Volume Title
*ASP Conference Series, Vol. **Volume Number***
Author
*© **Copyright Year** Astronomical Society of the Pacific*

Organizing community-based data standards: lessons from developing a successful open standard in systems biology

Michael Hucka

Department of Computing and Mathematical Sciences, California Institute of Technology, Pasadena, CA 91125, USA

Abstract. In common with many fields, including astronomy, a vast number of software tools for computational modeling and simulation are available today in systems biology. This wealth of resources is a boon to researchers, but it also presents interoperability problems. Despite working with different software tools, researchers want to disseminate their work widely as well as reuse and extend the models of other researchers. This situation led in the year 2000 to an effort to create a tool-independent, machine-readable format for representing models: SBML, the Systems Biology Markup Language. SBML has since become the de facto standard for its purpose. Its success and general approach has inspired and influenced other community-based standardization efforts in systems biology.

Open standards are essential for the progress of science in all fields, but it is often difficult for academic researchers to organize successful community-oriented standards. I draw on personal experiences from the development of SBML and summarize some of the lessons learned, in the hope that this may be useful to other groups who seek to develop open standards in a community-oriented fashion.

1. Introduction

Interpreting the staggering amount of biological data produced in recent years is a daunting challenge. To study biological entities and the properties that arise from their interactions in the context of an overall system, systems biologists build and test formal models of cellular components and processes. The results from these activities are used to refine the models and suggest the next course of action. (See Figure 1).

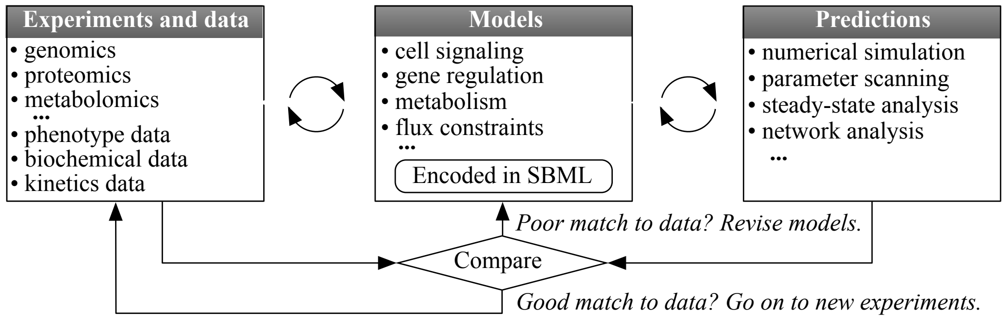

Figure 1. Where SBML is situated in the biological modeling enterprise.

To be useful embodiments of our understanding of biological systems, models must be put into a consistent and widely-supported format that can be communicated directly between software tools. The Systems Biology Markup Language (SBML) serves this purpose; it is an open, machine-readable, XML-based format for representing computational models (Hucka et al. 2003). By supporting SBML as an input/output format, different tools can all operate on the same representation of a model, removing opportunities for errors and assuring a common starting point for analyses and simulations.

Standards are essential for the progress of science, but taking a proposed standard from inception to adoption is not straightforward. SBML has been the most successful format in its domain, a de facto standard supported by over 260 software tools (SBML Team 2014), used in prominent multi-group modeling efforts, supported by the most significant public databases today (Le Novère et al. 2006), and more. This is uncommon success for an academic effort. This paper provides a summary some of the lessons and ideas from the SBML experience, in the hope that it can benefit others who also want to engage in the development of standards.

2. Positive lessons

The following are eleven positive lessons (based on SBML's success) about what were good decisions and actions, especially in SBML's formative years. Most, if not all, are general enough that they should be transferrable to other efforts.

1. Address a significant problem faced by many people. It is easy to find situations where a lack of standards or infrastructure creates hurdles or friction. Motivated individuals often respond with "this can be done better!" and embark on developing a solution. However, not all problems affect other people in the same way—*you* may think you have a great solution, but others may not feel the problem is significant enough. An essential ingredient for a successful standardization effort is addressing a problem that many other groups face. SBML was originally a by-product of an effort to develop a software interoperability framework: the framework needed a data exchange format. It was not until later that it became clear SBML addressed a real need for many people, and that it should be the focus of a standardization effort in its own right.

2. Time it well. A standardization effort needs to come at a time when people are receptive to a new proposal. If it comes too early, there will not be enough interested people to sustain the effort. If it comes too late, another standard or approach will have been adopted by then and people and institutions will be too entrenched in the existing approach. Dislodging incumbent standards is difficult. SBML was introduced at just the right time, when many people became interested in developing new software tools.

3. Gather a small subset of actual stakeholders. By bringing together a small number of members of your target audience, representing a reasonable gamut of use-cases, the design of a standard can be focused and proceed faster. At this stage, it is better to limit involvement to a small number of groups (say, 8–10); too many leads to an attempt to satisfy too many requirements at once. Design by a small group of domain experts stands a better chance of succeeding in creating a coherent workable solution. This runs against the goal of eventually having a democratic, open process, but it is a temporary bootstrapping phase.

