```
        <xsd:extension base="SimpleSpeciesReference">
            <xsd:attribute name="stoichiometry" type="xsd:positiveInteger" use="optional"
                default="1"/>
            <xsd:attribute name="denominator" type="xsd:positiveInteger" 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" minOccurs="1"
                    maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="Reaction">
    <xsd:complexContent>
        <xsd:extension base="SBase">
            <xsd:sequence>
                <xsd:element name="listOfReactants" type="ListOfSpeciesReferences"
                    minOccurs="0"/>
                <xsd:element name="listOfProducts"  type="ListOfSpeciesReferences"
                    minOccurs="0"/>
                <xsd:element name="listOfModifiers" type="ListOfModifierSpeciesReferences"
                    minOccurs="0"/>
                <xsd:element name="kineticLaw" type="KineticLaw" minOccurs="0"/>
            </xsd:sequence>
            <xsd:attribute name="id" type="SId" use="required"/>
            <xsd:attribute name="name" type="xsd:string" use="optional"/>
            <xsd:attribute name="reversible" type="xsd:boolean" use="optional" default="true"/>
            <xsd:attribute name="fast" type="xsd:boolean" use="optional" default="false"/>
        </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<!--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" minOccurs="1"
                    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"
                        minOccurs="1"/>
                </xsd:sequence>
                <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: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`.

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/`.

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 `ftp://ftp.cds.caltech.edu/pub/caltech-erato/notation/`.

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.cds.caltech.edu/erato`.

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/`.

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`.

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.

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., Levchenko, A., and Mjolsness, E. (2000). Cellerator: A computational technique for automatic model generation of signal transduction pathways. In *Smart Systems 2000*. The Institute for Advanced Interdisciplinary Research, The Atlas Building, 16821 Buccaneer Lane, Suite 206, Houston, TX 77058, USA.

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., Wold, B. J., Meyerowitz, E. M., and Mjolsness, E. D. (2003). Cellerator: Extending a computer algebra system to include biochemical arrows for signal transduction modeling. *Bioinformatics*. In Press.

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/`.