The Systems Biology Markup Language in a glimpse

Riku Louhimo

Helsinki April 29, 2007

Seminar on Computational Systems Biology

UNIVERSITY OF HELSINKI
Department of Computer Science

1 Introduction

The focus in contemporary biological research is on interactions between biological components. This new approach has been named systems biology. The research done in this field and the vast amount of data it produces has led to a general entropy of data modelling methods, which in turn introduces incompatibilities between different datasources. Developed models in old systems generally become inusable after system support has stopped. Currently system-to-system and model-to-model importing is error prone because time-consuming programming tasks are required when comparing existing models from different sources.

The *Systems Biology Markup Language (SBML)* is "a computer-readable format for representing models of biochemical reaction networks" [FHLN06]. It offers a solution to data integration and general compatibility issues between varied sources. It is an XML-based markup language. The motivations for using XML are its portability and de facto usage as a bioinformatics lingua franca [HFS+03].

This report has the following structure. First, XML and its syntax are introduced in section 1. Then, in section 2, a description of SBML and its uses is given. Finally, section 3 offers a short discussion and a recap of the topic.

2 XML

The *eXtensible Markup Language (XML)* is a general purpose markup language derived from the *Standard Generalized Markup Language (SGML)*. One other such derivative is HTML. By definition XML is an agreed-upon textual format for representing tree-structured data [W3C06]. The goal and motivation behind developing XML was that SGML is very extensive and hence not very attractive for e.g. web-publishing. However, a uniform way to represent structured data is nonetheless required.

Developers of XML aimed at constucting a simplified version of SGML, while retaining its best characteristics - mainly the demands for well-formedness or validity. A well-formed document obeys the syntax of XML, while a valid document conforms to the logical structure defined in a separete schema, such as a DTD.

2.1 Basics of XML

XML documents consist of the document and its *document type definition (DTD)*. The DTD is used to validate the XML-document. It is a document spesific grammar. Alternatively some other schema language might be used. Technically, DTD is a simple and rigid implementation of the schema language standard in XML. Another popular one is the W3C XML Schema language (XSD), which sports greater specificity, is namespace aware and supports types (such as boolean, integer etc.). It has more expression power and is more programmable. As a consequence, it is harder to learn and less used.

A typical XML file has a tree-like structure of elements identified by tags, such as those in HTML. Unlike its simpler cousin, XML does not allow sloppiness. This is to say that e.g. end-tags are compulsory and that there is zero tolerance for interlacing elements. Ambigious element names are disallowed and the names should be self-describing per normal programming conventions. Here is some hypothetical code.

```
<?xml version="1.0" encoding="iso-8859-1">
<!DOCTYPE example SYSTEM "model.dtd">
<experiment>
  <specimen>Escherichia Coli</specimen>
  <pathway>
     <cycle name="TCA" />
  </pathway>
</experiment>
```

Only one root element (`experiment` in the above example) is allowed, which is due to the tree-structure. Names of elements are user-defined but must conform to the DTD or schema. Elementwise attribute values are given inside the start-tag. XML allows a lot of freedom in data representation: depending on the use, the information can be contained in the attributes with no free text allowed or alternatively XML only provides a framework to which the user may "cast" his own content.

For example, if we wanted the above code to be a wordbook type description, then the name of the specimen ("E. Coli") would be better as an attribute with the general description written between the tags.

2.2 Usage in bioinformatics

The use of XML in bioinformatics, as put forward by [AVB01], was introduced to aid interoperability. This is greatly enhanced with SBML, where the system modelled is a biochemical network or pathway. Normal bioinformatical data is complex and vast. Also, the amount of available data grows rapidly and complexity increases with the shift into a systems biological point of view. However, one of the biggest problems is that current analysis not only produces more data, but also changes our perception of old data. These put a huge strain on legacy data management (i.e. updating and reuse of models).

Benefits of XML include its high flexibility. Once a DTD is written, it can also be easily modified or written anew. Neither XML nor DTD require an interpreter (that is, they are human-readeble) and thus allow later modifications in a simple but efficient manner. Being made for the Internet gives XML an edge over its competitors. Especially cross-references between databases are manageable [AVB01]. Its commoness also guarentees that most people wanting to use it are already at least relatively familiar with it.

On the downside, XML has no build-in inheritance and no method to update documents dynamically. This can however be achieved with external programs. XML does not (as such) support numerical values, tables or matrices, a major drawback from a systems biology point of view.

