To appear in

Foundations of Systems Biology
ed. Hiroaki Kitano
MIT Press, 2001

The ERATO Systems Biology Workbench: An Integrated Environment for Multiscale and Multitheoretic Simulations in Systems Biology

Michael Hucka [a,b] (MHucka@cds.caltech.edu)
Andrew Finney [a,b] (AFinney@cds.caltech.edu)
Herbert Sauro [a,b] (HSauro@cds.caltech.edu)
Hamid Bolouri [a,b,c,d] (HBolouri@caltech.edu)
John Doyle [a,b] (Doyle@cds.caltech.edu)
Hiroaki Kitano [a,b,e] (Kitano@symbio.jst.go.jp)

[a] ERATO Kitano Symbiotic Systems Project, M-31 Suite 6A 6-31-15 Jingumae Shibuya-ku, Tokyo 150-0001, Japan

[b] Control and Dynamical Systems 107-81, California Institute of Technology, CA 91125, USA

[c] Science and Technology Research Centre, University of Hertfordshire, AL10 9AB, UK

[d] Division of Biology 216-76, California Institute of Technology, CA91125, USA

[e] Sony Computer Science Laboratories, 3-14-13 Higashi-gotanda Shinagawa-ku, Tokyo 141-0022, Japan

6 The ERATO Systems Biology Workbench: An Integrated Environment for Multiscale and Multitheoretic Simulations in Systems Biology

Michael Hucka, Andrew Finney, Herbert Sauro,
Hamid Bolouri, John Doyle, and Hiroaki Kitano

Over the years, a variety of biochemical network modeling packages have been developed and used by researchers in biology. No single package currently answers all the needs of the biology community; nor is one likely to do so in the near future, because the range of tools needed is vast and new techniques are emerging too rapidly. It seems unavoidable that, for the foreseeable future, systems biology researchers are likely to continue using multiple packages to carry out their work.

In this chapter, we describe the ERATO *Systems Biology Workbench* (SBW) and the *Systems Biology Markup Language* (SBML), two related efforts directed at the problems of software package interoperability. The goal of the SBW project is to create an integrated, easy-to-use software environment that enables sharing of models and resources between simulation and analysis tools for systems biology. SBW uses a modular, plug-in architecture that permits easy introduction of new components. SBML is a proposed standard XML-based language for representing models communicated between software packages; it is used as the format of models communicated between components in SBW.

INTRODUCTION

The goal of the ERATO *Systems Biology Workbench* (SBW) project is to create an integrated, easy-to-use software environment that enables sharing of models and resources between simulation and analysis tools for systems biology. Our initial focus is on achieving interoperability between seven leading simulations tools: *BioSpice* (Arkin, 2001), *DBSolve* (Goryanin, 2001; Goryanin et al., 1999), *E-Cell* (Tomita et al., 1999, 2001), *Gepasi* (Mendes, 1997, 2001), *Jarnac* (Sauro, 1991; Sauro and Fell, 2000), *StochSim* (Bray et al., 2001; Morton-Firth and Bray, 1998), and *Virtual Cell* (Schaff et al., 2000, 2001). Our long-term goal is to develop a flexible and adaptable environment that provides (1) the ability to interact seamlessly with a variety of software tools that implement different approaches to modeling, parameter analysis, and other related tasks, and (2) the ability to interact with biologically-oriented databases containing data, models and other relevant information.

In the sections that follow, we describe the Systems Biology Workbench project, including our motivations and approach, and we summarize our current design for the Workbench software environment. We also discuss the *Systems Biology Markup Language* (SBML), a model description language that serves as the common substrate for communications between components in the Workbench. We close by summarizing the current status of the project and our future plans.

Motivations for the Project

The staggering volume of data now emerging from molecular biotechnology leave little doubt that extensive computer-based modeling, simulation and analysis will be critical to understanding and interpreting the data (e.g., Abbott, 1999; Gilman, 2000; Popel and Winslow, 1998; Smaglik, 2000). This has lead to an explosion in the development of computer tools by research groups across the world. Example application areas include the following:

- Filtering and preparing data (e.g., gene expression micro- and macroarray image processing and clustering/outlier identification), as well as performing regression and pattern-extraction;
- Database support, including remote database access and local data storage and management (e.g., techniques for combining gene expression data with analysis of gene regulatory motifs);
- Model definition using graphical model capture and/or mathematical description languages, as well as model preprocessing and translation (e.g., capturing and describing the three-dimensional structure of subcellular structures, and their change over time);
- Model computation and analysis, including parameter optimization, bifurcation/sensitivity analysis, diffusion/transport/buffering in complex 3-D structures, mixed stochastic-deterministic systems, differential-algebraic systems, qualitative-qualitative inference, and so on; and
- Data visualization, with support for examining multidimensional data, large data sets, and interactive steering of ongoing simulations.

