Bioengineered Bugs 1:4, 274-281; July/August 2010 © 2010 Landes Bioscience

Computer-aided design of biological circuits using TinkerCell

Deepak Chandran,[1,*] Frank T. Bergmann[1,2] and Herbert M. Sauro[1]

[1]Department of Bioengineering; University of Washington; Seattle, WA USA; [2]Keck Graduate Institute; Claremont, CA USA

Key words: synthetic biology, modeling, software, CAD, simulation, systems biology, design, standards, computational

Synthetic biology is an engineering discipline that builds on modeling practices from systems biology and wet-lab techniques from genetic engineering. As synthetic biology advances, efficient procedures will be developed that will allow a synthetic biologist to design, analyze and build biological networks. In this idealized pipeline, computer-aided design (CAD) is a necessary component. The role of a CAD application would be to allow efficient transition from a general design to a final product. TinkerCell is a design tool for serving this purpose in synthetic biology. In TinkerCell, users build biological networks using biological parts and modules. The network can be analyzed using one of several functions provided by TinkerCell or custom programs from third-party sources. Since best practices for modeling and constructing synthetic biology networks have not yet been established, TinkerCell is designed as a flexible and extensible application that can adjust itself to changes in the field.

Introduction

Synthetic biology can be described as an amalgam of wet-lab techniques from genetic engineering, modeling techniques from systems biology and design concepts from electrical and control engineering.[1,2] The goal of synthetic biology is the ability to build biological "circuits" or networks made using individual components such as genes and promoter regions, that produce a desired behavior. The process by which this goal should be attained is unclear, and therefore the precise definition of synthetic biology is also unclear.[3] However, the success of synthetic biology depends on a few key ingredients: efficient design process, standardized engineering protocols,[4] and some form of modularity[5] allowing one engineer to build on another's work. The potential of such a technology is immense for understanding fundamental science or solving real-world problems. By constructing oscillators,[6-8] bistable switches,[9,10] noise controlling networks[11] and synchronizing circuits,[12] synthetic biologists are able to bring theory of biological systems into practice. For the purpose of real-world application, synthetic biology may become a key player in bioremediation,[13] drug production[14,15] and bio-fuel production.[16]

While synthetic biology has great potential, it is important to understand the present limitations. There are unresolved issues at several levels: wet-lab protocols, exchange of information and computational modeling. Synthetic biology relies on the hypothesis that biological parts or functional elements encoded as DNA, can be assembled in order to build circuits with predictable behavior.[17] Wet-lab protocols for assembling biological parts are time-consuming and labor intensive, especially when building large circuits. For the purpose of sharing, there are no standard protocols for exchanging complete parts between research labs, including the DNA sequence and information about the part's function. For computational analysis, there is no clear consensus on what types of models are best suited for reliably predicting biological circuit behavior, especially in the presence of so many unknown parameters in an average synthetic circuit. Nonetheless, it is equally important to realize that significant progress has been made to address each of the above issues. Standardized assembly has made exchange of parts between labs easier.[18,19] Design strategies have been explored for controlling uncertainty due to noise[20] and parameters.[21] Modeling methods have been demonstrated to be predictive for steady state behaviors of cells,[22,23] and the importance of intermediate stages, such as mRNA and protein folding, have been shown to capture the dynamics of a circuit.[7]

Although progress has been made, the future of synthetic biology is unclear. From one perspective, it can be argued that the idea of engineering biology is simply an impossible task due to overwhelming complexity. On that other hand, it is also arguable that scientists will be able to discover ways of controlling some subset of all possible biological systems, turning biology into a reliable technology. In this later optimistic future, research labs would make frequent use of biological parts databases, as it is already being done to a certain extent using the Registry of Biological Parts (http://partsregistry.org).[24] The databases would contain structured information with supporting experimental and kinetic data. It is important to adopt this optimistic point of view in order to justify why computer-aided design (CAD) will be helpful in synthetic biology.

Different approaches have been taken for constructing CAD programs for synthetic biology. Existing CAD applications for

*Correspondence to: Deepak Chandran; Email: deepakc@u.washington.edu
Submitted: 05/02/10; Revised: 05/25/10; Accepted: 05/28/10
Previously published online: www.landesbioscience.com/journals/biobugs/article/12506

