Supporting SBML as a Model Exchange Format in Software Applications

2 authors:

Sarah M. Keating
California Institute of Technology
54 PUBLICATIONS   **2,033** CITATIONS

Nicolas Le Novère
Babraham Institute
348 PUBLICATIONS   **17,756** CITATIONS

Some of the authors of this publication are also working on these related projects:

JSBML: a Java library for manipulating SBML View project

Impact of Technology on an Organization Performance (Motorbike Industry of Pakistan) View project

Chapter 11

Supporting SBML as a Model Exchange Format in Software Applications

Sarah M. Keating and Nicolas Le Novère

Abstract

This chapter describes the Systems Biology Markup Language (SBML) from its origins. It describes the rationale behind and importance of having a common language when it comes to representing models. This chapter mentions the development of SBML and outlines the structure of an SBML model. It provides a section on libSBML, a useful application programming interface (API) library for reading, writing, manipulating and validating content expressed in the SBML format. Finally the chapter also provides a description of the SBML Toolbox which provides a means of facilitating the import and export of SBML from both MATLAB and Octave (http://www.gnu.org/software/octave/) environments.

Key words SBML, Systems biology standards, Reproducibility, Exchange format, Language

1 Introduction

Systems Biology projects take into account the interactions between a very large number of physical entities and the analysis of many parameters. As seen in the previous chapters, the quantitative relationships between entities and their interactions are often described using mathematical models and there exists a variety of software applications that can be used for different types of analysis. Early modellers and software developers in systems biology quickly realised that if their efforts were to be of benefit to the wider community it must be possible to share and reuse the models. The best way to facilitate this, and to enable concurrent use of multiple software packages with different capabilities, was to agree a common format for describing the models.

There are many ways to describe models in a standardised manner. One can use natural languages, graphical languages, sets of equations, logical relationships between elements, etc. The need for a language to exchange models became manifest at the end of the last century, with efforts starting in the field of metabolic networks [1] and physiology modelling [2]. A similar need was expressed

during the first Workshop on Software Platforms for Systems Biology held at the California Institute of Technology in early 2000.

The result was the Systems Biology Markup Language (SBML) [3]; a machine-readable model definition language based on XML, the eXtensible Markup Language [4].

2   SBML

An SBML document contains all the information pertaining to the structure of a model, including the list of symbols, variables and constants, the list of mathematical relationships linking them, and all the numbers needed to instantiate simulations. SBML was originally viewed as being aimed towards models of molecular pathways [5]. However, its versatility means that SBML can be, and today is being, used in a variety of modelling contexts. For instance, BioModels Database [6] contains SBML representations of models including cell signalling [7], metabolism [8], gene regulation [9] but also nonmolecular representations of cellular processes [10], models of neurons [11], treatment of tumours [12] or even of zombie invasions [13]. In general, SBML enables the encoding of any mathematical model based on pools of entities and processes that modify them. This versatility is currently expanding towards rule-based modelling, reaction–diffusion, logical modelling etc. Since its creation in 2000, SBML has continued to evolve as an international community effort, and has grown in terms of the levels of acceptance to the point where, at the time of this writing, it is used by over 200 software packages worldwide and required as a format for model encoding by many journals.

2.1   SBML Development

SBML has been, and continues to be, developed in stages, with specifications released at the end of each development stage. This approach, which effectively freezes SBML development at incremental points, allows users to work with stable standards and gain experience with the standard before further development. Future development can then benefit from the practical experiences of users and developers.

Major editions of SBML are termed *levels* and represent substantial changes to the composition and structure of the language. The latest level being developed is Level 3 [14] representing a major evolution of the language through Level 2 [15] from the introduction of Level 1 in the year 2001 [3, 16]. SBML Level 3 is being developed as a modular language, with a central core comprising a self-sufficient model definition language, and extension packages layered on top of this core to provide additional, optional sets of features.

The separate levels of SBML are intended to coexist. All of the constructs of Level 1, i.e., the elements and attributes of the SBML representation can be mapped to Level 2; likewise, the majority

of the constructs from Level 2 can be mapped to Level 3 Core. In addition, a subset of Level 3 constructs can be mapped to Level 2 and a subset of Level 2 constructs can be mapped to Level 1. However, the levels remain distinct; a valid SBML Level 1 document is not a valid SBML Level 2 document and so on.

Minor revisions of SBML are termed *versions*, and constitute changes within a Level to correct, adjust and refine language features.

2.2 Structure of SBML Level 3 Core

SBML is a structured language with a strict syntax and very precise semantics. A serious understanding of the language can only be achieved through the SBML specification document [14]. In this section we will present a basic overview of the common constructs of SBML. The later sections look at code designed to intuitively work with the SBML constructs and attributes and will provide more insight into the SBML language itself.

This text restricts itself to SBML Level 3 Core and does not go into any detail relating to the L3 packages that are being developed to extend the core both in terms of SBML development and also the development of the software discussed in the later sections of the chapter.

2.2.1 The SBML Element

An SBML document is essentially an XML document containing an **sbml** element which declares the *namespace*, *level* and *version* of SBML. The **sbml** element MUST contain a **model** element which itself consists of lists of one or more components. This SBML snippet illustrates an **sbml** element containing a **model** element.

```
<?xml version="1.0" encoding="UTF-8"?>
<sbml
xmlns="http://www.sbml.org/sbml/level3/version1/
core"
      level="3" version="1">
   <model/>
</sbml>
```

2.2.2 The Model Element

Some components in SBML (see Table 1) represent items that have a numerical value that may be constant or may vary throughout a simulation. The constructs that represent possible variables are compartment, species and parameter. In all these cases, the *id* attribute of the component is used throughout the model to represent the numerical value of that component at the point in time specified by any simulation/analysis that is being undertaken.