3 SBML

Although XML is versatile and easy to use in bioinformatics in general, the inherit complexity of biochemical network models creates a need for a language of its own. The *Systems biology markup language (SBML)* was coined to deal with the same problems that XML tackles in the general bioinformatical case. Similar XML-based languages, such as CellML, PSI MI and BioPAX have also been developed but are not covered here, even though compatibility between CellML and SBML is a goal in development.

SBML is defined in iterations known as levels, which are again broken down to versions. The atomicity of the levels makes compatibility issues less likely [HFS+03], as the schema for each level is kept in storage. Every new level addresses some problem reported by the community. For example as of the current level (level 2), SBML has

been using MathML to model the complex mathematical formulae required. This was previously done with ASCII text, which was the standard in databases at that time. MathML is *"an XML application for describing mathematical notation and capturing both its structure and content"* [W3C03]. Details of that language are not discussed here.

Different modelling tasks that can be tackled with SBML include cell signalling, metabolism and gene regulation. The ultimate goal of SBML is to unify the presentation of systems biology models in different databases. Currently, the amount of time used by researches simply to get data from two different sources and then unifying them, is huge.

Note that SBML is not meant for biologists (or even bioinformaticians!) to write by hand. It is meant as an underlying interface to software packages and it simply guarentees, that different models are universally usable, despite one's software preference.

3.1 Basics

SBML is an XML-based markup language so its syntax is that of XML. Every model definition (i.e. document in XML jargon) starts with the XML declaration and the location of the XMLSchema that all SBML documents must conform to as it defines the language. The root element is always `sbml`. Only ASCII characters are allowed in attributes. All elements reside in the same namepace. Therefore no two attributes, compartments or identifiers can share a name. This does not affect reaction components nor unit identifiers (if the model has unit definitions), since they form their own private local namespace [FHLN06].

A model is split into different elements called lists. These lists define the model. Types of lists include compartments, species, reactions, parameters, unit definitions and rules. Level 2 adds lists for functions, events and constraints. Normal XML-data can be added anywhere with a specific `<?xml>` declaration and MathML with a `math` element containing the MathML schema path and the formulation.

SBML has built in datatypes absent from regular XML. These include, but are not limited to, integer, double, boolean and so on (for a complete list, see the SBML specification)[FHLN06]. Arrays of objects, say, from an indexed collection of data, are not yet supported.

```
<?xml version="1.0" encoding="iso-8859-1">
```

```
<sbml xmlns="http://www.sbml.org/sbml/level2"
      level="2" version="2">
<model name="my_model">
  <listOfCompartments />
  <listOfSpecies />
  <listOfReactions>
      <listOfReactants />
      <listOfProducts />
  </listOfReactions>
  ...
</model>
</sbml>
```

A dummy example of raw SBML code is given above. The syntax is explained in the next section, but we can already see that SBML does not differ much from regular XML.

3.2 Components

Elements in SBML are called components. Each component contains model data in attributes. No free text is allowed to be used. In a sense, an SBML document contains two roots, since `model` is the top-most structure after `sbml`, and there can only be one `model` definition per document. In fact, all the other elements are optional.

The highest level component is called a *compartment*. It represents the boundaries in which the species – SBML jargon for chemical molecules – are contained and where the reactions take place. In Figure 1 we see two adjacent compartaments with a third within another. The `listOfCompartments` element also contains the information whether the element is within another compartment. The name attribute is mandatory for unique reference. All compartments can have a `CompartmentType` association in order for the user to be able to group several compartments. The current level offers no functionality yet beyond a unique identity.

The *species* component contains a unique identifier (name of molecule or metabolite), the initial amount of the molecule and a compartment pointer. The `listOfSpecies` element contains all the chemical substances within a specific compartment. As with compartments, species can also be grouped together in a `SpeciesType` element. This enables the user to differentiate between enzymes and cofactors for example.

Reactions are broken down into `listOfReactions` elements with a subelement for

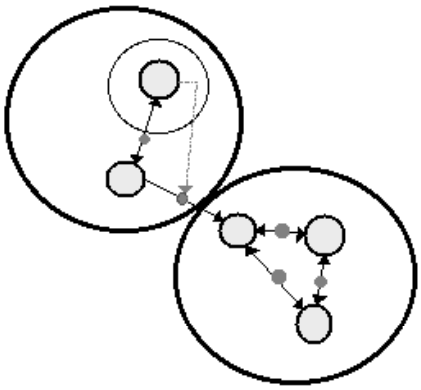

Figure 1: White circles are compartments, light grey are species and the lines represent reactions.