This explosive rate of progress in tool development is exciting, but the rapid growth of the field has been accompanied by problems and pressing needs. One problem is that simulation models and results often cannot be compared, shared or re-used directly because the tools developed by different groups often are not compatible with each other. As the field of systems biology matures, researchers increasingly need to communicate their results as computational models rather than box-and-arrow diagrams. But they also need to reuse each other's published and curated models as library elements in order to succeed with large-scale efforts (e.g., the Alliance for Cellular Signaling, Gilman, 2000; Smaglik, 2000). These needs require that models implemented in one software package be portable to other software packages, to maximize public understanding and to allow building up libraries of curated computational models.

A second problem is that software developers often end up duplicating each other's efforts when implementing different packages. The reason is that individual software tools typically are designed initially to address a specific set of issues, reflecting the expertise and preferences of the originating group. As a result, most packages have niche strengths which are different from, but complementary to, the strengths of other packages. But because the packages are separate systems, developers end up having to re-invent and implement much general functionality needed by every simulation/analysis tool. The result is duplication of effort in developing software infrastructure.

No single package currently answers all the needs of the emerging systems biology community, despite an emphasis by many developers to make their software tools omnipotent. Nor is such a scenario likely: the range of tools needed is vast, and new techniques requiring new tools are emerging far more rapidly than the rate at which any single package may be developed. For the foreseeable future, then, systems biology researchers are likely to continue using multiple packages to carry out their work. The best we can do is to develop ways to ease sharing and communication between such packages now and in the future.

These considerations lead us to believe that there is an increasingly urgent need to develop common standards and mechanisms for sharing resources within the field of systems biology. We hope to answer this need through the ERATO Systems Biology Workbench project.

THE SYSTEMS BIOLOGY MARKUP LANGUAGE

The current inability to exchange models between simulation/analysis tools has its roots in the lack of a common format for describing models. We sought to address this problem from the very beginning of the project by developing an open, extensible, model representation language.

The Systems Biology Workbench project was conceived at an ERATO-sponsored workshop held at the California Institute of Technology, USA, in December, 1999. The first meeting of all the collaborators at *The First Workshop on Software Platforms for Molecular Biology* was held at the same location in April, 2000. The participants collectively decided to begin by developing a common, XML-based (Bosak and Bray, 1999), declarative language for representing models. A draft version of this Systems Biology

Markup Language (SBML) was developed by the Caltech ERATO team and delivered to all collaborators in August, 2000. This draft version underwent extensive discussion over mailing lists and then again during *The Second Workshop on Software Platforms for Molecular Biology* held in Tokyo, Japan, November 2000. A revised version of SBML was issued by the Caltech ERATO team in December, 2000, and after further discussions over mailing lists and in meetings, a final version of the base-level definition of SBML was released publicly in March, 2001 (Hucka et al., 2001).

The Form of the Language

SBML Level 1 is the result of merging modeling-language features from the seven tools mentioned in the introduction (BioSpice, DBSolve, E-Cell, Gepasi, Jarnac, StochSim, and Virtual Cell). This base level definition of the language supports non-spatial biochemical models and the kinds of operations that are possible in these analysis/simulation tools. A number of potentially desirable features were intentionally omitted from the base language definition. Subsequent releases of SBML (termed *levels*) will add additional structures and facilities currently missing from Level 1. By freezing sets of features in SBML definitions at incremental levels, we hope to provide the community with stable standards to which software authors can design to, while at the same time allowing the simulation community to gain experience with the language definitions before introducing new elements. At the time of this writing, we are actively developing *SBML Level 2*, which is likely to include the ability to represent submodels, arrays and array connectivity, database references, three-dimensional geometry definition, and other features.

Shown at right is an example of a simple, hypothetical biochemical network that can be represented in SBML. Broken down into its constituents, this model contains a number of components: reactant species, product species, reactions, rate laws, and parameters in the rate laws. To analyze or simulate this network, additional components must be made explicit, including compartments for the species and units on the various quantities. The top level of an SBML model definition simply consists of lists of these components:

$$X_0 \xrightarrow{k_1 X_0} S_1$$

$$S_1 \xrightarrow{k_2 S_1} X_1$$

$$S_1 \xrightarrow{k_3 S_1} X_2$$

beginning of model definition
  list of unit definitions (optional)
  list of compartments
  list of species
  list of parameters (optional)
  list of rules (optional)
  list of reactions
end of model definition

The meaning of each component is as follows:

Unit definition: A name for a unit used in the expression of quantities in a model. Units may be supplied in a number of contexts in an SBML model, and it is convenient to have a facility for both setting default units and for allowing combinations of units to be given abbreviated names.

