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

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

Principal Investigators: John Doyle and Hiroaki Kitano

SBML Level 2, Version 1, Working Draft

Contents

1 Introduction

We present the **S**ystems **B**iology **M**arkup **L**anguage (SBML) Level 2, a description language for simulations in systems biology. SBML is oriented towards representing biochemical networks common in research on a number of topics, including cell signaling pathways, metabolic pathways, biochemical reactions, gene regulation, and many others. By default SBML models are encoded using XML, the eXtensible Markup Language (Bosak and Bray, 1999; Bray et al., 1998). This document contains many examples of SBML encoded in XML.

Releases of SBML are termed *levels*. SBML Level 2 evolved out of SBML Level 1 (Hucka et al., 2001). The structures of SBML Level 1 can be mapped in a straight forward fashion to SBML Level 2. A large subset of the structures in SBML Level 2 can be mapped to SBML Level 1. A valid SBML Level 1 document is not a valid SBML Level 2 document and vice versa.

SBML Level 2 was the result of studying the features of the following simulation systems: *BioSpice* (Arkin, 2001), *Cellarator* (Shapiro et al., 2000), *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), *NetBuilder* (Schilstra and Bolouri, 2002), *ProMot/DIVA* (Tränkle et al., 1997), *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. In addition SBML Level 2 was developed in close collaboration with the authors of CellML (Physiome Sciences, 2001).

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 the 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

The changes in SBML introduced in Level 2, given Level 1 Version 1 as a starting point, are:

- `formula` attributes on `KineticLaw` and `Rule` elements are replaced by `math` elements as defined in MathML (W3C, 2000b). See Sections 3.6, 4.9.3 and 4.7. All new elements use MathML to describe numerical expressions. The MathML subset includes logical operators that enable the expression of discontinuous expressions.

- New `id` fields replace `name` fields as the fields identifying components in the model. The `id` field has type `SId` (similar to `SName` in level 1). `name` fields remain but are optional and are of type string. See Section 3.3.

- A new list of elements, `listOfModifiers`, is added to the `Reaction` type. This list contains those species that affect the reaction but are neither created nor destroyed by the reaction. See Section 4.9.
- All elements can be annotated with one optional RDF (Lassila and Swick, 1999) element each using the form described in the CellML Metadata specification (Cuellar et al., 2002). See Section 3.1.1. The placement of RDF elements is more restricted than CellML.
- `Model` has an optional list of global function definitions, of type `MathDefinition`, which use MathML. See Section 4.2.
- `Model` has an optional list of `Event` structures which describe the time and form of explicit instantaneous discontinuous state changes in the model. See Section 4.8.
- The main identifier namespace does not contain any built-in symbols. We use a MathML mechanism to refer to two built-in entities: a symbol representing time and a delay function. See Section 3.6.1.
- Unit identifiers are in a separate namespace from the namespace used for models, functions, species, compartments, reactions and parameters.
- `Compartment`, `Species` and `Parameter` structures have boolean `constant` fields. These fields determine whether the variables represented by these structures can be changed by events, rules and reactions. See Sections 4.4, 4.5 and 4.6.
- A model does not have to contain species, compartments or reactions. See section 4.1.
- A reaction can have no products or no reactants but must have at least one reactant or product. See Section 4.9.
- The term ‘specie’ has been replaced by ‘species’ in all element and attribute names. There are no duplicate names that contain ‘specie’.
- A rule is not a substitute for a component definition, for example a `Parameter` structure for a given identifier must precede a `ParameterRule` structure for the same identifier.
- The form of scalar rules is constrained. See Section 4.7.
- `ParameterRule` structures have the field `parameter` (instead of `name`) which contains the identifier for the parameter assigned a rate or value by the rule. See Section 4.7.
- `ParameterRule` structures no longer have a units field. The units of the parameter are given by the corresponding `Parameter` structure. See Section 4.7.

1.3 Notational Conventions

SBML is intended to be a common XML-based format for encoding systems biology models in a simple form that software tools can use as an exchange format. However, for easier communication to human readers, 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 UML 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 biochemical network that can be represented in SBML:

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

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

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

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 math 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 events (optional)*
- *list of reactions (optional)*

end of model definition

The meaning of each component is as follows:

Math definition: A math function definition for use 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 species in SBML Level 2 are its initial amount and the compartment in which it is located.

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.

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 added to the differential equations constructed from the set of reactions and can be used to set parameter values, establish constraints between quantities, etc.

Event: A statement that describes both the exact point in time, and the form of, a set of discontinuous state changes in the model. For example, an event my describe that one species concentration is halved when another species concentration exceeds a given threshold value.

A software package can read a 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 of SBML Level 2. Section 5 provides several complete examples of models encoded in XML using SBML Level 2.

3 Preliminary Definitions

This section covers certain 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 types composing an SBML Level 2 model definition has a specific data type that is derived directly or indirectly from a single base type called `SBase`. This inheritance hierarchy is depicted in Figure 1.

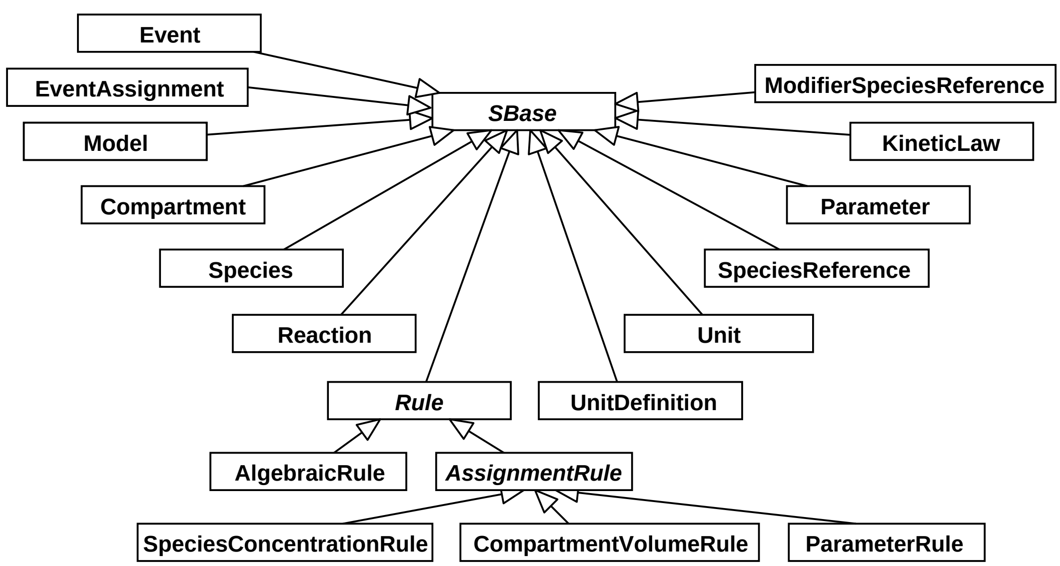

Figure 1: *A UML diagram of the inheritance hierarchy of major data types in SBML. Open arrows indicate inheritance, pointing from inheritors to their parents (Eriksson and Penker, 1998; Oestereich, 1999).*

The definition of `SBase` is presented in Figure 2 on the next page.