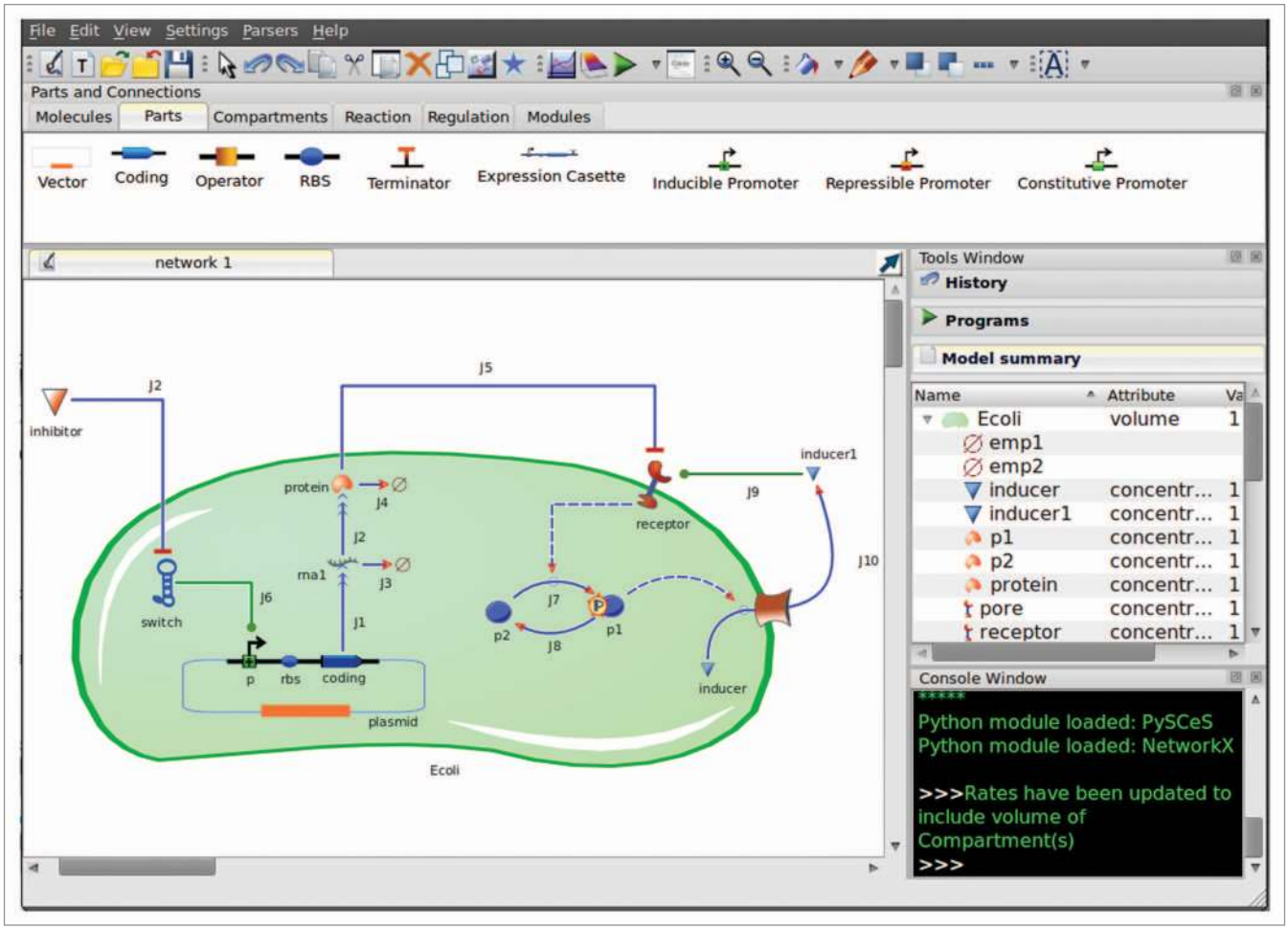

Figure 1. A screenshot of TinkerCell, showing a simple model of lactose import. The bacterial cell in the model contains a plasmid with a promoter, RBS and a coding region. The protein produced from the coding region is the membrane protein that is responsible for importing lactose, which in turn inhibits the transcription factor, LacI. LacI negatively regulates the promoter on the plasmid.

synthetic biology include BioJade,[25] GenoCAD,[26] SynBioSS,[27] ProMoT,[28] Clotho[29] and TinkerCell.[30] Each of these CAD applications are unique in their own ways and have their own respective focus areas. The focus of this article is TinkerCell, which is an application for visually constructing biological networks and analyzing its dynamics. A screenshot of TinkerCell is shown in **Figure 1**.

Motivation for TinkerCell

TinkerCell is a CAD software application for visually constructing and analyzing biological models or circuits. Unlike classical engineering disciplines, there are no established best practices in synthetic biology for taking a circuit from the design stage to the construction stage. In this situation, the goal of TinkerCell is to serve as an application that can adapt with the continuously evolving field of synthetic biology. The following hypothetical use cases were used to guide the design TinkerCell:

A model needs to define its mathematics in context of the underlying biology. A researcher constructs a model using some biological parts, such as a promoter, a ribosomal binding site and a protein coding sequence. The researcher wishes to study the effect of changing the promoter on the system. Changing the promoter implies changing one or more parameters in the model, depending on how the model is constructed. To satisfy this requirement, the CAD application needs to identify the parameter(s) that belong with each part in the model. In the anticipated future, there will be some sort of database of biological parts that the researcher would access. If the researcher selects a specific part from the database, the CAD application should be able to incorporate the parameters from the database into the model. Other information available from the database, such as equations that describe the dynamics of the part and the DNA sequence, should also be incorporated into the model as well.

Enforcing one type of modeling methodology is not a good idea for a developing field. Repeated experience with modeling and experiments might lead researchers to recognize that a specific modeling method is best suited for certain types of synthetic circuits. The CAD application should be able to incorporate new modeling methods without having to remove its existing modeling framework. For this purpose, the CAD application should allow different ways of defining the dynamics of the model.

