A Model Browser for Biosimulation

G. Yngve[1], M.S., J.F. Brinkley[2], M.D., Ph.D.,
D. Cook[2], M.D., Ph.D., L.G. Shapiro[1], Ph.D.
[1]Department of Computer Science and Engineering
[2]Department of Biological Structure
University of Washington, Seattle, WA, U.S.A.

ABSTRACT

The complexities of biological simulation present difficulties with modeling and experimenting. Simulators process models represented as code, whereas biologists think about abstract models. Our ModelBrowser addresses this difficulty through interactive visualization. Variables and equations appear as a directed graph of nodes and edges, and the user can search and browse this graph by performing queries on metadata associated with the variables and the connectivity of the edges. The browser also supports a hierarchical categorization of the variables, such as by an ontology. We believe that the ModelBrowser will help biologists reason about code in the context of the abstract model, so that they can understand and modify others' code and debug their own.

INTRODUCTION

Biological simulation is a fast-growing field today with a wide range of applications. Researchers are generating bigger and more complex models, and they are sharing them with each other. Many challenges are arising, including debugging, tuning, and validating a model, as well as sharing, publishing, and merging models. There need to be better ways to reason about models beyond perusing source code, which is often cryptic or poorly documented. In this paper we describe our ModelBrowser, which is a visualization application that assists the researcher in reasoning about a model through interaction with a graph of the model.

Our test example is a cardiovascular model with baro- and chemoreceptor feedback; it represents anatomy as a lumped network. The lumps (nodes in the network) represent homogenized individual anatomical parts or aggregates at various scales, and the connectivity (edges) represents physiology such as flow or control. Variables and equations range from tangible to abstract. The lumped network represents state and behavior from fluid dynamics to chemical reactions, drawing analogies from L-R-C electrical circuits. A lump encodes several variables of different physical properties, such as pressure, volume, and flow. Other variables exist too, including temporary variables that hold common subexpressions or observational variables such as vital signs.

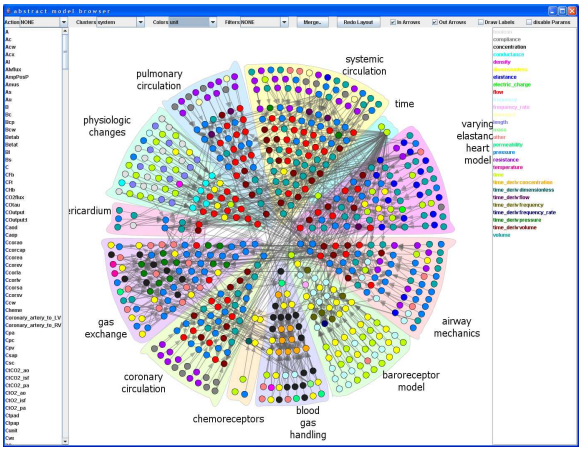

Figure 1: Overview of a large model using a pie layout. Parameters and variables are color-coded by scientific unit and clustered by module. Arrows show dependencies.

Realized into code, the model has around 300 parameters, 300 time-dependent variables, and 300 equations, of which 60 are differential. The variables use 25 different physical units. The developer divided the code into eleven sections. The module tags followed the sections that the developer already partitioned. The hierarchy that we created for this model contains 75 terms and has a depth of six, and most nodes in the hierarchy have six to twelve children. Most of the hierarchical relationships are part-of, but some are is-a or address functionality. Previously, the model already had description tags; we worked with the developer to add module and anatomy tags. Figure 1 is a screenshot of our system showing the whole model. The different sections are pie wedges. The circles are parameters (outer) and variables (inner), and the arrows represent dependencies. An excerpt of code from the model is in Figure 2.

```
real Ro = 0.025      mmHg*sec/ml;
Ro.system  = "systemic circulation";
Ro.anatomy = "vena cava";
Ro.desc    = "Vena cava resistance offset parameter";

real Rvc(t)   mmHg*sec/ml;
Rvc.module  = "systemic circulation";
Rvc.anatomy = "vena cava";
Rvc.desc    = "Resistance of Vena cava";

Rvc = (KR*(Vmax_vc/Vvc)^2) + Ro;
// Vena cava: Lu et al. Eq.(4)
```

Figure 2: Excerpt of code from the test model showing declarations of a parameter and a variable, each with metadata, and an equation.