Compartment: A container of finite volume for substances. In SBML Level 1, a compartment is primarily a topological structure with a volume but no geometric qualities.

Specie: A substance or entity that takes part in a reaction. Some example species are ions such as Ca^{2++} and molecules such as glucose or ATP. The primary qualities associated with a specie in SBML Level 1 are its initial amount and the compartment in which it is located.

Parameter: A quantity that has a symbolic name. SBML Level 1 provides the ability to define parameters that are global to a model, as well as parameters that are local to a single reaction.

Reaction: A statement describing some transformation, transport or binding process that can change the amount of one or more species. For example, a reaction may describe how certain entities (reactants) are transformed into certain other entities (products). Reactions have associated rate laws describing how quickly they take place.

Rule: In SBML, a mathematical expression that is added to the differential equations constructed from the set of reactions, and can be used to set parameter values, establish constraints between quantities, etc.

A software package can read in a model expressed in SBML and translate it into its own internal format for model analysis. For instance, a package might provide the ability to simulate a model, by constructing a set of differential equations representing the network and then performing numerical time integration on the equations to explore the model's dynamic behavior. The output of the simulation might consist of plots of various quantities in the model as they change over time.

SBML allows models of arbitrary complexity to be represented. We present a simple, illustrative example of using SBML in Appendix A, but much more elaborate models are possible. The complete specification of SBML Level 1 is available from the project's World Wide Web site (`http://www.cds.caltech.edu/erato/`).

Relationships to Other Efforts

There are a number of ongoing efforts with similar goals as those of SBML. Many of them are oriented more specifically toward describing protein sequences, genes and related elements for database storage and search. These are generally not intended to be computational models, in the sense that they do not describe entities and behavioral rules in such a way that a simulation package could "run" the models.

The effort closest in spirit to SBML is CellML™ (CellML Project, 2001). CellML is an XML-based markup language designed for storing and exchanging computer-based biological models. It includes facilities for representing model structure, mathematics and additional information for database storage and search. Models are described in terms of networks of connections between discrete components; a component is a functional unit that may correspond to a physical compartment or simply a convenient modeling abstraction. Components contain variables and connections contain mappings between the variables of connected components. CellML provides facilities for grouping components and specifying the kinds of relationships that may exist between components. It uses MathML (Ausbrooks et al., 2001) for expressing mathematical relationships and provides the ability to use ECMAScript (formerly known as JavaScript; ECMA, 1999) to define functions.

The constructs in CellML tend to be at a more abstract and general level than those in SBML Level 1, and it provides somewhat more general capabilities. By contrast, SBML is closer to the internal object model used in model analysis software. Because SBML Level 1 is being developed in the context of interacting with a number of existing simulation packages, it is a more concrete language than CellML and may be better suited to its purpose of enabling interoperability with existing simulation tools. However, CellML offers viable alternative ideas and the developers of SBML and CellML are actively engaged in ensuring that the two representations can be translated between each other.

THE SYSTEMS BIOLOGY WORKBENCH

In this section, we describe how we approached the development of the Systems Biology Workbench from both philosophical and technical standpoints; we also summarize the overall architecture of the system and explain how it enables integration and sharing of software resources.

Driving Principles

The Systems Biology Workbench is primarily a system for integrating resources. It provides infrastructure that can be used to interface to software components and enable them to communicate with each other. The components in this case may be simulation codes, analysis tools, user interfaces, database interfaces, script language interpreters, or in fact any piece of software that conforms to a certain well-defined interface.

We knew from the outset that the success of the Workbench would be contingent on contributors benefitting from sharing resources through the system. For this reason, we made three commitments toward this goal:

- The Systems Biology Workbench software will be made publicly and freely available under open-source licensing (O'Reilly, 1999; Raymond, 1999). The agency funding the development of the Workbench (the Japan Science and Technology Corporation) has formally agreed that all SBW code can be placed under open-source terms. At the same time, the license terms will not force contributors to apply the same copying and distribution terms to their contributed software—developers will be free to make their components available under license terms that best suit them. They may choose to make a component available under the same open-source license, in which case it may be packaged together with the Systems Biology Workbench software distribution; however, there is nothing preventing an author from creating an SBW-compatible component that is closed-source and distributed privately.
- The Workbench architecture is designed to be symmetric with respect to facilities made available to components. All resources available through the Workbench system are equally available to all components, and no single component has a controlling share. All contributors thereby benefit equally by developing software for the Workbench.
- The direct interface between a software component and the Systems Biology Workbench is a specific application programming interface (API). The component's authors may chose to implement this API directly and publish the details of the operations provided by the component. Alternatively, they may enter into a formal agreement with us (the authors of the Workbench) in which they reveal only to us their component's API, and we will write an interface between the Workbench and this API. The latter alternative allows contributors to retain maximum confidentiality regarding their component, yet still make the component available (in binary executable form) for users of the Workbench.

