Proceedings of the 2006 Winter Simulation Conference
L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds.

SBW – A MODULAR FRAMEWORK FOR SYSTEMS BIOLOGY

Frank T. Bergmann
Herbert M. Sauro

Computational and Systems Biology
Keck Graduate Institute
535 Watson Drive
Claremont, CA 91711, U.S.A.

ABSTRACT

A large number of software packages are available to assist researchers in systems biology. In this paper, we describe the current state of the Systems Biology Workbench (SBW), a modular framework that connects modeling and analysis applications, enabling them to reuse each other's capabilities. We describe how users and developers will perceive SBW and then focus on currently available SBW modules. The software, tutorial manual, and test models are freely available from the Computational and Systems Biology group at Keck Graduate Institute. Source code is available from SourceForge. The software is open source and licensed under BSD.

1 INTRODUCTION

The increasing popularity of systems biology (Kitano 2005) brings with it a demand for computational modeling and analyzing tools to aid researchers. Many modeling and model-analysis applications have been written, and using the popular exchange format SBML (Hucka et al. 2003) most of these applications allow import/export of SBML models. Still, the process of exporting and importing models this way is rather cumbersome and disrupts the workflow of researchers. There are two possible solutions for this problem. The first would be to form an application that combines all the features a researcher would want. This of course would be hard to achieve and even harder to maintain. The alternative is to allow applications, that are specialized in different areas of computation, to communicate with each other.

Instead of developing monolithic applications, we focused on a modular approach. We chose to focus on a platform/language independent framework of loosely coupled applications: the Systems Biology Workbench (SBW) (Hucka et al. 2002). This allows newly developed applications to reuse functionality from existing applications, rather than re-implement them. For example, rather than implementing a new simulator for a new project, we simply take advantage of an existing SBW-enabled one.

An alternative approach has been taken by BioSPICE (Garvey et al. 2003). Here the approach is to "wrap" applications into Analyzer objects with specified input and output behavior. These Analyzers can then be used within an IDE to create workflows between applications. While this approach works for applications that do not require user input at runtime, it does not work well for a modeling process. A modeling process requires more flexibility than a workflow can provide.

1.1 Architecture

The Systems Biology Workbench (SBW) is a resource-sharing framework based on a broker architecture. SBW uses a message based system that employs binary formatted messages sent over TCP/IP (Sauro et al. 2003). This basic architecture was chosen with regard to performance, portability, simplicity, and language-neutrality. We also considered XML-RPC, Java RMI (Sun Microsystems 2006), and CORBA (OMG 2006) but ultimately favored a simple peer-to-peer technique.

Binding libraries, provided for most programming languages, allow applications to become SBW-enabled. This means that the application can make use of functionality provided by other SBW-enabled applications or provide functionality for other applications. The binding libraries are written to fit the natural pattern of the programming language of choice. SBW-enabled applications are called modules. Each module in turn will provide one or more services and each service will have one or more methods. Each method is defined by its unique signature and help string. Help strings are also available at the service and module level. Help strings provide a degree of internal documentation to the SBW interface and are useful in a number of situations (see SBW add-on). Services are also classified into categories, which allow the discovery of modules not only by name but also by functionality. Ex-

ample categories include simulators (with various levels), SBML translators, or a simple analyzer category. The convention is that every service belonging to a certain category implements specific method signatures that allow the services to be exchanged at a later time.

At the core of SBW stands the SBW Broker. The Broker negotiates communication between SBW modules. Its main function is to provide other modules with events such as the startup/shutdown of applications, a system shutdown, or advertising newly registered modules. The Broker also allows the user to find SBW modules either by name or by category. While the broker is most often used to negotiate communications between local modules, it can also communicate with remote brokers. Furthermore, SBW allows all SBW modules to be available as Web Services (W3C 2006). This communication flow can be summarized as in Figure 1.

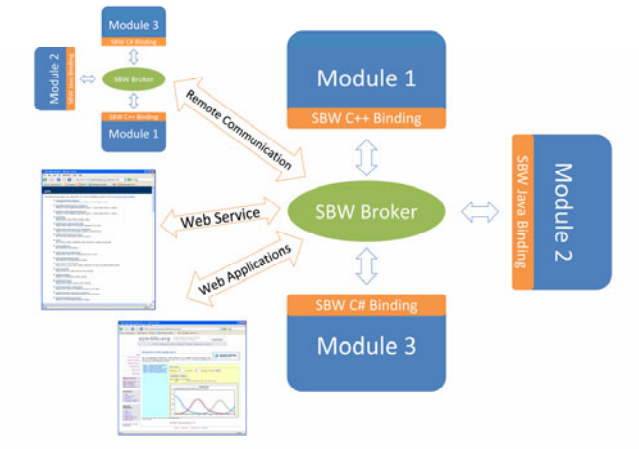

Figure 1: SBW Communication Flow

SBW can be used in a hosting environment where it provides analysis support for SBML models. This has been done by BioSPICE, BioUML (Kolpakov 2002), CellDesigner (Kitano et al. 2005), and JDesigner (Sauro et al. 2003). BioSPICE allows integrating SBW modules as analyzers into the generated workflow. A simpler way of integration has been done in CellDesigner and JDesigner by incorporating a SBW menu.