By interacting with our visualization, one can answer questions about the model such as,

- "Show me the equations for just the fluid dynamics portions of the model,"
- "Does the model have baroreceptors in the aorta or the carotids or both?"
- "Do the lungs exert pressure on the heart?"
- "What parameters tweak the P-V curves?"
- "In the model, what are the paths of blood through the cardiovascular system?"

We first describe related work, then we present the features of the ModelBrowser, and finally we conclude and discuss future work.

RELATED WORK

The creators of CellML[9] recognized the need for a standard medium that was both human- and computer-readable and could aid in exchanging cellular models, a task previously costly and fraught with error. They discussed three forms of model representation: high-level conceptualization, mathematical model, and computer code. The high-level conceptualization reads as prose but is insufficient to rigorously define the model, whereas the computer code lacks the semantic information of the model. CellML is an open-source markup language for defining mathematical models of cellular function to help bridge the gap between computational and mathematical models. Our ModelBrowser lets one reason more mathematically about a computational model by visualizing it and allowing interactions on the qualitative information and graph connectivity.

Johnson[8] posed the need for integrated modeling environments as an important problem, noting that visualizations are often considered afterthoughts to the model and simulation. Cook et al.[4] proposed using ontologies to generate code for models through an icon-based GUI. They noted that much physiology obeys canonical equations, such as resistance or capacitance laws. Rubin et al.[11] followed, using both canonical and custom equations to recreate a cardiovascular model; using this technique they discovered several errors and inconsistencies in the original model. Cook[3] later developed Chalkboard, a graphical modeling tool for pathways that lets the user analyze feedback qualitatively, and automatically generates quantitative simulation code. The Physiome Project[7] has spawned several visualization packages, e.g. CMGUI and SCIRUN, mostly aimed at scientific visualization. Spotfire[1] seminally showed the power of the dynamic query; it uses data without any strong underlying topology, whereas our system takes advantage of the irregular hierarchical and adjacency networks present in model equations. Pathway Analytics, commercial software produced by TeraNode[13], offers a full suite for the domain of biological pathways using model authoring, collaborating, simulating and analyzing, incorporating lab data, and visualizing.

Holten's work on hierarchical edge bundles[6] reduces clutter and enhances clarity in graphs consisting of both hierarchical and adjacency relationships. His algorithm renders edges as translucent B-splines using nodes in the hierarchy as control points. Transparency helps multiple edges be discernible, even when superimposed. Color gradient indicates edge direction, which has less clutter than arrows. A bundling strength adjusts how edges are grouped by straightening the control points; the user can can vary it interactively to obtain low- or high-level connectivity information.

MODELBROWSER

The ModelBrowser is a step toward giving the developer better tools for reasoning about models that have been implemented into code. The browser consists of an index of displayed variables on the left, a legend of colors on the right, and in the center the graphical representation of the model. Nodes represent variables from the code, and edges represent dependencies from the equations. The interface provides easy navigation through a complex graph, such as traversing neighbors or pruning irrelevant parts.

Our visualization runs as a plugin to JSim[1], a Java-based simulation system for building quantitative numerical models and analyzing them with respect to experimental reference data. JSim can

[1] http://www.physiome.org/jsim/

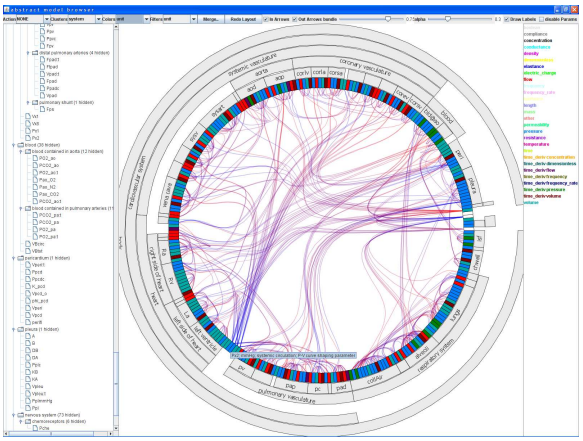

Figure 3: View of flow variables. Note the clump of P-V curve parameters at 7-o'clock.

constrain units in equations to be balanced, and it supports the embedding of metadata in the model file, as part of JSim's mathematical modeling language. JSim's capabilities are focused toward biological simulation, and it can import SBML and CellML models. The ModelBrowser uses Prefuse,[5] a Java-based toolkit for building interactive information visualization applications.