3.1.1 Metadata

Elements of type `SBase` can optionally contain a single RDF `rdf` element as their first sub-element. To support RDF SBase has an optional field `metaid` of type `ID`. RDF `description` elements can be created

SBase
metaid : ID {use="optional"} rdf:rdf : (RDF) {minOccurs="0"} notes : (XHTML) {minOccurs="0"} annotation : (any) {minOccurs="0"}

Figure 2: *The definition of* `SBase`*. Text enclosed in braces next to attribute types (i.e.,* {`minOccurs="1"`}*) 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.*

in which the `describes` fields contain the values of the `metaid` fields of SBML elements. The form of the RDF element content should follow the form described in the CellML Metadata Specification (Cuellar et al., 2002) with the restriction that RDF elements can only occur as the first sub-element of any SBase element. This restriction on RDF placement is designed to simplify the parsing of SBML.

3.1.2 Annotations

The type `SBase` is designed to allow a modeler or a software package to attach arbitrary information to each component in an SBML model. `SBase` contains two fields to support the attachment of arbitrary information, both of which are optional: `notes` and `annotations`. The field `notes` is a container for XHTML content. It is intended for recording optional user-visible annotations. 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. The second field, `annotations`, is provided for software-generated annotations. It is a container for arbitrary data (XML type `any`) and is intended to store information not intended for human viewing. As with the user-visible `notes` field, every data object can have its own `annotations` value.

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

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

The `annotations` 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 `annotations`. 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 `annotations`: (a) Information about graphical layout of model components; (b) application-specific processing instructions that do not change the essence of a model; (c) bibliographic information pertaining to a given model; and (d) identification information for cross-referencing components in a model with items in a database. (We expect to introduce an explicit scheme for recording bibliographic information and making database references in SBML Level 3, at which time using annotations for these purposes will become unnecessary.)

Different applications may use XML Namespaces (Bray et al., 1999) to specify the intended vocabulary of a particular annotation. Here is an example of this kind of usage. Suppose that a particular application wants to annotate data structures in an SBML model definition with screen layout information and a time stamp. The application 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 as follows:

```
...
<annotations 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>
```

```
</annotations>
...
```

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 `annotations` itself rather than on a higher-level enclosing construct or the enclosing document level, but other placements would be valid as well (Bray et al., 1999).

The use of XML Namespaces permits multiple applications to place annotations on XML elements of a model without risking interference or element name collisions. Annotations stored by different simulation packages can thus coexist in the same model definition. Although XML Namespace names ("`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 tools developer 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 `annotations` field on `SBase`. The namespace issue here is unrelated to the namespaces discussed in Section 3.5 below in the context of `SName` and symbols in SBML.

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

SBML components include two fields: `id` and `name`. The `id` field is used to identify the component. Other SBML structures refer to a component using this identifier. Section 3.4 describes the form of this field. Section 3.5 describes the scoping and namespace rules for these identifiers.

The `name` field is a XML string field and is optional. The only function of this field is to contain a humanly readable label for the component and thus there are no restrictions as to its content. In user interfaces the `name` field should be displayed to identify the component. The `id` field can be used as an alternative when either the `name` field is not present or the user interface cannot support unconstrained XML strings as component labels. Systems automatically generating the content of `id` field should be aware some parsers may display the results to the user. The `notes` sub-element on `SBase` should be used for containing formatted data to be associated with SBML elements (see Section 3.1).

3.4 Type `SId`

The type `SId` is the type of the `id` field in the majority of component types where the component identifier has global scope (see Section 3.5 for a description of the scoping rules). `SId` is a data type derived from the

`http://www.sbml.org/2001/ns/biospice`
`http://www.sbml.org/2001/ns/dbsolve`
`http://www.sbml.org/2001/ns/ecell`
`http://www.sbml.org/2001/ns/gepasi`
`http://www.sbml.org/2001/ns/jarnac`
`http://www.sbml.org/2001/ns/jdesigner`
`http://www.sbml.org/2001/ns/stochsim`
`http://www.sbml.org/2001/ns/vcell`
`http://www.sbml.org/2002/ns/promot`
`http://www.sbml.org/2002/ns/cellerator`
`http://www.sbml.org/2002/ns/copasi`
`http://www.sbml.org/2002/ns/netbuilder`

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

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 conventional Backus-Naur Form (Naur, 1960). The meta symbols { and } signify "zero or more times" the items they enclose. Note that although XML permits the use of Unicode characters (Unicode Consortium, 1996), SBML limits the allowable characters in `SId` to plain ASCII text characters for compatibility with existing simulation software.*

The `SId` is not derived from the XML `ID` type because that would force all SBML identifiers to exist in a single global namespace. Use of the `ID` type for SBML identifiers would affect the form of local parameter elements and proposed modularity extensions. Identifier namespaces are described in more detail in section 3.5. Use of `ID` type for SBML identifiers has limited utility as MathML `ci` elements aren't of the type `IDREF` (see Section 3.6).

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 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 components 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 Math

Math in SBML Level 2 is expressed using MathML (W3C, 2000b). It is used in the definitions of functions (Section 4.2), kinetic laws (Section 4.9.3), events (Section 4.8) and rules (Section 4.7). The `KineticLaw`, `Rule` and `Event` types have `math` subelements. A function definition has a single MathML `lambda` subelement. The URI `http://www.w3.org/1998/Math/MathML` namespace should be used as the namespace for these MathML elements. A W3C document (Bray et al., 1999) describes the general form for declaring and using XML namespaces.

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 SBML MathML subset 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`
- *calculus* `diff`
- *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 the logical operator, relational operator, `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).

3.6.1 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 specie identifier occurs in a `ci` element, it represents the concentration (i.e., *substance/volume*) of the specie. 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 13.

