Application of Technology ■

ModelDB: An Environment for Running and Storing Computational Models and Their Results Applied to Neuroscience

BRET E. PETERSON, PHD, MATTHEW D. HEALY, PHD, PRAKASH M. NADKARNI, MD, PERRY L. MILLER, MD, PHD, GORDON M. SHEPHERD, MD, DPHIL

Abstract Research groups within the Human Brain Project are developing technologies to help organize and make accessible the vast quantities of information being accumulated in the neurosciences. The goal of this work is to provide systems that enable this complex information from many diverse sources to be synthesized into a coherent theory of nervous system function. Our initial approach to this problem has been to create several small databases. While addressing the issues of each individual database, we are also considering how each might be incorporated into an integrated cluster of databases. In this paper, we describe a pilot project in which we construct a database of computational models of neuronal function. This database allows models to be created and run and their results reviewed through a World Wide Web interface. Because models encapsulate knowledge in a formal manner about how neuronal systems function, we also discuss how this database forms a natural center for our initial attempts at creating a cluster of related databases. General issues of database development in the context of the Web are also discussed.

■ **JAMIA.** 1996;3:389–398.

Neuroscience research is generating vast quantities of data in subdisciplines, ranging from the molecular to the behavioral level. Each level of study has had surprising success at describing phenomena at that level, but no one area has been sufficient in itself to provide a satisfactory description of nervous system function as a whole. Furthermore, the complexity of the systems being studied requires that, even within a given subdiscipline, an increasingly narrow focus of specialization is required in order for progress to be made. This necessary specialization makes compilation and integration of results across disciplines all the more difficult.[1,2]

To help overcome these barriers and to help facilitate cross-discipline exchange of information between different laboratories, we have developed a pilot modeling database environment called ModelDB (http://senselab.med.yale.edu/models/help/). ModelDB is a system that acts as a front end for both a database and a complex modeling program and ties the two together. ModelDB provides the necessary functionality for developing models in a collaborative environment through the use of a World Wide Web interface. Because the models exist in a database, they are easily linked to data and results in other databases. This capability is critical because the data and results upon which the model is based can be easily reviewed. Because these links are made using the HyperText Transfer Protocol (HTTP),[3] the databases

Affiliations of the authors: Yale University School of Medicine, New Haven, CT, Department of Anesthesiology and Center for Medical Informatics (BEP, MDH, PMN, PLM); Section of Neurobiology (BEP, GMS).

This work was supported by the National Institute for Deafness and other Communicative Disorders (National Institutes of Health), by the National Institute for Neurological Diseases and Stroke (National Institutes of Health), and by the National Aeronautics and Space Administration, National Institute of Mental Health, and the National Institute for Deafness and other Communicative Disorders (National Institutes of Health) through the Human Brain Project.

Correspondence and reprints: Bret Peterson, PhD, 1 Kearny Suite 501, San Francisco, CA 94108. email: bret@scilearn.com

Received for publication: 3/25/96; accepted for publication: 7/18/96.

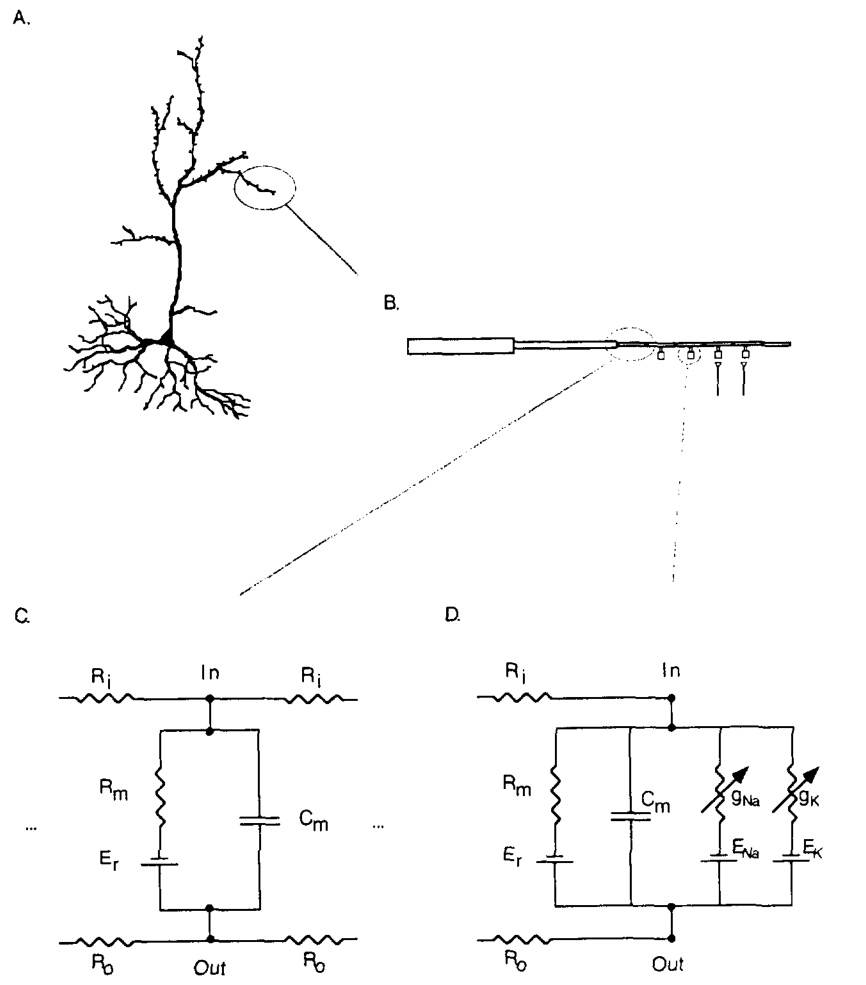


Figure 1 Construction of a canonical model of part of the spiny dendritic tree of a neuron. Morphology of a dendritic branch (A) is abstracted into a series of cylindrical compartments (B). Each compartment is modeled as an equivalent electrical circuit (C). Voltage-dependent conductances can be added to appropriate compartments to simulate active properties in the branch (D).