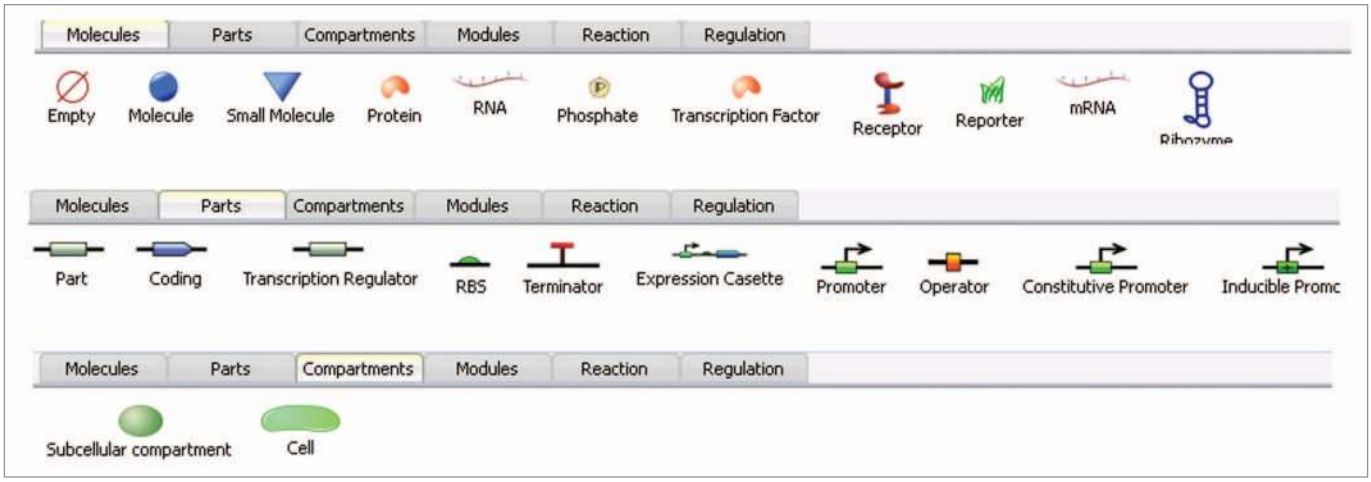

Figure 2. TinkerCell uses a catalog of biological components for constructing models. Each component in the catalog belongs to an ontology, which is transparent to the user. Models that are built using components in the catalog will contain the mathematical descriptions as well as the biological descriptions.

The ability to integrate new programs is beneficial for a developing field. A researcher has constructed a model with existing parts from a database. The researcher is interested in checking to see which parts in the circuit contain a particular restriction site. This is a specific function that the CAD application may not provide by default. However, if a program exists for performing this analysis, the CAD application should allow the researcher to add the new program. The CAD application should allow full integration of this new program. For example, as an output, the new program should be able to request the CAD application to visually highlighting the parts that contain the restriction sites of interest. This will allow the researcher to construct models in the CAD program, analyze general aspects of the model, and use custom programs to analyze specific aspects of the model.

The ability to share new programs is beneficial for building a community. Continuing from the previous example, it might be the case that a custom program used by one researcher is a valuable tool for other researchers in the field. Allowing a user to share custom programs will not only enhance the CAD applications functionality but also foster community development. Therefore, a CAD application should provide a simple procedure for each user to share and retrieve custom programs.

The ability to reuse models is beneficial for engineering. A research lab has built several small synthetic circuits and has relatively good models for each. A researcher at the lab wishes to connect some of these small circuits to construct larger circuits. The CAD application should support such reuse of circuits by allowing users to construct modules and connect modules to build larger circuits. The user interface should allow an option for hiding the internal details of the modules, providing the user with a compact view of the larger circuit in terms of the smaller modules.

Models should account for uncertainties that exist in real systems. For biological parts, exact quantitative values are rarely available. For some parameters, even rough estimates may not be available. For a model to reflect the reality, the uncertainty related to parameters in a model should be taken into account. The CAD program should allow parameters to be defined as a range of values or a distribution of values rather than a single exact number and provide methods for analyzing the model by taking the uncertainties into consideration.

TinkerCell's underlying software structure is designed with these use cases in mind. The current version of TinkerCell does not fully satisfy all of these use cases because some of the features are still under development. For example, TinkerCell currently allows a user to add custom programs that interact with TinkerCell's visual interface, but the option for sharing custom programs between users does not exist. Similarly, models can be constructed my composing existing models, but no feature is available for sharing models between users. The visual diagrams are converted to mathematical models by TinkerCell *extensions* (discussed in the TinkerCell design section), so TinkerCell does not enforce one type of modeling approach. Finally, while parameter uncertainties can be stored in TinkerCell models, there are no functions available for using this information at present. However, TinkerCell's underlying design will be able to support all the missing features.

TinkerCell's Design

Component based modeling. The first design feature in TinkerCell is a well structured model. Users build TinkerCell models by selecting and connecting components from the parts catalog that is displayed at the top of the TinkerCell window (see **Fig. 2**). Components in this catalog include proteins, small molecules, cells, promoters and coding regions. The list of components is flexible because the list is loaded from an Extensible Markup Language (XML) file. The XML file represents a structured definition of each component. For example, the file describes "transcriptional repression" as a connection from a "transcription factor" to a "repressible promoter". Similarly, a "transcription factor" is described as a special case of a "protein". This file captures what is called ontology or a structured description of

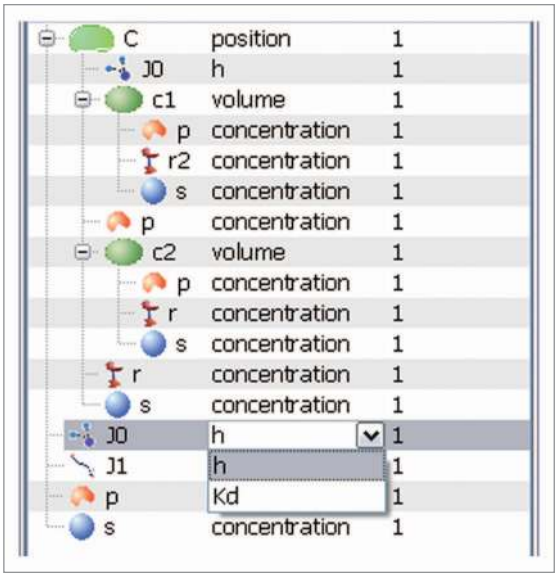

Figure 3. The model summary window is an interface that allows the user to view and edit any of the parameters in the model. The window shows the parameters according to the component that they belong with, e.g., a promoter's strength parameter.