every reactant and product for a given reaction. Every reactant and product must reference a species by its identifier. Each reaction has a stoichiometry and optionally a rate equation associated with it. These equations are expressed using MathML [HFB+04]. Every reaction can be indicated to be reversible or irreversible with a simple boolean attribute.

```
<listOfReactions>
  <reaction name="R1" reversible="false">
    <listOfReactants>
       <speciesReference species="CoA" />
          <stoichiometryMath>
            <math>
            ...
            </math>
          </stoichiometryMath>
    </listOfReactants>
    <listOfProducts>
       <speciesReference species="Fumarate" />
    </listOfProducts>
    <kineticLaw>
       <math xmlns="http://www.w3.org/1998/Math/MathML">
       ...
       </math>
    </kineticLaw>
```

```
  </reaction>
  ...
</listOfReactions>
```

The *parameter* component assigns a fixed floating point value to a parameter. Thus a parameter can replace the exact value in formulae elsewhere in the document. Parameters can also be made global to the model by declaring them outside of the `listOfParameters` element, normally at the beginning of the model.

```
<listOfParameters>
  <parameter name="Pi" value="3.14159" />
</listOfParameters>
```

It is possible for the user to add new units through a `listOfUnitDefinitions` element. All of these units must however be derived from the base units that SBML offers. These standard units are in the SI system and include the likes of liter, joule and so on. A complete list can be obtained from the website. The user is allowed to override these explicitly by using a `Unit` element that specifies the difference from the base units in SBML. One application of this would be the scaling of a unit, say, to the power of -1. Unit definitions can reside almost everywhere in the model structure but the recommended place is on the top of the model definitions.

All use of MathML can be made easier by defining a function. A mathematical formula defined in the `FunctionDefinition` component can be referenced from anywhere in the model. The idea is more or less similar to writing a method that has to be used repeatedly here and there in software code.

Rules declared in a `listOfRules` element are mathematical expressions written with MathML. Their main uses are creating constraints between quantative variables and to set parameter values using equations. Rules can be either assingments (variable value setting), rates of change or algebraic, if the first two are not applicable. Rules should only be used, if the constraints cannot be expressed within the *reactions* component or by simple value definitions.

Events carry out state changes in the model happening in discrete time steps but in a discontinuous way. One example would be the halving of one metabolite quantity when another exceeds a predefined threshold. Effects on the model by events can also be delayed. An `Event` element has three compulsory parts: an id, a trigger in MathML and an `EventAssignment` that executes the actual changes to the model attributes. A piece of example code with some MathML is given in Figure 2.

