SBML Models and MathSBML

Bruce E. Shapiro*, Andrew Finney, Michael Hucka, Benjamin Bornstein, Akira Funahashi, Sarah M. Keating, Nicolas LeNovère, Joanne Matthews, Maria Schilstra

*Corresponding Author: bshapiro@caltech.edu; 626-395-8161; Division of Biology and Biological Network Modeling Center, California Institute of Technology, Mail Code 139-74, Pasadena, CA, USA

Andrew Finney,Ph.D., Physiomics, Inc, Oxford, UK

Michael Hucka, Ph.D. Division of Control and Dynamical Systems and Biological Network Modeling Center, California Institute of Technology, Pasadena, CA USA

Benjamin Bornstein, Machine Learning Systems Group, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA USA

Akira Funahashi, Ph.D., Kitano Symbiotic Systems, Japan

Sarah M. Keating, Science and Technology Research Center, University of Hertfordshire, Hatfield, UK

Nicolas LeNovère, Computational Neurobiology, EMBL-EBI, Wellcome-Trust Genome Campus, Hinxton, UK

Joanne Matthews, Science and Technology Research Center, University of Hertfordshire, Hatfield, UK

Maria Schilstra, Ph.D., Science and Technology Research Center, University of Hertfordshire, Hatfield, UK.

Keywords: SBML, libSBML, MathSBML, Systems Biology, XML, Biomodels

Abstract

MathSBML is an open-source, freely-downloadable *Mathematica* package that facilitates working with Systems Biology Markup Language (SBML) models. SBML is a tool-neutral, computer-readable format for representing models of biochemical reaction networks, applicable to metabolic networks, cell-signaling pathways, genomic regulatory networks, and other modeling problems in systems biology that is widely supported by the systems biology community. SBML is based on XML, a standard medium for representing and transporting data that is widely supported on the internet as well as in computational biology and bioinformatics. Because SBML is tool-independent, it enables model transportability, reuse, publication and survival. In addition to MathSBML, a number of other tools that support SBML model examination and manipulation are provided on the sbml.org website, including libSBML, a C/C++ library for reading SBML models; an SBML Toolbox for MatLab; file conversion programs; an SBML model validator and visualizer; and SBML specifications and schemas. MathSBML enables SBML file import to and export from *Mathematica* as well as providing an API for model manipulation and simulation.

1. Motivation

The Systems Biology Markup Language (SBML) is a tool-neutral, computer-readable, text file (XML) format for representing models of biochemical reaction networks. It is especially applicable to descriptions of cell signaling pathways, metabolic networks, genomic regulatory networks, and other modeling problems in systems biology[1, 2]. SBML is based on XML (the eXtensible Markup Language), a standard medium for representing and transporting data that is widely supported on the Internet [3] as well as in computational biology and bioinformatics (a recent PubMed search on "XML" returned 475 hits; INSPEC 7637 hits; and Web of Science 3009 hits; scholar.google.com, 15,900 hits) [4]. The central goal of SBML is model portability. By encoding models in SBML, they can be freely interchanged between users, regardless of which software tool, hardware platform, or operating system each uses. So long as each modeler uses SBML compliant software, they will both be able to run simulations from the same model, without modification, on their on platform, and compare results.

The benefits of this interoperability are enormous. Not only can users share models, but they can use multiple simulation tools and techniques within a single research project without rewriting their models [5]. Say, for example, that a modeler wants to perform a combination of discrete stochastic and continuous dynamic simulations. Usually this means that he will need to use two different simulation tools. Typically, each software program has a unique model description format that is incompatible with other programs. If both tools are SBML compliant then the model only needs to be encoded once.

A second benefit of this standardization is model publication and dissemination in the peer-reviewed literature. Published models are described in a variety of formats: differential equations, algebraic equations, reactions, pathway diagrams, event rules, etc. If, in addition, the author encodes his model in SBML and makes it available to the publisher (and eventually, the journal's readers) via an auxiliary web site, then computationally astute peer-reviewers can test the models and verify the purported results independently of the authors. Furthermore, when the final paper is published, readers can easily reproduce the same results and incorporate them into and/or compare/contrast them with their own simulations. Journal editors appear to agree; for example, the instructions for authors of Nature Molecular Systems Biology include the statement "Where relevant and possible, authors are encouraged to submit datasets in SBML format." [6]

Finally, SBML can help ensure model survivability [7]. When models are described in unique data formats, particularly when their authors code their own simulation engines, the software model survives only as long as the program is being used. Typically this means that once a student graduates or a post-doctoral researchers moves on to a more permanent positions this technology is lost by the original host institution. If commercial or widely available tools from the public domain are used, on the other hand, models typically only survive until a new version or software release requires a new data format, or more commonly, the program stops being supported on the modeler's preferred hardware/software environment. While there is no guarantee that SBML will always be around, the designers of nearly a hundred different tools are have already made their software SBML compliant or announced an intention to do so in the near future [The growing list is regularly updated on sbml.org].

2. The Evolution of SBML

SBML has been developed through an evolving international collaboration that reflects the wide variety of research being performed in systems biology. Owing to both the geographical diversity as well as the size of this group most discussions have taken place electronically. Moderated (to edit out spam) discussion lists (sbml-discuss, libsbml-discuss) are maintained and archived at http://sbml.org/forums. Over 2500 messages were posted to these lists between Oct. 2002 and Oct.2005; another thousand or so were posted on earlier discussion lists that were combined with Systems Biology Workbench development, and countless other private messages have been sent between list members (for example, the SBML Team has exchanged some 1900 messages amongst itself between Feb 2003 and Oct 2005, and the authors of this chapter another 2000 or so during the same period). These lists currently contain over 200 members coming from academic, commercial and private environments, from all continents.

The ideas developed and discussed through these forums were crystallized through a series of open workshops and working groups starting in 2000, many of which were followed by detailed specification documents or proposals. To date, ten workshops and three "hackathons" have been held in Japan, the US, the UK, Sweden, and Germany (Table 1). Workshops provide a forum for users to become aware of new developments in SBML software, discussion of proposed SBML features so that consensus decisions can be made, and maximizing software interoperability by discussing issues that have arisen in the various implementations. Hackathons, on the other hand, provide a forum for software developers to gather and work simultaneously to solve interoperability issues. The minutes of all workshops and hackathons are available at http://sbml.org/.

The specification of the original language, called SBML Level 1, was released on 2 March 2001. Minor deficiencies and corrections were incorporated in the next release, SBML Level 1, Version 2 (L1V2), on 28 August 2003. Level 1, Version 2 replaces Level 1, Version 1 as it primarily corrects errors in the original document [8]. A major revision, SBML Level 2, Version 1 (L2V1), was released on 28 June 2003 [9]. An initial draft for SBML Level 2, Version (L2V2) was released on 26 March 2005 and posted on the SBML Wiki but has not been finalized. **[Update after Boston Forum]** [10] SBML compliant authors may choose to use either L1V2 or L2V1. All specification documents and related resources are maintained on the web site at http://sbml.org.

The sbml.org web site aims to provide for SBML what w3.org provides to the world wide web. While formal membership in an equivalent consortium is neither required – nor likely practical because of the smaller relative size of the community – the intent is to provide a formal, nonbiased (from the perspective of individual modeling tools) location where specifications, schemas, technical reports, discussion lists, a Wiki, and various online tools can be maintained. The tools provided, such as a validator, visualizer, conversion libraries, and libraries for reading and maintaining SBML files, are designed to aid all modelers in their SBML implementations, and will be the subject of section 5 of this chapter. The group that maintains sbml.org, the "SBML Team," is an international research team distributed at institutions around the world. The SBML Team is not the "keeper" of SBML – that role is for the systems biology community – merely organizers, editors, and fellow tool developers.

