Preprint of a paper to be presented at the Fourth International Conference on Bioinformatics of Genome Regulation and Structure (BGRS-2004), Novosibirsk, Russia, July 2004.

A SOFTWARE ARCHITECTURE FOR DEVELOPMENTAL MODELING IN PLANTS: THE COMPUTABLE PLANT PROJECT

[1]Gor, V.,*[1]Shapiro, B.E., [2]Jönsson, H., [1]Heisler, M, [1]Venugopala Reddy, G., [1]Meyerowitz EM, and [3]Mjolsness E

[1]California Institute of Technology, Pasadena, CA, USA; [2]Lund University, Lund, Sweden; [3]University of California, Irvine, CA, USA
email: bshapiro@caltech.edu
*Corresponding Author

Keywords: *Arabidopsis*, *Cellerator*, developmental modeling, *Mathematica*, meristem, SBML, systems biology

Summary

Motivation. We present the software architecture of the Computable Plant Project, a multidisciplinary computationally based approach to the study of plant development. *Arabidopsis thaliana* is used as a model organism, and shoot apical meristem development as a model process. Meristems are the plant tissues where regulated cell division and differentiation lead to plant parts such as flowers and leaves. We are using green fluorescent proteins to mark specific cell types and acquire time series of three-dimensional images via laser scanning confocal microscopy. To support this we have developed an interoperable architecture for experiment design that involves automated code generation, computational modeling, and image analysis.

Results. Automated image analysis, model fitting, and code generation allow us to explore alternative hypothesis *in silico* and guide *in vivo* experimental design. These predictions are tested using standard techniques such as mutants and altered hormone gradients. The present paper focuses on the automated code generation architecture.

Availability. http://www.computableplant.org

Introduction

Scientists who probe the functionality of dynamic developmental systems often express their models mathematically; to make precise system-specific predictions these models are typically encoded with high-level computer languages and standard support libraries and solved numerically. However, high-level languages and libraries typically trade efficiency for generality, and thus may not be appropriate for large hybrid dynamical systems. They also typically lack state-of-the-art technologies in such computationally intensive areas as model optimization and fitting. Finally, custom designed systems are rarely interoperable, making it difficult for researchers to disseminate models.

We have developed an architecture aimed at production-scale model inference. We generate simulation code from models specified in biological and/or mathematical language. Other computational tools are used to analyze expression imagery and other data sources, and the simulator combined with nonlinear optimization is used to fit the models to the data. Key elements include: a *mathematical framework* combining transcriptional regulation, signal transduction, and dynamical mechanical models; a *model generation package* (Cellerator) based on a computer algebra representation; *extensions to SBML* (Systems Biology Modeling Language), an exchangeable model representation format, to include dynamic objects and relationships; *a C^{++} code gen-*

erator to translate SBML into highly efficient simulation modules; *a simulation engine* including standard numerical solvers and plot capability; *a nonlinear optimize*r; and ad hoc *image processing* and *data mining* tools. This architecture is capable of simulating processes such as intercellular signaling, cell cycling, cell birth and death, dynamic cellular geometry, changing topology of neighborhood relationships, and the interactions of mechanical stresses.

Methods and Algorithms

Models are input in Systems Biology Markup Language (SBML), an XML-based language for exchanging biological models. SBML is currently supported by more than fifty different software packages used by biological modelers and has become the *de facto* standard for exchanging models among the systems biology community [Hucka et al 2003; Finney and Hucka 2003]. The modeling interface is provided by *Cellerator* [Shapiro et al 2003], which allows users to specify models in an arrow-based biochemical notation, and translates them automatically to differential equations using a variety of different schemes. *Cellerator* produces extended SBML Level 2 code utilizing *MathSBML* [Shapiro et al 2004]. SBML encoded models are parsed into internal data structures with a *libSBML*-based parser [Bornstein et al 2004].

Several extensions to SBML have been proposed and will likely be adopted in SBML Level 3 [Finney et al 2004]. In particular, SBML Level 2 does not support spatially-dependent models where each biological entity is *individually defined and enumerated*, and further, does not provide any easy way to describe dynamic geometry and variable size models resulting from cell birth, death, and differentiation. Therefore we have adopted [Finney et al 2003] to describe dynamic topology and connectivity in terms of arrays, and have extended *Cellerator*, *MathSBML* and *libSBML* accordingly.

