# 17 Software Infrastructure for Effective Communication and Reuse of Computational Models

Andrew Finney, Michael Hucka, Benjamin J. Bornstein, Sarah M. Keating, Bruce E. Shapiro, Joanne Matthews, Ben L. Kovitz, Maria J. Schilstra, Akira Funahashi, John Doyle, and Hiroaki Kitano

Until recently, the majority of computational models in biology were implemented in custom programs and published as statements of the underlying mathematics. However, to be useful as formal embodiments of our understanding of biological systems, computational models must be put into a consistent form that can be communicated more directly between the software tools used to work with them. In this chapter, we describe the Systems Biology Markup Language (SBML), a format for representing models in a way that can be used by different software systems to communicate and exchange those models. By supporting SBML as an input and output format, different software tools can all operate on an identical representation of a model, removing opportunities for errors in translation and assuring a common starting point for analyses and simulations. We also take this opportunity to discuss some of the resources available for working with SBML as well as ongoing efforts in SBML's continuing evolution.

## 17.1 Introduction

The chapters of this book testify to the rising importance of computational modeling in biological research as a means of helping to better understand biological function. The increasing interest in this approach, coupled with our modern ability to generate ever-more complex models more rapidly than ever before, make it clear that practical computational modeling requires the use of software tools.

Until recently, the majority of models were implemented in custom programs and published only as statements of the underlying mathematics (that is, intended for human consumption). However, to be useful as formal embodiments of our understanding of biological systems (Bower and Bolouri, 2001), computational models must be put into a consistent form that can be communicated more directly between the software tools used to work with them. This format must help overcome a number of problems facing a systems biologist:

- Users often need to work with complementary resources from multiple software tools in the course of a project because different tools have different strengths and capabilities. For example, one tool may have a good model editing interface, another tool may provide novel facilities for analyzing system properties, yet another may implement an advanced simulation capability but lack a good graphical interface, etcetera. If the tools do not share a common model storage format, users are forced to re-encode their models in each tool separately, a time-consuming and error-prone practice.
- Models published in peer-reviewed journals are sometimes accompanied by instructions for obtaining the definitions in electronic form. However, because each author may use a different software environment (and associated model representation language), such definitions are often not straightforward to examine, test, and reuse. Researchers who wish to use a published model typically must transcribe it manually into a format compatible with their particular software.
- When simulation software packages are no longer supported, models developed in those systems can become stranded and unusable. This has already happened on a number of occasions, with a resulting loss of usable models to the research community. Continued innovation and development of new tools will only aggravate this problem unless the issue of standard formats is addressed.
- Reuse of existing models requires that those models can be clearly identified, easily retrieved, and related to their published descriptions in the scientific literature. Moreover, because of the increasing size and complexity of models continually being developed, the model structure should be documented to allow for efficient handling and sound modification.

We developed the Systems Biology Markup Language (SBML) in an effort to address these problems. SBML is a format for representing computational models in a way that can be used by different software systems to communicate and exchange those models (Finney and Hucka, 2003; Hucka et al., 2003, 2004). By supporting SBML as an input and output format, different software tools can all operate on an identical representation of a model, removing opportunities for errors in translation and assuring a common starting point for analyses and simulations. SBML is by no means a perfect format, but it has proven useful and achieved widespread acceptance within the domain of modeling at the level of biochemical reaction networks. Over 90 open-source and commercial software tools support SBML as of November 2005.

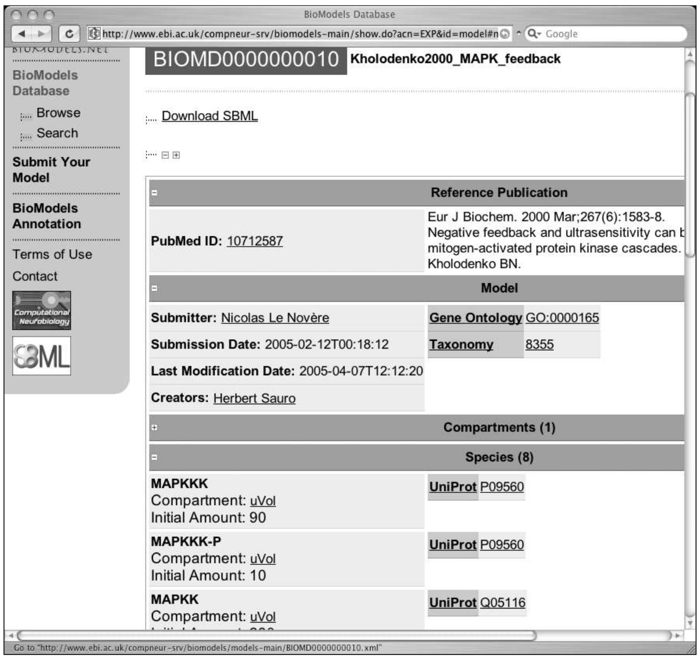
A gratifying by-product of the SBML project has been the way it has catalyzed a community of interested researchers, developers, and users who are now collaborating on evolving SBML and creating new resources around it. This is undoubtedly a reflection of an urgent need in the community for *any* format such as SBML to address issues of interoperability. At the same time, we suspect that the challenges faced by the SBML community and the solutions that are arising have underlying components that would be faced by any effort to define a similar standard exchange format. We discuss two examples in this chapter. One is the difficulty of balancing ease of language implementation against representational power. Today this is being answered by progress towards SBML Level 2 Version 2, which is expected to be ratified in 2005, and the modular SBML Level 3, which is expected in 2006. A second is the unexpected difficulty of ensuring correct interpretation of SBML by different software applications. We describe our current attempts to address this problem using a combination of (i) a carefully-designed software library, libSBML, which among other features provides rule-based model consistency testing, and (ii) a semantic validation suite for testing correct interpretation of SBML constructs by software applications.

## 17.2 Software Assistance for Biological Modeling

As an example of how software technologies such as SBML assist modelers today, consider the following hypothetical (but still quite plausible) sequence of events.