As shown with PySCeS (Olivier, Rohwer, and Hofmeyr 2005) scripting can be a powerful tool for researchers. We developed a set of scripts that allow SBW to be used in the Python interactive mode. Users will be notified of every module startup and are able to access all SBW modules from the shell. Similarly, users can access SBW via scripting from Matlab (Wellock et al. 2005).

A more detailed description of the inner workings of SBW can be found in (Hucka et al. 2002) and (Sauro et al. 2003).

2 SBW FROM A USER'S PERSPECTIVE

A user of an SBW-enabled application will probably never perceive SBW working at all. For a user, SBW is a collection of loosely coupled applications, all of which support SBML. One example would be the use of an SBW-enabled model editor such as JDesigner, which is a program that allows the user to load or construct a biochemical model and then analyze the model. When loading a model created by another software tool, JDesigner will ask another SBW module to generate a layout for the loaded model. As JDesigner has no inherent simulation capabilities, it uses another SBW module to simulate the loaded model. Similarly, if a user performs structural analysis on the model, JDesigner will acquire this information via SBW from Metatool (Pfeiffer et al. 1999).

JDesigner (and CellDesigner) also feature an SBW menu. This menu is populated dynamically on startup and lists all SBW modules implementing the SBW analysis category. When the user selects a program from this menu, the SBW-enabled program is invoked, the model is passed to it, and the program performs further analysis on the model. There is a wide variety of modules available, such as bifurcation analysis, frequency analysis, (stochastic) simulation, and 3D Visualization.

Because the SBW menu is shared between many applications, users now have a new option for moving between these applications. A user will work on an SBML model in one program, and by selecting another SBW module through the SBW menu, sends the model to the next program where the user can further analyze the model, and so forth. This "hopping" from program to program is rather different from the usual export/import scenario and allows users to efficiently move from tool to tool.

3 SBW FROM A DEVELOPER'S PERSPECTIVE

SBW is a modular framework that gives developers tools for basic modeling and analysis tasks such as interrogating models, simulating models, and analyzing models (such as structural or stoichiometry analysis). Therefore, the developer of a new software application can use these tools as a foundation and focus on novel tasks instead of re-inventing the wheel. It should be noted that the basic analysis tools are exchangeable, so that, for example, one simulator can easily be replaced by another one.

Similarly, an existing application written in any supported programming language can be modified to interact with SBW with minimal programming overhead. This enables other applications to use its functionality.

3.1 Language Bindings

Native SBW language bindings are available for the most common programming languages such as C/C++, Java,

.NET languages, Python, Delphi/Kylix, and Matlab. Other languages, such as FORTRAN, would use the C libraries to access SBW. Other languages can be supported; the only requirement is that the programming language provides string handling routines and IP socket access.

The language bindings encapsulate all communication with SBW making remote communication to another module as simple as any other method call. A default method call will result in the application blocking until a result comes back. Alternatively, asynchronous methods can be implemented. Should a call to another SBW module result in failure, exceptions will be thrown in a language dependent way. In other words, for languages that support exception handling (such as C++, Java, or the .NET languages) the caller will receive a proper exception, otherwise error flags have to be checked (for older languages such as C or FORTRAN).

It should be noted that the binding libraries only support simple data types such as bytes, characters, complex numbers, floating point numbers, and strings. Furthermore, one- and two-dimensional regular arrays of these types are supported. More complex data types can be constructed by the use of lists. Lists are recursively defined and can therefore contain any combination of other types. With this set of types, any other data type can be represented.

3.2 SBW Visual Studio Add-on

In order to call a method with SBW, the following four steps must be executed: First, the chosen SBW module must be selected; second, the SBW Service containing the method has to be found; third, the method to execute needs to be selected; finally, a call has to be made. This procedure is approximately the same for all program languages. For example, these steps could be written in C# as follows:

```
// get module instance of a "simulator"
Module oSimulator = new Module("simulator");

// obtain simulation service by category
// lookup
Service oSimService = oSimulator.
findServicesByCategory("simService");

// get a method handle
Method oSimulate = oSimService.
getMethod("double[][] simulate()");

// finally call the method
oSimulate.Call();
```

The code shown above must be repeated, at least in part, for every method that an application wishes to use. For any given module there may be tens or possibly hundreds of method calls. As a result we have devised an add-on to the Microsoft Visual Studio programming environment (Microsoft 2006b) that allows any SBW module to be automatically wrapped ready for use. The add-on can generate interface code for Visual Basic (.NET), C# (.NET), and C++ (unmanaged). The developer simply selects the SBW module and service to be wrapped, along with several options, and a wrapper file will be created and added to the current project. The programming language will be detected from the current project. Possible options that modify the behavior of the wrapper include optimization options that influence when and how the methods are acquired and an option to automatically restart an application, in case it was closed by someone else and is still needed.

