WEB SIMULATION OF BRAIN MODELS

Amanda Alexander and Michael Arbib
Brain Simulation Lab
USC Brain Project
University of Southern California
Los Angeles, CA 90089-2520
e-mail: aalx@java.usc.edu,
arbib@pollux2.usc.edu

Alfredo Weitzenfeld
Instituto Technologico Autonomo de Mexico
Rio Hondo 1
San Angel, C.P. 01000 Mexico D.F.
e-mail: alfredo@lamport.rhon.itam.mx

KEYWORDS

Continuous simulation, Hierarchical, Neural networks, Neuroscience, Java

ABSTRACT

In 1994, we set out with a grant from the Human Brain Project to develop a database of exemplary models that would be freely accessible from the Web. In achieving this goal, we prototyped the Brain Models on the Web database (BMW) and the Summary Database (SDB). BMW contains models in an Informix (trademark) database with links into the Summary Database which contains empirical neuroscience information to support the model: published literature on experiments, summaries of articles, etc. To make it easier and faster for the modeler to create new models, we created a schematic capture system that allows the modeler to "drag-and-drop" icons/modules on a page and interconnect them. Being able to see the schematic of the model will help the experimentalist understand the model better and contribute to the model delineation. We also have created several interface modules which mimic many of the experimental protocols used by experimentalists. With a rich set of modules available, the modeler will only need to develop those features unique to his model. We have also created a "Web friendly" simulator we call NSL. NSL is a hierarchical simulator, completely written in java and also available in C++. It provides a wide variety of output displays, reports, and soon will provide parallel processing support.

INTRODUCTION

The Neural Simulation Language (NSL) provides a modeling and simulation environment for large scale, general purpose neural networks of the brain, and Brain Models on the Web (BMW) provides a repository of exemplary models. Because of the need to create “living” models, that is models that change over time to reflect advances in research or better experimental data, it was decided that NSL3.0 should be web based so that modelers from around the world could run and comment on models in the BMW database easily. Researchers are free to add on to and modify existing models, and if granted access to BMW, they can upload their models to the database (Arbib 1995). For example, modelers involved with research into Huntington’s disease may want to examine the Cortico-Subcortical Model for the Generation of Spatially Accurate Sequential Saccades (Dominey and Arbib, 1992) which models the “distractibility” syndrome common in Huntington’s disease patients. Or researchers of Parkinson’s disease may want to modify the levels of the neurochemical, dopamine, represented in the Basal Ganglia Model (Crowley, 1997) to see what effects it has on the oculomotor system. Also, when experimenting on a model, you can control how a drug is affecting the system, and then apply that knowledge to what happens in an actual empirical experiment. Eventually, the continuous updating of a model will cause the model to become more and more accurate and will reduce the need for empirical testing. Performing drug experiments on animals is expensive, but performing drug experiments on models is comparatively cheap.

To be completely compatible with almost all computer platforms available to researchers, the USC Brain Project has written NSL3.0 completely in Java; a companion effort at ITAM has written NSL3.0 in C++. We will focus here on the Java version which contains a schematic capture system, a preprocessor/compiler, a simulator, a scripting language and a post processor (see figure 2). In all, it contains more than 94,000 lines of Java code. Fortunately for our sponsors, we only wrote two thirds of it from scratch. The grammar for the compiler came from “guavac” from Effective Edge Technologies (trademark), the compiler generator (FLEX, BISON, and JBF) came from GNU. The Tool Command Language (TCL) scripting software, originally by John Ousterhout, was implemented in Java by Ioi Lamb from Cornell (JACL). Because all of these free packages were written in Java, it was relatively easy to integrate them into NSL.