*A computationally-savvy biologist named Albert is investigating one of the mitogen-activated protein kinase (MAPK) cascades. The MAPK pathways lead from growth factor receptors on cell membranes to effector molecules located in the cell cytoplasm and nucleus. This family of signaling pathways is one that has received much attention in both experimental (Seger and Krebs, 1995) and computational biology (Schoeberl et al., 2002).*

*Our hypothetical biologist might begin with a body of experimental data gathered by himself and other members of the laboratory in which he works. In order to understand his experiments in the context of other data and other published results, he decides to develop a computational model so that he can integrate different sources of existing knowledge and his own hypotheses into a common, formalized framework. Since the MAPK system is a popular topic of study, he has no trouble finding related work in the literature, including existing computational models. He chooses to begin with a relatively simple model by Kholodenko (Kholodenko, 2000). The original publication gives a complete listing of the mathematical equations that define Kholodenko's model, but no software implementation. (Even though that particular article is from this decade, it still predates the development of SBML and most of today's software tools.) The model is not complex, but he knows that recreating a model from a research paper will take time, so before starting, he visits the BioModels Database (BioModels Team, 2005) to check if the model is available in a machine-readable format. He searches the database and quickly finds an existing implementation (figure 17.1), which he can download in SBML format.*



BioModels Database

http://www.ebi.ac.uk/compneur-srv/biomodels-main/show.do?acn=EXP&id=model#n

BIOMODELS.NET

BioModels Database
- Browse
- Search

Submit Your Model

BioModels Annotation

Terms of Use

Contact

BIOMD0000000010 **Kholodenko2000_MAPK_feedback**

Download SBML

| | **Reference Publication** |
|---|---|
| **PubMed ID:** 10712587 | Eur J Biochem. 2000 Mar;267(6):1583-8. Negative feedback and ultrasensitivity can b mitogen-activated protein kinase cascades. Kholodenko BN. |

| | **Model** | |
|---|---|---|
| **Submitter:** Nicolas Le Novère | **Gene Ontology** | GO:0000165 |
| **Submission Date:** 2005-02-12T00:18:12 | **Taxonomy** | 8355 |
| **Last Modification Date:** 2005-04-07T12:12:20 | | |
| **Creators:** Herbert Sauro | | |

**Compartments (1)**

| | **Species (8)** | |
|---|---|---|
| **MAPKKK** Compartment: uVol Initial Amount: 90 | **UniProt** | P09560 |
| **MAPKKK-P** Compartment: uVol Initial Amount: 10 | **UniProt** | P09560 |
| **MAPKK** Compartment: uVol | **UniProt** | Q05116 |

Go to "http://www.ebi.ac.uk/compneur-srv/biomodels/models-main/BIOMD0000000010.xml"

**Figure 17.1** Screenshot of a model display page in the BioModels Database (BioModels Team, 2005).

*Once he has the SBML file, Albert starts up his favorite Windows-based model editing package, JDesigner (Sauro et al., 2003). This package provides a friendly, graphical diagram view of a model (figure 17.2). He spends a significant amount of time experimenting with the model running time-course simulations to examine the behavior under different conditions, as well as making modifications and exploring the results. After becoming familiar with the Kholodenko model, he next begins to make modifications based on his own experimental work and that of his colleagues.*

*Eventually, Albert's model grows and becomes substantially different from the original. He reaches a point where he has to find values for parameters in the model that are not directly measurable, but he believes he has enough converging evidence from other data that he can search for plausible values by a process known as parameter estimation. This is a resource-intensive task requiring many repeated simulation runs and analyses—more than he can comfortably run on his laptop computer. Albert enlists the aid of a colleague, Bernadette, who works at another institution and who has access to clusters of computers on which she can quickly perform large computations. Bernadette is less a biologist and more a computational scientist, but she has had enough exposure to biological modeling that she can perform the parameter estimation tasks for Albert. Despite the geographical distance separating them and the fact that Bernadette is adamant about using Linux rather than Windows as her computer operating system of choice, Albert has no difficulty conveying an unambiguous model definition to Bernadette*

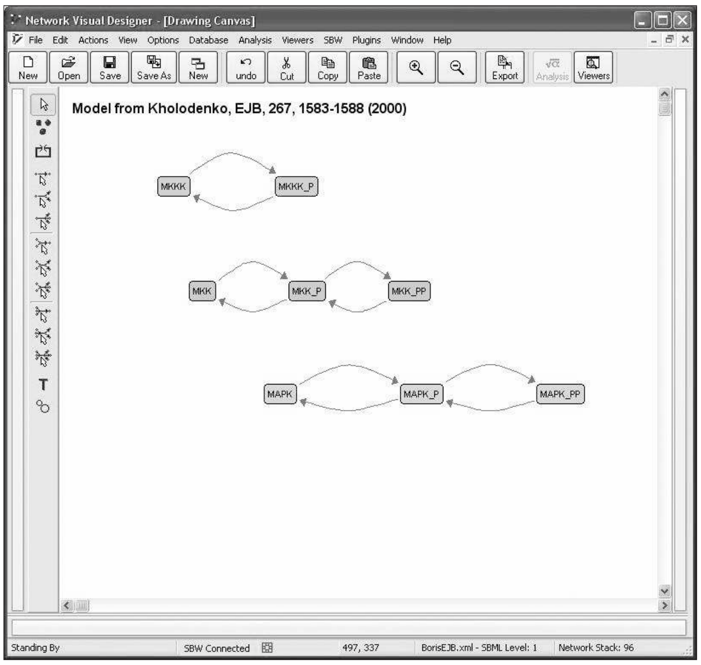
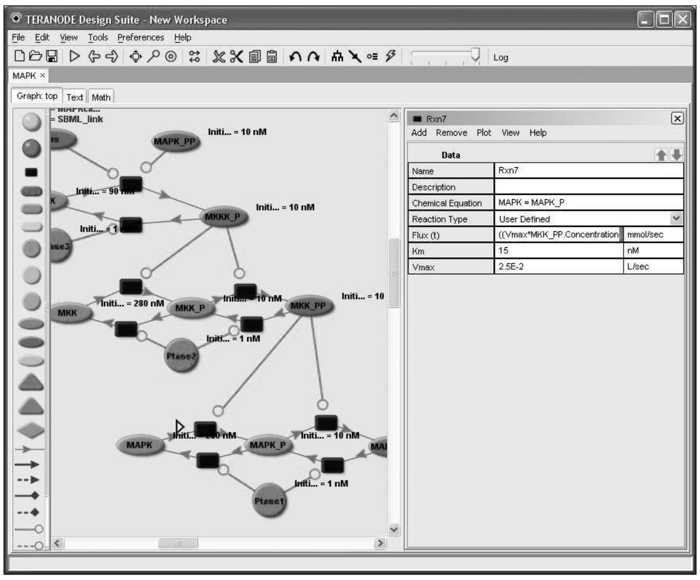

Network Visual Designer - [Drawing Canvas]
File Edit Actions View Options Database Analysis Viewers SBW Plugins Window Help
New Open Save Save As New undo Cut Copy Paste Export Analysis Viewers
Model from Kholodenko, EJB, 267, 1583-1588 (2000)
MKKK
MKKK_P
MKK
MKK_P
MKK_PP
MAPK
MAPK_P
MAPK_PP
Standing By
SBW Connected
497, 337
BorisEJB.xml - SBML Level: 1
Network Stack: 96

**Figure 17.2** Screenshot of JDesigner (Sauro et al., 2003), a free computational modeling system for biochemical reaction networks. It runs on the Microsoft Windows operating system.