The Overall Architecture of the Workbench

Although our initial focus is on enabling interaction between the seven simulation/analysis packages already mentioned, we are equally interested in creating a flexible architecture that can support future developments and new tools. We have approached this by using a combination of three key features: (1) a framework divided into layers, (2) a highly modular, extensible structure, and (3) inter-component communications facilities based on message-passing.

Layered Framework

We sought to maximize the reusability of the software that we developed for the Workbench by dividing the Workbench infrastructure into two layers: the Systems Biology Workbench itself, and a lower-level substrate called the *Biological Modeling Framework* (BMF). The latter is a general software framework that can be used in developing a variety of biological modeling and analysis software applications. It not directly tied to the current architecture of SBW, allowing us the freedom to evolve and change SBW in the future while still maintaining a relatively stable foundation.

BMF provides basic scaffolding supporting a modular, extensible application architecture (see below), as well as a set of useful software components that can be used as black boxes in constructing a system (cf. Fayad et al., 1999). Other projects should be able to start with BMF, add their own domain- and task-specific elements, and thereby implement a system specialized for other purposes. This is how the neuroscience-oriented Modeler's Workspace (Hucka et al, 2000) is being implemented. Computational biologists and other users do not need to be aware of the existence of BMF—it is scaffolding used by the developers of SBW and other tools, and not a user-level construct.

SBW is a particular collection of application-specific components layered on top of BMF. These collectively implement what users experience as the "Systems Biology Workbench". Some components add functionality supporting the overall operation of the Workbench, such as the message-passing communications facility; other components implement the interfaces to the specific simulation/analysis tools made available in the Workbench. Figure 6.1 illustrates the overall organization of the layers.

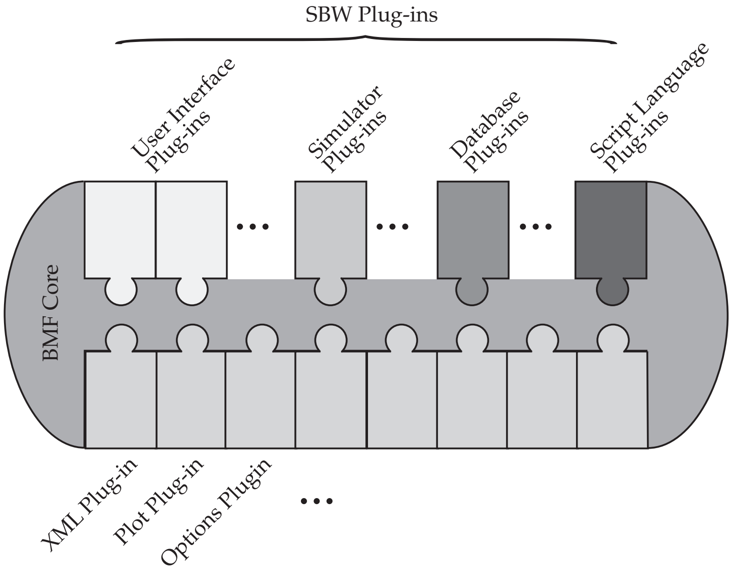

Figure 6.1 The Systems Biology Workbench (SBW) is a collection of software components layered on top of a simple plug-in architecture called the Biological Modeling Framework (BMF).

Highly Modular, Extensible Architecture

The Biological Modeling Framework layer that underlies SBW is implemented in Java and provides (1) support for managing pluggable components ("plug-ins") and (2) a collection of basic components (such as an XML Schema-aware parser, file utilities, basic plotting and graphing utilities, etc.) that are useful when implementing the typical kinds of applications for which BMF is intended.

The kinds of application-specific plug-ins that comprise SBW can generally be grouped by their purposes: user interfaces, simulator/analyzer interfaces, scripting language interfaces, and database interfaces. Other kinds are possible, but these are the primary application-specific plug-in types that we foresee being developed. There can be any number of plug-ins in a given system, subject to the usual limitations on computer resources such as memory. Each plug-in needs to conform to certain minimal interface requirements dictated by the framework, discussed further below. A plug-in can make use of any public services provided by BMF, the core plug-ins, and application-specific plug-ins.

By virtue of the software environment provided by the Java 2 Platform (Flanagan, 1999; Gosling et al., 1996), plug-ins can be loaded dynamically, without recompiling or even necessarily restarting a running application. This can be used to great advantage for a flexible and powerful environment. For example, an application could be designed to be smart about how it handles data types, loading specialized plug-ins to allow a user to interact with particular data objects on an as-needed basis. If the user does not already have a copy of a certain plug-in stored on the local disk, the plug-in could be obtained over the Internet, much like current-generation Web browsers can load plug-ins on demand. In this manner, plug-ins for tasks such as displaying specific data types or accessing third-party remote databases could be easily distributed to users.