can be in a variety of sites around the world. Because the interface to the models is also through HTTP, the collaborative team of modelers and experimenters can be distributed around the world as well, as envisaged by the Human Brain Project.[4,5]

In this paper we describe the application of ModelDB to a series of dendritic models using a particular neuroscience simulator. ModelDB has potentially broader applications, however, because it can easily be adapted to run other simulators as well. The approach of combining Web and database technologies can also be generally applied.

Background

An important step toward integrating results at the level of the neuron has been the creation of computational models. Starting with the introduction of the compartmental modeling approach,[6,8] there have been successive steps in the development of specialized programs,[9-12] adaptation of general-purpose simulation programs,[13,14] and finally the recent emergence of large-scale programmable environments for developing neural simulations.[15-21] ModelDB attempts to extend these environments further.

The preliminary use of ModelDB has been to create a series of compartmental models of a simplified canonical representation of a dendritic system using the GENESIS simulator.[22] As shown in Figures 1A and B, the behavior of a small branch of a dendrite is approximated by first separating its properties into a series of discrete cylindrical compartments. These compartments represent the main branch of the dendrite and the head and necks of the dendritic spines. The simulator models the electrical flow between compartments by means of equivalent electrical circuits (Figs. 1C and D).[6-8,23-25] This particular model extends previous models[26-28] and was constructed to investigate how nonlinear (voltage-dependent) ion channel properties in the dendrite and spines (Fig. 1D) might influence coupling between synaptic input and the charge spread to more proximal parts of the neuron. It is this spread that leads to the generation or modulation of impulses in the axon, which constitute the output of the neuron that excites or inhibits other cells in the pathway. The models demonstrate how this particular set of dendritic properties could make important contributions to the network properties underlying cortical functions.

Computational models serve as precise representations of our understanding of studied or, as in this case, theoretical circuits, neurons, and neuronal components. This precision not only helps organize what is known but also helps make explicit what is unknown. It thus provides an extremely useful means for experimenters to consolidate their findings. Modeling does require, however, a significant level of computing and mathematical sophistication, and it usually requires expertise in a particular programmable simulation package as well. Therefore, experiments and modeling together tend to require more skills than are generally possessed by a single individual. Consequently, close collaboration has traditionally been required between modeler and experimenter.

The quantity of information that needs to be exchanged between experimenter and modeler in order to create these models has often prevented this interaction from extending beyond the confines of individual laboratories. Consequently, a model seldom encompasses more than the particular expertise of a single laboratory. ModelDB attempts to make easier the exchange of information between modeler and experimenter. Because it is based on Web technology, ModelDB can operate independent of the collaborators' locations. It thus allows a distributed group of multidisciplinary experts to work together on developing a single model.

Design of the Environment

In the initial stage of this project, there were three primary goals. The first was to provide a system for archiving models in order to maintain a structured, continuous record of the models that were created as well as to maintain an ongoing record of the parameters that were tested on these models. The second goal was to provide a mechanism for experimenters who were not primarily modelers to run existing models and test new parameter sets. The third goal was to create a system that encouraged collaboration by making it independent of the simulator, the operating system, and the physical location of the collaborators.