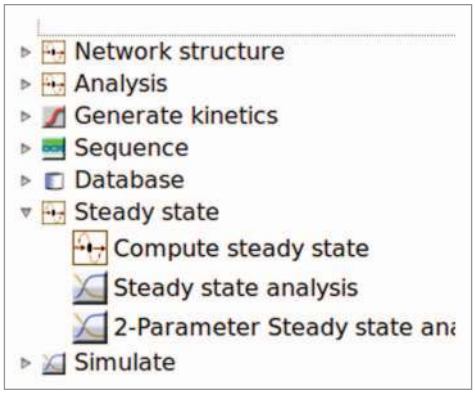

Figure 4. This figure shows the control coefficients of different fluxes in an incoherent feed-forward network on the steady state value of the second protein (p2 in the Figure). The control coefficients are computed using PySCeS. The output from PySCeS is displayed visually: the reaction arcs are colored green for positive control coefficients and red for negative and the line widths are also adjusted according to the control coefficients. This example illustrates how Python scripts in TinkerCell can produce visual outputs.

relationships between concepts. The particular XML file in use currently is meant to be temporary, and it will be replaced with a standard ontology in the future.

Default models derived from structure. The purpose of the ontology becomes clear as the user starts to build a model using components from the catalog. By using definitions of the components, TinkerCell is able to derive much of the dynamics automatically. For example, when a user connects a promoter, a ribosomal binding site and a protein coding region together, TinkerCell is able to identify the fact that all of these are DNA components, and therefore, the relative locations of the components are relevant to their functions. By using the ordering of the components, TinkerCell is automatically able to assign rate equations describing the dynamics of the transcription and translation reactions. This example shows knowledge of the biology can automate model construction. Of course, the user can modify any of the default equations if needed.

Flexibility provided through extensions. The second design feature in TinkerCell is extensibility. Much of the work in TinkerCell is done by *extensions*. Extensions are programs that can be added to TinkerCell without altering the existing program. In the earlier, when the user places DNA components together, the transcription rate equations are automatically derived by a TinkerCell extension. Because extensions are optional features, they can be removed or replaced. For instance, if the default reaction rate equations derived by the current extension are not preferred, it is possible to write a different extension that provides a different way of defining the dynamics, e.g., Boolean logic. The role of extensions in TinkerCell fits nicely with the fact that TinkerCell is an open-source project. In an ideal scenario, different extensions would be available, each providing a different type of default modeling method. Users will be able to choose the extension that fits their interests. Several user interface features are also provided by extensions. For example, the model summary window shown in **Figure 3** and also in **Figure 1** is a user interface extension. Although the current set of extensions are all written by the original designer of TinkerCell, they could have easily been written by other programmers who wanted to contribute to TinkerCell.

Expandable set of functions. The third design feature is support for third-party functions. When the user is finished constructing the model, the model can be analyzed using one of many functions listed in the programs menu (see **Fig. 4**). These functions are not built-in functions. Rather, they are loaded from a folder containing programs written in C and Python programming languages. TinkerCell provides a extensive application programming interface (API) with over two hundred functions that are callable from C and Python. These functions allow third-party programs to get information about the model from TinkerCell, analyze the model, and report the results back to TinkerCell. Because of the rich API, the results can be presented visually. **Figure 5** shows an example output produced from a Python program in TinkerCell that uses PySCeS[31] to perform sensitivity analysis. The results are presented by coloring the reactions in the network according to the control coefficients. TinkerCell automatically loads Python scripts and C programs from designated C and Python folders, allowing easy integration of third-party code.

TinkerCell's flexibility is due to its layered architecture (see **Fig. 6**). TinkerCell consists of a Core library, which provides all the basic drawing capabilities. The TinkerCell extensions are C++ programs that build on the Core library. C and Python extensions build on the Core library as well as the C++ extensions. Major changes to TinkerCell can be made through C++ extensions. New functions can be added to TinkerCell via C or Python extensions.

Future plans. There are two features in TinkerCell that can potentially make it valuable as a medium through which the

synthetic biology community can share models and analysis programs. The first feature is the ability to construct and connect modules. Individual modules can be described using a TinkerCell model. In future, modules will also be able to encapsulate models represented using Antimony scripts[32] or the Systems Biology Markup Language.[33] A feature that is under development is the ability to encapsulate the internal details of a module, similar to the way electrical circuit diagrams hide the internal details of common components such as amplifiers. One can imagine a similar interface for biological circuits where individual modules might represent feed-forward networks or bistable switches. These modules might represent circuits that have been built by different research labs. TinkerCell will allow users to upload and download modules, which would permit different labs to use each other's works to create new circuits. **Figure** 7 shows a screenshot of the interface that is presently being developed to enable this functionality.