*because JDesigner can produce SBML output and Bernadette has at her disposal several software tools that can read SBML.*

*Bernadette writes command scripts in Linux that take Albert's model and his experimental data (which he stored in ordinary comma-delimited tabular format) and perform parameter estimation using an optimization package written in MATLAB (The Mathworks, Inc., 2005). To convert the SBML model into appropriate MATLAB data structures, she uses one of the free MATLAB toolboxes available for this purpose (Keating, 2005). After some iterations back and forth with Albert to clarify his goals, and many computer runs, the pair eventually determine best-estimate values for the unknown parameters in Albert's model. Bernadette also performs a large number of additional simulation and analysis runs on her Linux computers using COPASI (Mendes, 2003) to explore the behaviors of the model. The results enable Albert to continue further with his research, comparing his predictions to experimental data and refining his model to incorporate new hypotheses. The model and its results are novel enough that Albert writes an article about them with Bernadette. They also submit the SBML model to the BioModels Database, where the curators annotate the model and enter it into the database for other researchers to use and build upon.*

*Some time after the article is published, a researcher working at a pharmaceutical company reads Albert and Bernadette's paper on MAPK signaling. It turns out*



Data

| | | |
|---|---|---|
| Name | Rxn7 | |
| Description | | |
| Chemical Equation | MAPK = MAPK_P | |
| Reaction Type | User Defined | |
| Flux (t) | ((Vmax*MKK_PP.Concentration | mmol/sec |
| Km | 15 | nM |
| Vmax | 2.5E-2 | L/sec |

**Figure 17.3** Screenshot of TERANODE Design Suite (TERANODE, Inc., 2005), an example of a modern commercial software package using SBML and integrating model editing, analysis, and simulation.

*that this researcher, Carl, has been investigating novel therapeutic interventions on this same pathway. Thanks to the availability of the model in SBML form, Carl is able to quickly obtain and try out the model in his software tool of his choice (figure 17.3), a full-featured commercial package called TERANODE Design Suite (TERANODE, Inc., 2005). The model's structure and behavior are consistent with his own findings, and moreover, it provides new insights that could lead to an investigation of new pharmacological agents. Carl is interested in pursuing this further. The copyright on the model stipulates that commercial users must contact the authors, so he contacts Albert and Bernadette and begins a promising new collaboration.*

## 17.3 The SBML Representation of Models

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

### 17.3.1 Brief Summary of the Form and Features of SBML

SBML is a machine-readable *model definition language* defined neutrally with respect to programming languages and software encoding. It is defined using a subset of UML, the Unified Modeling Language (Eriksson and Penker, 1998; Oestereich, 1999), and in turn, this definition is used to create an XML Schema (Biron and Malhotra, 2000; Fallside, 2000; Thompson et al., 2000) for SBML. The XML Schema specifies how SBML can be expressed using XML, the eXtensible Markup Language (Bosak and Bray, 1999; Bray et al., 2000). XML is a simple and portable text-based substrate that has been gaining widespread acceptance in computational biology and bioinformatics (Achard et al., 2001; Augen, 2001).

The main focus of SBML is encoding models consisting of biochemical entities (species) linked by reactions to form biochemical networks. An important principle in SBML is that models are decomposed into explicitly-labeled constituent elements, the set of which resembles a verbose rendition of chemical reaction equations. The representation deliberately does not cast the model directly into a set of differential equations or other specific mathematical frameworks. This explicit, modeling-framework-agnostic decomposition makes it easier for different software tools to interpret the model and translate the SBML form into whatever internal form each tool actually uses.

SBML is being developed in levels, with each higher SBML *level* adding richness to the model definitions that can be represented by the language. Level 2 is the highest level of SBML currently defined; it represents an incremental evolution of the language resulting from the practical experiences of many users and developers working with Level 1 since its introduction in the year 2001. A definition of a model in SBML Level 2 consists of lists of one or more of the following components:

- *compartment*: a container of finite dimensions where one or more chemical substances (well-mixed) are located;
- *species*: a pool of a chemical substance located in a specific compartment (a species represents the concentration or amount of a substance and not a single molecule);
- *reaction*: a statement describing some transformation, transport or binding process that can change one or more species (each reaction is characterized by the stoichiometry of its products and reactants and optionally by a rate equation);
- *parameter*: a quantity that has a symbolic name;
- *unit definition*: a name for a unit used in the expression of quantities in a model;
- *rule*: a mathematical expression that is added to the model equations constructed from the set of reactions (rules can be used to set parameter values, establish constraints between quantities, etcetera.);
- *function*: a named mathematical function that can be used in place of repeated expressions in rate equations and other formulas; and
- *event*: a set of mathematical formulas evaluated at a specified moment in the time evolution of the system.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
  <model id="EnzymeKinetics">
    <listOfCompartments>
      <compartment id="Cell" size="1"/>
    </listOfCompartments>
    <listOfSpecies>
      <species id="S"  compartment="Cell" initialAmount="1" boundaryCondition="true"/>
      <species id="E"  compartment="Cell" initialAmount="1"/>
      <species id="ES" compartment="Cell" initialAmount="0.01"/>
      <species id="P"  compartment="Cell" initialAmount="0.01" boundaryCondition="true"/>
    </listOfSpecies>
    <listOfReactions>
      <reaction id="Reaction1">
        <listOfReactants>
          <speciesReference species="S"/>
          <speciesReference species="E"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="ES"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply> <minus/>
              <apply> <times/>  <ci> k_1 </ci>  <ci> S </ci>  <ci> E </ci>  </apply>
              <apply> <times/>  <ci> k_r </ci>  <ci> ES </ci>  </apply>
            </apply>
          </math>
          <listOfParameters>
            <parameter id="k_1" value="3"/>
            <parameter id="k_r" value="6"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
      <reaction id="Reaction2" reversible="false">
        <listOfReactants>
          <speciesReference species="ES"/>
        </listOfReactants>
        <listOfProducts>
          <speciesReference species="E"/>
          <speciesReference species="P"/>
        </listOfProducts>
        <kineticLaw>
          <math xmlns="http://www.w3.org/1998/Math/MathML">
            <apply> <times/>  <ci> k_2 </ci>  <ci> ES </ci>  </apply>
          </math>
          <listOfParameters>
            <parameter id="k_2" value="9"/>
          </listOfParameters>
        </kineticLaw>
      </reaction>
    </listOfReactions>
  </model>
</sbml>
```

**Figure 17.4** Simple SBML Level 2 model of a system of reactions involving enzyme kinetics.

Additional features in SBML Level 2 include support for a systematic way of including metadata, and support for delay functions. The latter are useful for representing biological processes having a delayed response, but where the details of the processes and the actual delay mechanism are not relevant to the operation of the model.

To make this discussion concrete, figure 17.4 gives the complete SBML Level 2 listing of a simple model of enzyme kinetics, $E + S \rightleftharpoons ES \rightarrow P$, where E, S, and P represent the enzyme, substrate, and product species, respectively, and ES is an intermediate complex formed during the reaction. In this particular SBML rendition, the system is represented as two reaction structures: the reversible reaction $E+S \rightleftharpoons ES$, here defined with a forward reaction rate of $k_1 * [S] * [E]$ and a reverse reaction rate of $k_r * ES$, and the irreversible reaction $ES \rightarrow P$, here defined with a forward reaction rate of $k_2 * [ES]$. The symbols `E`, `S`, and `ES`, when used in rate expressions (SBML's `kineticLaw` elements), stand for the concentrations of the different species, and the parameters $k_1$, $k_r$, and $k_2$ are set to values $k_1 = 3$, $k_r = 6$, and $k_2 = 9$. When specific units are omitted from quantities in an SBML model (as they are here), the model is assumed to use the default units for those quantities, which in SBML are moles for substance amounts and liters for volumes. Other formulations of this model might, for example, express this system explicitly as three irreversible reactions, change the units on quantities to be millimoles and microliters, and so on. This model is presented here only to give a sense for the structure of SBML and the relative simplicity, and we reiterate that people are not meant to edit models directly at this level; instead, software tools read and write this kind of representation on the user's behalf.

SBML's representational power extends far beyond the kind of simple enzyme kinetics model used here as an illustration. Its simple formalisms allow a wide range of biological phenomena to be modeled, including cell signaling, metabolism, gene regulation, and more. There is no assumption about the kinds of kinetics or interactions or network organizations that can be represented. Significant flexibility and power come from the ability to define arbitrary formulas for the rates of change of variables as well as the ability to express other constraints mathematically.

### 17.3.2 Relationships to Other Efforts

Many XML-based formats have been proposed for representing data and models in biology; however, we know of only two XML-based formats that are suitable for representing compartmental reaction network models with sufficient mathematical depth that the descriptions can be used as direct input to simulation software. The two are SBML and CellML (Hedley et al., 2001b,a; Lloyd et al., 2004).

CellML is built around an approach of composing systems of equations by linking together the variables in those equations; this is augmented by features for declaring biochemical reactions explicitly, as well as encapsulating arbitrary components into modules. Its focus is on a component-based architecture to facilitate reuse of models and parts of models, and the mathematical description of models. By contrast, SBML provides constructs that are more similar to the internal data objects used in many contemporary simulation/analysis software packages specialized for biochemical networks.

These differences notwithstanding, the SBML and CellML efforts share much in common and represent somewhat different approaches to solving the same general problems. They were initially developed independently, but the primary developers of both languages are actively engaged in exchanges of ideas and are seeking ways of making the languages more interoperable. SBML Level 2 borrows a number of approaches from CellML, making it that much easier to translate between the two formats.

## 17.4 The Continued Evolution of SBML

The need for a language like SBML was manifest during the first Workshop on Software Platforms for Systems Biology, held at the California Institute of Technology in early 2000. The two or three dozen attendees at the time represented less than a dozen software projects, yet even within this small group, it proved impossible to share models without having to re-encode them anew in each software tool. This needless impediment to collaboration directly inspired the SBML effort.

Defining a language such as SBML and encouraging its use by other groups has always involved balancing conflicting demands. For example, there is pressure to include a wide variety of features to support the various kinds of modeling and analysis capabilities being explored in different tools. But if the capabilities are too advanced or too specialized for most tools, then few if any software packages will implement support for the entire language specification, with the consequence that most tools will still not be able to exchange models in a meaningful way. On the other hand, if SBML does not expand quickly enough to support features satisfying more advanced research efforts, then SBML risks losing the groups' patience, potentially leading to the creation of incompatible dialects of the language.