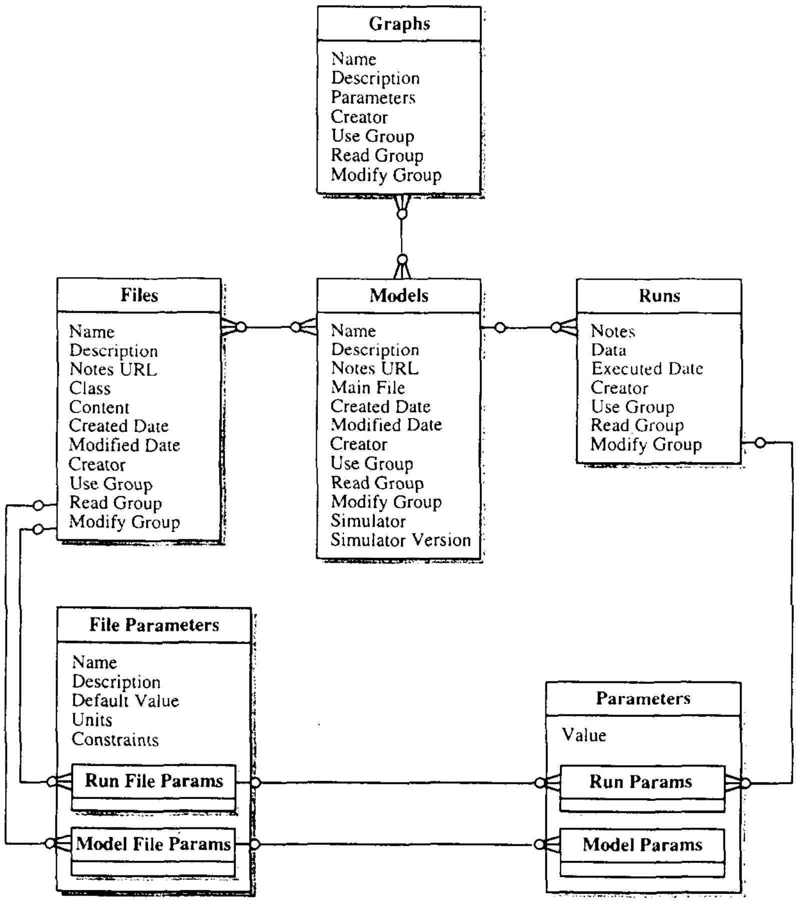


Figure 2 A schematic of the database schema. The three-pronged connectors indicate a "many" relation, and the single-pronged connectors indicate a "one" relation. The bridging tables for the many-to-many relations are not shown. The tables within tables indicate Illustra's support of subtables (analogous to object-oriented subclasses). These allow the scripts to reference the supertable rather than having to explicitly join the two subtables. An entry is created in the Models table for every version of every model. The entry references a set of files that define the model. When the model is run, the Run and Model parameters in each file are replaced with their assigned values in the Parameters tables. If no value has been assigned, they take on their default values from the parameter definitions in the File Parameters tables. Each run generates a new entry in the Runs table that contains the resulting data. Entries in the Graph table are also associated with Models entries. They specify which variables in the resulting data should be graphed together. Selecting a pair of Graph and Run entries for a particular model allows a helper application to graph the data in the specified format.

Archiving

Modeling is an art of constant revision. A difficulty in this art is maintaining careful records so that the results associated with a model that has since undergone many revisions can be recreated by duplicating the previous version of the model. The relational database engine used by ModelDB maintains referential integrity between a model version and runs related to that version, so it is impossible to accidentally delete a version on which one or more runs depend. ModelDB explicitly maintains synchronization between a particular version of a model and the files/parameters that constitute it. In collaborative efforts, the database can also help prevent costly duplicative efforts. Results from models with parameter sets that have already been tested can be presented immediately rather than regenerated by running the model again.

Interface

Many modeling programs require that a fairly complex language be learned before an actual model can be implemented. Programming models in such environments requires a considerable investment of time. Once the model is built, however, the code can be effectively hidden from experimenters to whom it has no meaning. Experimenters are provided with an interface that allows them to directly manipulate familiar parameters and to initiate runs. The results are easily accessible, so experimenters can determine how well the output of the model reflects the particular system that they are studying.

ModelDB uses a Web interface that affords a series of advantages. Because browsers are available on almost every platform, this interface is effectively platform independent. Only minor modifications need to be made by the environment developers to support new simulators. The interface, however, need not change; thus, from the point of view of those running the models (experimenters) rather than those developing them (modelers), the system can appear to be simulator independent as well. Finally, since the system is accessible over the Internet, it makes running and demonstrating models possible from almost anywhere in the world; i.e., the system is also location independent.

Database Structure

The database schema in ModelDB is centered on the Models table (Fig. 2). An entry in this table references a set of files that the simulator uses to run the corresponding model. Each file has two sets of parameters associated with it: Model and Run parameters. Values for the Model and Run parameters are defined when the model is created and run, respectively. Dividing parameters into these two groups is left to the discretion of the modeler and depends on the aspect of the model being tested. In short, Run parameters are those whose influence on the model are being tested and are thus modified often, whereas the Model parameters are those that define the attributes of the model. When the model is run, an entry in the Run table that holds the resulting data is created and associated with the model. Entries in a Graph table can also be associated with a model. These entries define what parameters of the run data should be graphed together.

ModelDB uses an Entity-Attribute-Value (EAV) representation for the Run and Model parameters. A file is the entity, the Run and Model parameters for each file are its attributes, and each parameter has an assigned value. The EAV approach, originally pioneered in the List Processing (LISP) language in the form of association lists,[29] has been widely used to represent knowledge in medical databases, most notably by the Columbia Informatics group[30,31] to represent significant positive and negative findings in clinical patient records.

The EAV design approach gives a great deal of flexibility in the way that files can be parameterized, from simply representing numerical values as parameters as described in this paper to representing variable sections of code as parameters. Because of this inherent flexibility, we believe that ModelDB will be able to support several kinds of simulators. While ModelDB has thus far been used for neural circuitry modeling, there is nothing within its schema that contains hard-coded neurocircuitry concepts (voltage, resistance, number of compartments, etc.). The only explicit assumption made in ModelDB is that a neural circuitry model is characterized by a particular set of parameters, some of which will change often during simulations, while others will remain stable. In practice, however, the EAV approach does have some significant inherent inefficiencies (notably in retrieval speed and the task of building easy-to-use, fault-resistant user interfaces) that limit its use as a panacea for database design. Therefore, similar domains in which the number of parameters to be tested is relatively modest are likely to be the most practical for adaptations of this environment.

Running in the Environment

ModelDB was created using cgi-bin scripts written in Perl that are accessed via an NCSA Web server. These scripts access an Illustra database (Fig. 3).[32] Illustra is a hybrid database system that combines relational and

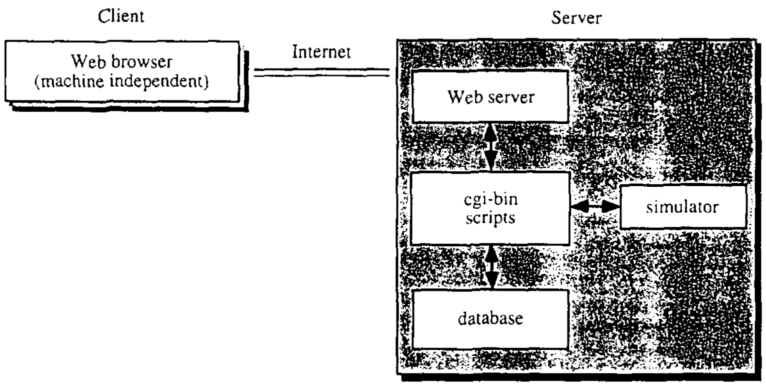


Figure 3 The environment is implemented via a Web server. The client uses a Web browser to access the server. The server invokes scripts that access the database as well as the simulation software.

object-oriented features. The current modeling environment primarily uses the relational features and could easily be ported to other relational databases. In the future, the object-oriented features will be employed for allowing analysis-based searches of modeling and experimental data.