It is also possible to introduce variable stoichiometry into reactions using the id of the speciesReference listed as a product or reactant within a reaction.

Other components represent mathematical constructs that define some level of interaction between the components that can be varied. These constructs include the reaction, rules and event components.

The remaining constructs: initialAssignment, functionDefinition, constraint and unitDefinition provide methods of adding further information or mathematical detail to a model.

Table 1
Components of an SBML Level 3 core model element

Component description	
Compartment	A container of finite size for well-stirred substances
Species	A pool of undistinguishable entities
Parameter	A quantity of whatever type is appropriate
Reaction	A statement describing some transformation, transport or binding process that can change one or more species
Rule	A mathematical expression that is added to the model equations
Event	A set of mathematical formulas evaluated at specified moments in the time evolution of the system
Initial assignment	A mathematical formula to assign the initial value of a component
Function definition	A named mathematical function that can be used in place of repeated expressions
Constraint	A mathematical formula for stating the assumptions under which the model is designed to operate
Unit definition	A name for a unit used in the expression of quantities in a model

2.2.3 Elements Providing Variables

In Subheading 2.2.2 it was noted that the compartment, species and parameter components of the **model** element represent items that have a numerical value that may be varied in the simulation of a model. The SBML specification assigns attribute values to these components that allow the user to specify initial values, units and whether the particular instance of a component can be varied by other constructs within the model. Some attributes are required and others have default values in some SBML Levels. Full details are available in the SBML specifications [14].

As a quick illustration consider a model that contains two species and a parameter. Since species must be located within a compartment, it will also contain a compartment. The **compartment** is a constant, 3D container of volume 2.3 L. Within this compartment are two **species**. There is also a fixed **parameter** with value 3000 per second. It is possible to define this unit in SBML but that is left as an exercise for the reader.

The **species** component in SBML does not represent a single molecule but rather a pool, that is an ensemble of indistinguishable entities, represented by its concentration or amount in a **compartment**. The environment is well stirred and thus no concentration gradients need to be considered. The first species has an initial amount of 4.6 mol and the second an initial amount of 1 mol. These are reacting and therefore the amounts will vary throughout a simulation.

An SBML model specifying these components is shown below.

```
<model>
   <listOfCompartments>
      <compartment id="cell" spatialDimensions="3"
               size="2.3" units="litre"
         constant="true"/>
   </listOfCompartments>
   <listOfSpecies>
      <species id="s1" compartment="cell"
               initialAmount="4.6"
               substanceUnits="mole"
         hasOnlySubstanceUnits="false"
               boundaryCondition="false"
         constant="false"/>
      <species id="s2" compartment="cell"
               initialAmount="1"
               substanceUnits="mole"
         hasOnlySubstanceUnits="false"
               boundaryCondition="false"
         constant="false"/>
      <species id="s2" compartment="cell"
               initialAmount="1"
               substanceUnits="mole"
         hasOnlySubstanceUnits="false"
               boundaryCondition="false"
         constant="false"/>
   </listOfSpecies>
   <listOfParameters>
         <parameter id="p" value="3000"
   constant="true"/>
   <listOfParameters>
</model>
```

The species specified above have initial amounts specified in moles. However, the *hasOnlySubstanceUnits* attribute has a value of false, indicating that whenever the *id* of the species appears in the model it refers to concentration.

Thus for any analysis, it may be necessary to convert between amount and concentration using

$$\text{Concentration} = \frac{\text{Amount}}{\text{Size}}$$

where size refers to the *size* of the **compartment** in which the **species** is located.

It is possible to create models in SBML without the need to consider units and thus units have largely been ignored within this text. However, in the situation where a model uses species that have been located within a compartment whose size is not unity the issue of concentration and amount **must** be considered.

There are several elements in SBML that allow the user to define relationships between variables within the model.

We will use a **reaction** to illustrate this, but there are other constructs (listed in Table 1) that can also be used.

A **reaction** in SBML represents any kind of process that can change the quantity of one of more **species**. It may be a mass action reaction, or involve transport, catalysis or any process that changes the species involved (note that transport changes species because they are located in compartments). It is necessary to define the participating reactants and/or products. This is done using a **speciesReference** component that identifies the species from the model's **listOfSpecies** and assigns a *stoichiometry* value to that species role within the reaction. Species that merely influence a reaction, such as a catalyst, are listed as objects of type **modifierSpeciesReference**. This construct is similar to **speciesReference** without the *stoichiometry* attribute. Attributes for a **reaction** object allow the modeller to specify whether the reaction is *reversible* or *fast*. The mathematics describing the velocity of the reaction can be encoded in the **kineticLaw** component. SBML uses a subset of the MathML 2.0 standard [17] to encode mathematical formula directly within SBML components. Note that an SBML **kineticLaw** represents the extent of the reaction per unit of time, and not the rate of the reaction. In other words, the result is not a concentration per time, but a quantity per time. This is why the rate is multiplied by the volume in the **kineticLaw**.