The second feature is TinkerCell's ability to incorporate new functions written in Python and C programming languages. Other programming languages such as Octave, R, Ruby and Perl will be added in future. If users are able to upload and download programs from a central repository, the sharing of useful programs for analysis of synthetic biology circuits may be greatly enhanced by TinkerCell. It is important to note that all of the programs that can be incorporated into TinkerCell are not dependent on TinkerCell; for instance, a Python program that is incorporated in TinkerCell can still be used as a separate program outside TinkerCell. The intent of TinkerCell is simply to provide an interface to existing code. One side project that might be required for this sort of sharing of programs will be a repository of code that perform small functions, ranging from numerical analysis to sequence analysis. This repository of code will be independent of TinkerCell, but TinkerCell can serve as a channel through which the repository can be accessed. Often, a third-party program will be difficult to use because the inputs and outputs might have not be clearly documented. When code is added to TinkerCell, the programmer has the option of adding several user interface features without extensive code writing. This can add to the ease of code sharing.

One of the greatest challenges of modeling biological systems is the large number of unknown parameters and missing information describing possibly important details of biological processes. Depending on the model, many of the details may or may not be needed. Nonetheless, future modeling techniques in synthetic biology would probably give much consideration to the uncertainty related to each parameter. TinkerCell's design allows uncertainties to be specified along with the parameters of the model. Managing uncertainty in synthetic biology is an active area of research,[34-37] and there are plans to add some form of uncertainty analysis to TinkerCell.

Standards and exchange formats. TinkerCell is part of a greater plan for facilitating exchange of synthetic biology parts. The Synthetic Biology Open Language (http://openwetware.org/wiki/The_BioBricks_Foundation:Standards/Technical/Exchange), abbreviated as SBOL, is a collaborative endeavor for establishing standards for exchanging synthetic biology circuits.

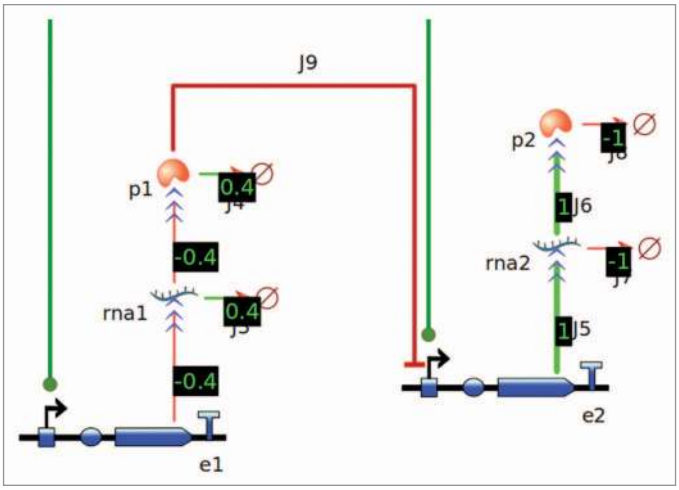

Figure 5. TinkerCell integrates third-party functions written in Python and C with its user interface. Python programs and C programs are loaded from designated folders and made available as buttons in TinkerCell, as shown in this figure. In future, TinkerCell will also support programs written in other languages such as Ruby, R and Perl.

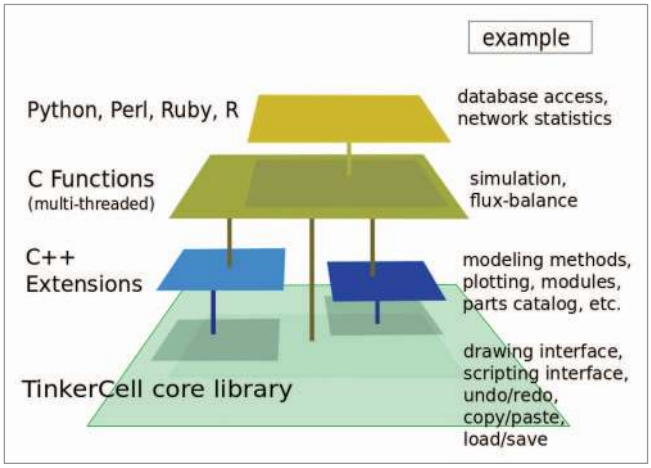

Figure 6. TinkerCell is extensible at different layers. The bottom-most layer is a Core library that provides all the basic drawing functions. The C++ extensions form a second-layer. These extensions provide the modeling framework and various use interface features. A C programmer interface is built on the C++ extensions and the Core library, providing over two hundred functions that can be used to add new C extensions. Each C function is extended to higher level languages such as Python, allowing Python extensions. The right-hand side of the figure lists some example features that are provided by each layer.

SBOL aims to establish standards for visual representation, text-based representations and a comprehensive semantic representation of biological parts and circuits. TinkerCell already supports much of the SBOL visual standard (see **Fig. 2**). Integrating SBOL semantic with TinkerCell is a long-term goal. When this goal is achieved, users will be able to use TinkerCell as a front-end for querying biological parts repositories, biological models and analysis functions. Due to the detailed descriptions provided by SBOL semantic, it would be possible for TinkerCell to automatically link concepts in mathematical models with biological parts.