Once this is completed, the SBW module will look like any other static class in the current project. For the example above, this means we can replace the four lines of code with a single call:

```
Sim.simulate();
```

Furthermore, as soon as the model service name is typed in the Visual Studio editor ("Sim" in the example above), the developer will be presented with a complete list of available methods, including help strings that explain the purpose of the method.

3.3 SBW Web Service Interface

Another useful tool for developers is the SBW Web Service Interface. The Web Service Interface allows other SBW modules to be wrapped into a Web Service (W3C 2006). Once an SBW module and service are selected, the generator will wrap every method in this service into a proper web method signature. Method and service help strings are used to populate web method help fields. In this way they appear in the WSDL as `wsdl:documentation` in the corresponding `wsdl:operation`. Special care has to be given to the data types. All simple types have direct correspondents in web methods. The same holds for one-dimensional arrays. Two-dimensional regular arrays have to be represented as jagged arrays. Finally, the SBW lists are represented as "`ArrayOfAnyType`".

The web services created in this way can be hosted with Microsoft's IIS (Microsoft 2006a) (provided ASP.NET is available) or with Mono's XSP (Mono-Project 2006) implementation (which can also be integrated into Apache). This allows the hosting on all supported platforms (WIN32, Linux, and OS X) and so presents a very easy way to make legacy applications (e.g., applications written in FORTRAN or C) available as a Web Service.

A sample implementation can be found on the author's website. There, a web service can be created for any installed SBW module / service and tested. (Created Web Services will only be available for a limited amount of time.)

4 AVAILABLE SBW MODULES

Since the publication of the initial description of SBW (Hucka et al. 2002) considerable progress has been made in module development.

4.1 SBW Utility Modules

The following SBW modules provide the core functionality for SBML analysis. They provide the capability to read any version of SBML and SBML layout annotations. Furthermore, basic mathematical analysis can be performed with them. It should be noted once again that any of these modules can be called from any language supported by SBW. Of course, they also can be wrapped up as a Web Service and so can be easily used in more dynamic environments.

Network Object Model (NOM): This module provides a unified SBW interface around libSBML (Bornstein et al. 2006). By creating another layer on top of libSBML we will be able to support future versions of SBML (Finney and Hucka 2003) with the same interface. libSBML provides platform independent SBML reading and writing capabilities for many programming languages and thus is an indispensable tool for application developers supporting SBML. The NOM performs a slightly different role. In its current version, it supports mainly SBML reading, validation, and conversion capabilities. Furthermore, the NOM allows several SBW modules working together on an analysis to share an SBML model in a clipboard-like manner. Instead of the SBML model being parsed repeatedly, this task has to be performed only once, which can improve the overall runtime.

SBW CLAPACK: This module provides a wrapper around functions of CLAPACK, which is the C version of the commonly used Linear Algebra PACKage (Netlib 2006). Since many analysis tools in systems biology need to compute eigenvalues or singular values, or to perform matrix factorizations/inversions, we generated an SBW module that performs these tasks.

SBMLLayoutModule: For the upcoming third version of SBML, the SBML community decided on a layout extension (Gauges et al. 2006). This extension will allow the overall layout (i.e., the position and dimensions of species/compartments as well as the way reactions are drawn) to be stored in an SBML model. Until level three of SBML is finalized, this SBML extension will be stored in an annotation in current models. Based on this layout extension we developed a rendering extension that specifies all rendering information (i.e., color, font, gradients, and grouping information). We implemented the *SBMLLayoutModule* so that all SBW modules can take advantage of a unified layout interface. This module allows reading and modifying a given layout, or generating a new layout from scratch. The module also supports the generation of a bitmap containing the layout for use in another program. We also embedded this SBW module in a Web Application on the authors' website (Bergmann 2006). Finally, since the SBML Layout Extension is not yet widely adopted, the module provides separate support for JDesigner and CellDesigner models, as they are the most commonly used graphical model editors.

DrawNetwork: Since many available SBML models do not contain any layout information, the next logical step was to develop an auto-layout SBW module. This has been done with *DrawNetwork.* This SBW module uses various force-directed layout algorithms based on work by (Fruchterman et al. 1991). Optionally the user of the module can decide to generate alias nodes (i.e., multiple copies of one species that refer to the same element in the SBML); this simplifies the generated layout immensely. The module can be used in server or interactive mode. In server mode, it listens for SBW calls and serves them. In interactive mode the user can manually modify the generated layout, moving or locking certain species or alias species individually. Figure 2 displays the generated layout (with aliasing) of BioModel # 14 (from model repository at www.biomodels.net).