Message-Passing for Inter-Component Communications

One of the challenges in developing a modular system, especially one that allows incremental addition and change by different developers, is designing appropriate interfaces between components. Knowledge of an element's interface necessarily becomes encoded in its structure; otherwise, component *A* would not know how to communicate with component *B*. Many frameworks are designed around a hierarchy of object classes and interfaces; this lets them provide a rich selection of features. However, for this project, class hierarchies have two important disadvantages:

- Methods and calling conventions for accessing different objects become scattered throughout the structure of each component in the system. The effects of changing an interface are not localized in client code: changing the interface of a fundamental object may require rewriting code in many different locations in every other component that uses it.

• The task of creating interfaces to components written in other programming languages is more complex. If the component is a library, the foreign-function bridge (whether it is implemented using the Java Native Interface [JNI] or other) must expose all of the methods provided by the component's interface, which requires substantial programming effort and significant maintenance. Similarly, if the component is a stand-alone application, the mechanism used to communicate with it must provide access to all or most of the classes and methods provided by the component's interface. CORBA (Object Management Group, 2001; Seetharman, 1998; Vinoski, 1997) is one technology that could be used to cope with these issues, but we decided to avoid requiring its use in SBW because we feared its complexity would be too daunting to potential contributors.

We began developing SBW using the common approach of designing object class hierarchies representing different functions and simulation/analysis capabilities, but soon decided that the problems above would become too onerous. We chose instead to base inter-component communications on passing messages via byte streams.

In this approach, each component or component wrapper needs to expose a simple programmatic interface, consisting of only a handful of methods. The main method in this interface (`PluginReceive`) is the entry point for exchanging messages. Other methods in the interface provide a way for starting the execution of the component (`PluginStart`), and for obtaining its name and a list of strings describing the kinds of capabilities it implements (`PluginRegistrationInformation`). The latter can be used by other components to discover programmatically what services a new component provides.

A *message* in this framework is a stream of bytes that encodes a service identifier and a list of arguments. The service identifier is determined from the list of services advertised by the component. The arguments are determined by the particular service. For example, a command to perform steady-state analysis on a biochemical network model would require a message that encodes the model and a list of parameters on the kind of analysis desired. The result passed back by the component would be in the form of another message.

The representation of the data in a message is encoded according to a specific scheme. The scheme in SBW allows for the most common data types to be packaged into a message. Each element in a message is preceded by a byte that identifies its data type. The types currently supported include character strings, integers, double-sized floating-point numbers, arrays of homogeneous elements, and lists of heterogeneous elements.

How does this approach help cope with the two problems listed above? At first glance, it may seem that this approach merely hides functionality behind a simple façade. After all, changing the operation of a component still requires other components to be changed as well, for example to compose and parse messages differently as needed. However, in this approach, the effects of actual interface changes are more *localized*, typically affecting fewer objects in other components.

The message-passing approach also simplifies the task of interfacing to components implemented in different programming languages. Rather than have to provide native-code interfaces (say, using Java JNI) to every method in a large class hierarchy, only a few methods must be implemented. Likewise, it is much simpler to link components that run as separate processes or on remote computers. A simple message-passing stream is easily implemented through a TCP/IP socket interface, the simplest and most familiar networking facility available nearly universally on almost every computer platform.

The current message-passing scheme can be used to exchange messages encoded in XML, which makes this approach similar to XML-RPC (Winer, 1999) and SOAP (Box et al., 2000). However, our message protocol allows other data types to be encoded as well. Using XML exclusively would require binary data to be encoded into, and unencoded from, a textual representation, which would impact performance and potentially affect floating-point accuracy. We therefore designed the protocol to allow binary data as well as XML to be exchanged.

Advantages of an Extensible Framework Approach

The modular framework approach is pervasive throughout the design of the system. Both the underlying BMF layer and an application layer such as SBW are implemented as plug-ins attached to a small core. Nearly all of the functionality of both layers are determined by the plug-ins themselves.

The primary benefits of using a modular framework approach accrue to software developers. For a developer who would like to build upon BMF and create a new system, or take an existing system such as SBW and create enhancements or specialized components, the following are some of the benefits (Fayad et al., 1999):

- *Modularity*. A framework is composed of modules, each of which encapsulates implementation details behind a stable interface. Design and implementation changes to individual modules are less likely to have an impact on other modules or the overall system.
- *Reusability*. A framework offers a set of elements that represent common solutions to recurring problems. Reusing the elements saves design and development time.
- *Extensibility*. Extensibility is designed into a framework by providing well-defined ways to add new modules and extend existing modules. In its most essential form, a framework is a substrate for bringing software modules together.