Implementation and Results

The *automatic code generator* is central to the architecture. It consists of an *inferencer*, a *rule segmenter and optimizer*, and *application code writer* modules (Fig. 1). It queries the parser for SBML structures and produces efficient C++ application code. The resulting C++ code is then compiled into object code optimized for the desired application. The first two modules of the automatic code generator – the *inferencer* and *rule segmenter* – are pre-processors. They are called once for each SBML model, independent of the application software to be generated. The *inferencer* receives parsed SBML structures from the parser and infers element attributes given the element name. This reflects the inverse relationships between SBML elements and their attributes. For example, the extended SBML has a `parameter` attribute `foreach` that indicates the compartment; the *inferencer* creates a list of inferred elements, such as the list of parameters in each compartment.

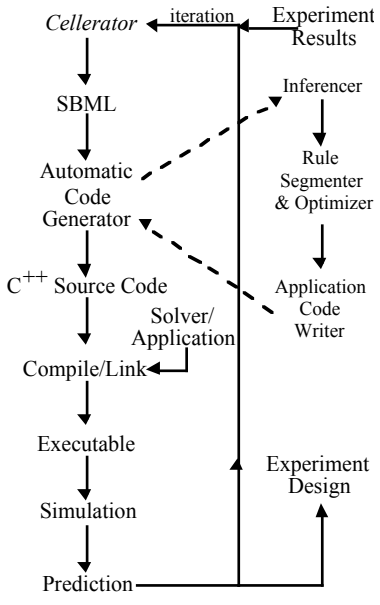

Fig. 1. System architecture.

The *rule segmenter and optimizer* translates SBML rules (which represent mathematical equations using a subset of MATHML) into C++ and performs all

necessary renaming of SBML model objects into C variables. Portions of SBML formulas that have no immediate C++ representation, such as the MATHML function `sum` (which sums a formula over an index) are broken up into sub-rules with intermediate variables; these are later translated into loops or other appropriate control and data structures. Future enhancements will include formula optimization. Identical portions of the formula will be separated into intermediate rules that are only executed once; scalar formulas inside loops will be pre-evaluated outside of the loop. The renaming function completes the work of this module. For example, individual array elements are referenced by index with an SBML model utilizing the MATHML `selector` operator; this is replaced by the appropriate C array reference such as `name[j]`.

The *application code writer* takes as input the C++ model representation generated by the *rule segmenter and inferencer*, along with an application request, chosen from a menu of available applications. The output is application source code that can be compiled and linked with the chosen application. The *application code writer* consists of a three-level library. The top level contains all of the application-dependent code. This *application level software* is high-level code that is updated as new applications are added. Applications that exist or are being developed include various forward developmental simulators including genetic regulatory network (GRN) temporal synthesis; 4^{th} and 5^{th} order Runge-Kutta differential equation solvers; and optimizers such as Lam-Delosme simulated annealing. In addition, this top level includes overloaded routines that originate at the second level thereby allowing the top level to access this lower level functionality. The second level, *SBML level software,* contains all processes that are not application dependant. This library has entry points for accessing all SBML attributes and elements. The third, and lowest level, is the *utility library*, which contains common operations such as vector algebra and memory maintenance.

Discussion

We are using this simulation environment to extend and enhance our previously reported developmental simulations of the shoot apical meristem (SAM) [Jönsson et al 2003; Mjolsness et al 2004]. Our working hypothesis is that SAM development can be described by the differential expression of key regulatory proteins such as CLV1 (a receptor kinase), CLV3 (thought to be the CLV1 ligand), WUS (a transcription factor negatively regulated by CLV1), and a layer-1 specific protein (L1SP). The dependence of CLV1 and CLV3 on WUS, perhaps through a hypothetical diffusible intermediary X, has been inferred from experiments. A second diffusive signal originates from L1SP and diffuses into the rest of the meristem via messenger Y. CLV3 is turned on only if the sum X+Y exceeds threshold. Finally, an unknown diffusible messenger Z creates a surface specific expression pattern for L1SP, which is itself inhibited by STEM, a hypothetical gene expressed only in the lowest meristem layer.