The SBML snippet shows the description of the reaction

$$s1 \xrightarrow{\ k\ } s2$$
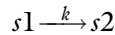
with a rate of

$$p \times s1$$

where $s1$ and $s2$ are two species residing in compartment *cell* and p is a parameter.

```
<model>
 …
<listOfReactions>
<reaction reversible="false"
   fast="false">
<listOfReactants>
     <speciesReference species="s1"
            stoichiometry="1"/>
   </listOfReactants>
   <listOfProducts>
     <speciesReference species="s2"
          stoichiometry="1"/>
</listOfProducts>

<kineticLaw>
   <mathxmlns="http://www.w3.org/1998/Math/MathML">
      <apply>
```

```
            <times/>
            <ci> p </ci>
            <ci> s1 </ci>
            <ci> cell </ci>
          </apply>
        </math>
      </kineticLaw>
     </reaction>
    </listOfReactions>
  ...
 </model>
```

2.2.5 The Mathematical Model

An SBML document contains all the information pertaining to the structure of a model. However, it does not directly contain the system of mathematical equations that describes the behaviour of the model. It is therefore necessary for software using SBML to reconstruct the mathematics needed to perform the required analysis. In some cases, such as assignments, the correct equations can be extracted fairly directly from the SBML constructs.

The SBML snippet here shows two rules, an **assignmentRule** and a **rateRule** from which the equations can be easily extracted.

$$y = 2x + 1$$

$$\frac{\mathrm{d}g}{\mathrm{d}t} = g - 1$$

```
<model>
  ...
  <listOfRules>
    <assignmentRule variable="y">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <plus/>
          <apply>
            <times/>
            <cn> 2 </cn>
            <ci> x </ci>
          </apply>
          <cn> 1 </cn>
        </apply>
      </math>
    </assignmentRule>
    <rateRule variable="g">
      <math xmlns="http://www.w3.org/1998/Math/MathML">
        <apply>
          <minus/>
          <ci> g </ci>
          <cn> 1 </cn>
        </apply>
      </math>
    </rateRule>
  </listOfRules>
  ...
</model>
```

In other cases, the process necessitates more complex procedures for instance extracting the equations from a set of reactions. As mentioned in Section/refsec:reactions one of the specificities of SBML is the representation of biological networks as a set of processes (SBML **Reactions** that are converting pools of entities (SBML **Species**) into others). Each process (SBML **Reaction**) is associated with a kineticLaw and lists the entities affected (SBML **SpeciesReferences**) characterised by their stoichiometry for this process. SBML does not contain the ODEs describing the evolution of the pools. Software must reconstruct them from the list of all reactions and associated stoichiometries. This approach, which is in direct contrast to CellML [2] where the focus is on describing the mathematical model, makes the maintenance of models easier. Adding or removing a reaction does not require carefully exploring all the equations. It also permits the use of a given computational model within different simulation frameworks, for example both deterministic or stochastic simulations can be constructed from the same set of SBML **Reactions**.

3 LibSBML

LibSBML [18] is an application programming interface (API) library for reading, writing, manipulating and validating content expressed in the SBML format. It is written in ISO C and C++, provides language bindings for .NET, Java, Python, Perl, Ruby, MATLAB and Octave, and includes many features that facilitate the adoption and use of both SBML and the library. LibSBML is freely available as source code and binaries for all major operating systems under the LGPL open source terms.

Developers can embed libSBML in their applications, saving themselves the work of implementing their own SBML parsing, manipulation and validation software.

3.1 SBML and the libSBML API

LibSBML uses objects (classes) that correspond to SBML components with member variables that represent the attribute values. The API is constructed to provide an intuitive way of relating SBML and the code needed to create or manipulate it. LibSBML has extensive documentation available both online and as a separate documentation archive available with each libSBML release. The C++ documentation is the most extensive but documentation specifically tailored for many of the other language bindings is available.

Here we provide an outline of key features of the library used in conjunction with some of the constructs of SBML that should provide a user with a basic starting point. The code examples use a combination of sample code and notes to provide further detail

and should be considered as part of the text. Code snippets use python but the API is identical for all the major languages supported. It should be noted that the bindings for MATLAB and Octave do differ from the general libSBML API and are considered separately in Subheading 4.

3.2 Reading and Writing SBML

LibSBML enables reading from and writing to either files or strings. Once read in libSBML stores the SBML in an **SBMLDocument** object which can later be written out. Thus any of functions shown could be used to read and write SBML from a file or string.

```python
>>> import libsbml
# read a document
>>> doc = libsbml.readSBMLFromFile(filename)
>>> doc = libsbml.readSBMLFromString(string)
# helper function that takes either a string
# or filename as argument
>>> doc = libsbml.readSBML(filename)
>>> doc = libsbml.readSBML(string)
# write a document
>>> libsbml.writeSBMLToFile(doc, filename)
>>> True
>>> libsbml.writeSBMLToString(doc)
>>>'<?xml version="1.0" encoding="UTF-8"?>\n
<sbml
xmlns="http://www.sbml.org/sbml/level3/version1/
core"
   level="3" version="1">\n
 <model/>\n
</sbml>\n'
```

The doc object produced is an instance of the **SBMLDocument** class which represents the **sbml** element.