SBML Level 2 uses the MathML `csymbol` element to represent standardized math entities without introducing built-in identifiers into the component identifier namespace. The `encoding` field of `csymbol` should be set to `SBML`. The `definitionURL` should be set to one member of the set of the predefined SBML symbol URLs. 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.

In SBML Level 2 there are two URLs in the set of predefined SBML symbols:

- `http://www.sbml.org/symbols/time`, which represents the current simulation time. The units of this entity is determined from the built-in `time` of Table 3 on page 13.

  As an example the following XML fragment encodes the equation $x + t$ where t is the built-in symbol for time.

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

- `http://www.sbml.org/symbols/delay`, which represents the function $delay(x, d)$. The result of this function is the value of x at d time units before the current time. The units of the d parameter are determined from the built-in `time` of Table 3 on page 13.

  As an 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>
          <add/>
          <ci> k </ci>
          <apply>
              <csymbol encoding="SBML"
                          definitionURL="http://www.sbml.org/symbols/delay">
                  delay
              </csymbol>
              <ci> x </ci>
              <ci> 0.1 </ci>
          </apply>
      </apply>
  </math>
  ```

  Section 5.5 contains an SBML model which uses delay to represent gene expression.

3.7 Valid Headers for SBML Level 2

An SBML Level 2 XML document consists of a single `sbml` element enclosing a single `model` element. The namespace URI for SBML Level 2 is `http://www.sbml.org/sbml/level2`.

The SBML element has two attributes: `version` and `level`. For the SBML described in this document these attributes should be set to 1 and 2 respectively. As an example a valid header for SBML Level 2 is a follows:

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

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 XML encodings of SBML model components, but we leave full XML examples to Section 5.

4.1 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 math definitions, compartments, species, reactions, parameters, events, rules and unit definitions that define a given model. Only one component of type `Model` is allowed per instance of an SBML document or data stream, although it does not necessarily need to represent a single biological entity. The UML definition of the `Model` structure is shown in Figure 4.

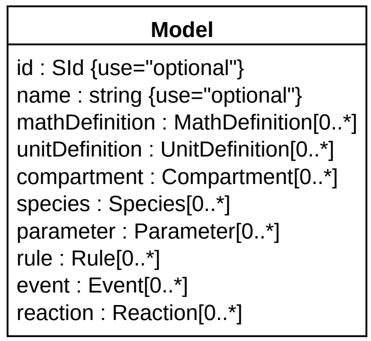

Figure 4: *The definition of `Model`. Additional fields are inherited from `SBase`.*

A `Model` data object may optionally have lists of `Species`, `Compartment`, `MathDefinition`, `UnitDefinition`, `Parameter`, `Reaction`, `Event` and `Rule` components (they are optional because the lists in fields `species`, `compartment`, `mathDefinition`, `unitDefinition`, `parameter`, `reaction`, `event` and `rule` are permitted to have zero length).

A model may also have an optional `id` field that can be used to assign 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. A model also has an optional string field `name`. The `name` and `id` fields should be used as described in section 3.3.

In the XML encoding of an SBML model, the lists of species, compartments, and optional unit definitions, parameters, reactions, math definitions, events and rules, 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">
    <listOfMathDefinitions>
        ...
    </listOfMathDefintions>
    <listOfUnitDefinitions>
        ...
    </listOfUnitDefinitions>
    <listOfCompartments>
        ...
    </listOfCompartments>
    <listOfSpecies>
        ...
    </listOfSpecies>
    <listOfParameters>
        ...
    </listOfParameters>
    <listOfRules>
        ...
    </listOfRules>
    <listOfEvents>
        ...
    </listOfEvents>
    <listOfReactions>
        ...
    </listOfReactions>
```

```
</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.

4.2 Math Definitions

The `MathDefinition` data structure associates an identifier with a function. The function can be used in any subsequent MathML `apply` element. The definition of `MathDefinition` is shown in Figure 5.

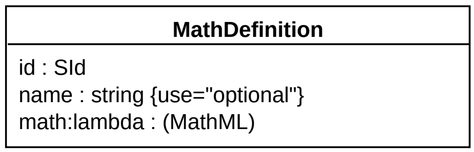

Figure 5: *The definition of* `MathDefinition`*.*

The `MathDefinition` has three fields, `id`, `name` and `math:lambda`. The `id` and `name` fields operate in the manner described in section 3.3. `id` is a `SId` field and `name` is an optional string field. MathML elements can refer to the function contained in a `MathDefinition` using the value of the `id` field. The `math:lambda` field is a MathML `lambda` element which defines the function associated with the `id` field. The function is only available for use in subsequent MathML elements.

The following is an example of a `MathDefinition` element, which defines the function $pow3(x)$ to be x^3:

```
<mathDefinition id="pow3">
    <lamdba xmlns="http://www.w3.org/1998/Math/MathML">
        <bvar><ci> x </ci></bvar>
        <apply>
            <power/>
            <ci> x </ci>
            <cn> 3 </cn>
        </apply>
    </lamdba>
</mathDefinition>
```

In future levels of SBML the set of possible subelements of the MathDefinition could be extended to other MathML elements. In SBML Level 2 simple symbol declarations, other than functions declarations, should be made using `Parameter` structures (see Section 4.6).

4.3 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 6.

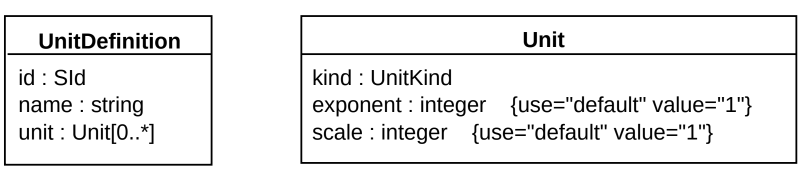

Figure 6: *The definition of* `UnitDefinition`*.*

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 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 `kind` field whose value must be taken from `UnitKind`, an enumeration of SI units. The possible values of `UnitKind` are listed in Table 2. 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`
<u>`dimensionless`</u>	<u>`item`</u>	`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"). Strictly speaking, "`celsius`" should be capitalized; however, for simplicity, SBML requires that the values of `UnitKind` be treated in a case-insensitive manner by software reading and writing SBML. Also, note that the gram and liter/litre are not strictly part of SI (Taylor, 1995); however, they are so useful in SBML's areas of application that they are included in the `UnitKind` enumeration of 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>
```

Many of the components in a model refer to quantities that have associated units. SBML Level 2 has three predefined quantity types: amount of substance, time, and volume. SBML defines default units and scales for these quantities. The defaults are summarized in Table 3.