Although frameworks were invented by software developers to simplify the implementation of complex software, users also benefit when a system is based on a framework approach. For a biologist or other user who would like to employ a tool built on top of BMF and SBW, there are two primary gains:

- *Control*. Users are given greater control over the composition of a framework-based system than a system based on a more traditional architecture. They can use just those modules that they need, and they have the freedom to chose different implementations of the same functionality, or even develop their own implementations, all without altering the rest of the system.
- *Reusability*. A successful framework may be reused to implement other domain-specific tools, reducing the burden on a user by allowing them to carry over their experiences involving the common parts.

Motivations for Using Java

We chose Java as the implementation language for the underlying BMF layer of SBW because it offers a number of attractive features and meets several objectives. In particular, Java arguably provides one of the most portable cross-platform environments currently available. Java also provides a built-in mechanism for dynamic loading of code, simplifying the implementation of an architecture oriented around plug-ins. Finally, Java provides a rich platform for development, with such things as remote invocation facilities and GUI widgets, on all supported platforms.

It is worth noting that plug-ins for the system are *not required* to be written in Java. Java provides mechanisms for interfacing to software written in other languages, through the Java Native Interface. Thus, although Java is used to implement the core of the system, plug-ins can be written in other languages and incorporated into an application built on top of the framework.

Although Java has received negative publicity in the past with respect to performance (Tyma, 1998), we do not feel that choosing Java will have a significant impact on run-time performance. The reason is that the core of the Systems Biology Workbench is a thin layer and most of the execution time in an application is spent within application-specific plug-ins. Those can be written in other languages if performance becomes an issue.

SUMMARY AND STATUS

The aim of the Systems Biology Workbench project is to create a modular, open software environment that enables different simulation and analysis tools to be used together for systems biology research. As part of this effort, we have also developed a model description language, the Systems

Biology Markup Language, that can be used to represent models in a form independent of any specific simulation/analysis tool. SBML is based on XML for maximal flexibility, interchangeability, and future compatibility.

Availability

We will make the software available under open-source terms from the Caltech ERATO team's web site, `http://www.cds.caltech.edu/erato/`. At the time of this writing, we are in the process of developing and implementing the core functionality of the Systems Biology Workbench, along with an initial set of plug-ins. The aim of this effort is to demonstrate the concepts described above and provide a medium through which we will develop and refine the APIs. We expect to make this initial implementation available in the first half of 2001, and to release the first full version of the Workbench by the end of 2001.

Future Plans

The final specification for SBML Level 1 was released in March, 2001. The relevant documents are available from the Caltech ERATO team's web site, mentioned above. SBML Level 2 is currently under development, and we anticipate making a preliminary specification available later in the year 2001. We will publish the specification documents on the web site as they become available.

ACKNOWLEDGMENTS

We are grateful for comments, advice and help from fellow ERATO Kitano Systems Biology project members Mark Borisuk, Mineo Morohashi, Eric Mjolsness, Bruce Shapiro, and Tau-Mu Yi.

SBML Level 1 was developed with the help of many people. We wish to acknowledge in particular the authors of BioSpice, DBSolve, E-Cell, Gepasi, StochSim, and Virtual Cell, and members of the `sysbio` mailing list. We are especially grateful to the following people for discussions and knowledge: Dennis Bray, Athel Cornish-Bowden, David Fell, Carl Firth, Warren Hedley, Martin Ginkel, Igor Goryanin, Jay Kaserger, Andreas Kremling, Nicolas Le Novère, Les Loew, Daniel Lucio, Pedro Mendes, Eric Mjolsness, Jim Schaff, Bruce Shapiro, Tom Shimizu, Hugh Spence, Joerg Stelling, Kouichi Takahashi, Masaru Tomita, and John Wagner.

APPENDIX

A EXAMPLE OF A MODEL ENCODED IN XML USING SBML

Consider the following hypothetical branched system:

$$X_0 \xrightarrow{k_1 X_0} S_1$$

$$S_1 \xrightarrow{k_2 S_1} X_1$$

$$S_1 \xrightarrow{k_3 S_1} X_2$$

The following is the main portion of an XML document that encodes the model shown above:

```
<sbml level="1" version="1">
    <model name="Branched">
        <notes>
            <body xmlns="http://www.w3.org/1999/xhtml">
                <p>Simple branched system.</p>
                <p>reaction-1:   X0 -> S1; k1*X0;</p>
                <p>reaction-2:   S1 -> X1; k2*S1;</p>
                <p>reaction-3:   S1 -> X2; k3*S1;</p>
            </body>
        </notes>
        <listOfCompartments>
            <compartment name="A" volume="1"/>
        </listOfCompartments>
        <listOfSpecies>
            <specie name="S1" initialAmount="0" compartment="A"
                    boundaryCondition="false"/>
            <specie name="X0" initialAmount="0" compartment="A"
                    boundaryCondition="true"/>
            <specie name="X1" initialAmount="0" compartment="A"
                    boundaryCondition="true"/>
            <specie name="X2" initialAmount="0" compartment="A"
                    boundaryCondition="true"/>
        </listOfSpecies>
        <listOfReactions>
            <reaction name="reaction_1" reversible="false">
                <listOfReactants>
                    <specieReference specie="X0"
                                     stoichiometry="1"/>
                </listOfReactants>
                <listOfProducts>
                    <specieReference specie="S1"
                                     stoichiometry="1"/>
                </listOfProducts>
                <kineticLaw formula="k1 * X0">
                    <listOfParameters>
                        <parameter name="k1" value="0"/>
                    </listOfParameters>
                </kineticLaw>
            </reaction>
            <reaction name="reaction_2" reversible="false">
                <listOfReactants>
                    <specieReference specie="S1"
                                     stoichiometry="1"/>
```

```
                </listOfReactants>
                <listOfProducts>
                    <specieReference specie="X1"
                                     stoichiometry="1"/>
                </listOfProducts>
                <kineticLaw formula="k2 * S1">
                    <listOfParameters>
                        <parameter name="k2" value="0"/>
                    </listOfParameters>
                </kineticLaw>
            </reaction>
            <reaction name="reaction_3" reversible="false">
                <listOfReactants>
                    <specieReference specie="S1"
                                     stoichiometry="1"/>
                </listOfReactants>
                <listOfProducts>
                    <specieReference specie="X2"
                                     stoichiometry="1"/>
                </listOfProducts>
                <kineticLaw formula="k3 * S1">
                    <listOfParameters>
                        <parameter name="k3" value="0"/>
                    </listOfParameters>
                </kineticLaw>
            </reaction>
        </listOfReactions>
    </model>
</sbml>
```

The XML encoding shown above is quite straightforward. The outermost container is a tag, `smbl`, that identifies the contents as being systems biology markup language. The attributes `level` and `version` indicate that the content is formatted according to version 1 of the Level 1 definition of SBML. The `version` attribute is present in case SBML Level 1 must be revised in the future to correct errors.

The next-inner container is a single `model` element that serves as the highest-level object in the model. The model has a name, "Branched". The model contains one compartment, four species, and three reactions. The elements in the `listOfReactants` and `listOfProducts` in each reaction refer to the names of elements listed in the `listOfSpecies`. The correspondences between the various elements should be fairly obvious.

The model includes a `notes` annotation that summarizes the model in text form, with formatting based on XHTML. This might be useful for a software package that is able to read such annotations and render them in HTML.

References

Abbott, A. (1999). Alliance of US Labs Plans to Build Map of Cell Signalling Pathways. *Nature* 402:219–220.

Arkin, A. P. (2001). *Simulac* and *Deduce*. Available via the World Wide Web at `http://gobi.lbl.gov/~aparkin/Stuff/Software.html`.

Ausbrooks, R., Buswell, S., Dalmas, S., Devitt, S., Diaz, A., Hunter, R., Smith, B., Soiffer, N., Sutor, R., and S. Watt. (2001). Mathematical Markup Language (MathML) Version 2.0: W3C Proposed Recommendation 08 January 2001. Available via the World Wide Web at `http://www.w3.org/TR/2001/PR-MathML2-20010108/`.

Bosak, J., and Bray, T. (1999). XML and the Second-Generation Web. *Scientific American*, May. Also available via the World Wide Web at `http://www.sciam.com/1999/0599issue/0599bosak.html`.

Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). Simple Object Access Protocol (SOAP) 1.1: W3C Note 08 May 2000. Available via the World Wide Web at `http://www.w3.org/TR/SOAP/`.

Bray, D., Firth, C., Le Novère, N., and Shimizu, T. (2001). *StochSim*, Available via the World Wide Web at `http://www.zoo.cam.ac.uk/comp-cell/StochSim.html`.

CellML Project. (2001). CellML Project Home Page. `http://www.cellml.org/`.

ECMA. (1999). ECMAScript Language Specification: Standard ECMA-262, 3rd Edition. Available via the World Wide Web at `http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM`.

Gosling, J., Joy, B., and Steele, G. (1996). The Java™ Language Specification. Addison-Wesley.

Fayad, M. E., Schmidt, D. C., and Johnson, R. E. (1999). *Building Application Frameworks*. Wiley.