4. Aim to support what people are actually doing. People trained as computer scientists (e.g., the author) have a tendency to design elegant, clever solutions—which then never get used in practice because they are too far removed from what users work with on a day-to-day basis. It is important to focus instead on what people are *actually* using and doing. The resulting standard is frequently messy and inelegant, but worth it if it matches more closely what people use or do in practice because it will require less effort for people to adopt. (Many data standards are hidden "under the hood" behind software interfaces anyway—elegance often matters little in the end.)

5. Have a dedicated, small development team. A lone developer or a group of part-time developers can of course develop standards too, but progress is faster and the quality is often higher when a dedicated full-time team undertakes the effort. Developing a standard requires more than simply designing the standard itself: it requires writing careful and clear specification documents, creating online resources, communication, creating software frameworks, and much more.

6. Use a staged development approach. Attempting to support the kitchen sink in a standard right from the start is a recipe for failure, but so is producing an incomplete standard. Instead, borrow a tactic from product development: create the equivalent of a "minimal viable product" at each stage. A simple standard that is fully usable for a subset of possible use-cases will allow people to start using it in practice, which in turn produces feedback for the next stage and simultaneous begins creating a user base.

7. Create open source libraries. For information standards of any meaningful complexity, it is easier for software developers to start with an API library rather than create their own. Providing good-quality libraries promotes faster adoption of a standard. Open-source software is attractive because it gives users a greater sense of security: even if the original project loses funding, they won't be left stranded without access to the software infrastructure they have come to rely on.

8. Get adoption by respected efforts in the field. Sometimes you build it, but nobody comes. To jump-start a network effect, you need to gain a few significant, highly visible adopters. Their mere use of a standard is likely to be noticed by others, which will increase adoption, which in turn will promote still more users to adopt the standard. (Ideally, these groups are part of the initial small subset of lesson #3.)

9. Switch gears when the standard gains traction. Once adoption increases, it is no longer in the best interests of the user community to have the standard be controlled by a group of self-appointed authorities. This is the time to introduce a democratic process, with an editorial board elected from the community, a system for proposing changes to the standard, and other community-oriented processes. This helps spread the intellectual work load in continuing to evolve the standard, and it lets the community take ownership, which encourages increased involvement and adoption.

10. Have a shepherd. Volunteer contributions are essential for a community-oriented process, but a successful long-running project often needs more. Many standardization projects are successful due in part to having someone who devotes a majority of their time to the effort as a whole: mediating disputes in the community, leading a development team, seeking funding, etc. Unfortunately, that time is spent on activities that do not produce publications—the primary metric for academic success. Consequently, these are not tasks suitable for postdocs or faculty. They require paid staff.

11. Be creative about seeking funding. Funding standardization efforts is often difficult. Before a standard is established, a grant proposal to develop the new standard is likely to score poorly because reviewers naturally tend to err on a conservative side (e.g., questioning why existing standards are inadequate, or arguing that the proposed approach is unproven). Finding nontraditional sources of funding may be the only way to support the necessary development for a successful effort.

3. Negative lessons

What follows is a list of three negative lessons: wrong decisions and actions.

1. Not waiting for implementations before freezing specifications. Early specifications were often finalized prospectively, based on general community agreement and before any software had tested the features. Some features were later discovered to be poorly thought-out. Waiting for software implementations to debug a specification before finalizing it is an important development principle.

2. Not producing simple enough designs. The goal of producing a specification that meets actual, practical needs sometimes conflicts with the goal of producing simple designs. Some features in SBML are complex, and should have been made simpler.

3. Not formalizing the process sufficiently. Before the creation and implementation of SBML's formal development process, decisions were in the hands of the original development and later, the SBML Editors. Without explicit procedures for change proposals, voting on issues, and so on, decisions can seem opaque or arbitrary to the rest of the community. An explicit process helps.

4. Conclusions

Many efforts to develop standards for many fields have been undertaken by academics in the past, and more will undoubtedly be created in the future. Here I summarized some of lessons learned from my participation in the development of SBML as well as several other successful efforts in systems biology. It is my hope that this can help other budding standardization efforts, including some in astronomy.

Acknowledgments. The original SBML development team consisted of Hamid Bolouri, Andrew M. Finney, Herbert M. Sauro and the author; John C. Doyle (Caltech) and Hiroaki Kitano (Systems Biology Institute, Japan) were the principal investigators. Development of SBML has been continued by many people, notably the SBML Editors: Frank Bergmann, Stefan Hoops, Nicolas Le Novère, Chris J. Myers, Brett G. Olivier, James C. Schaff, Sven Sahle, Lucian Smith, Dagmar Waltemath, and Darren J. Wilkinson. Core SBML development is currently funded by NIH grant R01GM070923.

References

Hucka, M., et al. 2003, Bioinformatics, 19, 524
Le Novère, N., et al. 2006, Nucleic Acids Research, 34, D689
SBML Team 2014, The SBML Software Guide, `http://sbml.org/SBML_Software_Guide`