Name	**Allowable Units**	**Default Units**	**Default Scale**
`substance`	moles or no. of molecules	moles	1
`time`	seconds	seconds	1
`volume`	liters	liters	1

Table 3: *SBML's built-in quantities and their default scale values. The names in the left-hand column are reserved. These names may be used wherever units may be supplied in a model component.*

Wherever unit specifications are permitted in a model (for example, for the volume in a compartment), the relevant built-in name from Table 3 may be used. Such usage signifies that the units to be used for the quantity should be the designated defaults. 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.

The list of unit definitions in a Model-type structure is the only place where new units can be defined. The new unit names may be used anywhere in a model where unit specifications are permitted. The various components of a model, such as reaction parameters and rules, can only use the base units from Table 2, the global unit definitions in the model, or the three predefined keywords "substance", "time", and "volume".

4.4 Compartments

A Compartment represents a bounded container in which species are located. The definition of Compartment is shown in Figure 7.

Compartment
id : SId name : string {use="optional"} volume : double {use="default" value="1"} units : SId {use="optional"} outside : SId {use="optional"} constant : boolean {use="default" value="true"}

Figure 7: *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 XML 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 and defaults to a value of "1" (one).

A Compartment structure has an optional constant boolean field which indicates whether the compartment's volume can vary during the simulation. A false value indicates that the compartment's volume can be determined by rules and events. The default value for the constant field is true.

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 4.1.) 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. Thus, compartments may be adjacent to each other or have other spatial relationships. 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.)

4.5 Species

The term *species* refers to 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 8.

Species
id : SId name : string {use="optional"} compartment : SId initialAmount : double units : SName {use="optional"} boundaryCondition : boolean {use="default" value="false"} charge : integer {use="optional"} constant : boolean {use="default" value="false"}

Figure 8: *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 XML type `string`. The field `compartment`, also of type `SId`, is used to identify the compartment in which the species is located. The field `initialAmount`, of type `double`, is used to set the initial amount of the species in the named compartment. 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 base unit names from Table 2 on page 13, 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 13.

A `Species` structure has an optional `constant` boolean field which indicates whether the concentration of the species can vary during the simulation. A `false` value indicates that the species' concentration can be determined by events, rules and reactions. The default value for the `constant` field is `false`.

By default when a species is a product or reactant of one or more reactions the concentration of a 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. `boundaryCondition` defaults to a value of "`false`", indicating that by default, the species is part of the reaction system. Table 4 shows how the combined values of the `boundaryCondition` and `constant` fields on the `Species` structure should be interpreted. In practice the `boundaryCondition` attribute means that a differential equation, that is derived from the reaction system, should not be generated for the species.

`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: *The Interpretation of the `constant` and `boundaryCondition` attributes on the `Species` structure.*

On the `Species` structure the optional field `charge` is 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>
```

4.6 Parameters

A `Parameter` structure is used to declare a variable for use in MathML structures. Whilst a parameter is a variable, by default it is constant for the duration of a simulation. The definition of `Parameter` is shown in Figure 9.

Parameter
id : SId name : string {use="optional"} value : double units : SId {use="optional"} constant : boolean {use="default" value="true"}

Figure 9: *The definition of `Parameter`.*

`Parameter` has an `id` field of type `SId` and an optional `name` field of XML type `string`. The symbol in the `id` field represents the parameter. The field `value` determines the value (of type `double`) assigned to the identifer. 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 13; 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.

A `Parameter` structure has an optional `constant` boolean field which indicates whether the parameter's value can vary during the simulation. A `false` value indicates that the parameter's value can be determined by rules and events. The default value for the `constant` field is `true`.

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

4.7 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:

Case 1, left-hand side is zero:	$0 = f(X)$
Case 2, left-hand side is a scalar:	$x = f(X)$
Case 3, left-hand side is a rate-of-change:	$dx/dt = f(X)$

The vector of variables consists of the set of species, compartments and parameters.

As written above there is little to distinguish between the first two categories however the second category is constrained to eliminate algebraic loops amongst the rules in that category. There are no constraints on the rules in the first category. The constraints on rules are described in more detail in Section 4.7.5. The distinction between the first two categories is useful for the following reasons:

- given an incomplete vector of variable values, calculated using numerical methods from the rest of the model, the rules in the second category can be simply evaluated to complete the vector
- some simulators don't contain solvers capable of solving unconstrained algebraic equations in the first category,
- those simulators that can solve these algebraic equations do make a distinction between these categories and
- some specialized numeric analyses of Models may only be applicable to models that don't contain rules in the first category

Rules can be categorized by the role of the variable x in the equations above: x can be the name of a compartment (to constrain its volume), the name of a species (to constrain its concentration), or a parameter name (to constrain its value).

The approach taken to covering these cases in SBML is to define an abstract `Rule` structure that contains just one field, `math`, to hold the right-hand side expression, then to derive subtypes of `Rule` that add fields to cover the various cases above. Figure 10 gives the definitions of `Rule` and the subtypes derived from it. The figure shows that `AlgebraicRule` is defined directly from `Rule`, whereas `CompartmentVolumeRule`, `SpeciesConcentrationRule`, and `ParameterRule` are all derived from an intermediate abstract structure called `AssignmentRule`. `AlgebraicRule` represents case 1 above whereas `AssignmentRule` represents cases 2 and 3 above.

The `math` field consists of a MathML `math` subelement.

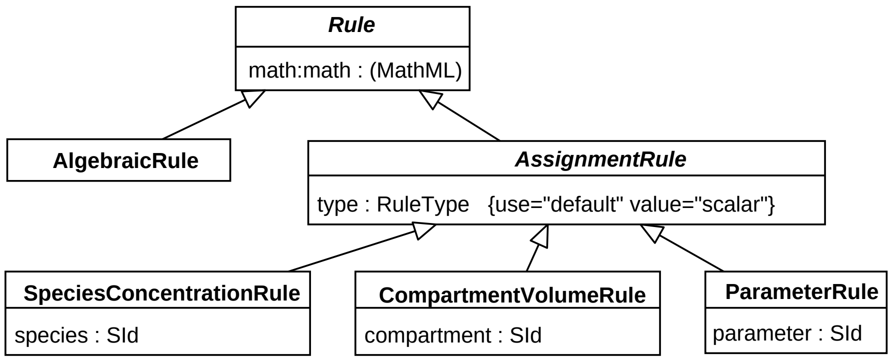

Figure 10: *The definition of `Rule` and derived types.*

The `type` field introduced in `AssignmentRule` is an enumeration of type `RuleType` that determines whether a rule falls into category 2 or 3 in the list of cases above. 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., $x = f(X)$, as in case 2 in the list above); the latter means that the expression has a rate of change differential on the left-hand side (i.e. $\frac{dx}{dt} = f(X)$ as in case 3 in the list above). Future releases of SBML may add to the possible values of `RuleType`.

4.7.1 `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 other cases. An example of the use of `AlgebraicRule` structures is given in Section 5.4.

4.7.2 `SpeciesConcentrationRule`

The `SpeciesConcentrationRule` structure adds one field, `species`, to the basic `AssignmentRule` type. The field `species` has type `SId` and is used to identify the species affected by the rule. The effect of the rule depends on the value of `type`: if the value is "`scalar`", the rule sets the referenced species's concentration to the value determined by the formula; 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`.

A `SpeciesConcentrationRule` structure and a `SpeciesReference` structure (see Section 4.9) cannot 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 specie the `boundaryCondition` field is set to "`true`".

Section 5.3 contains a model with a `SpeciesConcentrationRule` structure.

4.7.3 `CompartmentVolumeRule`

The `CompartmentVolumeRule` structure adds one field, `compartment`, to the basic `AssignmentRule` type. The field `compartment` has type `SId` and is used to identify the compartment affected by the assignment. The effect of the rule depends on the value of `type`: if the type is "`scalar`", the rule sets the referenced compartment's volume to the volume determined by the formula; if the type is "rate", the rule sets the rate of change of the compartment's volume to the volume determined by the formula. No more than one CompartmentVolumeRule can refer to a given compartment.

4.7.4 ParameterRule

The ParameterRule structure adds two fields, parameter to the basic AssignmentRule type. The parameter attribute has type SId and identifies the parameter affected by the assignment. The effect of this rule is to give a value to a parameter that can be used in subsequent formulas. This parameter has the value returned by the expression in the formula attribute. No more than one ParameterRule can refer to a given parameter.

4.7.5 Constraints on rules

No more than one assignment rule can defined for a given identifier. No assignment rule can be defined for an identifier whose corresponding structure has the 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 ParameterRule for that parameter.

The order of scalar rules is significant. scalar rules 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:

- those identifiers for which there exists a subsequent scalar rule and
- the identifier for which the rule is defined

These constraints are designed to eliminate algebraic loops amongst the scalar rules. As an example consider the following math, in the order shown:

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

If this math was 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.

4.7.6 Example of Rule Use

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

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

This math is encoded by the following valid scalar rule set:

```
<model>
    ...
    <listOfRules>
        <parameterRule 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>
        </parameterRule>
        <speciesConcentrationRule species="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>
        </speciesConcentrationRule>
        <compartmentVolumeRule compartment="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>
        </compartmentVolumeRule>
    </listOfRules>
    ...
</model>
```

4.8 Events

An *event* represents an entity which can invoke an instantaneous discontinuous change in the state of the model. An `Event` structure defines when the event can occur, which variables are affected by the event and how those variables are affected. The effect of the event can optionally be delayed after the occurrence of the condition which invokes the event entity. 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 11. An example of a model which uses events is given in Section 5.6.

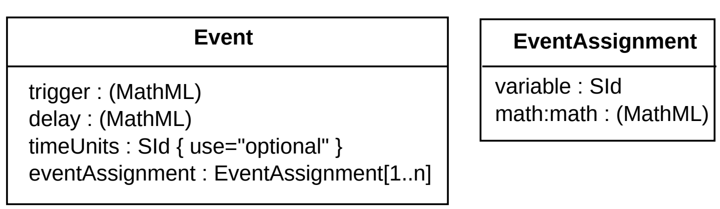

Figure 11: *The definitions of `Event` and `EventAssignment`*

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

4.8.1 `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.

4.8.2 `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.

4.8.3 `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 13.

4.8.4 `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 11. 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:

```
<event>
    <trigger>
        <math:math>
            <math:apply>
                <math:leq/>
                <math:ci> P1 </math:ci>
                <math:ci> t </math:ci>
            </math:apply>
        </math:math>
    </trigger>
    <listOfAssignments>
        <assignment variable="k2">
            <math:cn> 0 </math:cn>
        </assignment>
    <listOfAssignments>
</event>
```

This structure makes the assignment $k_2 = 0$ at the point when $P_1 \leq t$.

4.9 Reactions

A *reaction* represents some transformation, transport or binding process, typically a chemical reaction, that can change the amount of one or more species. The `Reaction` type is defined in Figure 12 on the following page.

In SBML, reactions are defined using lists of reactant species, products, and their stoichiometries, by modifier species and by parameter values for separately-defined kinetic laws. These various quantities are recorded

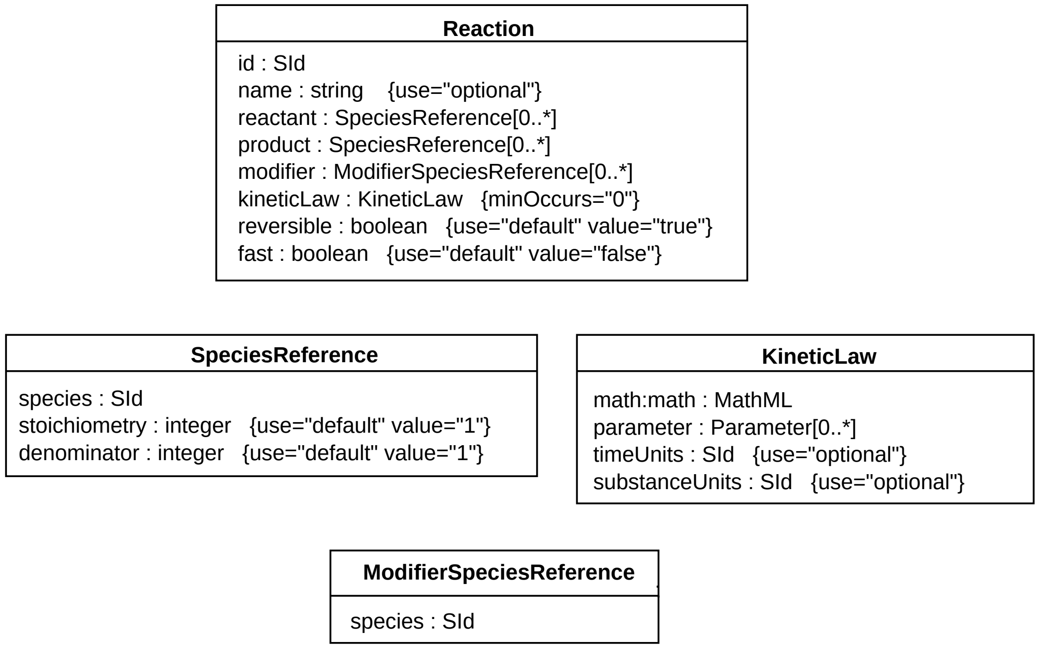

Figure 12: *The definitions of `Reaction`, `KineticLaw` and `SpeciesReference`.*

in the fields `reactant`, `product`, `modifier` and `kineticLaw`. Both `reactant` and `product` are references to species implemented using lists of `SpeciesReference` structures (defined in Section 4.9.1 below). The `SpeciesReference` structure contains fields for recording the names of species and their stoichiometries. `kineticLaw` is an optional field of type `KineticLaw` (defined in Section 4.9.3 below), used to provide a mathematical formula describing the rate of the reaction. `modifier` is a reference to species implemented using lists of `ModifierReference` structures (defined in Section 4.9.2).

In addition to these fields, the `Reaction` structure also has a boolean field, `reversible`, that indicates whether the reaction is reversible. The field is optional, and if left unspecified in a model, it defaults to a value of "`true`". Information about reversibility is useful in certain kinds of structural analyses such as elementary mode analysis.

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`

Each unique species involved in a reaction is listed once in a model, in a list contained in the `species` field of the `Model` data structure discussed in Section 4.1. Lists of modifiers, products and reactants in `Reaction` type structures refer to those species. The connection between the products and reactants in a reaction definition and the species names listed in the enclosing `Model` definition is achieved using the `SpeciesReference` type data structure defined in Figure 12.

The field `species` of type `SId` in `SpeciesReference` must refer to the name of a species defined in the enclosing `Model`-type structure. Refer to Table 4 to determine the attribute values of species that can be referenced by `SpeciesReference` structures.

The two fields `stoichiometry` and `denominator` together set the stoichiometry value for a species in a reaction. Both are integers, and both have default values of "`1`" (one). The use of these separate terms allows a simulator to employ rational arithmetic computations on the stoichiometry matrix, potentially reducing round-off errors and other problems during computations. Such computations are particularly important when working with large matrices and calculating such things as elementary modes.

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.

4.9.2 `ModifierSpeciesReference`

The connection between the modifiers (catalysts and/or inhibitors) in a reaction definition and the species names listed in the enclosing `Model` definition is achieved using the `ModifierSpeciesReference` type data structure defined in Figure 12 on the preceding page. The field `species` of type `SId` in `ModifierSpeciesReference` must refer to the name of a species defined in the enclosing `Model`-type structure.

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

4.9.3 `KineticLaw`

A `kineticLaw` structure describes the rate of the enclosing reaction. The use of a `KineticLaw` structure in a `Reaction` component is optional; however, in general there is no useful default that can be substituted in place of a missing kinetic law definition in a reaction.

The field `math`, a `math` element of MathML, expresses 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 13. The only species identifiers that can be used in the `math` field of the `kineticLaw` structure are those listed in the `Reactant`, `Product` and `Modifier` fields of the `Reaction` structure.

A `KineticLaw` type structure can contain zero or more `parameter` structures (Section 4.6) 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 `Reaction` structure that defines the reaction $J_1 : \ X_0 \longrightarrow S_1; \ k_1 X_0 S_2$ where S_2 is a catalyst. 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>
```

5 Examples of Full Models Encoded in XML Using SBML

In this section, we present several examples of complete models encoded in XML using SBML Level 2. Our approach to translating the UML-based structure definitions presented in the previous sections is described elsewhere (Hucka, 2000).

5.1 A Simple Example Application of SBML

Consider the following hypothetical branched system:

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

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

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

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"
      math:xmlns="http://www.w3.org/1998/Math/MathML">
    <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:math>
                <math:apply>
                    <math:times/>
                    <math:ci> k1 </math:ci>
                    <math:ci> X0 </math:ci>
                </math:apply>
            </math: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:math>
                <math:apply>
                    <math:times/>
                    <math:ci> k2 </math:ci>
                    <math:ci> S1 </math:ci>
                </math:apply>
            </math: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:math>
                        <math:apply>
                            <math:times/>
                            <math:ci> k3 </math:ci>
                            <math:ci> S1 </math:ci>
                        </math:apply>
                    </math:math>
                    <listOfParameters>
                        <parameter id="k3" value="0"/>
                    </listOfParameters>
                </kineticLaw>
            </reaction>
        </listOfReactions>
    </model>
</sbml>
```

The XML encoding shown above is quite straightforward. The outermost container is a tag, `<smbl>`, that identifies the contents as being Systems Biology Markup Language. The attributes `level` and `version` indicate that the content is formatted according to version 1 of the Level 2 definition of SBML. The `version` attribute is present in case SBML Level 2 must be revised in the future to correct errors.

The next-inner container is a single `<model>` element that serves as the highest-level object in the model. The model has a name, "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 includes a `<notes>` annotation that summarizes the model in text form, with formatting based on 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"
      math:xmlns="http://www.w3.org/1998/Math/MathML"
      html:xmlns="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:math>
                    <math:apply>
                        <math:divide/>
                        <math:apply>
                            <math:times/>
                            <math:ci> vm </math:ci>
                            <math:ci> s1 </math:ci>
                        </math:apply>
                        <math:apply>
                            <math:plus/>
                            <math:ci> km </math:ci>
                            <math:ci> s1 </math:ci>
                        </math:apply>
                    </math:apply>
                </math: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:math>
                    <math:apply>
                        <math:divide/>
                        <math:apply>
                            <math:times/>
                            <math:ci> vm </math:ci>
                            <math:ci> s2 </math:ci>
                        </math:apply>
                        <math:apply>
                            <math:plus/>
                            <math:ci> km </math:ci>
                            <math:ci> s2 </math:ci>
                        </math:apply>
                    </math:apply>
                </math: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:math>
                    <math:apply>
                        <math:divide/>
                        <math:apply>
                            <math:times/>
                            <math:ci> vm </math:ci>
                            <math:ci> x1 </math:ci>
                        </math:apply>
                        <math:apply>
                            <math:plus/>
                            <math:ci> km </math:ci>
                            <math:ci> x1 </math:ci>
                        </math:apply>
                    </math:apply>
                </math: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 modelled is

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

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

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 uses the approximate form.

