Properties of SBML documents and tools – capabilities

Jörg R. Weimar
Institute of Scientific Computing
Technical University Braunschweig
D-38092 Braunschweig, Germany
j.weimar@jweimar.de

May 26, 2003

1 Introduction

This is a proposal for a recommendation on the usage of a controlled vocabulary for the desctiption of tools and models using the systems biology markup language (SBML).

Different tools for handling SBML files have different capabilities of which constructs in SBML they support. A controlled way of specifying these capabilities is needed to find out if a tool can handle a certain model or class of models. Conversely, models could specify which capabilities they need. Apart from capabilities, specifications on the assumptions that a tool makes are also necessary.

A set of filters could be developed to overcome some limitations of certain tools. Such a filter would need to know the capabilities of the target tool.

An automatic tool could be written to find out which SBML features are used in a model. This tool could also try to make some inferences about the assumptions, but cannot necessarily resolve all assumptions.

2 List of features

A preliminary list of features of SBML that tools might or might not support follows:

2.1 SBML level 1

- Level 1 Version 1
- Level 1 Version 2
- Level 1 Version 3
- rules
- scalar AssigmentRules
- rate AssigmentRules

- AlgebraicRule
- KineticRateLaw
- Formulas
- only Mass Action
- predefined math functions
- predefined rate law functions
- local Parameters
- global Parameters
- Units
- Compartment volume not 1

2.2 SBML level 2

Additional features in SBML level 2:

- Level 2
- Level 2 Version 1
- Level 2 Version 2
- Level 2 Version 3
- Level 2 minus MathML
- logical operators in MathML
- function definition
- ModifierSpeciesReference
- Delays
- Events
- Names (in addition to IDs)
- initial concentration

2.3 SBML level 3

Additional features in SBML level 3:

- submodels
- Instances
- Arrays
- Sparse arrays
- dynamic arrays
- Parameter sets
- initialAssigmentRule
- Ports
- Links to Ports only

2.4 Notes and annotations

- XHTML notes
- Graph layout annotation
- ....

The above list of features is necessarily incomplete. Some features could be structured hierarchically, some are mutually exclusive, and especially regarding notes and annotations, extension to other features will be necessary.

3 Properties of tools regarding the features

Lokking at the list of features, we can try to find how those words could be used in describing a tool or a model.

3.1 Models

For models, one could state that

This model uses **feature**.

or

This model does not use **feature**.

and

This model is created with assumption **feature**.

3.2 Tools

The most important properties for tools are whether a tool can read SBML files and whether the models written by the tool contain (or can contain) a feature. Example sentences would be

This tool can correctly read and process models that use **feature**.

This tool cannot read models that use **feature**.

This tool creates models that may contain **feature**.

The models that this tool creates never contain **feature**.

This tool can optionally create all models without using **feature**.

This tool can convert models with **feature** into models without **feature**.

4 Formalizing

It is desirable to create a vocabulary in which features and properties are orthogonal, in that each feature can be combined with each property and the statement would make sense.

An example list of properties would be: ”Never” ”Read only”, ”Read and Write”, ”Used”, ”Not used”.

A capability of a tool is then a pair of feature and property. This could be formalized as am xml schema, such that the capabilities of a model could be described by the following XML fragment inside an annotation node of the SBML file:

```
<annotation xmlns:cap="http://www.sbml.org/2002/ns/capabilities/>
  </cap:capability cap:feature="Delays" cap:property="Not used">
  </cap:capability cap:feature="Events" cap:property="Used">
</annotation>
```

4.1 RDF

The information about models is a typical example of metadata. Therefore one should investigate whether the Resource Description Format (RDF) can approriately be used for formalizing the capabilities.

4.2 Modularization

Some of the issues in this proposal could be resolved by a systematic modularization of SBML anolng the lines of XHMTL modularization (see http://www.w3.org/TR/xhtml2/).

This modularization could first define a core module, and then add elements or attributes to the core to extend the functionality. It is not clear that all possible combinations of properties and features could be specified in this way, but I think most can.

A separate issue is how a model would state which modules it uses, and similarly for a tool to state which modules it can handle.

4.3 Modularization of SBML

As an example, consider the following modularization of SBML level 1:

SBML L1V2 Base module Models using only this base have the following properties:

No compartments (1 compartment with unit volume) No units (all units in litre, mole, second, and combinations thereof or otherwise consistent) No parameters No rules No rate laws! Integer stoichiometry

Models which only need the base module could be used by tools that do stoichiometric analyses, since they don't need any other information.

Note that I would have liked to leave out the initialAmount, but then the result is not SBML L1 conformant.

```
SBML Base module

SBase
--------------
(no attributes)

Model extends SBase
-------------------
name : SName {use="optional"}
species: Species [0..*]
reaction: Reaction [0..*]

Species extends SBase
---------------------
name: SName
initialAmount: double

Reaction extends SBase
----------------------
name: SName
reactant: SpecieReference [0..*]
product: SpecieReference [0..*]

SpeciesReference extends SBase
-----------------------------
species: SName
stoichiometry: integer {use="default" value="1"}
```

An extension to use compartments would add the following, which make the compartment not optional, but required in the species.

```
SBML Compartment module extends Base module

Compartment extends SBase
-------------------------
```

```
Model extends SBase
-------------------
compartment: Compartment [1..*]

Compartment extends SBase
-------------------------
name: SName
volume: double {use=default value = "1"}
outside: SName {use="optional"}

Species extends SBase
---------------------
compartment: SName
```

A separate extension could be specified to allow notes:

```
SBML Notes module extends Base module

SBase
---------------
notes: (XHTML)
```

Another extension is to allow kinetic rate laws (but no parameters yet). This is formulated here so that if the module is used, then all reactions must have a kineticLaw. It is also possible that partial models are useful (I know about work on hybrid static/dynamic models).

```
SBML RateLaw module extends Base module

Reaction extends SBase
----------------------
kineticLaw: KineticLaw

KineticLaw extends SBase
------------------------
formula: string
```

```
SBML GlobalParameters module extends Base module

Model extends SBase
------------------------
parameter: Parameter [0..*]

Parameter extends SBase
-----------------------
name : SName
value: double {use="optional"}
```

All modules would need more description about the semantics. Some modules would consist entirely of secriptions, such as the module for predefined mathematical functions, or the module for predefined rate law functions.

A module for units would add the units attribute at many elements, even ones that are not present on the other modules used. For example a model that needs units and rateLaws, but still assumes a standard compartment of volume 1 would need the Base module, the RateLaw module, and the Units module, but not the Compartment module. Still, the Units module would specify the units attribute for a compartment. One would have to state that extensions to elements that do not exist are to be ignored.

TO BE CONTINUED