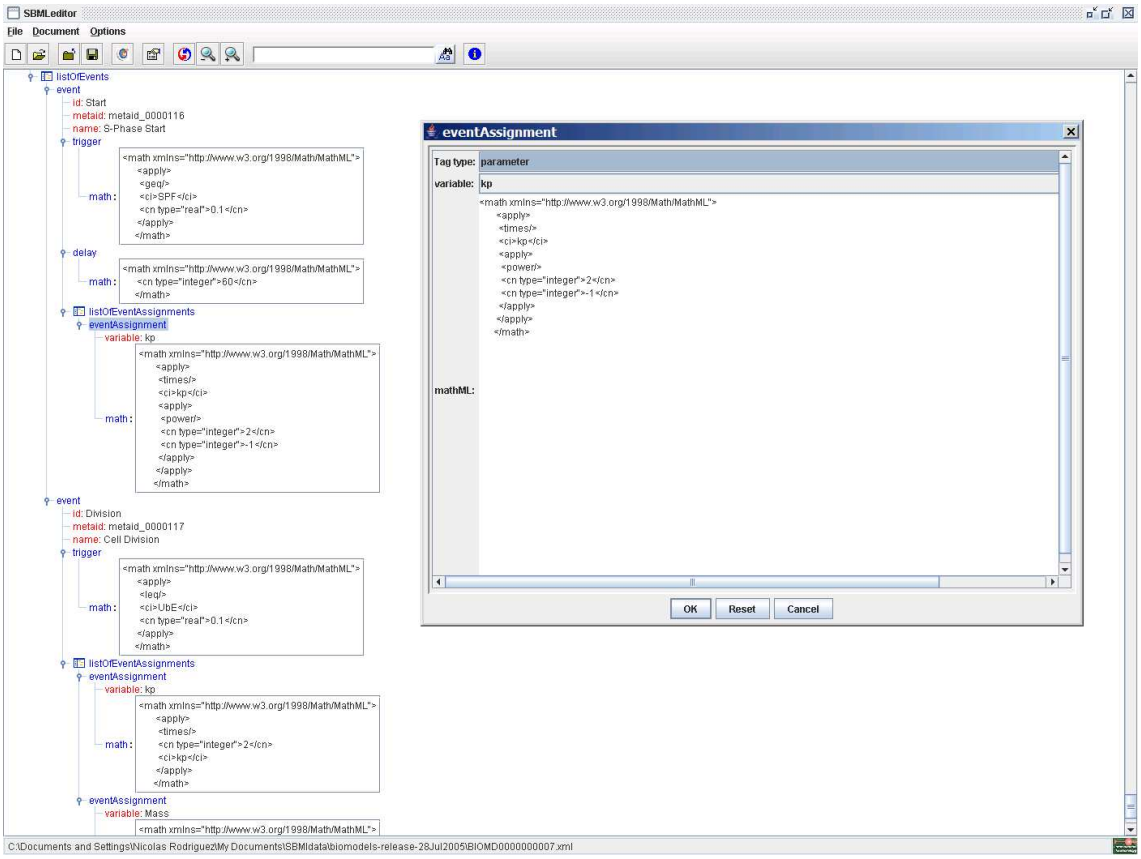

Figure 2: Event SBML code created with the SBMLEditor *http://www.ebi.ac.uk/compneur-srv/SBMLeditor.html*

Initial values can be set in two ways in SBML. The first way is directly to each component in the model. This is not always sufficient, however. Say we need to calculate the starting values of metabolite A from the concentrations of metabolites B and C. Simple value assignment fails to automate this type of assignment; hence, we use the *initial assignment* component. Values within the element are written in MathML expressions and can range from simple (i.e. a = "1") to complex calculations. All initial assignments only affect the values before the starting time point in the model.

Mathematical constraints in MathML on the model values at every relevant point in time are modelled with a *constraints* component. Constraints should never be used to model any dynamic behaviour in the model as this is already done with rules. A proper use is modelling linear programming constraints in metabolic flux balance analysis.

3.3 Using the SBML

SBML should always be used through software since it was not intended to be written by hand. There already exists a plethora of different analysis programmes capable of interpreting models defined with the language and the number keeps growing. This is due to the fact that SBML is open-source and the development group actively participates in creating new software. A list of available software can be obtained from the project website at *http://www.sbml.org*. Less than two years ago the language was supported (used or converted to) by 75 software systems (up from 60 in 2004) and the number has now grown by almost 50% to more than 110 today [SL05] [KHK+05].

The similarity of rules, reactions and events makes it hard to quickly grasp, when to use which. Here is an example that should help understand the difference between a rate rule and an event. Suppose we have an in-flux of some metabolite A in our metabolic model happening every other second. This we would model with a rate. On the other hand, after the amount of A exceeds some threshold, we want to limit its intake into the cell in order to model inhibition by over-production. For this we use an event.

Mathematical modelling in SBML has evolved in stretches. Using the MathML enables complex expressions to be used in a standard way. Currently the rule, event, function and reaction elements can all include mathematical notations and level 2 version 2 introduces an additional element – constraint. All in all while vastly improved from level 1, the different mathematical components could probably be structured in a more concise and intuitive way. On the other hand, the wide variety of ways to express something does not restrict the user when constructing a model, though this hightens the risk of SBML resolving into "XML with some biology" and hence the much needed unified modelling language for systems biology going down the drain.

3.4 Beyond SBML

While SBML and other similar tools have developed rapidly and offer huge benefits in machine-readable network data representation, one overlooked group of stakeholders are humans. Graphical representation of biological network models needs uniformity as much as machines do. A vaguely UML like standard named Systems Biology Graphical Notation *(http://www.sbgn.org)* aims to codify human-readable network

models [KFMO05].

The project is still in beta phase, as the first official version (1.0) is almost ready for release. The project codifies symbolic notation for molecules as nodes and differents types of reactions as edges in diagrams. Also a type of logical AND/OR port is introduced. The general idea is that biologists need to become as acquinted with these diagrams as engineers have become to electric circuit diagrams and that having one uniform notation makes the reading and understanding of new network diagrams a fairly trivial task.

At the heart of the SBGN project lie two pieces: a piece of software called CellDesigner *(http://celldesigner.org)* and SBML. Networks created with CellDesigner are automatically constructed as SBML and all diagrams created with the program are saved with .xml extension (which is the SBML standard). All kinetics put in to the model are automatically translated in to MathML. Example diagrams using SBGN notation are shown in Figure 3.