Metadata

The ModelBrowser can filter, color, cluster, or merge variables based on metadata associated with the model. The ModelBrowser reaches its full potential when the variables of the model possess a variety of metadata tags that can carry a variety of extra information. Some of these tags are part of JSim's math modeling language, such as units, datatypes, and comments, but others need to be defined explicitly, such as anatomical, physiological, or organizational tags. Tagging the variables does put extra work onto the developer on par with what is required to document code with a tool such as JavaDoc, but the benefits are huge and would likely save time in the long run. The tags give the user more fields to search and sort so that filtering and coloring can reveal interesting patterns. For our test model, we had metadata for unit, datatype, module, comment, and anatomy, of which the latter three were tagged explicitly.

Hierarchies

Hierarchies add significant meaning and capability to code. Though a computer scientist might be capable of designing a carefully architected object-oriented system, the same cannot be expected of a biologist writing a model. Furthermore, a biological model may have more interconnections than the average software. Thus a modular or hierarchical structure can enrich a flat list of equations and variables by using metadata to assign variables to entries in a hierarchy, perhaps derived from an ontology. In our case, an extra XML file holds the application hierarchy for the model. We are pursuing novel tools for developing hierarchies that build from existing ontologies such as the FMA.[10]

Dependencies

Each equation in the model consists of a lefthand variable and a righthand expression, which in turn consists of several terms, either implicitly or explicitly. The JSim compiler parses the simulation code and exposes these dependencies, needed for compilation and execution, to the plugin developer. The variables are represented visually as nodes, and the dependencies as edges. For example, the equation $F = m \cdot a$ would have three nodes and two edges, with arrows from m and a to F.

INTERACTION

The ModelBrowser has three types of interaction, as categorized by Card, et al.[2]. We illustrate the use of our tool with several screenshots.

Data Transformations

Data transformations operate on the data itself, but rather than doing an SQL query, the user can point and click through friendlier interfaces. For each field of discrete metadata, the user can select checkboxes for which values to show or hide. The system evaluates the conjuction over all the fields. If a filtered node becomes invisible, any edges going to or from it likewise become invisible. The user can select a set of children in a hierarchy to hide or unhide. To remind the user that nodes are hidden in the hierarchy, the number of hidden nodes is displayed in the tree index, as is done in Jambalaya[12]. Figure 3 shows the variables filtered to just pressure, flow, volume, and their time derivatives, using hierarchical edge bundles.

Our system can display details on nodes by selection either in the index or on the graph. When focusing on a node, the user can load equations, graphs, and other details for the node.

Visual Mappings

Our system supports coloring of nodes based on discrete values in the variables' metadata. Each value is mapped to a color, explained by a legend. Around thirty different colors can be displayed before shades become hard to discern. Most metadata are treated as-is, but units are colored by their canonical form, i.e. ignoring dimensionless constants (e.g. cubic meters and liters are both volume and are colored the same). The system also colors nodes based on connectivity, using different color channels to represent whether a node

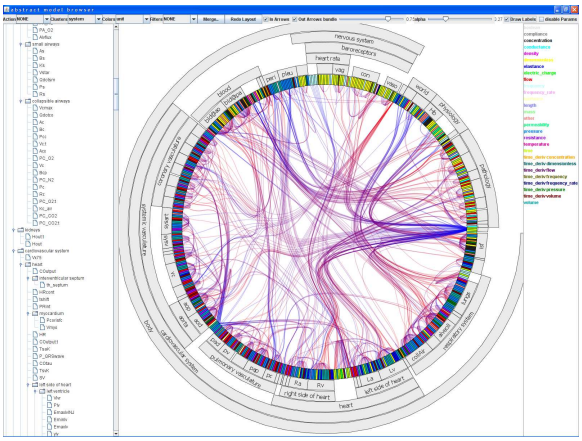

Figure 4: Full model, hierarchical edge bundling.