Creating a Model

The principal reason for associating multiple files with a model is that this allows modularization of function and reuse of code. Many required functions are used in every model, and these can be grouped into library files. Including these files in the model simply involves selecting the file names from a list of files in the database. The files can be assigned to a class (e.g., library) to simplify locating them.

In the dendritic models described above, the remaining function of the models was defined with two additional files, one describing the morphology of the dendrite and the other containing the code that implements the experimental test conditions.

The code was first tested on a single instance of the model with a single set of parameters to verify that it works. This step is necessary because debugging is easier within the simulator environment than within ModelDB. The next step is to parameterize the files and to enter them into the database. This critical step involves replacing values that might change from model to model with Model parameters, and replacing those values whose effect on the model are to be tested with Run parameters. Run parameters are easier to change, but, because they define a run, too many of them can make searching for and displaying runs cumbersome. In the morphology file (Fig. 4), the dimensions of every compartment were parameterized. The shapes of the main dendrite and the spine head compartments were rarely changed, so they were defined as Model parameters. On the other hand, the shapes of spine necks were critical variables in determining function, and thus they were defined as Run parameters so that they could be modified often.

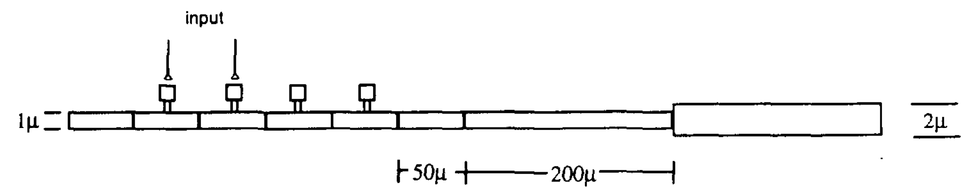
After the files were entered into ModelDB, they were added to a model. Before the model was run, it was further defined by setting the values of the Model parameters. If a Model parameter value is not set, it assumes its default value. Once a run is generated, the values of these parameters can not be changed because they are part of the model definition.

Running the Models

After a model has been defined by selecting files and setting the Model parameters, it is ready to run. Before each run, ModelDB presents an interface that allows the Run parameters to be modified. When these are submitted, ModelDB creates a local work space for the current user and places the files within this space while replacing the Run and Model parameters within these files to their currently assigned values. It then calls the simulator to run the model. Currently, the environment waits for the run to complete and then puts the results into the database. In the future, in order to support larger models, a queuing system will be created so that the user can submit a model but not necessarily have to wait for it to finish before going on to other tasks.

This issue was not addressed initially because our approach had been to develop small canonical models, not large-scale models containing thousands of compartments. Both types of models have advantages for dealing with the complexity of neuronal systems. Canonical models attempt to define the minimum complexity required to simulate essential neural operations. As discussed elsewhere,[33,34] this approach is critical for incorporating more realistic neurons into neural networks without adding more computational complexity to the neural nodes than is absolutely necessary. Developing such models could lead to close collaborations between system and neuronal model-

A.



B.

```
##d1Len model 50
##d2Len model 50
##d3Len model 50
##d4Len model 50
##d5Len model 50
##d6Len model 50
##d7Len model 200
##d8Len model 200
##d1Diam model 1
##d2Diam model 1
##d3Diam model 1
##d4Diam model 1
##d5Diam model 1
##d6Diam model 1
##d7Diam model 1
##d8Diam model 1
##n1Len model 1
##n2Len model 1
##n3Len model 1
##n4Len model 1
##h1Len model 1
##h2Len model 1
##h3Len model 1
##h4Len model 1
##h1Diam model 1
##h2Diam model 1
##h3Diam model 1
##h4Diam model 1
##h1NaDen model 5000
##h2NaDen model 5000
##h3NaDen model 5000
##h4NaDen model 5000
##h1KDen model 400
##h2KDen model 400
##h3KDen model 400
##h4KDen model 400

##n1Diam run 0.08
##n2Diam run 0.08
##n3Diam run 0.08
##n4Diam run 0.08
```

C.

```
//      Morphology
dend1   none    ##d1Len  0  0  ##d1Diam
dend2   dend1   ##d2Len  0  0  ##d2Diam
dend3   dend2   ##d3Len  0  0  ##d3Diam
dend4   dend3   ##d4Len  0  0  ##d4Diam
dend5   dend4   ##d5Len  0  0  ##d5Diam
dend6   dend5   ##d6Len  0  0  ##d6Diam
dend7   dend6   ##d7Len  0  0  ##d7Diam
dend8   dend7   ##d8Len  0  0  ##d8Diam

neck1   dend2   ##n1Len  0  0  ##n1Diam
neck2   dend3   ##n4Len  0  0  ##n2Diam
neck3   dend4   ##n2Len  0  0  ##n3Diam
neck4   dend5   ##n3Len  0  0  ##n4Diam

head1   neck1   ##h1Len  0  0  ##h1Diam   Na ##h1NaDen   K ##h1KDen  Ex 0
head2   neck2   ##h2Len  0  0  ##h2Diam   Na ##h2NaDen   K ##h2KDen  Ex 0
head3   neck3   ##h3Len  0  0  ##h3Diam   Na ##h3NaDen   K ##h3KDen
head4   neck4   ##h4Len  0  0  ##h4Diam   Na ##h4NaDen   K ##h4KDen
```