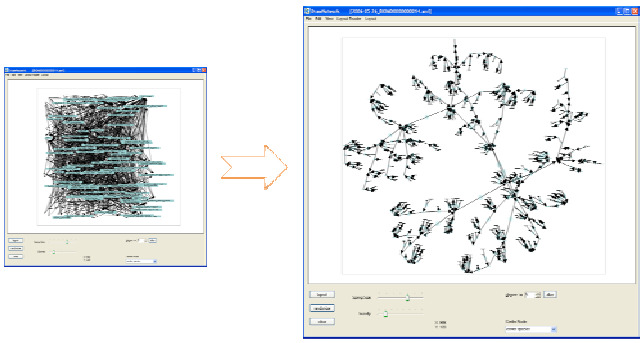

Figure 2: Autolayout of BioModel #14 (From Model Repository at www.biomodels.net)

Translators: Finally, SBW features a number of SBML translator modules that transform SBML into XPP-, Matlab-, Simulink-, Java-, Jarnac-files or others. This makes it easy to transfer models and to analyze them on these programs.

4.2 SBW Simulators

The dynamic behavior of biochemical networks is one of the main interests of researchers in systems biology. This has resulted in a multitude of simulators becoming available. While in some cases the need exists to devise specialized simulators for certain areas, in most cases it will be enough to consider existing simulators. With SBW this integration is performed simply. Hence, there is a large variety of SBW-enabled simulators available, as given below.

Jarnac: Jarnac is more than just a simulator. It is a complete scripting environment combining stochastic and ODE simulation capabilities with a powerful control language. It allows for modeling in a shorthand notation as well as analyzing the models and graphing the results. For ODE simulations, Jarnac uses the popular integrators CVODE (Cohen and Hindmarch 1996) or LSODA (Hindmarch 1983) which can be individually selected by the user. Stochastic simulations are performed using an implementation of the Gillespie algorithm (Gillespie 1976). The shorthand language of Jarnac can easily be converted into SBML and vice versa. It should be noted that some features of SBML Version 2 have not been implemented, such as events or delay equations. However, these features can be implemented with functions written in Jarnac-Script. Acting as an SBW module, Jarnac provides its simulation capabilities or analysis routines to any other SBW module. Probably the greatest shortcoming of Jarnac is that it was written for WIN32 systems only. To make Jarnac's functionality available on other systems, we devised a new SBW module called *JarnacLite*. This module has been written for the sole purpose of quickly transforming SBML into Jarnac-Script, allowing for modifications in the useful shorthand and then letting other modules analyze the model by transforming it back into SBML. As JarnacLite was written using the .NET language C#, it runs under WIN32 as well as all operating systems supported by Mono.

roadRunner: In an effort to create a fully SBML compliant simulator we next focused on creating roadRunner. Instead of interpreting model equations, roadRunner will compile the model equations dynamically, which results in much improved performance when compared with traditional simulators. RoadRunner uses the integrator CVODE and NLEQ (ZIB 2003) for steady state analysis. To further speed up the simulation, the model is separated into a system of independent and dependent variables. This separation process is described in detail in (Vallabhajosyula et al. 2006). Again, roadRunner has been written completely in C#. Provided that CVODE and NLEQ are available for a given operating system, roadRunner will run on that operating system. Thus, all major operating systems are supported.

Dizzy: A further collection of simulators has been developed by Stephen Ramsey at the Institute for Systems Biology (Ramsey 2006). Dizzy allows for stochastic simulations using Gillespie, Gibson-Bruck, or Tau-Leap algorithms. Dizzy can be called via the SBW menu in model editors such as JDesigner or CellDesigner.

Oscill8: Developed by Emery Conrad (Conrad 2006), Oscill8 is used for bifurcation analysis and time-course simulations. Oscill8 provides a user-friendly interface around AUTO (Doedel 1981), allowing for one and two parameter bifurcation diagrams (Figure 3) or bifurcation searches. For SBW, it provides a simulation service and additionally is available through the SBW menu to perform bifurcation analysis. Oscill8 is available for WIN32, Linux, and OS X. The front-end requires a .NET implementation such as the .NET Runtime or Mono.