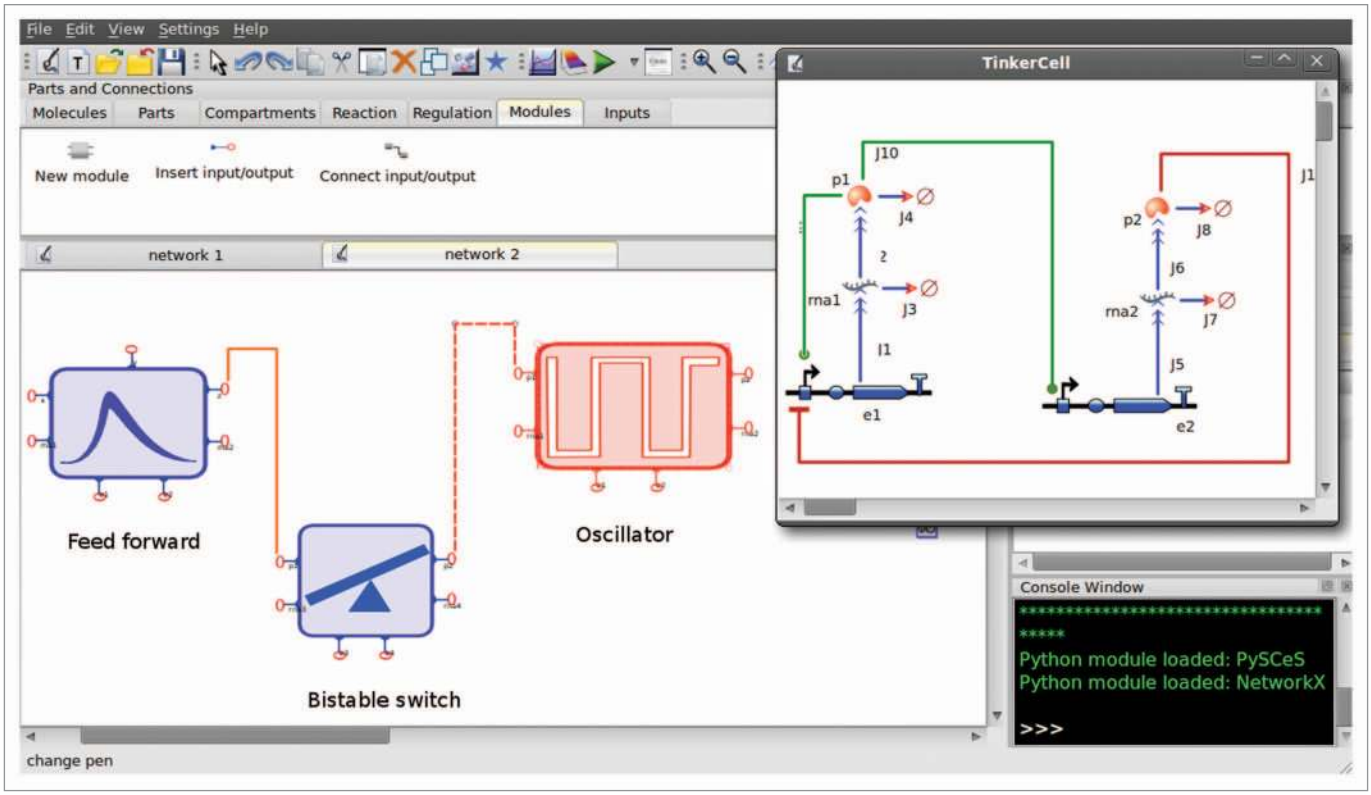

Figure 7. This figure shows three modules connected to form a larger circuit. The internal details of each module are hidden from view to provide the user with a concise view diagram, which can often provide a clearer conceptual understanding of the circuit. The internal details of each module still can be viewed and changed in a separate window, as shown at the right-hand side of the figure. One of the future plans of TinkerCell is to allow users to upload and download modules from a central repository. When that feature is complete, this interface can be used to construct circuits using other researchers' modules.

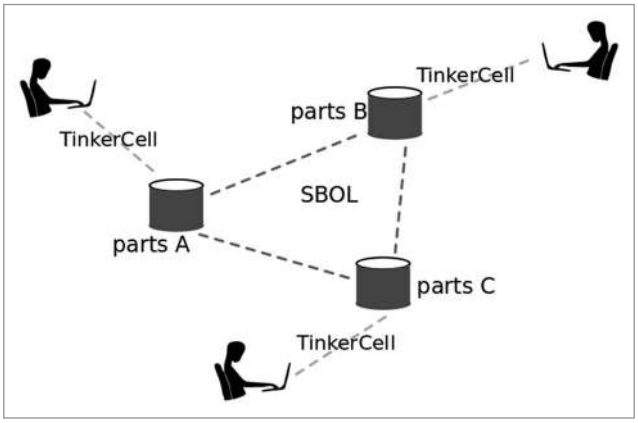

Figure 8. Integrating the Synthetic Biology Open Language semantic standards with TinkerCell will allow users to design circuits in TinkerCell and query local and remote databases for biological parts that are suitable for the design. The user would only interact via TinkerCell; standard exchange formats will make the database queries transparent to the user. The different components required to complete this process are under development at present.

Figure 8 illustrates how this integration can foster community development.