The **SBMLDocument** contains one instance of a **Model** object which in turn contains the ListOfXYZ classes representing the SBML elements contained within the model element (see Fig. 1). The API allows the user to retrieve sub elements from a parent. Assuming that the SBML snippet of Subheading 2.2.1 has been read in, each of these ListOfXYZ objects is an empty list since the **model** element is empty.
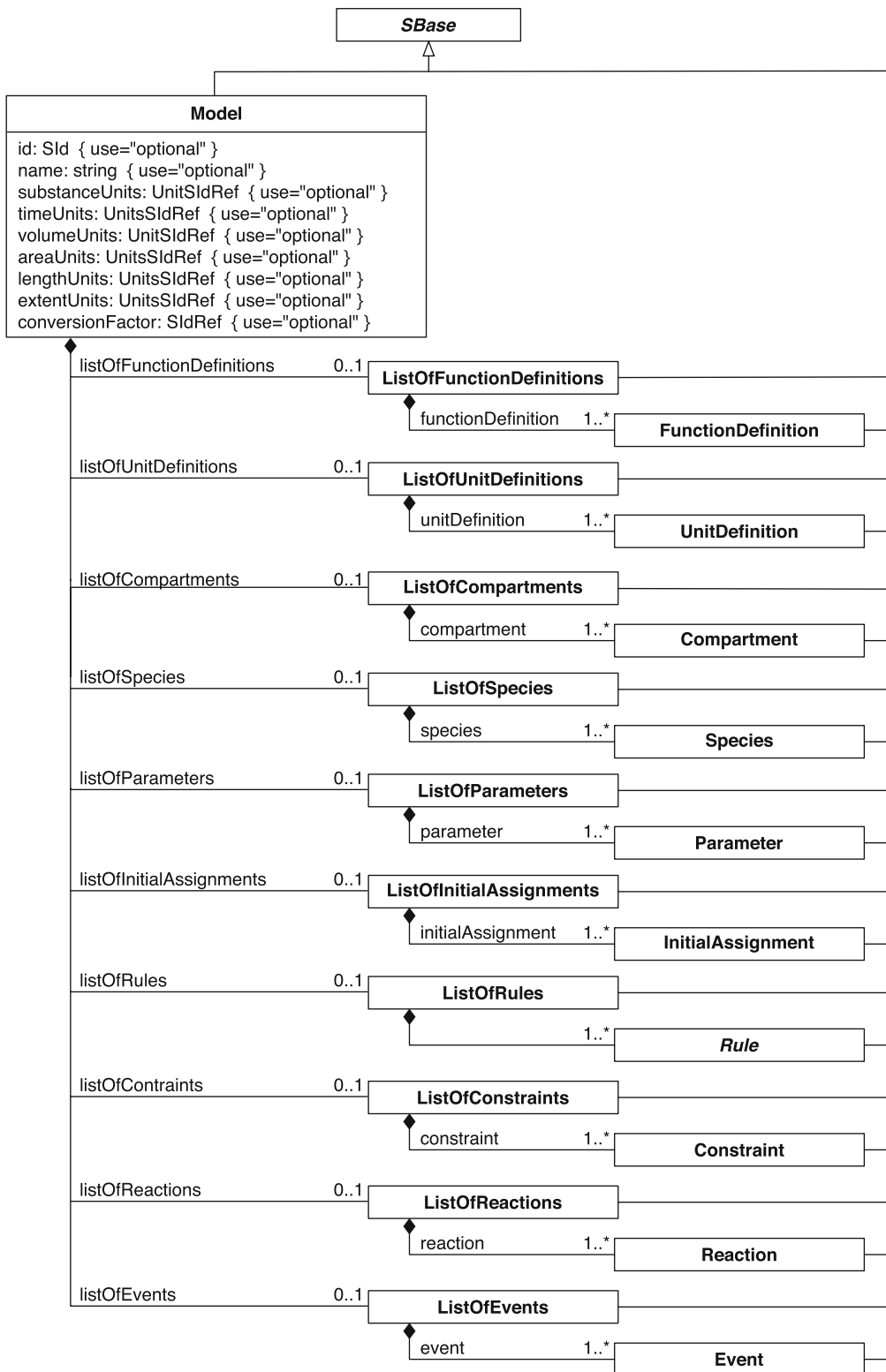
Fig. 1 This figure illustrates the structure of the **model** element. The various components are located within the relevant "ListOfXYZ" components

```
>>> import libsbml
>>> reader = libsbml.SBMLReader()
>>> doc = reader.readSBMLFromFile(filename)

>>> model = doc.getModel()

# there are no compartments in the list
>>> model.getNumCompartments()
0

# the listOfSpecies will exist but be empty
>>> model.getListOfSpecies()
<libsbml.ListOfSpecies;
  proxy of <Swig Object of type 'ListOfSpecies *'
  at 0x02D48F98> >
>>> model.getNumSpecies()
0

# individual objects (if they exist) can be retrieved
# by index or identifier
>>> p1 = model.getParameter(1)
>>> p2 = model.getParameter("p")
```

3.3 Creating and Manipulating SBML

The libSBML API allows easy creation of objects and sub objects representing SBML elements and the sub elements contained within them.

```
>>> import libsbml
# create an SBML Level 3 Version 1 document
>>> sbmlns = libsbml.SBMLNamespaces(3,1)
>>> doc = libsbml.SBMLDocument(sbmlns)

#create the model as a sub element of the document
>>> model = doc.createModel()
#create a compartment as a sub element of the model
>>> compartment = model.createCompartment()
# note the compartment is created and
# added to the listOfCompartments within the model
>>> model.getNumCompartments()
1
```

3.3.1 SBMLNamespaces

LibSBML allows you to work with all levels and versions of SBML. In order to facilitate this there is an **SBMLNamespaces** object that records the level, version and appropriate namespaces for each object. A sub object inherits the **SBMLNamespaces** from the containing document object. It is however possible to create objects prior to adding them to a document, in this case libSBML does check that there is consistency between **SBMLNamespaces** before adding objects. It is considered best practice to create a document and then create the sub elements directly from the document as in the example above.

```
<model>
   ...
   <listOfCompartments>
     <compartment id="cell" spatialDimensions="3"
                  size="2.3" units="litre" constant="true"/>
   </listOfCompartments>
   ...
</model>
```

SBML snippet 1: An SBML compartment

3.3.2 *SBML Component Example: Compartment*

The **compartment** component has attributes that specify its *spatialDimensions*, its *size* and the corresponding *units*, plus a *constant* attribute that determines whether the size can change or not during a simulation. The SBML snippet 1 represents a constant, 3D compartment with volume 2.3 L.