3. SBML Level 2 Models

In this section we will review the format of SBML Level 2 Models. Level 1 is omitted due to space limitations as well as the overwhelming (and growing) prevalence of SBML Level 2; the interested reader should consult the references for additional information. The overall structure of an SBML model is

```
beginning of model definition
        list of function definitions
        list of unit definitions
        list of compartment definitions
        list of species
        list of parameters
        list of rules
        list of reactions
        list of events
end of model definition
```

The order of the lists cannot be modified, e.g., *species* must precede *parameters*, etc. SBML models are encoded as XML files; each XML file contains a single "model" object, which is itself enclosed within an `sbml` object (see figure 1). Each of the "lists" is optional; when present it must contain a nonzero number of object definitions of the general form

```
<listOfFoos>
       <foo …> … </foo>
       <foo …> … </foo>
       …
</listOfFoo>
```

where `foo` is one of `functions`, `units`, `compartments`, `species`, `parameters`, `rules`, `reactions`, or `events`. With the exception of `species`, the final "s" is omitted in the individual object definitions; in all cases, the first letter of the object is upper case in the `listOf` definition, and lower case in the individual object definition (hence `listOfFoos` has an upper case “F” and is plural, and `foo` is singular and entirely lower case).

All SBML objects derive from a class SBASE (see figure 2). Class SBASE (and hence all other SBML objects) contains three optional fields: a `metaid`, `notes`, and `annotation`. The `metaid` field is present for supporting metadata annotations using RDF, and has a data type of ID as defined by XML. Other tool users may also choose to use this `metaid` field. The `notes` field is a container for XHTML content. There are no restrictions on what a user may include in this content; however, unlike other fields, which are designed to be read by machines, the notes field is intended to provide a place to store information that can be easily read by humans. Furthermore, when a web browser that does not support non-HTML XML display is used to view an SBML model, it is usually only the notes field that will be visible. Finally, the `annotation` field is a container for software-generate information that is not intended to be read by humans, but nevertheless contains information that cannot otherwise be encoded in SBML that is needed by particular software tools.

Nearly all SBML objects contain the following two fields: `id` and `name`. Most objects that have an `id` field will require that field, which is used to identify the particular instantiation of that object from other instantiations. The value of the id field must be an identifier that begins with a letter and contains only letters, numbers, and the underscore character. SBML is case-sensitive, so that “x” and “X” represent two different identifiers. No two identifiers in the same scope may have the same name; thus no species can have the same name as any compartment. Units are kept in a separate scope, and (as will be seen below) reactions may (optionally) use locally defined parameters that have a local scope. The `name` field is always optional and its value may be any string of Unicode characters.

Several SBML objects allow (or require) mathematical expressions, notably `kineticLaw` (for a reaction), `stoichiometry` (of a species in a reaction), event triggers, event assignments, and rules. All mathematical expressions and formulas are expressed using a subset of MathML [11]. MathML is an XML standard for encoding such expressions in a machine-readable format. MathML contains two flavors: presentation MathML, and content MathML. Presentation MathML is typically used to describe the placement of symbols on a page or a screen, while content MathML is used to describe the mathematical structure of an equation. For example the following expresses $E = mc^2$ in content MathML,

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
 <apply>
  <eq/>
  <ci>E</ci>
  <apply>
   <times/>
   <apply>
    <power/>
    <ci>c</ci>
    <cn type='integer'>2</cn>
   </apply>
   <ci>m</ci>
  </apply>
 </apply>
</math>
```

For the remainder of this chapter, whenever we refer to MathML we will implicitly be referring to that subset of content MathML that is implemented in SBML Level 2 (Table 2). MathML is intended to be both generated and read by computers, and not by human. While short pieces of MathML are readable, the language's verbosity quickly makes it difficult to follow longer expressions. Fortunately there are tools available to perform this translation; for example, in MathSBML (discussed in greater detail below) there are two functions `InfixToMathSBML[`*infix-expression*`]` and `MathMLToInfix[`*MathML-string*`]` that perform the conversion immediately.

A *function definition* associates a named identifier with a MathML lambda object that represents a mathematical function. For example

```
<functionDefinition id="cube">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
    <lambda>
      <bvar><ci> x </ci></bvar>
      <apply>
        <power/>
        <ci> x </ci>
        <cn> 3 </cn>
      </apply>
    </lambda>
  </math>
</functionDefinition>
```

defines a function `cube` that represents the mathematical expression x^3. A later MathML expression could then refer to the function `cube` via the `apply` command.

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
 <apply>
  <ci>cube</ci>
  <ci>x</ci>
 </apply>
</math>
```

A *unit definiton* defines physical units that can be applied to model objects in terms of a default set basic SI units (such as gram, litre, volt, etc.) For example, the user may define a unit “mmls” as millimoles per liter per second:

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

and then give the value of a rate constant, later in the model, in units of mmls, e.g.,

```
<parameter id="K"  value="0.007" units="mmls"/>
```

*Compartment*s are finite sized containers for species. In SBML Level 1, a compartments may be a hierarchy of a topological enclosures with volume but no geometric qualities. For example,

```
<compartment id="Membrane" spatialDimensions="2"/>
<compartment id="Cell" outside="Membrane" size="1"/>
```

defines a compartment "Cell" surrounded by a second compartment "Membrane." In this example Membrane is a two dimensional surface surrounding a 3-dimensional cell. The variable represents the compartment size, which may either be held fixed or allowed to change dynamically (by setting `constant='False'`) in a rule. Besides the topological nesting, no other geometric information is normally encoded in SBML Level 2, although such information could be encapsulated in rules.

Species are any chemical substances that can be measured by quantity or concentration that take part in a reaction. Examples include proteins, nucleic acids, and small molecules such as O2 or ATP:

```
<species id="Glucose" compartment="cell" initialAmount="4" />
```

Other fields allow specifying initial concentration (instead of amount), units, charge, and whether or not the value should be kept constant, held as a boundary condition (allowed to be changed by rules but not by reactions), or variable.

Parameters are constants or variables that do not represent substances. Parameters may be either global, or locally specified within reactions (discussed below); and example was given above. A parameter may be held fixed or allowed to change dynamically (by setting `constant='False'`). The values of dynamic *parameters* may be changed by *rules*, but not by *reactions*. Examples of parameters are rate constants, mass, and physical constants such as Avogadro's number. Rate constants and parameters that are referenced in multiple reactions should be defined globally; a rate constant that is only used in a single reaction should be defined as a local parameter.

Rules are mathematical expressions that describe the dynamics or values of variables. In SBML Level 2 there are three types of rules: assignment rules, rate rules, and algebraic rules. Assignment rules define the value of a parameter (or species) as a mathematical function of other variables in the system. Rate rules define the rate of change (derivative with respect to time) of a variable as a function of other system variables. Algebraic rules express algebraic constraints that should be satisfied by the system, such as $x + y - 7 = 0$. For example, the following defines a rate rule $dk_1 / dt = A / (1 + A)$, followed by an assignment rule $k = k_1 / k_2$ and an algebraic rule $0 = k_1 + k_2 + k_3$

```
<rateRule variable="k1">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
   <divide/>
   <ci>A</ci>
   <apply>
    <plus/>
    <ci>A</ci>
    <cn type="integer">1</cn>
   </apply>
  </apply>
 </math>
 </rateRule>
 <assignmentRule variable="k">
  <math xmlns="http://www.w3.org/1998/Math/MathML">
   <apply>
    <divide/>
    <ci>k1</ci>
    <ci>k2</ci>
   </apply>
  </math>
 </assignmentRule>
<algebraicRule>
 <math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
   <plus/>
   <ci>k1</ci>
   <ci>k2</ci>
   <ci>k3</ci>
  </apply>
 </math>
</algebraicRule>
```

The ordering of rules is critical: a program is expected to evaluate them *in the order listed in the model*. Furthermore, (a) no more than one assignment or rate rule can be defined for any given identifier; (b) assignment rules override any initial conditions for that variable; (c) the math field of a rule can only contain identifiers that have been previously defined, and (d) can not contain either the identifier for which the rule is defined or (e) any element for which there is a subsequent assignment rule.