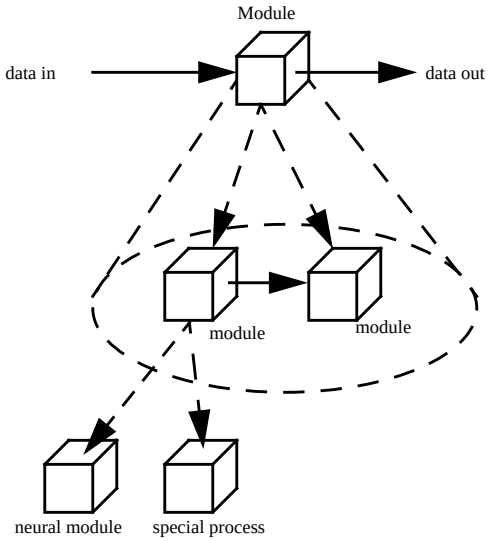

Figure 1: The NSL hierarchical computational model is based on modules to be implemented by neural networks or other neural modules in NSL.

To create a model, researchers must create "black box" objects we call modules. The structure of each model is hierarchical: characterized by a large module which can be broken down into smaller modules, connected by ports (see figure 1). These modules may in turn be further decomposed until we reach the leaf modules, which are either taken from a module library, or are new modules written directly in NSL.

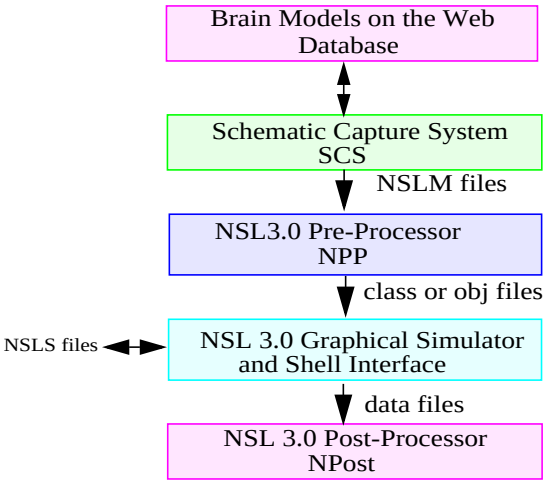

Figure 2: NSL System Processes - Schematic Capture, Pre-Processor, Simulator, and Post- Processor

As shown in figure 2, there are several steps in creating and running a model. To define a model, we first use the Schematic Capture System's (SCS) Schematic Editor to create the structure of the model and SCS's Language Sensitive Text Editor to define the leaf modules. Next we use SCS's NSLJ Generator to create the NSLM files and the Nsl_Link file to compile them. After that we invoke the Nsl_Link file, which calls the NSL Pre-Processor, NPP, which generates all of the model class files used by the simulator. Finally, we run the simulator either as an application or as an applet over the web. If we run the simulator as an applet, we can only view the output while the simulator is running due to the security restrictions placed on applets. If we run the simulator as an application, we can save the output in a variety of different formats: Postscript, NSLJ data files, and Matlab (trademark). In the future, we plan to have a postprocessing system that will allow you to plot and analyze the data files at a later date.

REQUIREMENTS FOR EXECUTING A MODEL

We should note here that all of the NSL tools require Sun Microsystems' "write once, run everywhere" java virtual machine. (Note: the virtual machine is available in the Java Development Kit from Sun which can be found at: "http://www.javasoft.com/products/jdk/1.1/index.html".) Currently, java runs on 18 platforms including Suns, Macintoshs, and PCs (Windows95 and Windows NT). The simulator can be executed as either a application or an applet. In either case, NSLJ models run faster if executed with one of the Just-In-Time (JIT) compilers available with most Web browsers.

RUNNING A BMW MODEL FROM THE WEB

Let us assume that we are visiting the BMW site http://www-hbp.usc.edu/HBP/bmw, and we want to execute the Cortico-Subcortical Model for Generation of Spatially Accurate Sequential Saccades (Dominey and Arbib 92) over the Internet using our favorite browser. First we would select it from a list of available exemplary models sorted according to conformance grade, simulator type, species, anatomy, task, author, name, and date.