D.

```
//      Morphology
dend1   none   50   0  0  1
dend2   dend1  50   0  0  1
dend3   dend2  50   0  0  1
dend4   dend3  50   0  0  1
dend5   dend4  50   0  0  1
dend6   dend5  50   0  0  1
dend7   dend6  200  0  0  1
dend8   dend7  200  0  0  2

neck1   dend2  1  0  0  0.08
neck2   dend3  1  0  0  0.08
neck3   dend4  1  0  0  0.08
neck4   dend5  1  0  0  0.08

head1   neck1  1  0  0  1   Na 5000   K 400  Ex 0
head2   neck2  1  0  0  1   Na 5000   K 400  Ex 0
head3   neck3  1  0  0  1   Na 5000   K 400
head4   neck4  1  0  0  1   Na 5000   K 400
```

Figure 4 Parameterization of the model. Each compartment is represented by a cylinder with a length and diameter (A). One file in the model is used to represent this morphology. When this file is defined, a list is generated that gives the parameter names and default values and indicates whether the parameter is a Run or Model parameter (B). This list is associated with the parameterized file contents (C). The approach taken here is to precede parameter names with "##" so that it is unlikely that a file would contain one by accident, but no restrictions on parameter names are enforced. Before a run, the parameters are replaced to create a working file (D). In this example, all parameters except for the spine neck diameters have been defined as Model parameters.

ers. This process could be facilitated by ModelDB because it provides a convenient mechanism for these modelers to share and run models over the Web.

Web-based technologies make it possible to present a consistent interface to experimenters for assigning values to parameters, independent of the simulator being used. An interface for starting a run on a simple model with four spines is shown in Figure 5. In this example, the Run parameters are the conductance increases induced by active synapses on two spine heads, the time delay between the activation of the two synapses, and the diameters of the necks of the four spines. Clicking on the "Submit" button begins the run. Information about the run, including the resulting data, are then entered into the database.

Handling Versions

One main advantage over using a simulator and the operating system file system is that ModelDB allows you to maintain previous versions, so models that have been used to create stored data are never lost. In order to be viewed as an asset, this maintenance must not significantly slow the modeling process. Parameterization of the files provides a mechanism for the modeler to pay a very low price for this maintenance. It only requires a little foresight in the design so that any item in a file that is likely to change is parameterized. Furthermore, those items likely to have many values tested on each model should be defined as Run parameters and the rest should be defined as Model parameters.

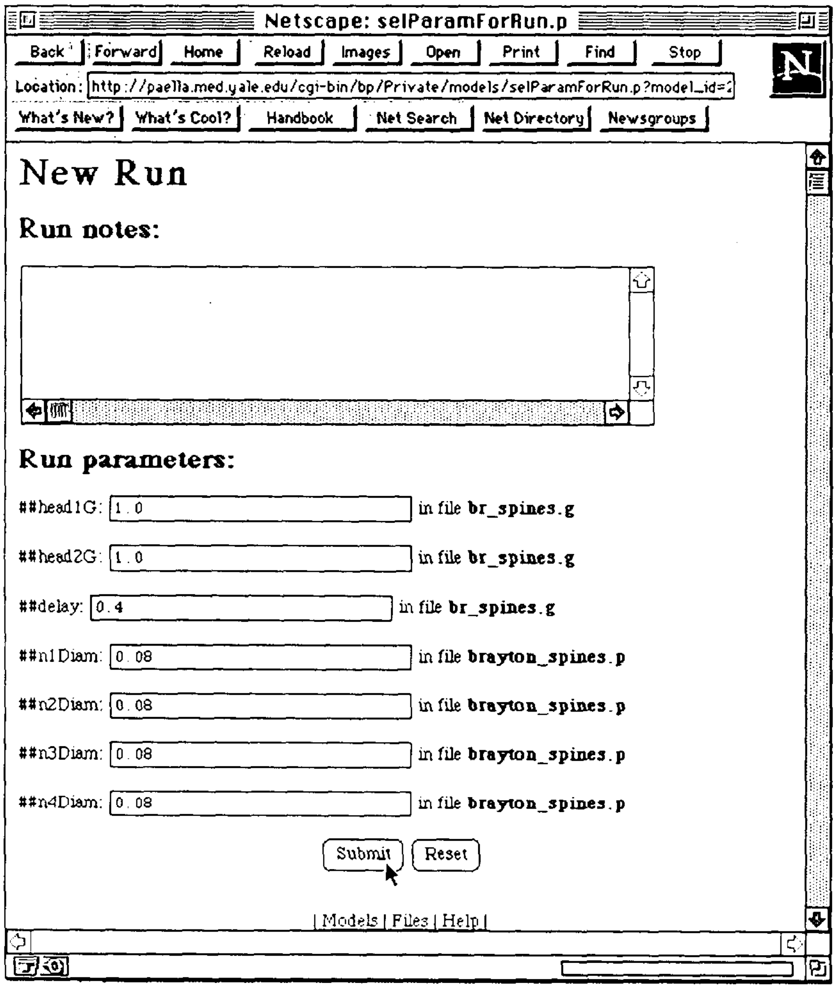
This organization leads to four levels of preparing a model for the next run. If only a Run parameter needs to be changed, then the model can be run immediately without modification. If one of the Model parameters has to be changed, then the experimenter need only ask the system to duplicate the model and then assign values to the Model parameters. If something that was not parameterized within one of the files has to be changed, the model and file must be duplicated. The new file can then be modified (probably by the modeler rather than the experimenter) and added to the new model in place of the previous copy. Finally, completely new models require that the modeler create new code in a new set of files. The system requires very little extra work, and that work is proportional to the size of the change being made. This minimal overhead is a small price to pay for guaranteed reproducibility of the data.