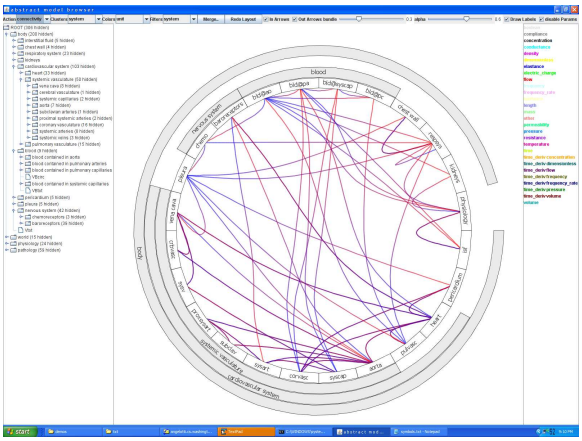

Figure 5: Summary view of model with hierarchy. Baro- and chemoreceptor ins/outs are clear.

has zero, one, or more dependencies, or is a dependency of zero, one, or more nodes. The system also allows filtering on the in- and out-degrees of nodes, as if they were also metadata.

Visual Transformations

Several visual transformations map the graph, with the hierarchy expanded to some degree, to an interactive diagram.

Expansion / Contraction. The hierarchy can be expanded or contracted by navigating through the tree on the index or by clicking on a node and telling it to expand or contract if legal. When a set of nodes are contracted, any dependencies inside the set disappear. Dependencies to or from an outside node are redirected to the contracted node. Expansion is an undo of contraction. Figure 4 is fully expanded, and Figure 5 is mostly contracted. When no hierarchy is present, contraction can be performed on metadata to merge like nodes.

Layout. We have implemented two different types of layouts for nodes and edges. The icicle layout emphasizes the dependencies across the hierarchies, whereas the pie layout optimizes the arrangement of nodes for dependencies within a group. The pie layout is space-filling and is best for giving an overview of the variables.

The icicle layout represents arbitrary hierarchies by concentric rings growing inward. For this layout, we implemented hierarchical edge bundles[6] to render edges, as discussed in related work. Directed edges go from blue to red. Figure 4 shows a visualization of the full model, with bundling defaulted to 0.75. We don't do any node rearrangement here because the layout is intended for edges across hierarchies, though it could have some benefit in Figure 5. Because of the shallower hierarchy and fewer edges, the user chose to reduce bundling to 0.3 and increase the opacity of the edges. Figure 5 shows how the baroreceptors sense the aorta and affect the heart and systemic arteries, and the chemoreceptors sense the blood in the aorta and affect the respiratory system.

The pie layout fills the variables into areas that are wedges of a pie, such that pie consists of a uniform density of nodes. The pie layout supports a partitioning just one level deep. A discrete optimization permutes nodes within a group to minimize the sum of squared edge lengths, which creates a much cleaner layout. Edges are rendered as straight lines with arrows, so that color can be used for other purposes. The system can put parameters further from the center than variables or intermix them. Figure 6 shows the advantage of a pie layout with edge optimization. The coronary circulation forms a nice chain of flow propagation.

Focus and Other Interaction. The user can also explore the in- and out-neighbors for a node recursively or browse feedback loops for a node. When focusing on a node and its neighbors, edges are highlighted and colored based on direction, and labels appear on the nodes, temporarily hiding unrelated nodes. Hovering pops up a tooltip that displays information on the variable. Figure 7 shows focusing on the set of merged baroreceptor variables and looking at second neighbors. In the screenshot, the user also loaded details of the equations associated with baroreceptors.

The whole system supports smooth panning and zooming on the graphs, and transitions are animated when possible to establish temporal coherency. For the hierarchical edge bundling, sliders exist to adjust bundling and opacity.

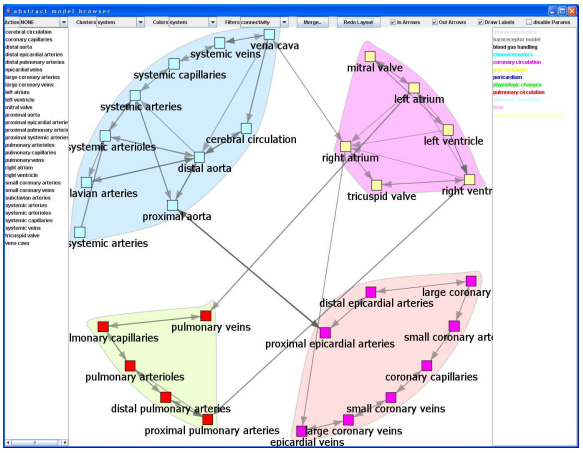

Figure 6: Interactions between anatomical parts, mostly fluid flow. Layout optimizes paths.