A *reaction* is a statement describing a transformation, transport or binding process that can change the amount of one or more species. For example,

```
<reaction id="R1">
  <listOfReactants>
    <speciesReference species="X" stoichiometry="1"/>
  </listOfReactants>
  <listOfProducts>
    <speciesReference species="Y" stoichiometry="2"/>
    <speciesReference species="Z" stoichiometry="1"/>
  </listOfProducts>
  <listOfModifiers>
    <modifierSpeciesReference species="A" />
  </listOfModifiers>
  <kineticLaw>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <times/><ci>k</ci><ci>A</ci><ci>X</ci>
      </apply>
    </math>
    <listOfParameters>
      <parameter id="k" value="0.1" />
    </listOfParameters>
  </kineticLaw>
</reaction>
```

represents the reaction $X \xrightarrow{kA} 2Y + Z$. The parameter *k* defined in this example is only defined locally; its existence is unknown outside of the reaction definition (specifically, a separate parameter namespace is defined for each reaction to contain its local parameters). Local parameters may not have the same id as global parameters, although local parameters in different reactions are permitted to have the same id. Kinetic laws can also reference global parameters. Stoichiometries can also be specified with MathML expressions.

Events are explicit instantaneous discontinuous state changes that are triggered as a result of changing conditions within a model. Events specify a *trigger*, the condition that causes the event to occur (e.g., $mass > 1$ and $A < 2$); an *eventAssignment*, an action that occurs as a result of the event's triggering (e.g., set $mass = mass / 2$); and a time *delay* (and associated *timeUnits*) between the occurrence of the trigger and the application of the eventAssignment. For example,

```
<event>
  <trigger>
    <math xmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
        <and/>
        <apply><gt/><ci>mass</ci><cn>1</cn></apply>
        <apply><lt><ci>A</ci><cn>2</ci></apply>
      </apply>
      <apply><leq/><ci> P1 </ci> <ci> t </ci></apply>
    </math>
  </trigger>
  <listOfEventAssignments>
    <eventAssignment variable="mass">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply><divide/><ci>mass</ci><cn>2</cn></apply>
   </math>
   </eventAssignment>
  <listOfEventAssignments>
</event>
```

sets $mass = mass / 2$ when the Boolean expressions $((mass > 1) \wedge (A < 2))$ changes from false to true. The event will only trigger when the condition changes from false to true. If the condition later because false, and then true again, the event will trigger a second time.

4. Proposed modifications to SBML

SBML is intended to meet the evolving needs of the systems biology community. Consequently SBML is being developed in levels, where each higher level adds additional features to the model definitions. These separate levels of SBML are intended to coexist; SBML Level 2 does not render SBML Level 1 obsolete. Software tools that do not need or cannot support higher levels may continue to use lower SBML levels; tools that can read higher levels are assured of also being able to interpret models defined in the lower levels. Minor changes in SBML are called versions; versions within the same level reflect minor changes within that level that were omitted from the earlier version, clarifications of intent and syntax, and typographical corrections to the model specifications.

As errors and omissions are discovered in the specifications they are posted on an errata page at sbml.org. These corrections are then added to the next version of the specification. No new major features were added in SBML Level 1, Version 2; however, it did introduce several variant spellings (e.g., allow both meter and metre; species instead of specie); and a number of typographical errors, earlier omissions, and clarifications were introduced. SBML Level 2, Version 1 did introduce a number of features, notably: events; functions; the use of MathML rather than C for formula strings; id and name fields for most objects; the removal of pre-defined rate laws; spatial dimensions; simplification of rule structure; and the addition of modifiers to a reaction defintion.

Several minor language extensions have been proposed for SBML Level 2, Version 2 [10]. (1) *Nested unit definitions* will allow new units to units may be defined in terms of other units defined in the same model, rather than merely in terms of the base SI units listed in the specification. (2) A new *list of species types* will represent classes of chemical entities independent of their locations; for example, two different species, one in the cytosol and one in the extracellular medium can both be labeled as calcium ions.

(3) A new boolean *constraint rule* will define conditions (e.g., $A + B < C$) under which a model is valid. If a specified constraint is violated then a simulator should halt and print a message indicating that the constraint was violated.

A substantial number of additional changes have been proposed for Level 3 [12]. Because of their greater specialization and the fact that not all modelers will have need for all of these features, it is likely that Level 3 will be modular, in the sense that users will be able to specify at the beginning of a model which Level 3 features the model uses. These features, which are summarized below in alphabetical order, are described in detail on the SBML Wiki at http://sbml.org/wiki.

Alternative reaction extensions would provide the additional data structures that might be required to describe reactions non-deterministically though such features as probability models, markov chains, petric nets, pi-calculus, grammar rules, etc. The present implementation of SBML is based on chemical reactions and rate laws, which lends itself quite well to differential equation formalisms but does not provide the proper set of information required for nondeterministic modeling. These reaction extensions could be closely related to *hybrid model* extensions.

Array and *set* extensions would describe collections of elements (bunches of things that are treated identically in some way) in terms of standard computational data structures such as arrays, vectors, lists, sets, etc. These extensions can interact with the model composition extensions, in that arrays or lists of models could be described. For example, the *Arabidopsis* shoot apical meristem, the hemispherical tip of the growing plant shoot that consists of around 500 cells, could be described by a dynamic array or list of models (or compartments), with new models (or compartments) being instantiated when cells divide, and old models being removed when cells die.

Complex-species extensions would allow models to describe a single species in terms of its different states, such as phosphorylated/non-phosphorylated, or having different numbers or types of ligands bound to different sites. It is related to the level 2 version 2 extension of *species types*, but takes the idea further, in that a species can have both a *type* (e.g., MAP-Kinase) and a *state* (double-phosphorylated).

Constraints would add minimal and/or maximal values for numerical fields; a simulation would then be required to ensure that constraints are met. For example, a geometric constraint would ensure that a species does not leave its compartment; a conservation of mass constraint would require that the sums of certain species are fixed; a parameter constraint would ensure that values are only allowed to range within certain observed physiological values. Constraints could also be used to describe uncertainties in numerical values.

Controlled Vocabulary extensions would provide common terms to describe multiple aspects of the same thing. Different models might used different controlled vocabularies. For example, a reaction might be labeled as “Michaelis-Menten” or “Bi-Uni-Uni-Bi-Pin-Pong-Ter-Ter”, or it might be described as “transcriptional”, “transport”, “activation”, etc; a species might be labeled “substrate” or “catalyst” or “Calcium” or “Sodium.” A controlled vocabulary could also include a mechanism for synonyms, indicating that “niacin”, “niacinamide”, “nicotinic acid”, and “Vitamin_B_3” refer to the same thing in a particular model.

Diagramming or *layout* extensions would allow a model to include specific descriptions of diagrams that describe the model. It would contain lists of graphical representations, or glyphs, of SBML model elements such as compartments, species, and reactions, and information about where to place the different glyphs on an diagram (digital or paper). The actual form of a specific *glyph* – e.g., whether a a species should be represented by a simple black character string or by filled green oval – would be left to the individual tool.

Dynamic model extensions provide ways to enable model structures to vary during a simulation. For example, a dynamic event might trigger cell division and add an additional compartment to the model. Dynamic extensions are closely related to array and model composition extensions.

Hybrid-model extensions would allow different parts of the same model to be described by different formalisms. For example one process could be described by a continuous differential equation, and another could be a discrete markov process. Hybrid models could also involve alternative reaction formalisms and rules that allow dynamic switching between the formalisms for specific processes, constraints that need to be enforced during a simulation, and instantiations of sub-models via model-compositions.

Model Composition would provide the capability to define one SBML model in terms of other models (either in the same file or linked to another file), and include mechanisms for creating a hierarchy of sub-models as "instances" of these models. For example, a model of a cell may contain multiple instances of a model of a mitochondria, with different parameter values, initial conditions, etc, or a tissue model may include various instances of a cell model.

Parameter Set extensions would facilitate the separation of initial conditions and parameter values of a model form the model structure itself. Some aspects are related to the idea of model composition. It is most basic form, a parameter set is a collection of key-value pairs where the key refers to an SBML object attribute; a specific parameter set could then be applied to an existing model, with the appropriate name-value pair substitutions made.

Spatial feature extensions would add geometric characteristics to a model. The only geometric aspects in SBML Level 2 are hierarchies of compartments that are described as being inside or outside of one another, and some aspect of area or volume. Spatial feature extensions could add information ranging from location, and adjacency lists to finite element or spline-models describing the surface shape and features of a compartment.