Viewing the Results

Once a model has been run one or more times, the experimenter must be able to access the results. Often the same variables, such as spine head voltages, are to be graphed for each run. Assigning a Graph structure to a model prevents the experimenter from having to define the variables to be graphed more than once. Since a Graph structure can belong to more than one model, the structure can be linked to a series of models as well as to a series of runs. Once a graph is selected for a model, the list of runs for that model is presented. When a run is selected, the data from that run and the graph definition are sent to a helper application to create a graph (Fig. 6). Currently, this helper application has only been written for the Macintosh, but we are taking steps to make this aspect system-independent as well. This independence will be achieved either by creating the graphs on the server side and outputting them in a browser readable format or by converting the graphing routines to Web-interpretable code (e.g., Java).[35-37]

Collaboration

The Web interface makes the models accessible via the Internet. Collaborators can therefore easily run models and view results whether they are working in the same office, in different buildings, or on different continents. Because computational neuroscience is interdisciplinary, as are other modeling fields, this ease of collaboration is important because it allows the cooperation of a necessarily diverse set of talents without being constrained by the physical location of the participants.



Location: http://paella.med.yale.edu/cgi-bin/bp/Private/models/selParamForRun.p?model_id=

New Run

Run notes:

Run parameters:

##head1G:	1.0	in file **br_spines.g**
##head2G:	1.0	in file **br_spines.g**
##delay:	0.4	in file **br_spines.g**
##n1Diam:	0.08	in file **brayton_spines.p**
##n2Diam:	0.08	in file **brayton_spines.p**
##n3Diam:	0.08	in file **brayton_spines.p**
##n4Diam:	0.08	in file **brayton_spines.p**

Figure 5 Interface for running the dendritic spines model. The Run parameters for this model are the synapse conductances on the first two spine heads, the delay between the arrival of these inputs, and the four spine neck diameters.

Two Lessons Learned

Two additional issues arose during this work: achieving adequate performance and creating a safe environment. These issues are common, especially in Web-based applications. Though workable solutions have been achieved, these issues will continue to require attention as the project expands.

Performance

Using a Web interface can lead to a number of performance problems. Because this interface does not easily allow the saving of state, it can require that the database be accessed more frequently. In the original implementation, each database access resulted in the launching of a large program through which communication with the database took place. These problems were partially alleviated by using Illustra's text capabilities and unlimited Structured Query Language (SQL) command size to allow multiple queries to be combined into a single database access. Because the cgi-bin scripts are written in Perl, which is a strong text-processing language, they could easily parse the returned values into separate responses for each query.

Naviserver (NaviSoft),[38] an alternative to the cgi-bin script interface, is now being investigated. This system allows one to embed C or TCL into the server so that the cgi-bin process initiation is avoided. Furthermore, it maintains constant contact with the database so that accesses are very efficient. Because Illustra charges per license where a license is needed for each connection to the database, a disadvantage of this system is that it requires that at least one Illustra license be dedicated to the Web server.

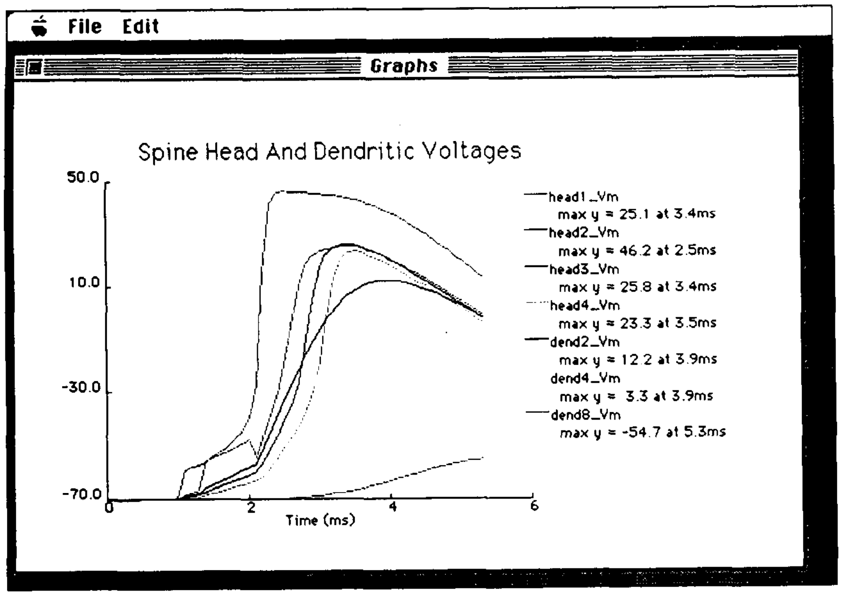
Providing a Safe User Environment

It was clear from the beginning of this project that some information about users would have to be maintained in order to provide a safe environment for each model developer. This was accomplished by giving each user a name and a password that permit access to the cgi-bin directory via the Web server. Each script can then determine the user by accessing the REMOTE_USER environment variable. A script can also access a separate database that maintains a list of the groups to which each user belongs. Similar to files in UNIX, files and models within ModelDB are associated with permissions that determine which groups can read, modify, and run them. This simple scheme makes it easy to develop a model in the privacy of a small group, then to make it public after it has been tested.