```
<?xml version="1.0"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2"
      math:xmlns="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>
    <speciesConcentrationRule species="S1">
        <math:math>
            <math:apply>
                <math:divide/>
                <math:ci> T </math:ci>
                <math:apply>
                    <math:add/>
                    <math:cn> 1 </math:cn>
                    <math:ci> Keq </math:ci>
                </math:apply>
            </math:apply>
        </math:math>
    </speciesConcentrationRule>
    <speciesConcentrationRule species="S2">
        <math:math>
            <math:apply>
                <math:times/>
                <math:ci> Keq </math:ci>
                <math:ci> S1 </math:ci>
            </math:apply>
        </math:math>
    </speciesConcentrationRule>
</listOfRules>
<listOfReactions>
    <reaction id="in">
        <listOfReactants>
            <speciesReference species="X0"/>
        </listOfReactants>
        <listOfProducts>
            <speciesReference species="T"/>
        </listOfProducts>
        <kineticLaw>
            <math:math>
                <math:apply>
                    <math:times/>
                    <math:ci> k1 </math:ci>
                    <math:ci> X0 </math:ci>
                </math:apply>
            </math: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:math>
                <math:apply>
                    <math:times/>
```

```
                                <math:ci> k2 </math:ci>
                                <math:ci> S2 </math:ci>
                            </math:apply>
                        </math: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:

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

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

$$S_2 = K_{eq} S_1$$

with the constraint:

$$S_1 + S_2 - T = 0$$

The following example SBML model uses this approximate form.

```
<?xml version="1.0"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" version="1" level="2"
      math:xmlns="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>
            <speciesConcentrationRule species="S2">
                <math:math>
                    <math:apply>
                        <math:times/>
                        <math:ci> Keq </math:ci>
                        <math:ci> S1 </math:ci>
                    </math:apply>
                </math:math>
            </speciesConcentrationRule>
            <algebraicRule>
                <math:math>
                    <math:apply>
                        <math:minus/>
                        <math:apply>
                            <math:plus/>
```

```
                        <math:cin> S2 <math:cin/>
                        <math:cin> S1 <math:cin/>
                    </math:apply>
                    <math:cin> T <math:cin/>
                <math:apply>
            <math:math>
        </algebraicRule>
    </listOfRules>
    <listOfReactions>
        <reaction id="in">
            <listOfReactants>
                <speciesReference species="X0"/>
            </listOfReactants>
            <listOfProducts>
                <speciesReference species="T"/>
            </listOfProducts>
            <kineticLaw>
                <math:math>
                    <math:apply>
                        <math:times/>
                        <math:ci> k1 </math:ci>
                        <math:ci> X0 </math:ci>
                    </math:apply>
                </math: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:math>
                    <math:apply>
                        <math:times/>
                        <math:ci> k2 </math:ci>
                        <math:ci> S2 </math:ci>
                    </math:apply>
                </math:math>
                <listOfParameters>
                    <parameter id="k2" value="0.15"/>
                </listOfParameters>
            </kineticLaw>
        </reaction>
    </listOfReactions>
</model>
</sbml>
```

5.5 Use of $delay$ function in a Model

This section contains a simple model which uses the built-in *delay* function described in Section 3.6.1. This model represents a gene that suppresses its own expression and consists of the 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 response time
m	is multiplier or equilibrium constant
q	is 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"
      math:xmlns="http://www.w3.org/1998/Math/MathML">
    <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>
            <speciesConcentrationRule species="P" type="rate">
                <math:math>
                 <math:apply>
                  <math:divide/>
                  <math:apply>
                   <math:minus/
                   <math:apply>
                    <math:divide/>
                    <math:cn> 1 </math:cn>
                    <math:apply>
                     <math:add/>
                     <math:cn> 1 </math:cn>
                     <math:apply>
                      <math:times/>
                      <math:ci> m <math:ci/>
                      <math:apply>
                       <math:power/>
                       <math:apply>
                        <math:csymbol encoding="SBML"
    definitionURL="http://www.sbml.org/symbols/delay">
                            delay
                        </math:csymbol>
                        <math:ci> P </math:ci>
                        <math:ci> delta_t </math:ci>
                       </math:apply>
                       <math:ci> q </math:ci>
                      </math:apply>
                     </math:apply>
                    </math:apply>
                   </math:apply>
                  </math:apply>
                  <math:ci> tau </math:ci>
                 </math:apply>
                </math:math>
            </speciesConcentrationRule>
        </listOfRules>
    </model>
</sbml>
```

5.6 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"
      math:xmlns="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>
            <speciesConcentrationRule species="P1" type="rate">
                <math:math>
                    <math:apply>
                        <math:minus/>
                        <math:ci> k1 </math:ci>
                        <math:ci> P1 </math:ci>
                        </math:apply>
                    </math:apply>
                </math:math>
            </speciesConcentrationRule>
            <speciesConcentrationRule species="P2" type="rate">
                <math:math>
                    <math:apply>
                        <math:minus/>
                        <math:ci> k2 </math:ci>
                        <math:ci> P2 </math:ci>
                        </math:apply>
                    </math:apply>
                </math:math>
            </speciesConcentrationRule>
        </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>
                <listOfAssignments>
                    <assignment variable="k2">
                        <math:cn> 1 </math:cn>
                    </assignment>
```

```
                <listOfAssignments>
            </event>
            <event>
                <trigger>
                    <math:math>
                        <math:apply>
                            <math:leq/>
                            <math:ci> P1 </math:ci>
                            <math:ci> tau </math:ci>
                        </math:apply>
                    </math:math>
                </trigger>
                <listOfAssignments>
                    <assignment variable="k2">
                        <math:cn> 0 </math:cn>
                    </assignment>
                <listOfAssignments>
            </event>
        </listOfEvents>
    </model>
</sbml>
```