5. Resources at sbml.org

In the following paragraphs we briefly describe the tools at sbml.org that have been designed to support SBML development and could be of use to nearly all SBML modelers. All of these tools described here are freely accessible at sbml.org.

5.1. Online Tools

The *online tools* enable the user to validate and visualize models in any version or level of SBML and convert Level 1 models to Level 2. The validator checks the model against the SBML XML schema and does limited consistency checks. It is possible for a model that is not valid of SBML to be passed by the tool (because it does not include complete consistence checking) but it will invalidate any model that does not follow the schema.

When a model is validated, the user will be provided with a model summary (e.g., number of each class of SBML object) and given the option to visualize the model or convert it to Level 2 (if the model is in Level 1). Errors are indicated by line number in the original model. The validator and converter are based on libSBML (see section 5.2). Visualization is provided utilizing Graphviz dot combined with an XSLT script, and displays the visualization as a gif image in the web browser. Because of server limitations visualization is limited to models containing 100 or fewer reactions.

5.2. LibSBML

LibSBML is a C/C++ library providing an application programming interface (API) for reading, writing and manipulation data expressed in SBML. LibSBML is a library designed to help read, write, manipulate, translate, and validate SBML files and data streams. It is not an application itself (though it does come with many example programs), but rather a library you can embed in your own applications. While it is implemented in C and C++, it includes Java, Python, Perl, Lisp and MATLAB language bindings in the distribution, and is written in such a way that users can write bindings from virtually any computer language implementation that allows cross-language bindings. The code is very portable and is supported on Linux, native Windows, and Mac OS-X operating systems.

The API (application programming environment) provides an exhaustive list of getters (e.g. species_getInitialAmount), setters/unsetters (species_unsetSpatialSizeUnits), field state booleans (species_isSetCompartment), object getters and creators (UnitDefiniton_addUnit, UnitDefinition_getUnit), enumerators, abstract classes corresponding to every SBML object, including full SBML field inheritence, and so forth. It also provides facilities for reading and writing SBML files, parsing models into abstract syntax trees, and SBML validation.

LibSBML understands all versions of SBML including Level 1, Versions 1 and 2, Level 2 version 1, and the draft SBML Layout Proposal. It is written in portable, pure ISO C and C++ and can be easily ported to nearly any operating system; the build uses GNU tools. It includes support for standard XML libraries, including both Expat and Xerces; and provides full XML schema validation (Xerces only).

5.3. SBML Tools for *MatLab* and *Mathematica*

The SBML Toolbox is a package for working with SBML models in MATLAB. Rather than providing a simulator per-se, the SBMLToolbox provides facilities for converting an SBML model into a MATLAB-accessible format, so that the both standard MATLAB solvers and/or user-developed simulators and libraries can be applied. The toolbox currently includes functions for reading and writing SBML models, converting SBML models into MATLAB data structures, viewing and manipulating those structures, converting them to MATLAB symbolic format, and simulating them using MATLAB's ODE solvers. At present the toolbox includes functions to translate an SBML document into a MATLAB_SBML structure, save and load these structures to/from a MATLAB data file, validate each structure (e.g. reaction structure), view the structures using a set of GUIs and to convert elements of the MATLAB_SBML structure into symbolic form and thus allow access to MATLAB's Symbolic Toolbox. There are a small number of functions to facilitate simulation and a function that will output an SBML document from the MATLAB_SBML structure definition of a model. The toolbox is based on libSBML and requires a prior MATLAB installation. It has been tested in Windows, Linux, Unix, Cygwin and MacOSX. Unix versions require a prior installation of libSBML; this is not required for the Windows version.

MathSBML provides facilities for reading and writing SBML models, converting them to systems of differential equations for simulation and plotting in *Mathematica*, and translating them to other formats. As with the SBMLToolbox, its main purpose is to get models in and out of *Mathematica*, so that the user can apply and or all of the standard features of that language to the SBML Model. MathSBML requires a prior installation of *Mathematica*, and is fully platform independent. MathSBML is the subject of Section 7 of this chapter.

5.4. SBML Conversion Utilities

SBML Conversion utilities provide the ability to convert models described in other modeling languages into SBML. So far we have implemented two different model conversion utilities: KEGG2SML and CELLML2SBML. In addition, the online tools provide conversion form SBML Level 1 models to SBML Level 2 models.

KEGG2SBML is a Perl script that converts KEGG (Kyoto Encyclopedia of Genes and Genomes, http://www.genome.jpg/kegg) Pathway database files to SBML (Systems Biology Markup Language) files using LIGAND database files [13, 14]. It is compatible with all levels and versions of SBML, and includes support for <annotations> tags for CellDesigner. KEGG is a suite of databases and associated software for describing high-order functional behaviors of cells, systems, and organisms, and for relating those behaviors to the organisms' genomes. It includes several databases that describe protein interaction networks (the pathway database); chemical reactions (the ligand database); full-organism networks (gene and SSDB); functional genomic (expression) and proteomic (BRITE) references. Despite the large amount of information (nearly one million different proteins and/or genes are in the gene database, for example) and extensive synonym and cross-links, it provides little or no information for actual reaction mechanisms or rate constants. KEGG2SBML requires Perl 5.6.1, expat, the Perl XML parser (XML::Parser) and libxml-perl, all of which are publicly available; and KEGG pathway database, KGML, and ligand database files that are available at the Kegg website. . It has been tested on FreeBSD and Linux platforms as well as Cygwin under Microsoft Windows.

CellML2SBML converts CellML models [15] to SBML. Like SBML, CellML (see chapter 40 for more details on CellML) is an XML-based modeling language used for storage and exchange of biological models. While there are some common facilities in both languages, the two languages have slightly different goals. In particular, CellML is closely affilitiated with anatomy and finite element modeling languages (AnatML and FieldML). The CellML developers have been involved in the development of the SBML standard, and is currently developing a second tool (SBML2CellML) that will perform the conversion in the opposite direction. CellML2SBML is available for Windows and Linux systems and requires an XSLT processor and four XSLT stylesheets to run.

5.5. Schemas, Specifications, and Test Suites

Full XML schemas (.xsd documents) have been defined for all versions of SBML, and are also included in the download of libSBML as well as the specification documents.

The *SBML Test Suite* is a collection of models and associated automation scripts intended to serve as a test set for developers of SBML-enabled software. It also includes sample models in SBML Level 1 and Level 2 format. Syntactic testing determines if a tool accepts only well-formed SBML and rejects any syntactically incorrect SBML input. This may be accomplished by validation against a full XML-schema. Semantic testing determines if the tool interprets well-formed SBML correctly: does the software construct the correct model and does that model behave correctly. This is usually tested by simulation and comparison with tabular output. The *semantic test suite* at sbml.org includes over 100 tests, including annotated SBML models and tabulated output, as well as an automated script for running the tests against your simulator, so long as the simulator can be invoked either from windows cygwin or a unix command line.

6. The Biomodels Database

The BioModels database [16-18], developed through an international collaboration between the SBML team (US/UK/Japan), EMBL-EBI (UK), the Keck Graduate Institute (US), and Systems Biology Institute (Japan), and JWS online (South Africa), provides access to published peer-reviewed quantitative biological models. The peer-review is provided by the publication process: a model must be published in some peer-reviewed form (e.g., a journal article) before it can be encoded in the database. The original paper's author does not have generate the SBML model, and the model can be described in any language (e.g., differential equations, stochastic, lists of chemical reactions, etc) within the paper, but only SBML (or CellML) models are incorporated within the database. Anybody an submit a model to the database, so long as it has been published and has the appropriate references, but it will not be propagated until the model has been verified by a database curator. Curators verify that the SBML model is valid, well-formed, syntactically correct and correctly represents the referenced publication and that simulations based on these models reproduce (at least some of) the published results. Curators also annotate the components of the models with terms from controlled vocabularies and links to other relevant data resources such as GO (the Gene Ontology database) and links to other databases (such as UniProt, KEGG, and Reactome). This allows the users to search accurately for the models they need and retrieve them in SBML format.