The libSBML API provides functions to set and retrieve each attribute, functions to check whether an attribute has been set and, in some cases, functions to unset an attribute. For all components and attributes these functions follow a standard form.

```
>>> import libsbml

# create an SBML Level 3 Version 1 document
>>> sbmlns = libsbml.SBMLNamespaces(3,1)
>>> doc = libsbml.SBMLDocument(sbmlns)

#create the model as a sub element of the document
>>> model = doc.createModel()

#create a compartment as a sub element of the model
>>> compartment = model.createCompartment()
# set the attributes on the compartment
# note a return value of 0 indicates success
>>> compartment.setId("cell")
0
>>> compartment.setSize(2.3)
0
>>> compartment.setSpatialDimensions(3)
0
>>> compartment.setUnits("litre")
0
>>> compartment.setConstant(True)
0

# get the attribute values
>>> compartment.getId()
```

```
'cell'
>>> compartment.getSpatialDimensions()
3

# examine the status of the attribute
>>> compartment.isSetSize()
True
>>> compartment.getSize()
2.3

#unset an attribute value
>>> compartment.unsetSize()
0
>>> compartment.isSetSize()
False
>>> compartment.getSize()
Nan
```

3.4   Validation

In addition to a strict syntax for the structure of the language the SBML specifications list a number of Validation Rules that indicate further requirements for fully valid SBML. The rules also include optional conditions for applying particular non-required types of consistency and those things recommended as best practices.

LibSBML provides a rich API for selecting and applying the various validation rules. It is possible to disable validators that deal with concepts that are not of interest to the user. For example, unit consistency is not a requirement of SBML. Software developers have differing views on whether they want units to be consistent and thus the ability to deselect unit validation is particularly useful. The following illustrates a case where turning unit validation off removes a warning from the list of validation errors reported.

```
>>> import libsbml

>>> reader = libsbml.SBMLReader()
>>> doc = reader.readSBMLFromFile(filename)

# validating the document returns the number of errors
>>> doc.validateSBML()
1
# querying the error log displays the error
>>> doc.getErrorLog().toString()
```

```
"line 9: (99505 [Warning]) In situations where a math-
ematical expression contains literal numbers or para-
meters whose units have not been declared,it is
not possible to verify accurately the consistency of
the units in the expression. The units of the
<assignmentRule> <math> expression 'k * 2' cannot be
fully checked. Unit consistency reported as either no
errors or further unit errors related to this object may
not be accurate.\n\n"
```

```
# turn the unit consistency checks off
>>>  doc.setConsistencyChecks(libsbml.LIBSBML_CAT_
UNITS_CONSISTENCY, False)
```

```
# validate the document again with unit checking off
>>> doc.validateSBML()
0
```

3.5 Working with Multiple SBML Levels/Versions

It has been noted in Subheading 2.1 that the levels and versions of SBML are intended to coexist. LibSBML provides a uniform API that seamlessly covers all SBML Levels and Versions, making it significantly easier for software developers to support the different definitions in their applications. The functions used to set attribute values will return error codes to indicate that the particular attribute or indeed the given value is not appropriate for the level and version of SBML being used. The code below provides examples of some of the error codes that may be returned.

```
>>> import libsbml
# create a L3 compartment and set spatialDimensions to
    4.5
# return code 0 indicates success
>>> compartmentL3 = libsbml.Compartment(3,1)
>>> compartmentL3.setSpatialDimensions(4.5)
0 # libsbml.LIBSBML_OPERATION_SUCCESS
```

```
# try the same with an L2 compartment
# return code -4 indicates that the value of 4.5 is not
allowed
>>> compartmentL2 = libsbml.Compartment(2, 4)
>>> compartmentL2.setSpatialDimensions(4.5)
-4 # libsbml.LIBSBML_INVALID_ATTRIBUTE_VALUE
```

```
# with L1
# return code -2 indicates that the spatialDimensions
# attribute does not exist in L1
>>> compartmentL1 = libsbml.Compartment(1,2)
>>> compartmentL1.setSpatialDimensions(4.5)
-2 # libsbml.LIBSBML_UNEXPECTED_ATTRIBUTE
```

In some cases developers prefer to target their own analysis software to one particular Level/Version of SBML.

LibSBML provides a method of converting models between levels and versions thus facilitating this approach.

It should be noted that it is not always possible to convert constructs from higher levels to lower levels. Events cannot be described in the Level 1 format. However there are some attributes, for example *sboTerm*, that provide semantic information that may not be crucial to the mathematical understanding of the model and thus converting to a lower level may remove the attribute, whilst leaving the mathematical model intact. Other constructs, such as initialAssignment and functionDefinition, can be converted. LibSBML will only perform a conversion if the set of mathematical equations that would be derived from both the source and target model are identical. The functions report success or failure; and querying the error log of the document will return any warnings about the conversion that has been done or errors that indicate why it could not proceed.

```
>>> import libsbml

>>> reader = libsbml.SBMLReader()
>>> doc = reader.readSBMLFromFile(filename)

# by default conversion will not happen if the source
model is invalid
# or the resulting model would be invalid
>>> doc.setLevelAndVersion(1,2)
False
>>> doc.getErrorLog().toString()
'line 4: (91003 [Warning]) Conversion of a model with
<constraint>s to
SBML Level 1 may result in loss of information.\n\n
line 1: (91014 [Warning]) SBML Level 2 Version 4 removed
the requirement
that all units be consistent. This model contains units
that produce
inconsistencies and thus conversion to Level 1 would
produce an invalid
model.\n\n'
```

```
# this behaviour can be overridden by changing the
strict flag to false
>>> strict = False
>>> doc.setLevelAndVersion(1,2,strict)
True
```

```
<species id="Ca_calmodulin" metaid="cacam" sboTerm="SBO:0000297"
        compartment="C" hasOnlySubstanceUnits="false"
        boundaryCondition="false" constant="false">
  <annotation>
    <rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
              xmlns:bqbiol="http://biomodels.net/biology-qualifiers/">
      <rdf:Description rdf:about="#cacam">
        <bqbiol:hasPart>
          <rdf:Bag>
            <rdf:li rdf:resource="http://identifiers.org/uniprot/P62158"/>
            <rdf:li rdf:resource="http://identifiers.org/obo.chebi/CHEBI%3A29108"/>
          </rdf:Bag>
        </bqbiol:hasPart>
      </rdf:Description>
    </rdf:RDF>
  </annotation>
</species>
```