Education. Education is perhaps a less emphasized area that TinkerCell will cover. Comments about TinkerCell at the NewScientist (http://www.newscientist.com) is indicative of the fact that a good percentage of the general public have an exaggerated view of synthetic biology, which can have negative consequences on the field. It is important for the people outside the field to be better informed, so that they are aware of the limitations and the risks involved in synthetic biology. Synthetic biology research requires understanding the biology as well as the dynamics of a circuit, which often prevents those

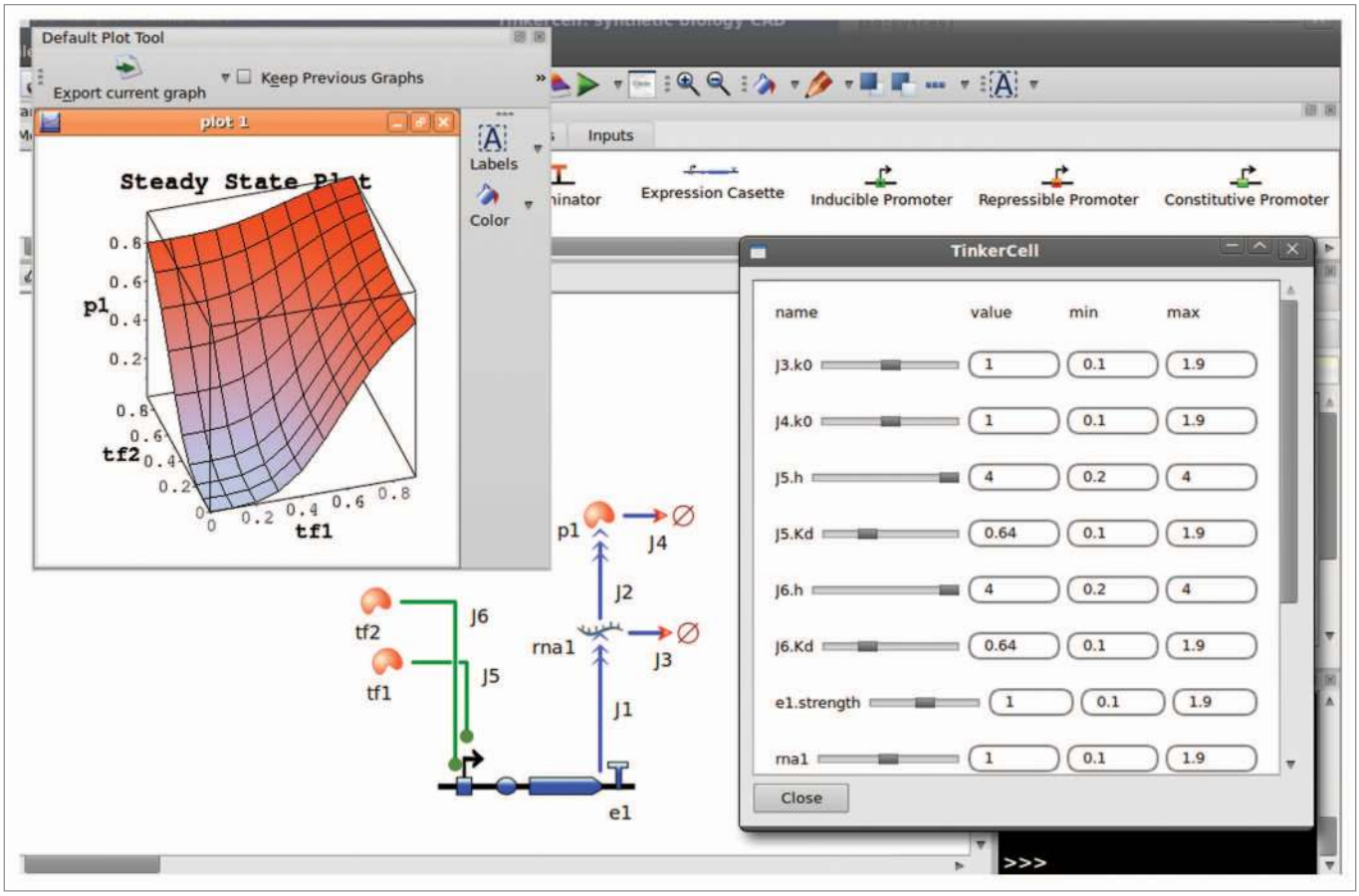

Figure 9. Some of the tools in TinkerCell, such as sliders, allows users to interactively study the effects of parameters on the dynamic behavior of a circuit. Such interactive features can be useful as educational tools. The figure shows a genetic network that behaves like an OR gate. The sliders can be used to show how the different parameters affect the threshold, steepness and height of the curve.

outside the field from understanding its limitations and potential. Tools such as TinkerCell that are visually appealing may serve as teaching tools for illustrating concepts when designing biological systems. Some interactive features in TinkerCell may also assist students in understanding the dynamics of biological circuits (see **Fig. 9**).

The community of young scientists at the International Genetically Engineered Machines Competition (iGEM)[38] is an ideal target audience for TinkerCell. This group would be able to use TinkerCell as a means of sharing computational models and algorithms that were developed as part of individual projects. Therefore, iGEM participants may represent a valuable group of contributors for TinkerCell.

Acknowledgements

This work was partly supported (Chandran) by a grant from the National Science Foundation (Id 0527023-FIBR) and Microsoft's Computational Challenges in Synthetic Biology 2006 Award. Bergmann was supported by a grant from NIH/NIGMS(GM081070).