7. Managing SBML with MathSBML

MathSBML [19] is an open-source *Mathematica* package that facilitates working with SBML models. Its primary purpose is to import SBML files into a *Mathematica* data-structure so that users can manipulate the models within *Mathematica* without having to worry about the details of SBML structure. *Mathematica* is one of several platforms widely used by biological modelers and is available in many academic and commercial environments (e.g., over 500 US colleges and universities have site licenses). *Mathematica* is a symbolic computation environment that includes a wide range of features of use to computational biologists, notably numerical computation, graphics, and a programming language. Symbolic computation environments, also known as computer-algebra systems, allow the users to process equations symbolically, using formats that are similar to mathematical equations. From the perspective of computational biologists, this means that reactions and kinetic laws can be expressed in that they are used to, such as $A + B \rightleftarrows C$ or $C'[t] == k_1 A[t]B[t] - k_2 C[t]$. Besides the import feature, MathSBML also includes functions for simulation and plotting of SBML models, including differential-algebraic equations and events; a complete API (Application Programming Environment; see Tables 3 and 4) for manipulating SBML Level 2 models; the ability to display models in human-readable form as annotated html (or within *Mathematica* notebooks); and the ability to export new or modified models back to XML format. A summary of MathSBML commands is given in Table 5.

MathSBML provides full model interoperability with *Mathematica* as well as a candidate reference implementation of SBML. MathSBML will run on any platform that has *Mathematica* 4.1 or higher installed. The solution of differential-algebraic systems (SBML models that have algebraic rules) requires *Mathematica* 5.0 or higher; purely differential systems (SBML without algebraic rules) can be solved on *Mathematica* 4.1. MathSBML is compatible with all levels and versions of SBML released to date, as well as several features proposed for future releases.

7.1 Model Format

Model import is performed using `SBMLRead`. Suppose, for example, that we are interested in modeling the cell cycle, and download the model "Novak1997_CellCycle" from the biomodels database into a local file `BIOMD0000000007.xml`. This file implements a model of DNA replicaton in the fission yeast *Schizosaccharomyces pombe*

[20]. We can read the model into the *Mathematica* computing environment with the command

```
m = SBMLRead["BIOMD0000000007.xml", context → None]
```

returns a *Mathematica* rule list (a standard technique used in *Mathematica* to describe complex data structures) as shown in figure 2. This type of data structure allows the user to access all features of the model directly with *Mathematica*; a more SBML-oriented approach would be to use the model builder, which is described in a later section. A user could get a list of all of the assignment rules in the model, for example, by entering

```
r = SBMLAssignmentRules /.m
```

which will return

```
{IEB[t] == 1 - IE[t], UbEB[t] == 1 - UbE[t], UbE2B[t] == 1 - UbE2[t],
  Wee1B[t] == 1 - Wee1[t], Cdc25B[t] == 1 - Cdc25[t],
  Rum1Total[t] == G1R[t] + G2R[t] + PG2R[t] + R[t],
  Cdc13Total[t] == G2K[t] + G2R[t] + PG2[t] + PG2R[t],
  Cig2Total[t] == G1K[t] + G1R[t],
  k2[t] == 0.0075 (1 - UbE[t]) + 0.25 UbE[t],
  k6[t] == 0.0375 (1 - UbE2[t]) + 7.5 UbE2[t],
  kwee[t] == 0.035 (1 - Wee1[t]) + 0.35 Wee1[t],
  k25[t] == 0.025 (1 - Cdc25[t]) + 0.5 Cdc25[t],
  MPF[t] == G2K[t] + 0.05 PG2[t], SPF[t] == 0.25 G1K[t] + MPF[t]}
```

The third rule can be obtained as `rule3 = r[[3]]`, which would return

```
Rum1Total[t] == G1R[t] + G2R[t] + PG2R[t] + R[t]
```

as the value of the variable `rule3`.

One particularly useful feature of `SBMLRead` is that it constructs the complete set of differential equations that describe the model by combining all of the kinetic laws and rate rules in the model. This set of differential equations is returned as the field `SBMLODES`. `SBMLRead` also returns the stoichiometry matrix as a separate field, and this can be used to simulate models that do not have complete sets of kinetic laws. The corresponding mass-action and mass-balance equations are also generated.

7.2. Variable Scoping and Names

MathSBML attempts to match all identifiers in the *Mathematica* version of the model as closely as possible to the name in the model. In addition, the hierarchies of variable scoping are preserved, e.g., units and reaction parameters are kept in their own namespaces. *Mathematica* represents the scope of a symbol by its context. The context of a variable is indicated by predicating it with a string of characters ending in the back-quote character (normally found to the left of the number 1 on American keyboards).

SBML model variables are defined in a local context; the name of the context is determined by the model "name" in SBML Level 1, and by the model "id" in SBML Level 2. Thus if the SBML model `foo` contains species `A` and `B`, and global parameters `f and k`, they will be represented as ``foo`A``, ``foo`B``, ``foo`f``, and ``foo`k``, respectively. Local parameters `k` and `kf` defined in reactions `R1` and `R2` will become ``foo`R1`k``, ``foo`R1`kf``, ``foo`R2`k``, and ``foo`R2`kf``, respectively. The only character that is allowed in an SBML identifier that is not allowed in a *Mathematica* identifier is the underscore ("_") character. The underscore has a special meaning in *Mathematica* that is used for pattern matching. SBMLRead replaces the underscore character with the

\[UnderBracket] character (Unicode bottom square bracket 9141), which looks like a bracket ("[") turned on its side, with the ends pointing up. The underbracket is translated back to an underscore when a model is written back out as an XML file.

Mathematica contains a number of standard contexts. In particular, any variables that you type in during a *Mathematica* session that do not explicitly include a context are placed in the `` Global` `` context. You do not have to explicitly include the context in Global` variables. Thus the identifiers `A` and `` Global`A `` represent the same variable. You can change the default context form `` Global` `` to something else by changing the value of the *Mathematica* identifier `$Context.`

In SBMLRead, the option `context→None` indicates that the model should be placed in the local context. Thus in the example in the previous section, we had a variable `Cdc13Total` and a global parameter `mu`, would normally be represented as represented as `` NovakTyson1997CellModel`Cdc13Total `` and `` NovakTyson1997CellModel`mu ``. The units of the compartment `Cell` are specified in `litre`, which is represented as `` NovakTyson1997CellModel`Units`litres ``, because units are kept in their own namespace. This particular model does not use any local parameters in the kinetic laws form reactions; however, if we were to add a parameter `k` to the reaction `Cdc25Reaction` it would become `` NovakTyson1997CellModel`Cdc25Reaction`k ``. By using the option `context→CellCycle` in our call to `SBMLRead` these would become `` CellCycle`Cdc13Total ``, `` CellCycle`Units`litre ``, and `` CellCycle`Cdc25Reaction`k. ``

7.3 Simulation and Plotting

Suppose you are interested in running a deterministic simulation of the model that was imported in the previous section. This feat is accomplished with `SBMLNDSolve`, which is a wrapper for the *Mathematica* numerical solver `NDSolve`. To run a simulation of NovakTyson1997CellModel for 400 minutes (the units of time are redefined as minutes in the model) you would enter

```
r = SBMLNDSolve[m,400]
```

The result is returned as a list of interpolation sets that are compatible with *Mathematica* interpolation and plotting functions. If you wanted to write a table of values of the model variables `Rum1Total` and `Cdc13Total` at intervals of 1 minutes from t =150 to t=200 to a comma-separated value file "results.csv,"

```
dt = dataTable[{Rum1Total, Cdc13Total}, {t, 150, 200, 1}, r];
Export["results.csv", dt, "csv"]
```

Other standard output file formats including .dif, .fit, fits, .hdf, .h5, .mat, .mtx, .tsv, .txt, .xls are also supported.