SBML snippet 2: An SBML species that corresponds to a "protein complex" (SBO:0000297). It is made up of two parts, the protein calmodulin (described by the UniProt entry P62158) and the divalent calcium cation (ChEBI term CHEBI:29108)

3.6 Working with MIRIAM RDF Annotation

In addition to the model semantics, that is the variables and their mathematical relationships, SBML provides a mechanism for adding attribution information and a layer of biological semantics on top of each component of the model. This is discussed in detail in another chapter.

LibSBML provides an API for interacting with **CVTerms** and **ModelHistory** elements. The following code would produce the SBML Snippet 2.

```
>>> import libsbml
```

```
# create the species and set the metaid attribute
>>> s = libsbml.Species(3,1)
>>> s.setMetaId("cacam")
```

```
# create a new cvTerm that is the biological qualifier
"hasPart"
>>> cv = libsbml.CVTerm(libsbml.BIOLOGICAL_
QUALIFIER)
>>> cv.setBiologicalQualifierType("hasPart")
```

```
O


# add resources to the cvterm
>>>    cv.addResource("http://identifiers.org/uni-
prot/P62158")
O


>>>    cv.addResource("http://identifiers.org/obo.
chebi/CHEBI\%3A29108")
O


# add the cvterm to the species
>>> s.addCVTerm(cv)
O
```

3.7 A Note on JSBML   With the exception of MATLAB and Octave, the language bindings for libSBML are automatically generated.

Unfortunately this limits the platform independence brought by the use of Java because the binding is only a wrapper around the C/C++ core, implemented using the Java Native Interface (JNI). As a consequence, some software developers have experienced difficulties deploying portable libSBML-based Java applications with high concurrent usage. JSBML [19], a Java library for SBML, addresses this issue by providing an API that maps all SBML elements to a flexible and extended Java type hierarchy whilst striving for 100 % compatibility with the libSBML Java API.

4   SBMLToolbox

Many modellers use a mathematical environments such as MATLAB (http://www.mathworks.com) as this provides the computational power they need for the simulation/analysis they wish to perform. Unfortunately models constructed in such an environment do not have a proscribed structure and this makes it difficult, if not impossible, to share and reuse these models. SBMLToolbox, together with the matlab binding for libSBML, provides a means of facilitating the import and export of SBML from both the MATLAB and Octave (http://www.gnu.org/software/octave/) environments. All the functionality of the libSBML binding and SBMLToolbox is compatible with Octave.

SBMLToolbox [20] is not intended to be a fully fledged simulation tool, although some simulation capability is available. It is primarily intended to demonstrate how structures in the MATLAB environment can be used to represent models and how these structures can be compatible with SBML. Here we provide a basic overview of SBMLToolbox. Full documentation is available online and with each SBMLToolbox release.

4.1   The MATLAB SBML Structure

The term *MATLAB SBML Structure* refers to the in-memory data structure used by the libSBML bindings and SBMLToolbox to represent an SBML model in the mathematical environment. A complete model is shown below. This mimics the elements of SBML and how they are contained within each other; again see Fig. 1. In the MATLAB SBML structure the name 'listOfXYZ' has been dropped as the structure contains arrays of the individual elements but the concept of the Model object remains identical to that of SBML and the classes used in libSBML.

```
model =

             typecode: 'SBML_MODEL'
               metaid: ''
                notes: [1x281 char]
           annotation: ''
           SBML_level: 2
         SBML_version: 1
                 name: ''
                   id: 'Branch'
   functionDefinition: [1x0 struct]
       unitDefinition: [1x0 struct]
          compartment: [1x1 struct]
              species: [1x4 struct]
            parameter: [1x0 struct]
                 rule: [1x0 struct]
             reaction: [1x3 struct]
                event: [1x0 struct]
          time_symbol: ''
         delay_symbol: ''
           namespaces: [1x1 struct]
```

The structure is a standard MATLAB/Octave structure, with the fieldnames representing SBML attributes or arrays of SBML sub elements. These can be easily accessed using the fieldnames and by indexing into any array.