The computable plant architecture provides a systematic, highly automated technique for predictive model generation. The approach combines computer-algebraic representations of biological and mathematical models to produce efficient and problem-specific simulation code. This code can be immediately linked with a menu of external solvers and quantitative predictions generated from the resulting simulations. This architecture is scalable and directly applicable to large-scale developmental systems such as the SAM. The use of extended SBML ensures that models will be interoperable, reusable, and readable by others. Novel to this approach are connections to external solvers by

way of automatic code generation and the ability to interpret and solve any biological developmental or cellular process via automatic generation of mathematical and computational tools. Thus no labor is expended writing and debugging problem-specific code, allowing researchers to spend more time on the wet bench.

Acknowledgments

This work was supported by the United States National Science Foundation (NSF) under a Frontiers in Integrative Biological Research (FIBR) grant. HJ was in part supported by the Knut and Alice Wallenberg Foundation through Swegene. Portions of the research described in this paper was performed at the California Institute of Technology.

References

1. Bornstein, B., Keating, S. Hucka, M., and Finney, A. (2004) "libSBML: A Software Toolkit for the Systems Biology Markup Language (SBML)". Poster presentation at *Pacific Symp. Biocomp.* (PSB-2004), http://www.sbml.org/libsbml.html
2. Finney, A. and Hucka, M. (2003) Systems Biology Markup Language: Level 2 and Beyond. *Biochem. Soc. Trans*, **31**: 1472-1473.
3. Finney, A., Gor, V., Bornstein, B., and Mjolsness, E. (2003). *Systems Biology Markup Language (SBML) Level 3 Proposal: Array Features*, http://www.sbml.org/wiki/arrays.
4. Finney, A., Hucka. M., Bornstein, B.J., Keating, S., Shapiro, B.E., Matthews, J., Kovitz, B., Funahashi, A., Schilstra, M., Doyle, J.C., and Kitano, H. (2004) Evolving a Lingua Franca and Accompanying Software Infrastructure for Computational Systems Biology: The Systems Biology Markup Language (SBML) Project. *IEE Systems Biology* (in press).
5. Hucka, M., Finney, A., Sauro, H.M., Bolouri, H., Doyle, J.C., Kitano, H., Arkin, A.P., Bornstein, B.J., Bray, D., Cornish-Bowden, A., Cuellar, A.A., Dronov, S., Gilles, E.D., Ginkel, M., Gor, V., Goryanin, II., Hedley, W.J., Hodgman, T.C., Hofmeyr, J.H., Hunter, P.J., Juty, N.S., Kasberger, J.L., Kremling, A., Kummer, U., Le Novere, N., Loew, L.M., Lucio, D., Mendes, P., Mjolsness, E.D., Nakayama, Y., Nelson, M.R., Nielsen, P.F., Sakurada, T., Schaff, J.C., Shapiro, B.E., Shimizu, S., Spence, H.D., Stelling, J., Takahashi, K., Tomita, M., Wagner, J., and Wang, J. (2003) The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models. *Bioinformatics* **19**:513-523
6. Jönsson, H., Shapiro, B.E., Meyerowitz, E.M., and Mjolsness E. (2003) Signaling in Multicellular Models of Plant Development, in *On Growth, Form, and Computers*, ed. Bentley P and Kumar S, Academic Press.
7. Mjolsness, E., Jönnson, H., Shapiro, B.E., and Meyerowitz, E.M. (2004) Modeling plant development with gene regulation networks including signaling and cell division, in *Bioinformatics of Genome Regulation and Structure*, ed. N. A. Kolchanov, Kluwer Publications.
8. Shapiro, B.E., Hucka, M., Finney, A., and Doyle, J. (2004) MathSBML: A package for manipulating SBML-based biological models. *Bioinformatics*. (In press).
9. Shapiro, B.E., Levchenko, A., Wold, B.J., Meyerowitz, E.M., and Mjolsness, E.D. (2003) Cellerator: Extending a computer algebra system to include biochemical arrows for signal transduction modeling. *Bioinformatics* **19**: 677-678.