Suppose instead of generating a table of data you want a plot of those same variables:

```
SBMLPlot[r, {Rum1Total, Cdc13Total}]
```

The plot will normally be displayed on the screen and remain embedded in the *Mathematica* notebook. `SBMLPlot` is a wrapper for the *Mathematica* function `Plot`. Any standard plotting options can be specified:

```
p = SBMLPlot[r, {Rum1Total, Cdc13Total},
            PlotStyles → {
                 {Dashing[{.02}], Thickness[.005], Blue},
                 {Thickness[.002], RGBColor[1,0,0]}},
            ImageSize→600,
            TextStyle→{FontFamily→Times, FontSize→18},
            holdLegend→True]
```

will generate a 600-pixel wide image (figure 2) with `Rum1Total` as a thick dashed blue line and `Cdc13Total` as a thinner solid red line. Figures can be exported, e.g., via

```
Export["myplot.jpg",p,"jpg"]
```

many standard graphics types are supported, including bmp, dcm, dic, eps, gif, jpg, pbm, pcx, pdf, pgx, pict, pnm, png, ppm, svg, tif, wmf and xbm file formats.

The MathSBML simulator, `SBMLNDSolve`, is a wrapper for *Mathematica's* `NDSolve`, which in turn evolved from the LSODA, IDA, and DASPK solvers. It incorporates a wide range of methods, including stiff and non-stiff integrators and switching methods and a framework for incorporating external solvers.

Events are implemented by the following algorithm, which ensures that events activate only when an event's trigger changes from false to true. Each event E in an SBML model in has a trigger expression T_E and assignments A_E. We replace the event E, with the following:

- a boolean variable V_E with initial value false
- an event E_1 with trigger $T_{E1} = (\sim V_E) \wedge T_E$ and assignments A_E and $V_E = \mathit{true}$

- an event E_2 with trigger $T_{E2} = (\sim T_E) \wedge V_E$ and assignment $V_E = \mathit{false}$.

The existence of the pseudo-events E_1 and E_2 and new model variable V_E is completely transparent to the user, who is only aware of the existence of the events specified in the model. Events with delays are similarly handled by creating a pseudo-event that triggers once when the specified delay has elapsed. Our cell cycle model actually has two events, one with a delay, and one with multiple assignments:

- Event `start` (the start of S-phase) occurs when `SPF` (S-phase promoting factor) crosses 0.1 from below; after a delay of 60 minutes, the model parameter `kp` is cut in half.

- Event `Division` (cell division) occurs when UbE crosses 0.1 from above. This triggers halving if the parameter `Mass` and doubling of the parameter `kp`.

Events are then detected in *Mathematica* 5.1 (and higher) by throwing and catching the event occurance via the `NDSolve StepMonitor` option; in earlier versions they are detected by the option `StoppingTest`. In all cases the precise event time is found by backward interpolation. If multiple events occur simultaneously all are detected and processed.

7.4. The Model Builder: An SBML API

MathSBML contains a simple model editor, allowing users to create SBML models compatible with other simulators, as well as a *Mathematica* text-command based API that can be used to produce arbitrarily complex SBML files. The model editor contains a suite of commands to add, modify, or remove single SBML objects (such as a reaction, chemical species, or equation) from the current model (Tables 2 and 3). The model may be either created de-novo or read from a file. After building the model, the user can test it by running simulations, continue to modify it, or write the results as an SBML file, in any order.

There is a set of functions `addX`, `modifyX`, and `removeX`, for each class of SBML model object: compartment, event, function, parameter, reaction, rule, species, and unit. Options allow users to specify specific object field values. For example, a partial list of the commands needed to create the cell-cycle model from scratch is illustrated in figure 3. The last step in the box creates an xml file `cellCycle.xml`. A large number of consistency checks are made as the commands are typed to ensure that the SBML specification is satisfied. For example every species must be associated with a compartment; if a compartment is not specified by the `addSpecies` statement, then the most recently referenced compartment is used. If no compartment has been defined yet, a new one is defined.

As the current model is built it is stored internally by MathSBML. At any point in model development, either before or after the xml file has been written, the model can be loaded into the simulator and tested via the `loadSimulator` command. The return value of `loadSimulator` is identical to the return value from `SBMLRead`, and therefore compatible with `SBMLNDSolve`. Similarly, `SBMLRead` will automatically load the model builder whenever a level-2 model is read in, so that it can be modified by add, remove, or modify commands.

Internally, an SBML model is stored as symbolic XML, a standard *Mathematica* data structure for handling XML files. The functions `getX[n]` return the nth object of class X in symbolic XML; the argument may be a number, an id, or a list of both. For example `getReaction[2]` returns the second reaction in the model. The function `XMLOut` is used there to generate the corresponding XML fragment. Other functions `XtoSymbolicSBML` and `XtoSBML` allow one to generate the corresponding symbolic XML or XML fragment for any SBML object.

MathSBML is freely downloadable (LGPL license) from sourceforge at http://sf.net/projects/sbml. Full documentation with examples of all entry points including the entire API is available on the sbml.org website at http://sbml.org/software/mathsbml/. Instructions for downloading and installing MathSBML are also provided on that site. MathSBML will run under any operating system or platform on which *Mathematica* is already installed; a complete list of compatible systems is given at the Wolfram Research web site.

Acknowledgements

SBML was started by Hamid Bolouri, Hiroaki Kitano and John Doyle in the year 2000. The projects were initially funded by a generous grant from the Japan Science and Technology Agency under the ERATO Kitano Symbiotic Systems Project. Support for the continued development of SBML and associated software and activities today comes from the following sources: the National Human Genome Research Institute (USA), the National Institute of General Medical Sciences (USA), the International Joint Research

Program of NEDO (Japan), the ERATO-SORST Program of the Japan Science and Technology Agency (Japan), the Ministry of Agriculture (Japan), the Ministry of Education, Culture, Sports, Science and Technology (Japan), the BBSRC e-Science Initiative (UK), the DARPA IPTO Bio-Computation Program (USA), the Army Research Office's Institute for Collaborative Biotechnologies (USA), and the Air Force Office of Scientific Research (USA). Additional support is provided by the California Institute of Technology (USA), the University of Hertfordshire (UK), the Molecular Sciences Institute (USA), and the Systems Biology Institute (Japan). Besides the authors, other SBML Team members include or have included Ben Bornstein (primary developer of libBML), Sarah Keating (SBML Toolbox), Ben Kovitz, Joanne Matthews (online tools), Akira Funahashi (KEGG/SBML conversion), Hamid Bolouri, Herbert Sauro, and Maria Schilstra (CellML conversion), Hiroaki Kitano, John Doyle and Roger Brent. The authors would also like to thank Nicolas Le Novère for his extraordinary efforts to develop the Biomodels database. *Mathematica* is a commercial product produced by Wolfram Research.

References

1. Hucka M et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 2003; 19: 524-531.

2. Hucka M et al. Evolving a lingua franca and associated software infrastructure for computational systems biology: the Systems Biology Markup Language Project. IEE Systems Biology. 2004; 1:41-53.

3. XML Core Working Group. Extensible Markup Language (XML). Available on the world wide web at http://www.w3.org/xml

4. Achard F, Vaysseix G, Barillot E. XML, bioinformatics, and data integration. Bioinformatics. 2001; 17: 115-125.2

5. Wanner BL, Finney A, Hucka M. Modeling the E. coli cell: The need for computing, cooperation, and consortia. In Alberghina L, Westerhoff HV, ed., Systems Biology: Definitions and Perspective. Berlin: Springer-Verlag. 2005; in press.

6. Nature Molecular Systems Biology. For Authors. Available on the world wide web at http://www.nature.com/msb/authors/index.html.

7. Finney A et al. Chapter 1. Software Infrastructure for Effective Communication and Reuse of Computational Models. In Szallasi Z and Stelling J, ed. System modeling in cellular biology: From concepts to nuts. MIT Press (in press).

8. Hucka M, Finney A, Sauro H, Bolouri H. Systems Biology Markup Language (SBML) Level 1: Structures and facilities for Basic Model Definitions. SBML Level 1, Version 2 (Final), 28 August 2003. Available on the world wide web at http://sbml.org/specifications/sbml-level-1/version-2/html/sbml-level-1.html.

9. Finney A, Hucka M. Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions. SBML Level 2, Version 1 (Final), June 28, 2003. Available on the world wide web at http://sbml.org/specifications/sbml-level-2/version-1/html/sbml-level-2.html.