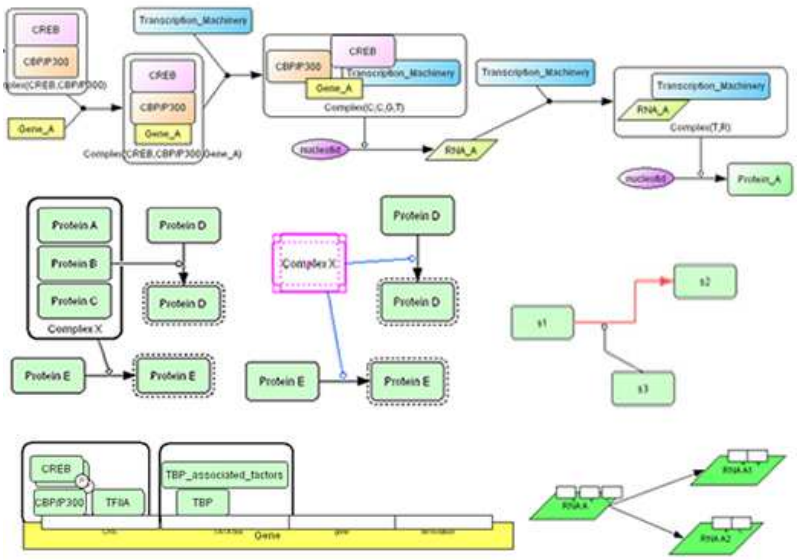

Figure 3: Diagrams created with CellDesigner from *http://celldesigner.org.*

4 Conclusions

One of the biggest contemporary problems in systems biology research is the non-conformity of different model data sources. This leads to poor efficiency, since inte-

grating models when conducting research takes up huge amounts of time [HFB+04]. The obvious remedy is uniforming the way models are defined. Standard definitions in turn simplify coding of software packages and should ideally make importing of models – not only from database-to-database, but also software-to-software – trivial.

This report introduced the systems biology markup language, SBML, but by no means is it the only one of its ilk. However, the number of applications using it is growing fast and the developers' goal of a de facto standard in system biological modelling appears to be a possibility. Other prominent standards include basic XML, CellML, PSI MI and BioPAX, which all have their respective advantages. A recent study found SBML especially good for pathway information representation, while falling short in protein structure representation. It is on the level 3 list of possible additions, though [SL05].

Since SBML evolves quite fast, it is evident that software developers – mainly form the academic world – have a hard time keeping up. While some software use level 2 version 1, some are stuck at level 1. This is a big hinderance, since many of the essential elements of SBML (such as the use of MathML) have only been introduced in recent versions.

The development of SBML is already in its third iteration and therefore many of the most pressing needs of the community have been met. Further levels, as they are called in SBML jargon, will introduce more advanced but less general features. Some commercial tools incorporating SBML already exist, but nonetheless the biggest challenge now is getting users and developers to adopt the standard.

References

AVB01 Achard, F., Vaysseix, G. and Barillot, E., XML, bioinformatics and data integration. *Bioinformatics review*, 17,2(2001), pages 115–125.

FHLN06 Finney, A., Hucka, M. and Le Novère, N., Systems biology markup language (sbml) level 2: Structures and facilities for model definitions, Available via the World Wide Web at http://sbml.org/documents/, 2006.

HFB+04 Hucka, M., Finney, A., Bornstein, B. J., Keating, S. M. H., Doyle, J. C., Kitano, H., Shapiro, B. E., Matthews, J., Kovitz, B. L., Schilstra, M. J. and Funahashi, A., Evolving a lingua franca and associated software infrastructure for computational systems biology: The systems biology markup language (SBML) project. *Systems biology*, 1,1(2004), pages 41–53.

HFS+03 Hucka, M., Finney, A., Sauro, H. M., Bolouri, H., Doyle, J. C. and Kitano, H., The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19,4(2003), pages 524–531.

KFMO05 Kitano, H., Funahashi, A., Matsuoka, Y. and Oda, K., Using process diagrams for the graphical representation of biological pathways. *Nature Biotechnolgy*, 23,8(2005), pages 961–966.

KHK+05 Klipp, E., Herwig, R., Kowald, A., Wierlig, C. and Lehrach, H., *Systems biology in practice*. WILEY-VCH Verlag GmbH & Co., 2005.

SL05 Strömbäck, L. and Lambrix, P., Representation of molecular pathways: an evaluation of SBML, PSI MI and BioPAX. *Bioinformatics*, 21,24(2005), pages 4401–4407.

W3C03 W3C, The MathML 2.0 specification, Available via the World Wide Web at http://www.w3.org/TR/MathML/, 2003.

W3C06 W3C, The XML 1.0 specification, Available via the World Wide Web at http://www.w3.org/TR/REC-xml, 2006.