In addition to fieldnames corresponding to SBML attributes the structures may contain additional fieldnames to assist in determining whether a value has been set or whether a default value is being used. SBML Levels 1 and 2 have inbuilt default values for some attributes whilst SBML Level 3 does not. SBMLToolbox facilitates the use of all SBML Levels and Versions.

```
>> model.species(1)

ans =

                    typecode: 'SBML_SPECIES'
                      metaid: ''
                       notes: ''
                  annotation: ''
                        name: ''
                          id: 'S1'
                 compartment: 'compartmentOne'
               initialAmount: NaN
        initialConcentration: 0
              substanceUnits: ''
            spatialSizeUnits: ''
        hasOnlySubstanceUnits: 0
           boundaryCondition: 0
                      charge: 0
                    constant: 0
          isSetInitialAmount: 0
   isSetInitialConcentration: 1
                  isSetCharge: 0


>> model.species(1).id

ans =
   'S1'

>> model.species(1).initialConcentration
ans =
   0
```

4.2 Creating SBML   The MATLAB SBML Structures directory of SBMLToolbox contains functions that provide identical functionality to that available in libSBML. Sub elements can be created and attributes values set and queried.

```
% create a model with level 3 and version 1
>> model = Model_create(3,1)

model =

              typecode: 'SBML_MODEL'
                metaid: ''
                 notes: ''
            annotation· ''
            SBML_level: 3
          SBML_version: 1
                  name: ''
                    id: ''
               sboTerm: -1
    functionDefinition: [1x0 struct]
        unitDefinition: [1x0 struct]
           compartment: [1x0 struct]
               species: [1x0 struct]
             parameter: [1x0 struct]
     initialAssignment: [1x0 struct]
                  rule: [1x0 struct]
            constraint: [1x0 struct]
              reaction: [1x0 struct]
                 event: [1x0 struct]
        substanceUnits: ''
             timeUnits: ''
           lengthUnits: ''
             areaUnits: ''
           volumeUnits: ''
           extentUnits: ''
      conversionFactor: ''
           time_symbol: ''
          delay_symbol: ''
       avogadro_symbol: ''
            namespaces: [1x0 struct]


% create a parameter within the model

>> model = Model_createParameter(model)


model =

              typecode: 'SBML_MODEL'
                metaid: ''
                 notes: ''
            annotation: ''
            SBML_level: 3
          SBML_version: 1
                  name: ''
                    id: ''
               sboTerm: -1
```

```
          functionDefinition: [1x0 struct]
              unitDefinition: [1x0 struct]
                 compartment: [1x0 struct]
                     species: [1x0 struct]
                   parameter: [1x1 struct]

                         ...


>> Parameter_isSetId(model.parameter)


ans =

     0


>> Parameter_getUnits(model.parameter)


ans =

     ''


>> Parameter_setId(model.parameter, 'p1')


ans =

        typecode: 'SBML_PARAMETER'
          metaid: ''
           notes: ''
      annotation: ''
         sboTerm: -1
            name: ''
              id: 'p1'
           value: NaN
           units: ''
        constant: 0
     isSetValue: 0
           level: 3
         version: 1


>> Parameter_setValue(model.parameter, 3)

ans =

        typecode: 'SBML_PARAMETER'
          metaid: ''
           notes: ''
      annotation: ''
         sboTerm: -1
            name: ''
              id: 'p1'
           value: 3
           units: ''
        constant: 0
     isSetValue: 1
           level: 3
         version: 1
```

4.3 Import and Export of SBML

Import and export of SBML to and from theMATLAB SBML Structure is achieved using the libSBML binding for either MATLAB or Octave. This binding essentially consists of two functions **TranslateSBML** and **OutputSBML**.

4.3.1 TranslateSBML

This function takes a filename (or where the environment allows will browse for a file if no argument is given) and returns the MATLAB SBML structure representing the model. It will optionally perform a full validation of the model, that is, validation with all available validators enabled as described in Subheading 3.4. Any errors reported can be saved to a separate structure.

```
% include the validate flag which is 0 by default
>> [model, errors] = TranslateSBML('test.xml', 1)

model =

              typecode: 'SBML_MODEL'
                metaid: ''
                 notes: [1x281 char]
            annotation: ''
            SBML_level: 2
          SBML_version: 1
                  name: ''
                    id: 'Branch'
    functionDefinition: [1x0 struct]
        unitDefinition: [1x0 struct]
           compartment: [1x1 struct]
               species: [1x4 struct]
             parameter: [1x0 struct]
                  rule: [1x0 struct]
              reaction: [1x3 struct]
                 event: [1x0 struct]
           time_symbol: ''
          delay_symbol: ''
            namespaces: [1x1 struct]

errors =

1x6 struct array with fields:
    line
    errorId
    severity
    message

% the errors structure can be indexed into to retrieve further information
>> errors(1)

ans =

        line: 34
     errorId: 99505
    severity: 'Warning '
     message: [1x405 char]

>> errors(1).message
```

```
ans =
```

```
In situations when a mathematical expression contains literal numbers or
parameters whose units have not been declared, it is not possible to verify
accurately the consistency of the units in the expression.  The units of
the <kineticLaw> <math> expression 'k1 * X0' cannot be fully checked. Unit
consistency reported as either no errors or further unit errors related to
this object may not be accurate.
```

4.3.2 OutputSBML

The function OutputSBML is the converse of TranslateSBML: it writes the MATLAB SBML structure to an XMLfile. The structure is checked to ensure it has the all the fieldnames it expects to find. Optionally the function can be configured to restrict this check to ensuring ONLY the expected fields are present. This facilitates the use of other fields by a user

```
% start with a structure that has user fields in addition to SBML fields
model =

               typecode: 'SBML_MODEL'
                 metaid: ''
                  notes: [1x281 char]
             annotation: ''
             SBML_level: 2
           SBML_version: 1
                   name: ''
                     id: 'Branch'
     functionDefinition: [1x0 struct]
         unitDefinition: [1x0 struct]
            compartment: [1x1 struct]
                species: [1x4 struct]
              parameter: [1x0 struct]
                   rule: [1x0 struct]
               reaction: [1x3 struct]
                  event: [1x0 struct]
            time_symbol: ''
           delay_symbol: ''
             namespaces: [1x1 struct]
            anotherfield: 'foobar'

# output
# by default extensions are allowed
>> OutputSBML(model, 'out.xml')
>> Document written.

# output with no extensions allowed
>> OutputSBML(model, 'out.xml', 0)
>> Unexpected field encountered.
```

5   Conclusion

The Systems Biology Markup Language has become the de facto standard for exchanging models in the Systems Biology arena. Supporting SBML in software applications is facilitated by both libSBML and SBMLToolbox enabling developers to concentrate on the functionality of their software and avoid the need to consider parsing, creation, manipulation and validation of the SBML language itself. The variety of language bindings available reduces the need for the developer to move away from their preferred programming language and thus, whether it is for a two line script or a full blown software application, the use of SBML should be accessible to a wide variety of users from many backgrounds.