10. Finney A and Hucka M. Systems Biology Markup Language (SBML) Level 2: Structures and Facilities for Model Definitions, SBML Level 2, Version 2 (Draft) 26 March 2005.

11. Asbrooks R, Buswell S, Carlisle D, Dalmas S, Devitt S, Diaz A, Froumentin M, Hunter R, Io P, Kohlase M, Miner R, Poppelier N, Smith B, Soiffer N, Sutor R, Watt S. Mathematical Markup Language (MathML) Version 2.0 (Second Edition), http://www.w3.org/TR/2003/REC-MathML2-20031021/.

12. Finney A. Developing SBML Beyond Level 2: Proposals for Development, in Computational Methods in Systems Biology, ed. V. Danos and V. Schachter, *Lecture Notes in Computer Science*, 2005; 3082:242-247.

13. Goto S, Nishioka T, and Kanehisa, M. LIGAND: Chemical Database for Enzyme Reactions, Bioinformatics, 1998; 14:591-599.

14. Goto S, Okuno Y, Hattori M, Nishioka T, Kanehisa M, LIGAND: database of chemical compounds and reactions in biological pathways. Nucleic Acids Research, 2002; 30:402-404.

15. Lloyd C, Halstead MDB, Nielson PF. CellML: its future, present, and past, Progress in Biophysics and Molecular Biology, 2004; 85(2-3):433-450.

16. LeNovère N, Finney A, Hucka M, Bhall U, Campagne F, Collado-Vise J, Crampin, Halstead M, Klipp, E, Mendes, P, Nielsen,P, Sauro,H, Shapiro B, Snoep J, Spence H and Wanner B. Minimal information requested in the annotation of models (MIRIAM). *Nature Biotechnol,* (in press), 2005.

17. LeNovère N, Bornstein B, Broicher A, Courtot M, Donizelli M, Dharuri H, Li L, Sauro H, Schilstra M, Shapiro B, Shoep JL, Hucka M. Biomodels, database of curated quantitative kinetics models. *Nucleic Acids Research*, **submitted**,2005.

18. Le Novère N, Donizell M, Bornstein B, Courtot M, Li L, Hucka M. Biomodels database, storage, and exchange of curated quantitative models in Systems Biology, *Nature Molecular Systems Biology*, **submitted**, 2005.

19. Shapiro BE, Hucka M, Finney A, Doyle JC. MathSBML: A Package for Manipulating SBML-Based Biological Models. *Bioinformatics*, 2004; 20(16):2829-2831.

20. Novak B, Tyson JJ. Modeling the control of DNA replication in fission yeast. PNAS, 1997; 94:9147-9152.

Figure 1

A skeleton SBML Level 2 model .

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
   <model id="My_Model">
      <listOfFunctionDefinitions>
      ...
      </listOfFunctionDefintions>
      <listOfUnitDefinitions>
      ...
      </listOfUnitDefinitions>
      <listOfCompartments>
      ...
      </listOfCompartments>
      <listOfSpecies>
      ...
      </listOfSpecies>
      <listOfParameters>
      ...
      </listOfParameters>
      <listOfRules>
      ...
      </listOfRules>
      <listOfReactions>
      ...
      </listOfReactions>
      <listOfEvents>
      ...
      </listOfEvents>
   </model>
</sbml>
```

Figure 2.

UML diagram of the SBML inheritance hierarchy showing the major data types in SBML.

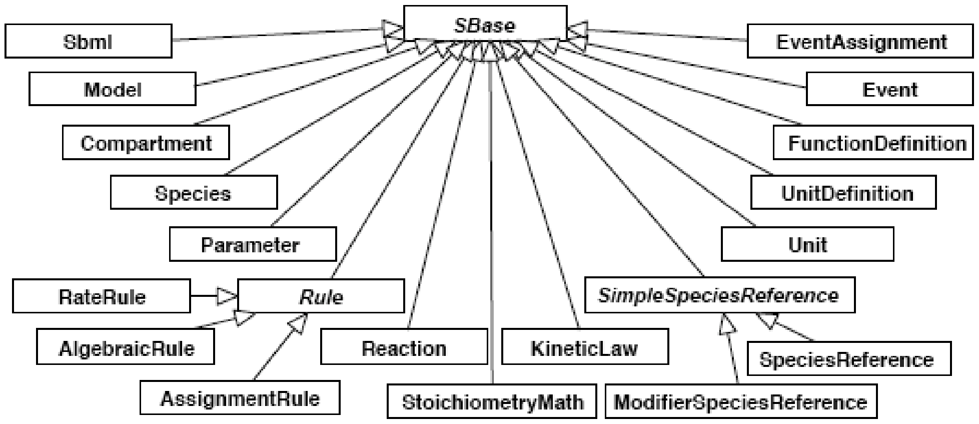

Figure 3.

Abbreviated form of data structure returned by SBMLRead after importing the cell cycle model described in the text. The ellipsis is used to indicate that some parts of the data structure have not been illustrated to save space in the present book chapter; in fact, MathSBML will display the entire data structure.

```
{SBMLAlgebraicRules → {},
 SBMLAssignmentRules → {IEB[t] == 1 - IE[t], …},
 SBMLCompartments → {Cell},
 SBMLConstants → {Cell → 1, …},
 SBMLEvents → {
   "Start" → {
     "trigger" → "SPF[t]  >= 0.1", "delay" → "60",
     "events" → {"kp[t]->kp[t]/2"}},
   …},
 SBMLFunctions → {},
 SBMLIC → {UbE[0] == 1, …},
 SBMLLevelVersion → 2.1`,
 SBMLMassActionEquations → {UbE′[t] == UbEB[t] v4[22], …},
 SBMLMassActionVariables → {UbE[t], …},
 SBMLMassBalanceEquations → {UbE′[t] == v4[22], …},
 SBMLModelid → "NovakTyson1997CellModel",
 SBMLModelName → "Novak1997_CellCycle",
 SBMLModelVariables → {UbE[t], …},
 SBMLNameIDAssociations → {"NovakTyson1997CellModel" → "Novak1997_CellCycle", …},
                                  0.25 Cdc25[t]       Cdc25B[t] MPF[t]
 SBMLODES → {Cdc25′[t] == - ─────────────────── + ────────────────── , …},
                                0.1 + Cdc25[t]      0.1 + Cdc25B[t]
 SBMLParameters → {mu, …},
 SBMLReactions → {∅ →  G2K, …},
 SBMLSpecies → {UbE[t], …},
 SBMLSpeciesCompartmentAssociations → {UbE → Cell, …},
 SBMLSpeciesTypeAssociations → {},
 SBMLStoichiometryMatrix → {{0, 0, …}, …},
 SBMLUnitAssociations → {Cell → Units`litre, …},
 SBMLUnitDefinitions → {Units`time → 60 Units`second, …}}
```

Figure 4.

A plot of the two variables Rum1Total (dashed line) and Cdc13Total (solid line)

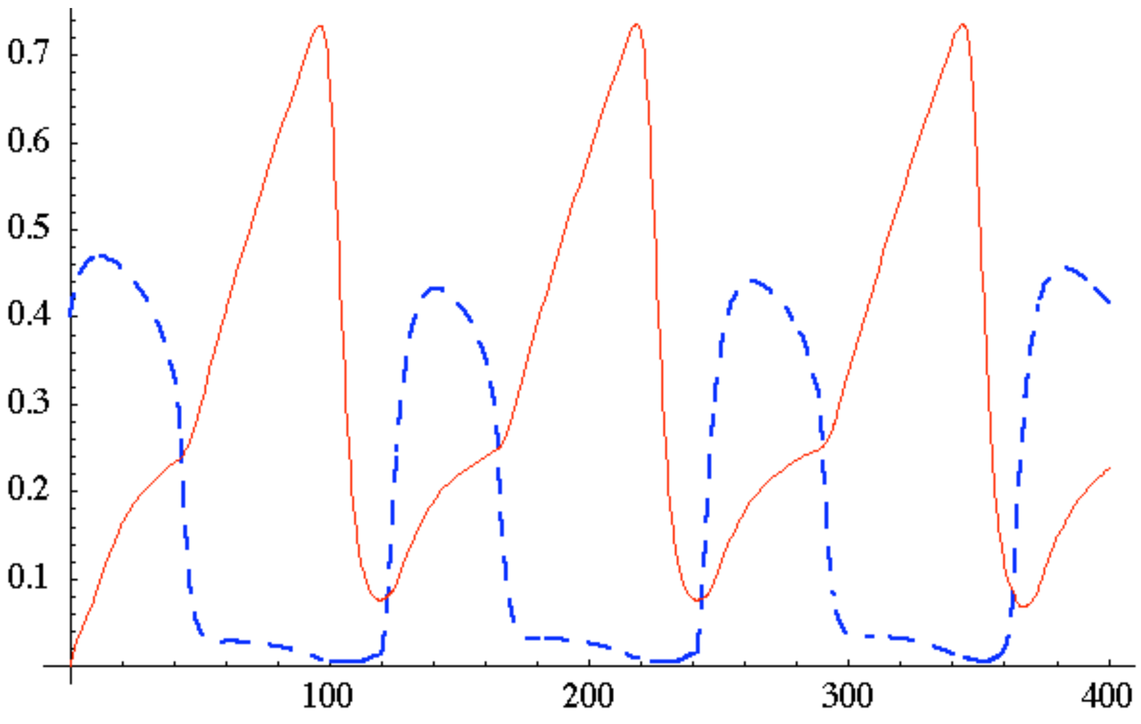

Figure 5

Model builder commands needed to create the cell cycle model. Due to space limitations only a subset of the commands are shown; the vertical ellipsis indicates that many commands were omitted. All of the omitted commands are of the form "addX". The last statement in the list, createModel, generates the SBML file cellCycle.xml.