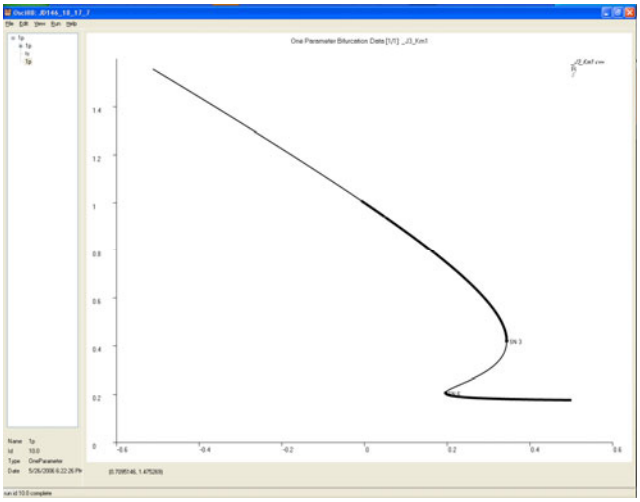

Figure 3: One Parameter Bifurcation-diagram Generated by Oscill8

Stochastic Simulators: Finally, we also include SBW simulation services that implement the Gillespie Algorithm (Direct Method and First Reaction Method), Gibson-Bruck Algorithm (Gibson and Bruck 2000), and Chemical Langevin Equation (Gillespie, 2000) for operating systems WIN32, Linux, and OS X.

SimDriver: To implement a unified user-interface to interact with all these simulators, we wrote the *SimDriver*. There are several implementations of this module available. For non-WIN32 operating systems, we use a Java implementation. On WIN32 systems, we use a .NET version of the SimDriver (Figure 4) with additional functionality such as steady state analysis, continuous time-course simulations, and modifications to parameter values. The SimDriver works for all simulators implementing one of the SBW simulator API's (Sauro and Bergmann 2006). There are several levels of this API with an increasing number of features. The user interface will disable all controls not supported by the selected simulator. The SimDriver also supports stochastic simulators and therefore provides additional facilities to aid the analysis of stochastic models, in particular, probability density function estimation, frequency analysis, and noise injection at particular points in the model.

3D Visualization Tool: Traditionally the simulation results are available as either data tables or X-Y plots. Data tables are helpful for further processing by other computational tools. X-Y plots, on the other hand, tend to get complex even for a limited number of species. In creating a new visualization tool, we had two goals in mind. The first

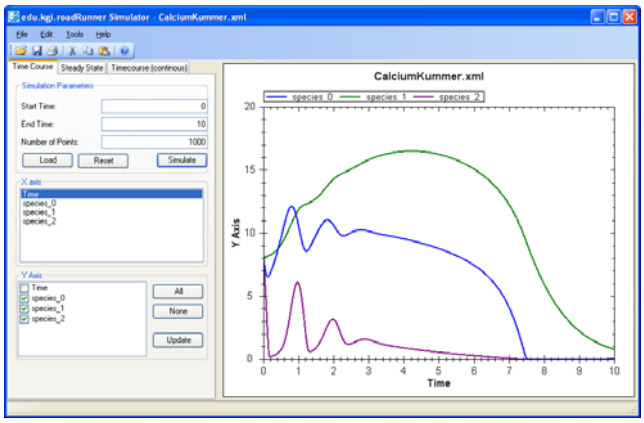

Figure 4: Time-course Simulation Using the SimDriver

goal was to strongly tie the simulation results to the model. The other goal was to be able to view the simulation in real time to refine or broaden it where necessary. These goals have been realized in form of a *3D Time-course Visualization* module. The layout of an SBML model, as obtained by the SBMLLayoutModule, or generated by the auto layout module, implements the basis of the 3D visualization tool and is projected onto a 3D plane. Furthermore, all positions of species are recognized and rendered as columns on top of the 3D plane. The height of the columns, representing the current species concentration, will vary during the time-course simulation (Figure 5). This way, interesting dynamic behaviors of the models can easily be seen. Since the time course simulation will be performed continuously, it is possible to dynamically change time-steps to focus on certain aspects of the model.

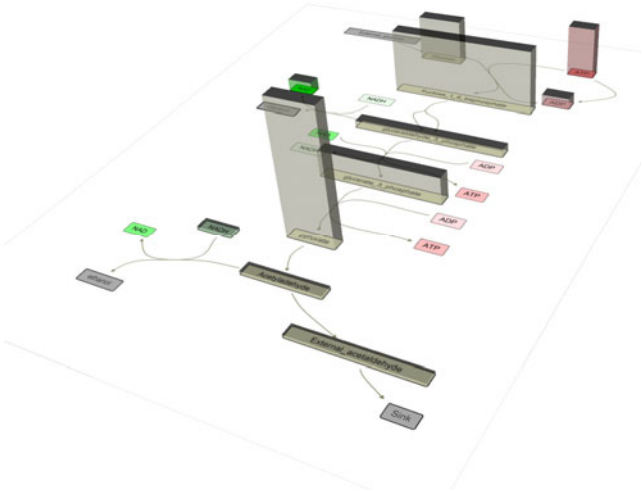

Figure 5: 3D Time-course Simulation of Jana Wolf's Glycolysis Model (Ruoff et al. 2003)

4.3 SBW Analysis Tools

The *Structural Analysis Module* is a graphical user interface that allows the user to analyze conservation laws of an SBML model. The tool uses SBW to identify the conservation cycles in a model as described in (Vallabhajosyula et al. 2006). This allows separating independent from dependent variables, which is important in order to compute a non-singular Jacobian matrix. The Jacobian matrix represents the basis for further analysis on the model. The Structural Analysis Module is available through the SBW menu.