References

1. Kell DB, Mendes P (2008) The markup is the model: reasoning about systems biology models in the Semantic. Web era. J Theor Biol 252:538–543

2. Hedley W, Nelson MR, Bullivant DP, Nielsen PF (2001) A short introduction to CellML. Philos Transact A Math Phys Eng Sci 359 (5):1073–1079

3. Hucka M, Finney A, Sauro HM, Bolouri H, Doyle JC, Kitano H, Arkin AP, Bornstein BJ, Bray D, Cornish-Bowden A, Cuellar AA, Dronov S, Gilles ED, Ginkel M, Gor V, Goryanin II, Hedley WJ, Hodgman TC, Hofmeyr JH, Hunter PJ, Juty NS, Kasberger JL, Kremling A, Kummer U, Le Novère N, Loew LM, Lucio D, Mendes P, Minch E, Mjolsness ED, Nakayama Y, Nelson MR, Nielsen PF, Sakurada T, Schaff JC, Shapiro BE, Shimizu TS, Spence HD, Stelling J, Takahashi K, Tomita M, Wagner J, Wang J, SBML Forum (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 19(4):524–31

4. Bray T, Paoli J, Sperberg-McQueen CM, Maler E (2000) Extensible Markup Language (XML) 1.0 (second edition), W3C recommendation 6-October-2000. Available via the World Wide Web at http://www.w3.org/TR/1998/REC-xml-19980210.

5. Stromback L, Lambrix P (2005) Representations of molecular pathways: an evaluation of SBML, PSI ML and BioPAX. Bioinformatics 21(24):4401–4407

6. Le Novère N, Bornstein B, Broicher A, Courtot M, Donizelli M, Dharuri H, Li L, Sauro H, Schilstra M, Shapiro B, Snoep JL, Hucka M (2006) BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. Nucleic Acids Res 34: D689–D691

7. Goldbeter A (1991) A minimal cascase model for the mitotic oscillator involving cyclin and cdc2 kinase. Proc Natl Acad Sci U S A 88 (20):9107–9111

8. Curto R, Voit EO, Sorribas A, Cascante M (1998) Mathematical models of purine metabolism in man. Math Biosci 151(1):1–49

9. ElowitzMB LS (2000) A synthetic oscillatory network of transcriptional regulators. Nature 403(6767):335–338

10. Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117(5):500–544

11. Izhikevich EM (2004) Simple model of spiking neurons. IEEE Trans Neural Netw 14(6): 1569–1573

12. Tham LS, Wang L, Soo RA, Lee SC, Lee HS, Yong WP, Goh BC, Holford NH (2008) A pharmacodynamic model for the time course of tumor shrinkage by gemcitabine + carboplatin in non-small cell lung cancer patients. Clin Cancer Res 14:4213–4218

13. Munz P, Hudea I, Imad J, Smith RJ (2009) When zombies attack!: Mathematical modelling of an outbreak of zombie infection. In: Tchuenche JM, Chiyaka C (eds) Infectious disease modelling research progress. Nova Science, New York, pp 133–150

14. Hucka M, Bergmann F, Hoops S, Keating SM, Sahle S, Wilkinson DJ (2009) The Systems Biology Markup Language (SBML) language specification for level 3 version 1 core. Available

via the World Wide Web at http://www.sbml.org/Documents/Specifications.

15. Hucka M, Hoops S, Keating SM, Le Nov'ere N, Sahle S, Wilkinson DJ (2008) Systems Biology Markup Language (SBML) level 2: Structures and facilities for model definitions. Available via the World Wide Web at http://www.sbml.org/Documents/Specifications.

16. Hucka M, Finney A, Sauro HM, Bolouri H (2001) Systems Biology Markup Language (SBML) Level 1: Structures and facilities for basic model definitions. Available via the World Wide Web at http://www.sbml.org/Documents/Specifications.

17. Ausbrooks R, Buswell S, Carlisle D, Dalmas S, Devitt S, Diaz A, Froumentin M, Hunter R, Ion P, Kohlhase M,Miner R, Poppelier N,

Smith B, Soiffer N, Sutor R, Watt S (2003) Mathematical Markup Language (MathML) version 2.0 (second edition). Available via the World Wide Web at http://www.w3.org/TR/MathML2/

18. Bornstein BJ, Keating SM, Jouraku A, Hucka M (2008) LibSBML: an API library for SBML. Bioinformatics 24(6):880–881

19. Dräger A, Rodriguez N, Dumousseau M, Dörr A, Wrzodek C, Le Novère N, Zell A, Hucka M (2011) JSBML: a flexible Java library for working with SBML. Bioinformatics 27 (15):2167–2168

20. Keating SM, Bornstein BJ, Finney A, Hucka M (2006) SBMLToolbox: an SBML toolbox for MATLAB users. Bioinformatics 22(10): 1275–1277