References

1. Daskalaki A. Handbook of Research on Systems Biology Applications in Medicine. IGI Global 2009.
2. Chandran D, Copeland W, Sleight S, Sauro H. Mathematical modeling and synthetic biology. DDT Disease Models 2009; 5:18-9.
3. Adam D. What's in a name? Nature 2001; 411:408-9.
4. Arkin A. Setting the standard in synthetic biology. Nat Biotech 2008; 26:771-3.
5. Sauro HM. Modularity defined. MSB 2008; 4:166-8.
6. Balagaddé FK, Song H, Ozaki J, Collins CH, Barnet M, Arnold FH, et al. A synthetic *Escherichia coli* predator-prey ecosystem. MSB 2008; 4.
7. Stricker J, Cookson S, Bennett MR, Mather WH, Tsimring LS, Hasty J. A fast, robust and tunable synthetic gene oscillator. Nature 2008; 456:516-9.
8. Tigges M, Marquez-Lago TT, Stelling J, Fussenegger M. A tunable synthetic mammalian oscillator. Nature 2009; 457:309-12.
9. Gardner TS, Cantor CR, Collins JJ. Construction of a genetic toggle switch in *Escherichia coli*. Nature 2000; 403:339-42.
10. Lou C, Liu X, Ni M, Huang Y, Huang Q, Huang L, et al. Synthesizing a novel genetic sequential logic circuit: a push-on push-off switch. MSB 2010; 6.
11. Becskei A, Serrano L. Engineering stability in gene networks by autoregulation. Nature 2000; 405:590-3.
12. Danino T, Mondragón-Palomino O, Tsimring L, Hasty J. A synchronized quorum of genetic clocks. Nature 2010; 463:326-30.
13. Viebahn M, Smit E, Glandorf DCM, Wernars K, Bakker PAHM. Effect of Genetically Modified Bacteria on Ecosystems and Their Potential Benefits for Bioremediation and Biocontrol of Plant Diseases-A Review. Sustainable Agriculture Reviews: Climate Change, Intercropping, Pest Control and Beneficial Microorganisms 2009; 45-69.
14. Anthony JR, Anthony LC, Nowroozi F, Kwon G, Newman JD, Keasling JD. Optimization of the mevalonate-based isoprenoid biosynthetic pathway in *Escherichia coli* for production of the anti-malarial drug precursor amorpha-4, 11-diene. Metab Eng 2009; 11:13-9.
15. Weber W, Fussenegger M. The impact of synthetic biology on drug discovery. DDT 2009; 956-63.

16. Steen EJ, Kang Y, Bokinsky G, Hu Z, Schirmer A, McClure A, et al. Microbial production of fatty-acid-derived fuels and chemicals from plant biomass. Nature 2010; 463:559-62.
17. Endy D. Foundations for engineering biology. Nature 2005; 438:449-53.
18. Shetty RP, Endy D, Knight T Jr. Engineering BioBrick vectors from BioBrick parts. J Biol Eng 2008; 2.
19. Anderson CJ, Dueber J, Leguia M, Wu G, Goler J, Arkin A, et al. BglBricks: A flexible standard for biological part assembly. J Biol Eng 2010; 4.
20. Murphy KF, Adams RM, Wang X, Balazsi G, Collins JJ. Tuning and controlling gene expression noise in synthetic gene networks. Nuc Acids Res 2010; 1:2712-26.
21. Bennett MR, Hasty J. Overpowering the component problem. Nat Biotech 2009; 27:450-1.
22. Basu S, Gerchman Y, Collins CH, Arnold FH, Weiss R. A synthetic multicellular system for programmed pattern formation. Nature 2005; 434:1130-4.
23. Entus R, Aufderheide B, Sauro HM. Design and implementation of three incoherent feed-forward motif based biological concentration sensors. Sys Syn Biol 2007; 1:119-28.
24. Goodman C. Engineering ingenuity at iGEM. Nature Chemical Biology 2008; 4.
25. Goler JA. BioJADE: A Design and Simulation Tool for Synthetic Biological Systems. Technical report 2004.
26. Czar MJ, Cai Y, Peccoud J. Writing DNA with genocadtm. Nuc Acids Res 2009; 37.
27. Hill AD, Tomshine JR, Weeding E, Sotiropoulos V, Kaznessis YN. SynBioSS: the synthetic biology modeling suite. Bioinformatics 2008; 24:2551.
28. Marchisio MA, Stelling J. Computational design of synthetic gene circuits with composable parts. Bioinformatics 2008; 24:1903-10.
29. Densmore DaVD A, Johnson M, Sritanyaratana N. A platform-based design environment for synthetic biological systems. The Fifth Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiatives, Insight and Innovations 2009; 24-9.
30. Chandran D, Bergmann FT, Sauro H. TinkerCell: modular CAD tool for synthetic biology. J Biol Eng 2009; 3.
31. Olivier BG, Rohwer JM, Hofmeyr JHS. Modelling cellular systems with PySCeS. Bioinformatics 2005; 21:560-1.
32. Smith LP, Bergmann FT, Chandran D, Sauro HM. Antimony: A modular model definition language. Bioinformatics 2009; 25:2452-4.
33. Hucka M, Finney A, Sauro H, Bolouri H, Doyle J, Kitano H, et al. The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. Bioinformatics 2003; 19:524-31.
34. Feng X, Hooshangi S, Chen D, Li G, Weiss R, Rabitz H. Optimizing genetic circuits by global sensitivity analysis. Biophys J 2004; 87:2195-202.
35. Marino S, Hogue IB, Ray CJ, Kirschner DE. A methodology for performing global uncertainty and sensitivity analysis in systems biology. JTB 2008; 254:178-96.
36. Kaltenbach HM, Dimopoulos S, Stelling J. Systems analysis of cellular networks under uncertainty. FEBS Lett 2009; 583:3923-30.
37. Chen BS, Chang CH, Lee HC. Robust synthetic biology design: stochastic game theory approach. Bioinformatics 2009; 25:1822-30.
38. Smolke CD. Building outside of the box: iGEM and the BioBricks Foundation. Nature Biotechnol 2009; 1099-102.