Metatool, by Stefan Schuster (Pfeiffer et al. 1999), is a command line tool for the calculation of conservation vectors, elementary modes, and the null-space and stoichiometry matrix. This tool has been wrapped up as an SBW menu. This module has been tightly integrated into JDesigner in order to highlight the modes in the graphical model.

The *Bifurcation Discovery Tool* is an SBW module that will perform a parameter scan of the given SBML model to search for interesting behavior such as switching and oscillation. The Bifurcation Discovery Tool is available through the SBW menu or supports loading SBML files directly. It has no simulation capabilities on its own but will use one of the SBW simulators to perform time course simulations and steady state analysis in order to perform the parameter scan. In principle, it implements a genetic algorithm with elitism, which evaluates slightly modified models in the parameter space. A detailed description is available in (Chickarmane et al. 2005).

The *Frequency Analysis Module* allows the user to select a parameter and a species of an SBML model and generates the corresponding frequency response. It is available through the SBW menu.

We also developed a suite of optimizers within the BioSPICE project. These optimizers fit experimental data to an SBML model. Five different algorithms were implemented for this purpose: a genetic algorithm with tournament selection, a hybrid algorithm combining a genetic algorithm with the simplex algorithm, the Levenberg-Marquardt algorithm, a hybrid combining simulated annealing and simplex, and finally the Nelder and Mead Simplex Algorithm. These algorithms have been implemented in Matlab and are accessible via the SBW-Matlab Bridge (Wellock, Chickarmane, and Sauro 2005), which allows Matlab scripts to be wrapped into SBW modules.

4.4 SBW Modeling Environments

JDesigner is a graphical modeling environment for biochemical reaction networks. It allows drawing the network on screen and selecting the appropriate kinetic laws from a wide selection of available rate laws or to define new rate laws. The current version of JDesigner has been redesigned to take advantage of modern rendering technologies.

JDesigner is the module that probably profits the most from SBW. Via the previously described auto-layout and SBML layout extension modules, it is able to render any SBML file. Using the SBW menu, it can pass the loaded model to a wide variety of analysis tools. Time-course simulation and steady state analysis can be performed by either roadRunner or Jarnac. Furthermore, it uses Metatool for structural analysis. Finally, it dynamically finds modules from the SBML exporter category at runtime so the model can be exported into a variety of file formats including Matlab, Java, or XPP.

Figure 6 demonstrates the analysis capabilities of JDesigner. Via the Metatool integration, one of the elementary modes of the model are highlighted. The pane on the bottom shows the time-course behavior of the model. The drawing area in the middle displays a generated layout for the model.

CellDesigner, a graphical modeling environment developed by the Systems Biology Institute, Tokyo, Japan, also features the SBW menu. While JDesigner uses the model of hyper-graphs to visualize the biochemical network, CellDesigner uses the Process Diagram (Kitano et al. 2005) notation (i.e., a state diagram). CellDesigner has been written in Java and is available for most operating systems.

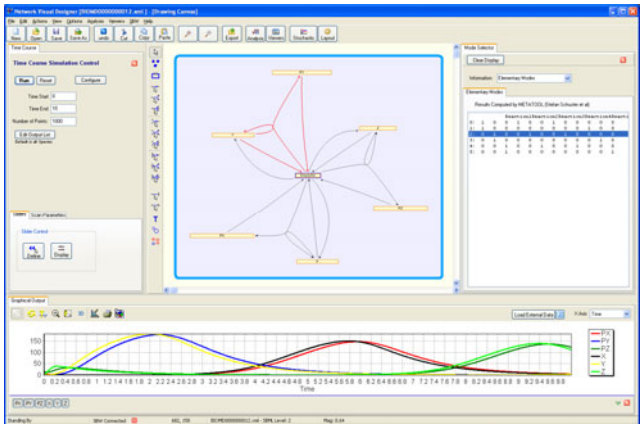

Figure 6: JDesigner Analyzing BioModel #12 (From Model Repository at www.biomodels.net)

5 FUTURE DIRECTIONS

So far, SBW development was primarily WIN32-centric. While the binding libraries were available for most operating systems, all of our modules were mainly developed for WIN32. As SBW nears its third version, we face new possibilities. Finally, open source alternatives to the .NET runtime (Mono-Project 2006) are stable enough for daily use. We will therefore dedicate more time to make SBW more useful on non-WIN32 systems.