Running the model can lead to additional problems because many simulators allow access to the underlying operating system. Intentional or accidental flaws within the modeling code can therefore lead to system problems. To help avoid these problems, a dummy account was instantiated with minimal permissions and limited disk space. The simulator processes were assigned to this account by using a program called cgiwrap.[39] Even with a responsible group of model developers, these precautions were well worthwhile.

Future Directions

The initial goal for ModelDB was to create an envi-



head1_Vm
max y = 25.1 at 3.4ms
head2_Vm
max y = 46.2 at 2.5ms
head3_Vm
max y = 25.8 at 3.4ms
head4_Vm
max y = 23.3 at 3.5ms
dend2_Vm
max y = 12.2 at 3.9ms
dend4_Vm
max y = 3.3 at 3.9ms
dend8_Vm
max y = -54.7 at 5.3ms

Figure 6 Viewing results from a run. A helper application is used to graph the voltages in the spine heads and in three of the main dendritic branch compartments.

ronment that satisfied an immediate set of needs for creating models. ModelDB was also designed so that its capabilities could be easily extended along a number of fronts. These included extending the number of supported simulators, providing complex search mechanisms for models and runs, and, finally, in the context of the Human Brain Project, incorporating this database into a set of interrelated databases that allow information to be shared. The first two of these extensions have been discussed within the context of this paper and are fairly straightforward; the last extension is more complex and merits further discussion.

Linking Multiple Databases

The Web provides a simple mechanism for representing knowledge as links between related material. These links can be predefined database queries. As the system is expanded, links will be used to connect the model and its parameters to other databases currently under construction (Fig. 7A). These databases include citation and researcher databases. This simple linking scheme could provide, for example, a list of citations and their abstracts relevant to a particular value chosen for one of the model parameters. It could lead to contact information for researchers working on this aspect of the system.

In the next stage of development, links will be used to reference relevant data in experimental databases (Fig. 7B). Because data from the models and experiments can be coerced into a common datatype, these databases could share a number of features. One of these shared features could be a common set of tools for display and analysis purposes that could be accessed via the browsers by defining Multipurpose Internet Mail Extension (MIME) types for the data. Further extension of capabilities will be possible by using Illustra's support for object-oriented database design. This support allows new data types to be defined along with corresponding functions for operating on those types. Incorporating aspects of current analysis tools into these functions will allow sophisticated searches based on analysis of model and/or experimental data. For example, one might search for model and experimental data in which most of the energy in the spike train is at a spike frequency of between 30 and 50 Hz, a frequency range that may be involved in feature binding.[40]

This proposed system is a fairly straightforward step from our current one. If there is one lesson to be learned from the Web it is that the best way to collect and connect large quantities of data is to distribute the task over many users. Our key focus therefore will be to develop frameworks that will aid the development of these databases and subsequent knowledge representations.

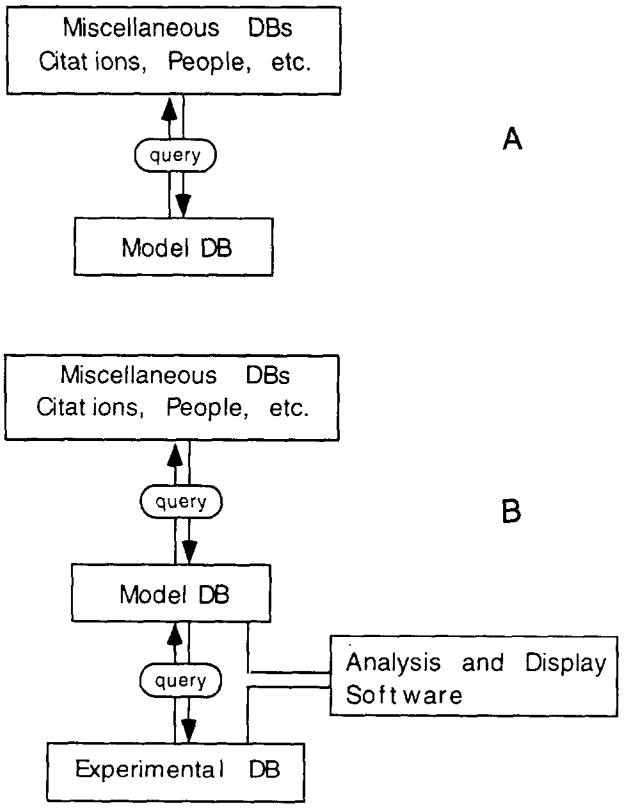


Figure 7 An overview of the progressive development of multiple databases proposed as a future research direction (see text). These databases will be extensively interlinked to help produce a useful research and clinical tool.

The Direction of the Human Brain Project

Within the Human Brain Project, the different laboratory groups share many common goals and strategies for reaching them, but there are differences in the details. At this early stage in the project, a broad overlap in general design with differing details of implementation is a healthy means for the research community to weigh the strengths and weaknesses of each approach.

In the second stage of this project, it will be important for standards to be created so that systems, tools, and data can be consolidated. This approach will allow a critical mass of information to accumulate, leading to the formation of a truly useful set of databases for both the research and clinical communities.

Summary

We have created an environment for building and running models within a database context. The database provides the means for storing and searching for models and their results in a structured manner. It also provides the means for ensuring that all results in the database are associated with the model that created them. The environment uses a Web interface that provides machine, simulator, and location independence. This independence simplifies the organization of collaborative modeling efforts. Because models encapsulate what is known about a system, this database provides an initial focus to which information from experimental and citation databases can be linked. In the future, a more abstract representation of the knowledge of systems under study will be used to bridge the modeling databases and these other databases. This system will provide a powerful tool for synthesizing what is known and exposing what is not.