In an attempt to help achieve this balance, we are proceeding with a staged approach to SBML development, embodied in the already-mentioned concept of SBML *levels*. Each higher SBML level adds richness to the model definitions that can be represented by the language. By delimiting sets of features at incremental stages, the SBML development process provides software authors with stable standards, and the community can gain experience with the language definitions before new features are introduced. Two levels have been defined so far (Finney et al., 2002; Hucka et al., 2001). Level 1 is simpler (but less powerful) than Level 2. The separate levels are intended to coexist; SBML Level 2 does not render Level 1 obsolete. Software tools that cannot support higher levels can go on using lower levels; tools that can read higher levels are assured of also being able to interpret models defined in the lower levels. The open-source software infrastructure we have been developing around SBML (see Section 17.5) allows developers to support both Levels 1 and 2 in their software with a minimum amount of effort.

### 17.4.1 Community Involvement

One component of SBML's success has been the community-oriented method of its continued evolution. SBML's popularity has led to the formation of an active international group of researchers and software developers who are now working together to push SBML in new directions. As is the case with many projects today, the primary mode of interaction between members is electronic mail, with discussions taking place on the community mailing list, sbml-discuss@caltech.edu. The list currently contains over 200 members coming from academic, commercial and private environments, from all continents. Besides discussions over the list, another important mode of interaction has been regular face-to-face meetings during the Workshops on Software Platforms for Systems Biology (also known informally as the SBML Forum meeting), held since mid-2000.

These meetings serve many vital functions. First, they provide a forum where proposals for potential new SBML features can be presented and where consensus decisions can be made about the development of SBML, with the aim of enabling SBML to support a wider range of model paradigms and modes of interoperability. Second, they ensure that systems biology software interoperability is maximized by discussing the correct use of SBML and (related to this) exposing software developers to issues in the correct interpretation and handling of SBML in all software. Third, they inform developers of the latest developments in software infrastructure for SBML. And finally, they educate the systems biology community about the range of modeling paradigms that are being used to understand biological phenomena. The ninth SBML Forum meeting was held on October 14–15, 2004, in Heidelberg, Germany, and was attended by 49 representatives of different international research groups. All presentation materials from SBML meetings are made publicly available on the project Web site (SBML Team, 2005b).

In 2003, a new type of meeting was instituted: *SBML Hackathons*, in which software developers gather together to work simultaneously on their software next to other developers, discovering and resolving interoperability problems as they go. The third SBML Hackathon was held on May 9–10, 2005, at the National Museum of Emerging Science and Innovation in Tokyo, Japan, and was attended by 45 delegates, nearly three times as many as attended the first SBML Hackathon in 2003.

### 17.4.2 SBML Level 2 Version 2

As a practical consequence of how SBML develops and evolves, it reflects how theoreticians and software developers conceptualize and structure their computational models of biochemical reaction networks. The exact form of the language matters less than the representational elements comprising the language. Though the incremental development path taken for SBML has led to a less-than-elegant structure, it is fair to say that SBML represents a consensus view of how computational models of reaction networks are understood today. The dedicated community of interested researchers has kept up the evolution of SBML and continues to result in improvements to meet increasingly sophisticated needs.

The next specification of SBML is expected to be an incremental update, Level 2 Version 2, to be followed closely by SBML Level 3, which has been in development for over a year. The following are illustrative of the enhancements likely to be introduced in SBML Level 2 Version 2 and the reasons for them.

- *SpeciesType*. In SBML, the amount (concentration or molecular count) of every chemical species must be defined with respect to a location. Locations in SBML are represented as compartments, where a compartment can represent a physical structure such as "cytoplasm" or a purely theoretical location used solely for modeling expediency. If the same kind of species appears in more than one location (for example, both inside a cell's cytoplasm and outside the cell), this must be represented as two different species, each having separate identifiers in the model. The reason is that when SBML models are translated into typical computational forms, those species are represented as variables (again, either concentrations or molecular counts) whose values can change over time. Species located in different compartments are assumed to comprise different *pools* of the species—that is the logical point of having compartments in the first place. However, a number of software developers have expressed the need for specifying that two species variables in SBML refer to the same kind or type of chemical irrespective of compartmental location. Therefore, one of the changes planned for SBML Level 2 Version 2 is the introduction of a SpeciesType data structure for this purpose. This will make it possible for a model to define a list of SpeciesType structures. Each species definition will then be able to refer to a particular SpeciesType definition, stating, in effect, that it is "of this species type." For example, a model could contain a SpeciesType for aspartate, and could have multiple species definitions, one for aspartate located in the cytosol and others for mitochondrial matrix compartments. The species representing the different pools of aspartate would have different identifiers (for instance, "aspartate_cytosol" and "aspartate_mitochon"), but each would refer to the common aspartate SpeciesType.

- *Nested Unit Definitions.* Not all software tools provide a means of changing the units of measurement used for the numerical quantities in a user's model; they often assume specific units for different quantities and rely on users to adjust numerical values as necessary when encoding models in the software environment. Unfortunately, sometimes different tools make different unit assumptions; thus, *some* capability for redefining units in SBML is necessary in addition to specifying default units. We thought that a small, simple scheme would serve best, and this is what we attempted in the first definition of SBML (Level 1 Version 1). The scheme turned out to be too limited; for example, it did not allow for the definition of some types of units that are not in the SI unit system, and it was significantly less capable than the unit scheme in CellML, making it difficult to translate some models between CellML and SBML. The consensus in the SBML community was that more definitional power was warranted, so SBML Level 2 Version 1 introduced a fuller unit scheme. Arguably the one feature it lacked was a provision to allow unit definitions to be defined in terms of other defined units rather than solely in terms of the base units. The reason was our continuing attempts to make the unit scheme simple—after all, what use is it if many software tools don't support it? But in the end, the consensus of users was that it should not be up to the SBML language to arbitrarily limit the capabilities in this area because it impacts a researcher's ability to represent their intentions precisely. The SBML community felt that tools that lack adequate support for units should either be enhanced appropriately, or else that unit manipulation functionality could be encoded in separate software tools and libraries such as libSBML (Section 17.5).

- *ConstraintRule*. It is sometimes important to be able to express the idea that certain model conditions should hold true and if, during a simulation, the conditions are exceeded, then the user should be alerted that the model is operating outside the assumptions made by the model's author. Although SBML has always had facilities for expressing mathematical relationships between quantities, it lacked a provision for expressing these kinds of constraints. SBML Level 2 Version 2 will extend the types of SBML rules available to include ConstraintRule. This structure will allow the statement of mathematical expressions whose values evaluate to a Boolean value (true or false). If at some point in time during a time-course evaluation of the model, the expression evaluates to false, the constraint is not satisfied. The ConstraintRule will contain an optional note (in XHTML format) that can contain a message to be displayed to the user if the constraint expression evaluates to false. An example of the application of this rule would be to make explicit the assumptions of an Henri-Michaelis-Menten rate law about relative species concentrations between product and substrate as well as between enzyme and substrate.