Additions to the SBW core will include more metadata to allow SBW modules to negotiate their data types in a more sophisticated way. This will resolve ambiguity of data types such as strings, which could represent a filename, an SBML model, or any other string. This will also allow for more interesting user interfaces.

We will also spend more time with the development of additional categories, possibly even with an enforcement policy that ensures that the methods of the category are implemented. Categories proved immensely effective as we have seen with the SBW menu and the SBML Exporter category.

6 SUMMARY

SBW represents a flexible framework allowing the integration of software components in a language-neutral way. The core framework is available for many operating systems, among them WIN32, Linux, and OS X. Binding libraries are available for the most common programming languages: C/C++, the .NET languages, Delphi, Java, Python, and Matlab. Other programming languages can be supported by calling the C bindings.

A growing collection of SBW-enabled applications are available for analyzing, modeling, simulating, and visualization. SBW is stable—open source and free downloads are available from the project website at <http://sys-bio.org>.

ACKNOWLEDGMENTS

The initial work has been funded by the Japan Science and Technology Corporation under the ERATO Kitano Systems Biology Project. Funded through the generous support of ERATO, DARPA (contract number MIPR 03-M296-01) and the DOE (under Grand No. DE-FG02-04ER63804, "Computational Resources for GTL"). Original Program Investigators: Hiroaki Kitano, John Doyle, in collaboration with Hamid Bolouri, Andrew Finney, and Mike Hucka.

We wish to acknowledge in particular the authors of BioSPICE, CellDesigner, COPASI, Dizzy, Oscill8, and Virtual Cell, as well as the SBML community for their support. We also thank Anastasia Deckard, Klaus Maier, Sri Rama Krishna Paladugu, and Ravishankar Rao Vallabhajosyula. Finally we would like to thank our user-base for their support and feedback.

REFERENCES

Bergmann, Frank T. 2006. SBML Layout Viewer. <http://sys-bio.org/Layout/> [accessed May 26, 2006].

Bornstein, B. J., Sarah Keating, and Andrew Finney. 2006. libSBML. <http://sbml.org/software/libsbml/> [accessed May 26, 2006].

Chickarmane, Vijay, Sri Rama Krishna Paladugu, Frank T. Bergmann, and Herbert M Sauro. 2005. Bifurcation discovery tool. *Bioinformatics* 21.18: 3688-3690.

Cohen, S. D., and A. C. Hindmarch. 1996. CVODE, a stiff/nonstiff ODE solver in C. *Computers in Physics* 10.2: 138-143.

Conrad, Emery. 2006. *Oscill8.* <http://oscill8.sourceforge.net> [accessed May 26, 2006].

Doedel, E. J. 1981. AUTO: a program for the automatic bifurcation analysis of autonomous systems. *Proceedings of the 10th Manitoba Conference on Numerical Mathematics and Computing,* 265-284.

Finney, Andrew, and Michael Hucka. 2003. Systems biology markup language: level 2 and beyond. *Biochemical Society Transactions* 31.6 (2003): 1472-1473.

Fruchterman, T. M. J., and E. M. Reingold. 1991. Graph drawing by force-directed placement. *Software - Practice and Experience* 21.11: 1129-1164.

Garvey, Thomas D., Patrick Lincoln, Charles John Pedersen, David Martin, and Mark Johnson. 2003. BioSPICE: access to the most current computational tools for biologists. *OMICS* 7.4: 411-420.

Gauges, Ralph, Ursula Rost, Sven Sahle, and Katja Wegner. 2006. A model diagram layout extension for SBML. *Bioinformatics* (to appear).

Gibson, M. A. and J. Bruck. 1999. Efficient exact stochastic simulation of chemical systems with many species and many channels. *Journal of Physical Chemistry A* 104:1876-1889.

Gillespie, D. T. 1976. A general method for numerically simulating the stochastic time evolution of coupled chemical species. *Journal of Computational Physics* 22: 403-434.

Gillespie, D. T. 2000. The chemical Langevin equation. *The Journal of Chemical Physics* 113.1: 297-306.

Hindmarch, A. C. 1983. ODEPACK: a systematized collection of ODE solvers. *Scientific Computing*.

Hucka, Michael, Andrew Finney, Herbert M. Sauro, Hamid Bolouri, John C .Doyle, and Hiroaki Kitano. 2003. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* 19.4: 524-531.

Hucka, Michael, Andrew Finney, Herbert M. Sauro, John C. Doyle, and Hiroaki Kitano. 2002. The ERATO Systems Biology Workbench: enabling interaction and exchange between software tools for computational biology. *Pacific Symposium on Biocomputing.* 450-261.