```
<<mathsbml.m
newModel["CellCycle"];
addCompartment["Cell",size->1];
addUnit[id->"time",name->"minutes",
      unit->{"second"->{multiplier->60}}];
addParameter[id->"mu",value->0.00495];
addParameter[id->"Mass",value->1,constant->False];
addRule[type->"RateRule",variable->"Mass",math->Mass*mu];
      ⋮
addSpecies[IEB,boundaryCondition->True,initialAmount->0];
addSpecies[UbEB,boundaryCondition->True,initialAmount->0];
      ⋮
addRule[type->AssignmentRule,variable->IEB,math->1-IE];
addRule[type->AssignmentRule,variable->UbEB,math->1-UbE];
      ⋮
addSpecies[Rum1Total];
addSpecies[Cdc13Total];
      ⋮
addRule[type->AssignmentRule,variable->Rum1Total,
      math->R+G1R+G2R+PG2R];
addRule[type->AssignmentRule,variable->Cdc13Total,
      math->G2K+G2R+PG2+PG2R];
      ⋮
addEvent[id->"Start",trigger->SPF≥0.1,
      eventAssignment->{kp->kp2},delay->60];
addEvent[id->"Division",trigger->UbE≤0.1,
      eventAssignment->{kp->kp*2,Mass->Mass/2}];
      ⋮
addReaction[products->{G2K},id->"G2K_Creation",kineticLaw->k1];
      ⋮
createModel["cellCycle.xml"];
```

Table 1. SBML Workshops and Hackathons.

Meeting	Date	Location
1st Workshop	April 2000	Pasadena, CA, USA
2nd Workshop	Nov. 2000	Tokyo, Japan
3rd Workshop	June 2001	Pasadena, CA,USA
4th Workshop	Dec. 2001	Pasadena, CA, USA
5th Workshop	July 2002	Hatfield, UK
6th Workshop	Dec. 2002	Stockholm, Sweden
7th Workshop	May 2003	Ft. Lauderdale, FL, USA
1st Hackathon	July 2003	Blacksburg, VA, USA
8th Workshop	Nov 2003	St. Louis, MO, USA
2nd Hackathon	May 2004	Hinxton, UK
9th Workshop	Oct. 2004	Heidelberg, Germany
3rd Hackathon	May 2005	Tokyo, Japan
10th Workshop	Oct. 2005	Boston, MA, USA

Table 2. The subset of MathML that is allowed in SBML Level 2.

Object	Elements* Allowed
Token	`cn**, ci, csymbol***, sep`
Basic content	`apply, piecewise, piece, otherwise`
Relational operators	`eq,neq, gt, lt, geq, leq`
Arithmetic operators	`plus`, `minus`, `times`, `divide`, `power`, `root`, `abs`, `exp`, `ln`, `log`, `floor`, `ceiling`, `factorial`
Logical operators	`and,or, xor, not`
Qualifiers	`degree, bvar, logbase`
Trigonometric	`sin, cos, tan, sec, csc, cot, sinh, cosh, tanh, sech, csch, coth, arcsin, rcos, arctan, arcsec, arccsc, arccot, arcsinh, arccosh, arctanh, arcsech, arccsch, arccoth`
Constants	`true, false, notanumber, pi, infinity, exponentiale`
Annotation	`semantics, annotation***, annotation-xml***`

*The attributes `style`, `class` and `id` may be used on any element. **The attribute type may only take on one of the following: `"e-notation","real"`, `"integer"`, or `"rational"`; `***` `encoding` and `definitionURL` attributes are allowed are `csymbol` elements, and `encoding` is permitted on `annotation` and `annotation-xml` elements.

Table 3. Summary of the MathSBML API. The "_" in the name can be replaced with any checked object, e..g., `addFunction` or `modifyRule`. Controllable options are summarized in Table 4.

	Model	Compartment	Event	Function	Parameter	Reaction	Rule	Species	Unit	annotation
add_		✓	✓	✓	✓	✓	✓	✓	✓	
_ToSBML		✓	✓	✓	✓	✓	✓	✓	✓	✓
_ToSymbolciSBML		✓	✓	✓	✓	✓	✓	✓	✓	✓
get_		✓	✓	✓	✓	✓	✓	✓	✓	
modify_		✓	✓	✓	✓	✓	✓	✓	✓	
remove_		✓	✓	✓	✓	✓	✓	✓	✓	
create_	✓									
createSymbolic_	✓									

Table 4. SBML attributes that can be controlled via the API commands in table 3. The "options" shown generally have a one-to-one correspondence with SBML attributes, although sometimes the spelling is different. For example, reaction products option refers to the SBML `speciesReferences` within the SBML `listOfProducts`; however, for the most part the correspondence is clear.

API commands for:	**Options***
`species`	`id, name, compartment, initialAmount, initialConcentration, substanceUnits, spatialSizeUnits, hasOnlySubstanceUnits, boundaryCondition, charge, constant`
`compartment`	`id, name, constant, outside, spatialDimensions, size, units`
`event`	`id, name, trigger, delay, timeUnits, eventAssignment`
`function`	`id, name, math`
`parameter`	`id, name, annotation, notes, value, units, constant`
`reaction`	`id, name, fast, kineticLaw, modifiers, name, products, productStoichiometry, reactants, reactantStoichiometry, reaction, reversible, parameters (`sub-options: `value, name), timeUnits, substanceUnits;`
`rule`	`type, variable, math`
`species`	`id, name, compartment, initialAmount, initialConcentration, substanceUnits, hasOnlySubstanceUnits, boundaryCondition, charge, constant`
`unit`	`id, name, unit (`sub-options: `exponent, scale, multiplier, offset)`
`model`	`id, name;` also: `comments` (XML Comments)

*All objects have modifiable annotation, notes, and metaid fields. Some options are mutually exclusive.

Table 5. Summary of MathSBML commands excluding the API.

Function	MathSBML Entry Points
Algebraic/MathML conversion	`InfixToMathML, MathMLToInfix`
Convert model file format	`SBMLCopy`
Plot results of a simulation	`SMBLPlot, SBMLGridPlot, SBMLListPlot`
Simulation	`dataTable, SBMLNDSolve`
Import a model	`SBMLRead`
Export a model	`SBMLWrite, createModel`
Annotation control	`setAnnotationPackage, setAnnotationURL, setModelAnnotation, setSBMLAnnotation`
Model display	`showModel`