### 17.4.3 SBML Level 3

As a language that is an intersection rather than a union of features needed by all tools, SBML currently cannot support all the representational capabilities that all software systems offer to users. Some tools offer features that have no explicit equivalent in SBML Level 2, and those tools currently can only store those features as annotations in an SBML model. But in many cases those features could potentially be used by more than one tool, and thus it would be appropriate to have some representation for them in SBML. Using Level 2 as a starting point, the SBML community has been developing proposals and prototype implementations of many new capabilities that will become part of SBML Level 3. The main current areas of interest are:

- Diagram layout: enabling the inclusion of diagrammatic renditions of a model of the sort visible in the screenshots of figures 17.2 and 17.3.
- Model composition: allowing construction of models from instances of submodels
- Multicomponent species: allowing species to be composed from instances of species types, enabling such things as the representation of complexes of phosphorylated proteins and generalized reactions acting on them
- Arrays: allowing models to contain indexed collections of objects of the same type
- Spatial features: allowing the representation of the geometric features of compartments, the diffusion rates of species and the spatial distribution of model parameters and boundary conditions
- Constraints: enabling the definition of constraints on model variables

It is unreasonable to expect a tool to support every feature planned for Level 3 in order to be called Level 3 compatible. One of the challenges for SBML Level 3 will be to design a modular feature set. The idea is to enable a model to contain explicit information about which capabilities are necessary to interpret it correctly, so that tools encountering the model may reject it gracefully if they do not possess the necessary facilities. For reasons of efficiency and correctness, an explicit indication is preferable to requiring tools to read and interpret the entire model and inferring the capabilities needed.

We anticipate that Level 3 will take the form of a core, consisting of minimal extensions to Level 2, and a set of Level 3 modules, each encapsulating the definition of one of the major features listed above. One of the extensions making up the Level 3 core will be explicit feature indicators, such that each of the modules has a corresponding feature tag which will appear in a list at the beginning of the model definition. The presence of a feature tag will signal to software tools reading the model that the model uses that particular feature. The software tool may then make a decision about whether it can handle the model or whether it should alert the user to a problem.

## 17.5 Enabling Efficient and Correct Interpretation of SBML Using a Dedicated Software Library

To make it easier for software developers and users to work with SBML, and more generally to promote the language's use as a common exchange format, our group has released and continues to develop a number of open-source SBML software tools. Here we describe one, libSBML, that many projects are using for implementing support for SBML in their software applications.

### 17.5.1 General Characteristics of libSBML

LibSBML is an application programming interface (API) library for reading, writing, and manipulating files and data streams containing SBML content. Developers can embed the library in their applications, saving themselves the work of implementing their own parsing, manipulation, and validation software. At the API level, the library provides the same interface to data structures independently of whether the model originated in SBML Level 1 or 2. The library currently also offers the ability to translate SBML Level 1 models to SBML Level 2.

LibSBML is written in ISO standard C and C++ and is highly portable. It is currently supported on the Linux, Solaris, MacOS X, and Microsoft Windows operating systems. The library provides language bindings for C, C++, Java, Python, Perl, MATLAB, and Common Lisp, with support for other languages planned for the future. We distribute the package in both source-code form and as precompiled dynamic libraries for the Microsoft Windows, Linux, and Apple MacOS X operating systems; they are available under terms of the LGPL (Free Software Foundation, 1999) from the *sbml* project on SourceForge.net (SourceForge.net, 2002), the world's largest open-source software repository and project hosting service. LibSBML is at release version 2.3.4 as of October 2005.

### 17.5.2 Advantages of a Dedicated Library for SBML

An often-repeated question is, why not simply use a generic XML parsing library? After all, SBML is usually expressed in XML, and there exist plenty of XML parsers, so why not simply tell people to use one of them, rather than develop a specialized library? The answer is: while it is true that developers *can* use general-purpose XML libraries, there are many reasons why using a system such as libSBML is a vastly better choice.

One of the features of libSBML is its facilities for manipulating mathematical formulas supporting differences in representation between SBML Level 1 and SBML Level 2. As discussed in more detail below, libSBML provides an API that allows working with formulas in both text-string and MathML (Ausbrooks et al., 2001) form, and to interconvert mathematical expressions between these forms. The utility of this facility extends beyond converting between SBML Level 1 and 2. Many software packages provide users with the ability to express formulas for such things as reaction rate expressions, and these packages' interfaces often let users type in the formulas directly as text strings. LibSBML saves application programmers the work of developing formula manipulation and translation functionality. It makes it possible to translate those formula strings directly into Abstract Syntax Trees (ASTs), manipulate them using AST operations, and write them out in the MathML format of SBML Level 2.

As discussed in Section 17.5.5, another feature of libSBML is the validation it performs on SBML inputs at the time of parsing files and data streams. This helps verify the correctness of models in a way that goes beyond simple syntactic validation. Still another invaluable feature of libSBML is the domain-specific operations it provides beyond simple SBML-specific accessor facilities. Examples of such operations include obtaining a count of the number of boundary condition species, determining the modifier species of a reaction (assuming the reaction provides kinetics), and constructing the stoichiometric matrix for all reactions in a model.

Finally, libSBML is solidly written and tested. The entire library has been written by seasoned, professional software engineers using the test-driven approach (Beck, 2002). The libSBML source code currently has 760 unit tests and over 3,400 individual assertions. It represents a robust and well-tested system that others can build upon.

### 17.5.3 Manipulating Mathematical Formulas

In SBML Level 1, mathematical formulas are represented as text strings using a C-like syntax. We chose this representation because of its simplicity, widespread familiarity, and use in applications such as Gepasi (Mendes, 1997) and Jarnac (Sauro, 2000), whose authors contributed to the initial design of SBML. For SBML Level 2, there was a need to expand the mathematical vocabulary of Level 1 to include additional functions (both built-in and user-defined), mathematical constants, logical operators, relational operators, and a special symbol to represent time. Rather than growing the simple C-like syntax into something more complicated and esoteric in order to support these features, and consequently having to manage two standards in two different formats (XML and text string formulas), we chose to leverage an existing standard for expressing mathematical formulas in Level 2: the content portion of MathML (Ausbrooks et al., 2001).