Kitano, Hiroaki. 2005. International alliances for quantitative modeling in systems biology. *Molecular Systems Biology.*

Kitano, Hiroaki, Akira Funahashi, Yuhiko Matsuoka, and Kanae Oda. 2005. Using process diagrams for the graphical representation of biochemical networks. *Nature Biotechnology* 23.8: 961-966.

Kolpakov, F. A. 2002. BioUML - framework for visual modeling and simulation of biological systems. *Proceedings of the International Conference on Bioinformatics of Genome Regulation and Structure*.

Le Novere, Nicholas, et al. 2005. BioModels Database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research* 34: 689-691.

Mathworks, The. 2006. MATLAB and Simulink for Technical Computing. <http://www mathworks.com/> [accessed May 26, 2006].

Microsoft. 2006a. Internet Information Services. <http://www.microsoft.com/WindowsServer2003/iis/default.mspx> [accessed May 26, 2006].

Microsoft. 2006b. Visual Studio 2005. <http://msdn.microsoft.com/vstudio/> [accessed May 26, 2006].

Mono-Project. 2005. ASP.NET-Mono. <http://www.mono-project.com/ASP.NET> [accessed May 26, 2006].

Netlib. 2006. LAPACK - Linear Algebra PACKage. <http://www.netlib.org/lapack/> [accessed May 26, 2006].

Olivier, Brett G., Johann M. Rohwer, and Jan-Hendrik S. Hofmeyr. 2005. Modelling cellular systems with PySCeS. *Bioinformatics* 21.4: 560-561.

OMG. The OMG's CORBA website. 2006. <http://www.omg.org/corba/> [accessed May 26, 2006].

Pfeiffer, T., I. Sanchey-Valdenebro, J. C. Nuno, F. Montero, and S. Schuster. 1999. METATOOL: for studying metabolic networks. *Bioinformatics* 15.3: 251-257.

Ramsey, S., D. Orrell, and H. Bolouri. 2005. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *Journal of Bioinformatics and Computational Biology* 3.2: 415-36.

Ruoff P., M. K. Christensen, J. Wolf, and R. Heinrich. 2003. Temperature dependency and temperature compensation in a model of yeast glycolytic oscillations. *Biophysical Chemistry* 106.2: 179-192.

Sauro, Herbert M., and Frank T. Bergmann. 2006. SBW simulation API. <http://sbw.kgi.edu/downloads/SimAPI.pdf> [accessed May 26, 2006].

Sauro, Herbert M., et al. 2003. Next Generation Simulation Tools: the Systems Biology Workbench and BioSPICE integration. *OMICS* 7.4: 35-372.

Sun Microsystems, Inc. 2006. Java RMI over IIOP. <http://java.sun.com/products/rmi-iiop/> [accessed May 26, 2006].

Vallabhajosyula, Ravishankar Rao, Vijay Chickarmane, and Herbert M. Sauro. 2006. Conservation analysis of large biochemical networks. *Bioinformatics* 22.3: 346-353.

W3C. 2006. Web Services. <http://www.w3.org/2002/ws/> [accessed May 26, 2006].

Wellock, Cameron, Vijay Chickarmane, and Herbert M. Sauro. 2005. The SBW-MATLAB interface. *Bioinformatics* 21.6: 823-824.

ZIB (Zuse Institute Berlin). 2003. Affin-invariant Newton Techniques (ANT) - NLEQ2. <http://www.zib.de/Numerik/numsoft/ANT/nleq2.en.html> [accessed May 26, 2006].

AUTHOR BIOGRAPHIES

FRANK BERGMANN is currently a PhD student under the supervision of Herbert Sauro at the Keck Graduate Institute. He received his first degree in computer science from the Johann Wolfgang Goethe University Frankfurt, Germany. For his diploma he specialized in computer graphics and carried out his senior thesis on visualization of reaction-diffusion systems in biology. He is the lead developer for the Systems Biology Workbench and his PhD is concerned with the development of tools and applications of computer science to Systems Biology. His e-mail address is <fbergman@kgi.edu>, and his web page is <http://public.kgi.edu/~fbergman>.

HERBERT SAURO was originally educated as a biochemist/microbiologist but became interested in the use of simulation and theory to understand cellular networks after accidentally coming across a paper by David Garkfinkel on the simulation of glycolysis. He wrote one of the first biochemical simulators for the PC (SCAMP) in the 1980s to assist work on extending metabolic control analysis (a theory closely related to biochemical systems theory). However, with the lack of community interest in systems biology during the late 80s and early 90s, he left science to start a software company and offer consultancy work to finance firms in the UK. With the surge in interest in systems biology in the US in the late 90s, he secured a position at Caltech to assist in the development of the Systems Biology Markup Language. Since then he moved to a faculty position at the Keck Graduate Institute where he continues to do research on network motifs, theory, and software. His e-mail address is <hsauro@kgi.edu>.