References ■

1. Huerta MF, Koslow SH, Leshner AI. The Human Brain Project: an international resource. TINS. 1993;16:436–8.
2. Bloom FE, Young WG. New solutions for neuroscience communications are still needed. Prog Brain Res. 1994;100:275–81.
3. Lowe HJ, Lomax EC, Polonkey SE. The World Wide Web: A review of an emerging internet-based technology for the distribution of biomedical information. J Am Med Inform Assoc. 1996;3:1–14.
4. Pechura CM, Martin JB (eds). Mapping the Brain and its Functions. Integrating Enabling Technologies into Neuroscience Research. Washington, DC: National Academy Press, 1991.
5. http://www-hbp-np.scripps.edu/
6. Rall W. Theoretical significance of dendritic trees and motoneuron input–output relations. In: Reiss RF (ed). Neural Theory and Modeling. Stanford, CA: Stanford University Press, 1964.
7. Rall W. Distinguishing theoretical synaptic potentials computed for different somadendritic distributions of synaptic input. J Neurophysiol. 1967;30:1138–68.
8. Rall W, Shepherd GM. Theoretical reconstruction of field potentials and dendrodendritic synaptic interactions in the olfactory bulb. J Neurophysiol. 1968;31:884–915.
9. Traub RD. Motoneurons of different geometry and the size principle. Biol Cybern. 1977;25:163–76.
10. Pellionisz A, Llinas R, Perkel DH. A computer model of the cerebellar cortex of the frog. Neuroscience. 1977;2:19–35.
11. Traub RD, Llinas R. Hippocampal pyramidal cells: significance of dendritic ionic conductances for neuronal function and epileptogenesis. J Neurophysiol. 1979;59:1352–76.
12. MacGregor RJ. Neural and Brain Modeling. San Diego, CA: Academic Press Inc., 1987.
13. Shepherd GM, Brayton RK. Computer simulation of a dendrodendritic synaptic circuit for self- and lateral-inhibition in the olfactory bulb. Brain Res. 1979;175:377–82.
14. Carnevale NT, Woolf TB, Shepherd GM. Neuron simulations with SABER. J Neurosci Met. 1990;33:135–48.
15. Hines M. A program for simulation of nerve equations with branching geometries. Int. J Biomed Comput. 1989;24:55–68.
16. Hines M. NEURON—A program for simulation of nerve equations. In: Eeckman F (ed). Neural Systems: Analysis and Modeling. Norwell, MA: Kluwer Academic Publishers, 1993:127–36.
17. Hines M. The NEURON simulation program. In: Skrzypek J (ed). Neural Network Simulation Environments. Norwell, MA: Kluwer Academic Publishers, 1993.
18. http://www.neuro.duke.edu/neuron/home.html
19. Wilson MA, Bower JM. The simulation of large-scale neural networks. In: Koch C, Segev I (eds). Methods in Neuronal Modeling: From Synapses to Networks. Cambridge, MA: The MIT Press, 1989.
20. Bower JM, Beeman D. The Book of Genesis. New York: Springer-Verlag, 1995.
21. http://www.bbb.caltech.edu/GENESIS
22. Peterson BE, Gale EA, Jensen RV, Shepherd GM. Models of dendritic spines containing active conductances display complex temporal processing capabilities. Soc Neurosci Abs. 1995;21:376.5.
23. Hille B. Ionic Channels of Excitable Membranes. Sunderland, MA: Sinauer Associates, 1984.
24. Jack JJB, Noble D, Tsien RW. Electric Current Flow in Excitable Cells, 2nd ed. Oxford: Clarendon Press, 1975.
25. Johnson D, Wu SM. Foundations of Cellular Neurophysiology. Cambridge, MA: The MIT Press, 1995.
26. Shepherd GM, Brayton RK, Miller JP, Segev I, Rinzel J, Rall W. Signal enhancement in distal cortical dendrites by means of interactions between active dendritic spines. Proc Natl Acad Sci U S A. 1985;82:2192–5.
27. Shepherd GM, Brayton RK. Logic operations are properties of computer-simulated interactions between excitable dendritic spines. Neuroscience. 1987;21:151–65.
28. Shepherd GM, Woolf TB, Carnevale NT. Comparisons between active properties of distal dendritic branches and spines: implications for neuronal computations. Journal of Cognitive Neuroscience. 1989;1:273–86.
29. Friedman C, Hripcsak G, Johnson S, Cimino J, Clayton P. A generalized relational schema for an integrated clinical patient database. SCAMC Proc. 1990;335–9.
30. Johnson S, Cimino J, Friedman C, Hripcsak G, Clayton P. Using metadata to integrate medical knowledge in a clinical information system. SCAMC Proc. 1990;340–4.
31. Winston PH. LISP, 2nd ed. Reading, MA: Addison-Wesley, 1984.
32. http://www.illustra.com/
33. Rall W. Perspectives on neuronal modeling. In: Binder MD, Mendell LM (eds). The Segmental Motor System. New York: Oxford University Press, 1990; 129–49.
34. Shepherd GM. Canonical neurons and their computational organization. In: McKenna T, Davis J, Zornetzer SF (eds). Single Neuron Computation. Cambridge MA: MIT Press, 1992;27–59.
35. van Hoff A, Shaio S, Starbuck O. Hooked on Java. New York: Addison-Wesley, 1996.
36. van Hoff A. Java and Internet programming: similar to C and C++ but much simpler. Dr. Dobb's Journal. 1995;20:56–61.
37. http://java.sun.com/
38. http://naviserver.navisoft.com/
39. ftp://ftp.cc.umr.edu/pub/cgi/cgiwrap/
40. Singer W, Gray CM. Visual feature integration and the temporal correlation hypothesis. Annu Rev Neurosci. 1995;18:555–86.