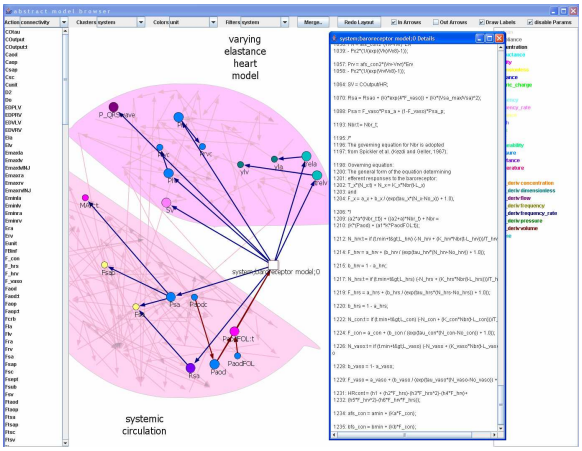

Figure 7: Exploring connectivity of baroreceptor module. All baroreceptor equations are shown.

CONCLUSIONS AND FUTURE WORK

We built the ModelBrowser to aid the researcher in reasoning qualitatively about a model specified in code. By interactively navigating graphs, filtering variables, and hierarchically browsing, a user can answer many questions that would be hard otherwise with just the source code. We have shown our tool to several bioengineers and have gotten positive feedback; a user study is next. We plan on both timing users and having a questionaire to evaluate insight for several tasks, such as answering the questions posed in the introduction. We also would like to try other models, especially ones involving metabolic pathways or ion channels.

Most of the queries in the ModelBrowser operate on nodes, and we would like to explore queries on edges and neighborhoods too. Many models contain canonical equations that are instantiated many times, e.g. conservation of flow. We would like to have a novice interface for searching for such equations, or be able to detect repeated patterns of equations. Advanced queries could aid in bug hunting or explaining auto-generated code.

We would like to extend the ModelBrowser to reason about a model quantitatively, by enabling dynamic queries on simulation data, visualizing sensitivity analysis, and clustering variables with similar behavior. In addition to being a generic model browser, we would like to focus our tool for specific tasks such as debugging or parameter tuning. Funded in part by NIH grant HL087706.

REFERENCES

1. C. Ahlberg and B. Schneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proc. of CHI*, 1994.
2. Stuart K. Card, Jock D. Mackinlay, and Ben Schneiderman. *Readings in Information Visualization: Using Vision To Think*. Morgan Kaufmann Publishers, Inc., 1999.
3. Daniel L. Cook, John H. Gennari, and Jesse C. Wiley. Chalkboard: Ontology-based pathway modeling and qualitative inference of disease mechanisms. *Pac Symp Biocomput*, 12:16–27, 2007.
4. Daniel L. Cook, Jose L. V. Mejino, and Cornelius Rosse. Evolution of a foundational model of physiology: Symbolic representation for functional bioinformatics. *MEDINFO*, 11:336–340, 2004.
5. Jeffrey Heer, Stuart K. Card, and James A. Landay. prefuse: a toolkit for interactive information visualization. In *ACM CHI*, 2005.
6. Danny Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Trans. on Vis.&Comp. Graphics*, 12(5), 2006.
7. Peter Hunter. Modeling human physiology: IUPS / EMBS physiome project. *Proc. IEEE*, 94(4), 2006.
8. Chris Johnson. Top scientific visualization research problems. *IEEE Comp. Gr.&Apps.*, 24(4), 2004.
9. Catherine M. Lloyd, Matt D.B. Halstead, and Poul F. Nielsen. CellML: its future, present and past. *Prog. Biophys.&Mol. Bio.*, 85:433–450, 2004.
10. Cornelius Rosse and Jose L. V. Mejino Jr. A reference ontology for biomedical informatics: the foundational model of anatomy. *Journal of Biomedical Informatics*, 36:478–500, 2003.
11. Daniel L. Rubin, David Grossman, Maxwell Neal, Daniel L. Cook, James B. Bassingthwaighte, and Mark A. Musen. Ontology-based representation of simulation models of physiology. In *AMIA Annual Symposium Proceedings*, 2006.
12. M. A. Storey, M. Musen, J. Silva, C. Best, N. Ernst, R. Fergerson, and N. Noy. Jambalaya: an interactive environment for exploring ontologies. In *Intl Conference on Intelligent User Interfaces*, 2002.
13. Teranode. Leveraging pathway analytics for life sciences research and development, 2005.