5.7 Use of Math Definition Feature in a Model

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

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

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"
      math:xmlns="http://www.w3.org/1998/Math/MathML">
    <model id="Branch">
        <listOfMathDefinitons>
            <mathDefinition id="f">
                <math:lamdba>
                    <math:bvar><math:ci> x </math:ci></math:bvar>
                    <math:apply>
                        <math:times/>
                        <math:ci> x </math:ci>
                        <math:cn> 2 </math:cn>
                    </math:apply>
                </math:lamdba>
            </mathDefinition>
        </listOfMathDefinitions>
        <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:math>
                            <math:apply>
                                <math:ci> f </math:ci>
                                <math:ci> S1 </math:ci>
                             </math:apply>
                        </math:math>
                    </kineticLaw>
                </reaction>
            </listOfReactions>
        </model>
    </sbml>
```

6 Discussion

The volume of data now emerging from molecular biotechnology leave little doubt that extensive computer-based modeling, simulation and analysis will be critical to understanding and interpreting the data (Abbott, 1999; Gilman, 2000; Popel and Winslow, 1998; Smaglik, 2000). This has lead to an explosion in the development of computer tools by many research groups across the world. The explosive rate of progress is exciting, but the rapid growth of the field is accompanied by problems and pressing needs.

One problem is that simulation models and results often cannot be directly compared, shared or re-used, because the tools developed by different groups often are not compatible with each other. As the field of systems biology matures, researchers increasingly need to communicate their results as computational models rather than box-and-arrow diagrams. They also need to reuse published and curated models as library components in order to succeed with large-scale efforts (e.g., the Alliance for Cellular Signaling; Gilman, 2000; Smaglik, 2000). These needs require that models implemented in one software package be portable to other software packages, to maximize public understanding and to allow building up libraries of curated computational models.

We offer SBML to the systems biology community as a suggested format for exchanging models between simulation/analysis tools. SBML is an open model representation language oriented specifically towards representing 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 to SBML: Level 3 and Beyond

As mentioned above, SBML Level 2 is intended to provide the most basic foundations for modeling biochemical networks. A number of significant capabilities are lacking from Level 2; these will be introduced in higher-level definitions of SBML. The following summarizes additional features that 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.
- *Database Interoperability.* In order to store models in a database, it will be necessary to add additional header information that provides information about authors, version numbers, revision dates, etc.
- *Geometry.* We will develop a scheme for representing the 3-D structure of compartments.

- *Submodels.* 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.
- *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. It will also enable multistate representations.
- *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 maths:

  $$\text{if } s > 0 \quad \tfrac{ds}{dt} = y$$

  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 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.

6.3 Tracking the XML Schema Standard

One of the problems in attempting to define an XML Schema for SBML is that, at the time of this writing, the XML Schema specification (Biron and Malhotra, 2000; Thompson et al., 2000) has not actually been finalized. This has been another motivation for defining SBML in terms of abstract data structures in a UML-based notation rather than directly as an XML Schema.

The moving-target status of the XML Schema standard definition requires that we plan to update the Schema corresponding to SBML. The following is our planned approach for handling changes in the Schema standard:

1. The definition of SBML Level 2 in this document is independent of XML Schema. Therefore, the definition of SBML Level 2 expressed here can remain the same regardless of what happens to the exact form of XML Schema. Among other benefits, this allows developers to leave their programs' internal data structures unchanged in the face of possible revisions in the Schema standard.

2. Whenever the definition of XML Schema is updated by the W3C in the future, we will issue a revised version of the XML Schema for SBML Level 2 that conform to the updated standard. We will leave the previous versions still available for reference. The updated XML Schemas for SBML Level 2 will be identical to the previous versions except where changes in XML Schema force a change in the definition of the Schema for SBML Level 2.

6.4 Availability

The SBML Level 2 definition, the XML Schema corresponding to SBML Level 2, and other related documents will be openly available from the Caltech ERATO web site, `http://www.cds.caltech.edu/erato/`.

Acknowledgments

SBML was first conceived at the JST/ERATO-sponsored *First Workshop on Software Platforms for Molecular 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 Molecular 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 Molecular Biology*, held in July 2002, at the University of Hertfordshire, UK. The participants collectively decided to revise the form of SBML in level 2.

SBML Level 2 was developed with the help of many people, especially the authors of BioSpice, DBSolve, Cellerator, COPASI, E-Cell, Gepasi, Jarnac, NetBuilder, Promot/DIVA, StochSim, and Virtual Cell, and members of the `sysbio` mailing list. We are particularly grateful to the following people for discussions and knowledge: Benjamin Bornstein, Dennis Bray, Claudine Chaouiya, Kwang Cho, Athel Cornish-Bowden, Autumn Cuellar, Serge Dronov, David Fell, Carl Firth, Akira Funahashi, Warren Hedley, Charles Hodgman, Stefan Hoops, Martin Ginkel, Victoria Gor, Igor Goryanin, Jay Kaserger, Andreas Kremling, Nick Juty, Nicolas Le Novère, Fred Livingston, Les Loew, Daniel Lucio, Joanne Matthews, Pedro Mendes, Eric Minch, Eric Mjolsness, David Morley, Poul Neilsen, Mark Poolman, Sven Sahle, Takeshi Sakurada, James Schaff, Maria Schilstra, Cliff Shaffer, Bruce Shapiro, Tom Shimizu, Herbert Sauro, Hugh Spence, Joerg Stelling, Kouichi Takahashi, Masaru Tomita, John Wagner 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 : (XHTML)
field4 : float {use = "default" value = "0.0"}
```

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.

A field whose type is shown in parentheses is implemented as an XML subelement rather than an XML attribute. The parentheses indicate that the type refers to the type of the subelement value.

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="default" value="0.0"`} indicates that the field `field4` is optional and that it has a default value of 0.0.

Fields with a name of the form `x:y` indicate that the field is in a separate XML namespace. `x` is the recommended name for the XML namespace. The text description of the field, rather than the diagram will indicate the URI of the XML namespace. `y`, following the XML form, will be the name of the field containing the field data. The type of the field will be a string representing the XML namespace, consistent throughout the document.

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*.

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., and Sperberg-McQueen, C. M. (1998). Extensible markup language (XML) 1.0, W3C recommendation 10-February-1998. Available via the World Wide Web at `http://www.w3.org/TR/1998/REC-xml-19980210`.

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://websites.ntl.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.

Naur, P. (1960). Revised report on the algorithmic language ALGOL 60. *Communications of the ACM*, 3(5):299–314.

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

Physiome Sciences, I. (2001). CellML™ home page. Available via the World Wide Web at `http://cellml.org/`.

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.

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.

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

Taylor, B. N. (1995). Guide for the use of the international system of units (SI). NIST Special Publication 811, 1995 Edition. National Institute of Standards and Technology, United States Department of Commerce. Available via the World Wide Web at `http://physics.nist.gov/Pubs/SP811/contents.html`.

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/`.

Tränkle, F., Gerstlauer, A., Zeitz, M., and Gilles, E. D. (1997). Application of the modeling and simulation environment ProMoT/DIVA to the modeling of distillation processes. *Computers & Chemical Engineering*, 21(Suppl. S):S841–S846.

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

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/`.