Using MathML in SBML has at least two advantages. First, instead of reinventing the wheel, we are building upon an existing and well-established W3C standard. Second, since the entirety of a model is expressed in XML, SBML is now more amenable to tools that can process, manipulate, and store XML, such as (for example) XSLT (Clark and DeRose, 1999), XQuery (Fernández et al., 2005), XPath (Fernández et al., 2005), and other XML technologies. That said, there are some disadvantages to using MathML. By introducing MathML part-way through the evolution of SBML, we have created a legacy support problem by having two formula representations with which to contend and interconvert. Also, most simulator packages cannot parse and understand MathML directly (but, we should point out the same would hold true had we chosen to expand the lowest-common-denominator C-like syntax of Level 1). Overcoming both of these disadvantages is easy with libSBML.

Abstract Syntax Trees (ASTs) are well-known in the computer science community; they are simple recursive data structures useful for representing the syntactic structure of sentences in certain kinds of languages (mathematical or otherwise). Much as libSBML allows programmers to manipulate SBML at the level of domain-specific objects, regardless of SBML level or version, it also allows programmers to work with mathematical formula at the level of ASTs regardless of whether the original format was C-like infix notation or MathML. LibSBML goes one step further by allowing programmers to work exclusively with infix formula strings and instantly convert them to the appropriate MathML whenever needed.

LibSBML ASTs provide a canonical, in-memory representation for all mathematical formulas regardless of their original format (that is, C-like infix strings or MathML). In libSBML, an AST is a collection of one or more ASTNodes. ASTNodes represent the most basic, indivisible part of a mathematical formula and come in many types. For instance, there are node types to represent numbers (with subtypes to distinguish integer, real, and rational numbers), names (for example, constants or variables), simple mathematical operators, logical or relational operators, and functions. Each ASTNode node may have none, one, two, or more child ASTNodes depending on its type. For instance, table 17.1 illustrates how the mathematical expression $1 + 2$, is represented as an AST with one *plus* node with two *integer* child nodes for the numbers 1 and 2, and the corresponding MathML representation.

### 17.5.4 Performance of LibSBML

XML parsers come in two popular varieties: Document Object Model (DOM) based and event-based. DOMs (Le Hors et al., 2000) are very generic in-memory structures that nearly duplicate the tree-like structure of the XML on disk. Using

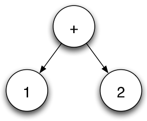
| **Infix** | | **AST** | | **MathML** |
|---|---|---|---|---|
| $1 + 2$ | $\Longleftrightarrow$ |  | $\Longleftrightarrow$ | `<math xmlns="http://www.w3.org/1998/Math/MathML">`<br>`  <apply>`<br>`    <plus/>`<br>`    <cn type="integer"> 1 </cn>`<br>`    <cn type="integer"> 2 </cn>`<br>`  </apply>`<br>`</math>` |

**Table 17.1** Illustration of a simple mathematical expression represented in both libSBML's AST structure and MathML (Ausbrooks et al., 2001).

a DOM simply moves the parsing bump under the rug. Instead of parsing a file, one now has to parse an in-memory data structure. Moreover, because DOMs are generic, needing to handle any XML that comes their way, one pays a penalty in terms of large memory consumption. Event-based parsers, on the other hand, allow programmers to intercept specific XML events (tags) and act on them. Event-based parsers are memory-efficient, but are often too low-level and fined-grained. They therefore lack the convenience of manipulating XML data in larger logical units.

LibSBML aims to strike a balance between DOM and event-based models of XML parsing. It provides the conveniences of a domain-specific object model while keeping memory usage to a minimum. Below, we compare the performance of libSBML, which uses the Xerces-C++ (Apache Software Foundation, 2004) event-based SAX parser under the hood, to parsing SBML with the Xerces-C++ DOM.

We obtained memory consumption statistics by writing two simple programs to read an SBML model from file into memory. One program used libSBML to read the model into domain-specific SBML objects and the other program used Xerces-C++ 2.6 to read the model into the W3C XML DOM format (Le Hors et al., 2000). Each program recorded its total resident memory consumption immediately before and after reading the model and reported the difference between these two numbers.

Total resident memory gives an estimate not only of the size of the model in memory, but also the size of the library and all supporting code that must be loaded into memory (often of concern to programmers). LibSBML was compiled with Xerces 2.6, so the amount of memory consumed by the Xerces library itself is the same for both programs.

We ran both programs over the 10,000+ models in the SBML Test Suite (SBML Team, 2005a) and models used in the first SBML Hackathon. Individual file sizes varied from 600 bytes to 5.76 MBytes. The runs were performed on computers running SuSE Linux 9.1 (Novell, Inc., 2005) with dual 64-bit AMD Opteron 2.2 GHz processors (Advanced Micro Devices, Inc., 2005).

Figure 17.5 shows a plot of the file size on disk versus the object model size in memory. While the Xerces-C++ 2.6 DOM is more efficient than previous implementations, the DOM consumed nearly five times as much memory for large multi-megabyte files. For small files (under five kilobytes), the DOM is ever so slightly more efficient. This is likely because Xerces uses string pooling and other reference counting techniques to optimize memory usage. For SBML files larger than five kilobytes and especially files larger than one megabyte, libSBML is the clear performance winner.

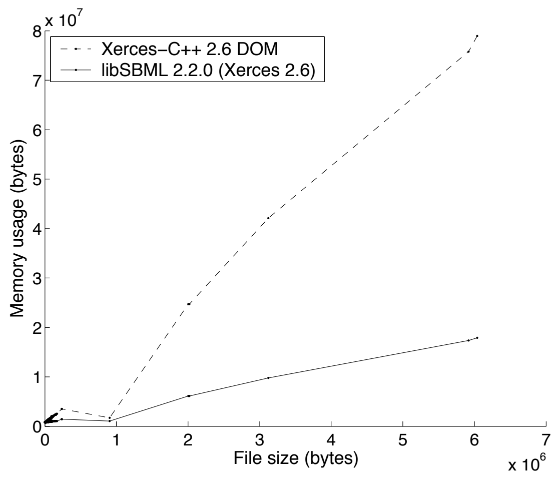


**Figure 17.5** A plot of memory consumption by libSBML (solid line) and the Xerces-C++ Document Object Model (dotted line), when each is used to read SBML models into computer memory. Data are based on over 10,000 sample models taken from the SBML Semantic Validation Suite and the first SBML Hackathon of 2003. File sizes (horizontal axis) varied from 600 bytes to 5.76 MBytes.

### 17.5.5 Helping Ensure Correctness and Consistency

Syntactic validation involves verifying that the SBML input is well-formed, and, for example, that data values are of the correct types. Consistency checking involves verifying the *contents* of an SBML model for self-consistency, referential integrity, and adherence to the SBML specifications. The tests are implemented as individual constraints within libSBML; the library reports back validation failures to the calling application via the libSBML API. The constraint checking system is modular, and the constraint set can be easily extended. We describe the design and intent of the constraint syntax below.

The design of SBML is driven by data models instead of the specifics of XML representation. To that end, the SBML specification is first described using UML static class diagrams. These class descriptions are mapped to XML representations using SCHUCS (Hucka, 2000), a technique we developed, tailored to producing efficient, reasonably succinct, and quasi-human-readable XML. We wanted to parallel our emphasis on data over representation, with a declarative language to express SBML model constraints (declarative languages state the *what* without specifying the *how*). For this, we took inspiration from the UML community and its development of OCL, the Object Constraint Language (Object Management Group, Inc., 2002; Warmer and Kleppe, 2003).

Although libSBML consistency checks are not expressed directly in OCL, we have created an OCL-like language on top of the libSBML C++ API. This language balances the readability of OCL with the efficiency and expressiveness of C++, which is sometimes necessary for more complicated validation procedures. The language allows the manipulation of only `const`ant C++ objects, which much like OCL, guarantees operations will be side-effect free. Further, this guarantee is enforced at compile time. Being side-effect free is an important property as we do not want the process of consistency checking to change the state of a model. An example will help make these concepts more clear.

One of the 50 consistency checks currently implemented ensures that if a model author overrides the default definition of the *substance* unit, a special unit name in SBML, the resulting unit definition is consistent with the notion of a substance. The consistency check constraint is written as:

```
START_CONSTRAINT (1202, UnitDefinition, ud)
{
  msg =
   "A 'substance' UnitDefinition must simplify to a single "
   "Unit of kind 'mole' or 'item' with an exponent of '1' "
   "(L2v1 Section 4.4.3).";

  pre( ud.getId() == "substance" );

  inv( ud.getNumUnits() == 1                          );
  inv( ud.getUnit(0).isMole() || ud.getUnit(0).isItem() );
  inv( ud.getUnit(0).getExponent() == 1               );
}
END_CONSTRAINT
```

The `START_CONSTRAINT` macro takes three arguments. The first is a number that uniquely identifies this constraint (that is, 1202). Assigning such identifiers to each constraint facilitates traceability and allows programmers to easily determine which rules have been violated. The next two parameters indicate the type of SBML object to which this rule applies (that is, `UnitDefinition`) and a shorthand name to use for the object being checked (that is, `ud`).

The body of the constraint consists of a message (`msg`) to be logged should the SBML object fail the check. After the message, zero or more preconditions (`pre`) may be listed. In order for the rule to apply to the SBML object in question, all preconditions must hold (in the order listed). If a precondition does not hold, the check is aborted without logging either a passage or failure. Finally, assuming all preconditions hold, the object's state must adhere to a set of one or more invariants (`inv`). Should any invariant fail, the constraint immediately fails and a message is logged.

In the above example, notice that preconditions and invariants are specified on the (lib)SBML object model. Each method invocation (operation) does not change the state of the model and specifies *what* not *how* (with apologies made for the standard names used for getter methods, for example, `getUnit()`, which arguably describes *how* and not *what*; even OCL falls victim to this slight, purely esthetic inconsistency.)

Finally, it's worth describing a case where OCL-like statements are not enough and having the full expressive power of C++ to write rules is advantageous. In SBML, compartments may be nested inside one another, with the limitation that this nesting may not be cyclic (an example of a cycle: compartment A is in B which is in C which is in A). While it is relatively easy to encode this constraint in the OCL-like language demonstrated above, reporting a user-friendly error message is another matter. Upon violation of this constraint, instead of simply stating that a cycle exists, it is better to indicate the chain of compartments that was followed to detect this cycle, thereby enabling the model author to quickly track down the cause of the error. Constructing such an informative error message is awkward in a purely declarative language like OCL. However, in C++, with its built-in Standard Template Library (STL) strings, sets, and the ability to iterate over collections, constructing an informative error message is straightforward.

### 17.5.6 Open-Source Development

We note with satisfaction that the open-source model of software development has been yielding dividends for libSBML. The user community has contributed not only several bug fixes, but new code as well. These include: support for the Expat parser library (Drake and Clark, 2005), a full Perl API, a full Lisp API, and an extension to support the use of a provisional SBML standard for storing model diagrams (see Section 17.4.3).

The libSBML open-source license allows it to be incorporated freely into other programs in whole or part. Several simulator programs and projects developed in academia already make use of libSBML to support both SBML import and export. Such simulator programs include: Gepasi (Mendes, 1997), COPASI (Mendes, 2003), Jarnac (Sauro, 2000), and the DARPA Bio-SPICE project (Kumar and Feidler, 2003). It is worth mentioning that since libSBML is distributed under the terms of the Lesser GNU Public License (LGPL) it may also be used without restriction in commercial applications (Free Software Foundation, 1999). We currently know of two commercial software applications using libSBML.

## 17.6 Validating Application Behavior

When we first developed SBML, we expected that most of the difficulties faced by developers in implementing software support would stem from issues of constructing and parsing valid model structures. We knew it would be impossible to write perfectly clear specifications for the language, but we expected that once issues of ambiguities and other problems in SBML's definition were overcome, interchange of models between software tools would naturally follow. And to a surprising extent, this was true for a few early applications such as Jarnac and Gepasi—exactly the same applications that informed the definition of SBML in the first place. It was not until a large number of other software developers began working with SBML that it became clear the community faced more subtle issues of model interpretation and consensual agreement about expected behaviors of simulation tools.

### 17.6.1 Types of Validation

At the highest level, we can partition the question of validity into two main categories:

1. *Syntactic*: does the software accept well-formed SBML input, and reject all syntactically invalid SBML input? (Note that a software package may reject some valid SBML inputs because it detects the presence of constructs it is not designed to handle. For the purposes of syntactic verification, such behavior is acceptable and presumably can be distinguished from a failure to accept well-formed SBML.)
2. *Semantic*: does the software interpret well-formed SBML correctly? This can be further divided:
   (a) *Model structure*: does the software construct the correct model structure based on the SBML input, independent of what it does with that structure?
   (b) *Model behavior*: does the software correctly interpret or generate the intended model behavior?

The difference between the two types of semantic validation is about structure versus dynamics. Going beyond verification of conformance to SBML syntax, the semantic interpretation of a model involves both creating the intended constructs based on the SBML and analyzing or simulating the model in the intended way. In both cases, correctness is something that has to be carefully specified.

Some models can only be evaluated based on their structure. For example, molecular interaction models may not contain any kinetic information, so it is not clear that there is a definable model behavior per se. In that case, the model may be only evaluable based on the model structure. Other models have dynamics, and software tools can be evaluated based on whether they produce agreed-upon simulation or analysis results.

### 17.6.2 A Problem Not Addressed by Definitions Alone

The problem of achieving "agreed-upon simulation and analysis results" goes deeper than stipulating the required syntactic and semantic aspects of SBML and providing model structure-based verification of consistency of the sort now available in libSBML (Section 17.5.5). At least two issues must be addressed. One is the problem of reaching a consensus in a community about how to to interpret different classes of models. This is a problem of education and communication, which in the case of SBML is being helped tremendously by the biannual SBML face-to-face events

(SBML Forums and Hackathons). A second problem is providing a way for software developers to verify the behaviors of their software tools vis-a-vis the consensus view of simulator behaviors. This requires testing the behavior of software that interprets and manipulates models encoded in SBML.

To help address this latter problem, we have recently introduced the first version of the SBML Semantic Validation Suite, described in the next section.

### 17.6.3 The SBML Semantic Validation Suite

The Semantic Validation Suite consists of (1) a set of valid SBML models each with representative, simulated time-course data, and (2) a scripted, automated testing framework for running software tools through the suite. This suite is designed to be used by software developers to check that their simulators produce results that are consistent with the SBML standard and thus with each other.

In the general case, verifying the interpretation of SBML by an arbitrary software package is an extremely challenging problem, since different applications use models in different ways, generate different types of outputs, and provide different user interfaces. The only realistic way to approach this problem systematically is to tackle different application types separately, treating ODE-based simulators as one type, stochastic simulators as another, pathway analysis tools as another, etcetera. We chose to develop tests for ODE-based simulators first because: (a) this kind of simulation software makes up a significant proportion of the applications that support SBML; (b) simulation is one of the more complex types of analysis that can be applied to SBML; and (c) apart from metadata, almost all SBML features impact a model's behavior in simulation.

The set of models in the SBML Semantic Validation Suite is still incomplete, but the current version covers the majority of SBML features. The suite is divided into categories of tests, where each category deals with a set of related features of SBML. The scripts in the suite allow a simulator to be tested systematically against the test set. Each test in the suite comes with: (1) the correct simulation output in a consistent documented format; (2) plots of correct simulation output, and (3) documentation for the test. The beta version of the test suite was announced in October 2004. Several developers have begun using the suite as part of their work and communicating feedback to us about the suite itself; this feedback process is helping us to improve every aspect of it.

Our long-term goal in this effort is to eventually produce a highly automated software evaluation system. We hope to be able to generate an in-depth guide that categorizes different tools along different dimensions related to their purposes and coverage of SBML features. This will be an important aid both to potential users (who will be able to easily compare the functionality of different software packages) and to developers (who will be able to use the evaluation tools to help guide their implementation of SBML support during software development). We also believe the content will be useful for researchers wishing to understand SBML on its own.

## 17.7 Summary

Computational modeling is becoming crucial for making sense of the vast quantities of complex experimental data that are now being collected. The systems biology community needs agreed-upon information standards if models are to be shared, evaluated, and developed cooperatively. The Systems Biology Markup Language (SBML) is an XML-based format for representing computational models in a way that can be used by different software systems to communicate and exchange those models. It is supported today by over 80 software tools worldwide and a vibrant community of modelers and software authors. A variety of resources are available for working with SBML; there is also an Internet MIME type defined for SBML (Kovitz, 2004) and a new public database of models based around SBML (BioModels Team, 2005).

In support of SBML and its community, we continue to develop and make available software infrastructure, including programming libraries, conversion utilities, interface packages for commonly-used software environments, and easy-to-access online tools. All of our software development follows the open-source tradition to maximize the accessibility and utility of the products.

The success of SBML has led to requests from the community for new features and continued evolution of the language. We view our role as organizers and editors in the development and evolution of SBML; the process is open and crucially dependent on the involvement of others in the computational modeling field. We invite interested individuals and groups to join the SBML Forum, the informal community of SBML users and developers, to participate in the process and help us improve SBML and its capacity for acting as a common exchange format for computational modeling software in systems biology. Information on this and other aspects of the SBML project is available on the project Web site (SBML Team, 2005b).

## 17.8 Acknowledgments

We thank Herbert Sauro for his fundamental work on SBML Level 1 as well as crucial discussions and software development work. We also thank Hamid Bolouri for organizing and leading the SBML effort during its first two and a half years, and we thank the SBML development community for their continuing enthusiasm, participation, feedback, and support. The SBML community includes the members of the sbml-discuss@caltech.edu international mailing list and the DARPA Bio-SPICE project Model Definition Language task force. Finally, we thank the following agencies and institutions for their generous support. The development of SBML was originally funded by the Japan Science and Technology Corporation (JST) under the ERATO Kitano Symbiotic Systems Project. Support for the continued development of SBML and associated software, meetings, 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 JST ERATO-SORST Program (Japan); the Japanese Ministry of Agriculture; the Japanese Ministry of Education, Culture, Sports, Science, and Technology; the BBSRC e-Science Initiative (UK); the DARPA IPTO Bio-Computation Program (USA); and the Air Force Office of Scientific Research (USA). Additional support 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).