Flanagan, D. (1999). *Java in a Nutshell*. O'Reilly & Associates.

Gilman, A. (2000). A Letter to the Signaling Community. Alliance for Cellular Signaling, The University of Texas Southwestern Medical Center. Available via the World Wide Web at `http://afcs.swmed.edu/afcs/Letter_to_community.htm`.

Goryanin, I. (2001). DBsolve: Software for Metabolic, Enzymatic and Receptor-Ligand Binding Simulation. Available via the World Wide Web at `http://websites.ntl.com/~igor.goryanin/`.

Goryanin, I., Hodgman, T. C., and Selkov, E. (1999). Mathematical Simulation and Analysis of Cellular Metabolism and Regulation. *Bioinformatics* 15(9):749–758.

Hucka, M., Beeman, D., Shankar, K., Emardson, S., and Bower, J. M. (2000). The Modeler's Workspace: A Tool for Computational Neuroscientists. *Society for Neuroscience Abstracts* 30, 21.69.

Hucka, M., Finney, A., Sauro, H. S., and 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.cds.caltech.edu/erato`.

Mendes, P. (1997). Biochemistry by Numbers: Simulation of Biochemical Pathways with Gepasi 3. *Trends in Biochemical Sciences* 22:361–363.

Mendes, P. (2001). *Gepasi 3.21*. Available via the World Wide Web at `http://www.gepasi.org/`.

Morton-Firth, C. J., and Bray, D. (1998). Predicting Temporal Fluctuations in an Intracellular Signalling Pathway. *Journal of Theoretical Biology* 192:117–128.

O'Reilly, T. (1999). Lessons from Open-Source Software Development. *Communications of the ACM* 42(4):32–37.

Object Management Group. (2001). *CORBA/IIOP 2.3.1*. Specification documents available via the World Wide Web at `http://www.omg.org/`.

Popel, A., and Winslow, R. L. (1998). A Letter From the Directors … Center for Computational Medicine & Biology, Johns Hopkins School of Medicine, Johns Hopkins University. Available via the World Wide Web at `http://www.bme.jhu.edu/ccmb/ccmbletter.html`.

Raymond, E. S. (1999). *The Cathedral & the Bazaar*, O'Reilly & Associates. (This is the book form; the paper alone is available via the World Wide Web at `http://www.tuxedo.org/~esr/writings/cathedral-bazaar/`.)

Schaff, J., Slepchenko, B., and Loew, L. M. (2000). Physiological Modeling with the Virtual Cell Framework. In *Methods in Enzymology*, Academic Press, 321:1–23.

Schaff, J., Slepchenko, B., Morgan, F., Wagner, J., Resasco, D., Shin, D., Choi, Y. S., Loew, L., Carson, J., Cowan, A., Moraru, I., Watras, J., Teraski, M., and Fink., C. (2001). *Virtual Cell*. Available over the World Wide Web at `http://www.nrcam.uchc.edu/`.

Seetharaman, K. (1998). The CORBA Connection. *Communications of the ACM* 41(10):34–36.

Smaglik, P. (2000). For My Next Trick ... *Nature* 407:828–829.

Sauro, H. M. (1991). SCAMP: A Metabolic Simulator and Control Analysis Program. *Mathl. Comput. Modelling* 15:15–28.

Sauro, H. M. and Fell, D. A. (2000). Jarnac: A System for Interactive Metabolic Analysis. In Hofmeyr, J-H. S., Rohwer, J. M., and Snoep, J. L. (eds.), *Animating the Cellular Map 9th International BioThermoKinetics Meeting*, Stellenbosch University Press.

Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J. C., and Hutchison, C. (1999). E-CELL: Software Environment for Whole Cell Simulation. *Bioinformatics* 15(1):72–84.

Tomita, M., Nakayama, Y., Naito, Y., Shimizu, T., Hashimoto, K., Takahashi, K., Matsuzaki, Y., Yugi, K., Miyoshi, F., Saito, Y., Kuroki, A., Ishida, T., Iwata, T., Yoneda, M., Kita, M., Yamada, Y., Wang, E., Seno, S., Okayama, M., Kinoshita, A., Fujita, Y., Matsuo, R., Yanagihara, T., Watari, D., Ishinabe, S., and Miyamoto, S. (2001). *E-Cell*. Available via the World Wide Web at `http://www.e-cell.org`.

Tyma, P. (1998). Why Are We Using Java Again? *Communications of the ACM* 41(6):38–42.

Vinoski, S. (1997). CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments. *IEEE Communications Magazine*. Feb.

Winer, D. (1999). XML-RPC Specification. Available via the World Wide Web at `http://www.xml-rpc.com/spec/`.