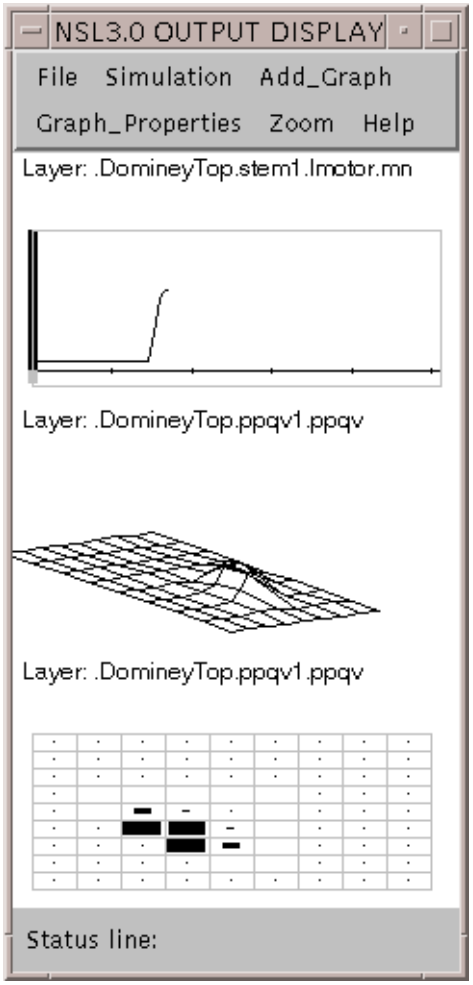

Figure 3. The Dominey Simulation Results

In order to run the simulation as an applet or as an application, we type "run" from the Nsl Shell or select "Run" from the Simulation menu. The "run" command causes all of the modules in the simulator's scheduler to be evaluated cyclically until the "end time" is reached (sequential simulation). Upon reaching this "end time" we would find simulation results similar to those displayed in figure 3.

In figure 3, we see the output of the double saccade experiment where we model a monkey that has been trained to stare at a light in the center of a grid; once the center light disappears then the monkey is free to move his eyes first to one target and then another. We have chosen to display several neural layers. The first one is the output tracings of some sample motor neurons (mn) firing in the brainstem; one for each direction. (To save space we only show the left motor neuron's output.) These motor neurons guide the eyes to new targets in the visual field. We display the quasi-visual cells from Lateral Inter Parietal Cortex within the Posterior Parietal (PPqv). The quasi-visual cells reflect where the eyes need to go next relative to where they are staring currently on the grid (Dominey and Arbib, 1992). We should note that all of the plots are update dynamically as the simulation is running; thus you can actually see the eyes chasing the targets as the simu-

lation progresses. The first plot is a temporal plot, the second is a 3 dimensional spatial plot, and the third is an area level graph.

Although the "Cortico-Subcortical Model" is a good example of what NSL is capable of, it is just one of many types of models that NSL can encode. In general, it is very flexible and can be use to simulate any type of modular network requiring continuous updating of information.

THE SCHEMATIC CAPTURE SYSTEM

We have already seen an example of how our Schematic Capture System, SCS, facilitates the creation of modular, hierarchical neural networks. We now discuss some general properties of SCS. The benefit of having a schematic capture system is that modules can be stored in libraries and easily accessed by the schematic capture system. The modeler can use the graphical user interface to easily connect icons representing the modules to form a neural network. SCS also provides the following capabilities:

•The ability to create modules using a text editor

•The ability to create icons that represent complex or simple modules

•The ability to connect icons together in a schematic

•The ability to encapsulate new schematics as new modules

•The ability to version new modules

•The ability to traverse hierarchy

•The ability to download and upload models to and from BMW

•The ability to interface with the Summary Database

Currently, the Schematic Capture System consists of seven subsystems: the Schematic Editor, the Icon Editor, the Language Sensitive Text Editor, the Simulator/Nsl_Link Script File Generator, the Consistency Checker, the BMW interface, and the SDB interface.

The Schematic Editor is responsible for building the whole schematic structure of the model (see figure 4) and acts as the control window for SCS allowing access to the other tools in the SCS system. The Schematic Editor also provides access to the libraries that SCS maintains It lets the user decide whether they want to use fixed or floating versions of a module. If the user chooses "float", then every time they enter SCS they will see the most recent version of that module; if the user chooses "fixed", then they will always see the version that existed at the time they first created their schematic.

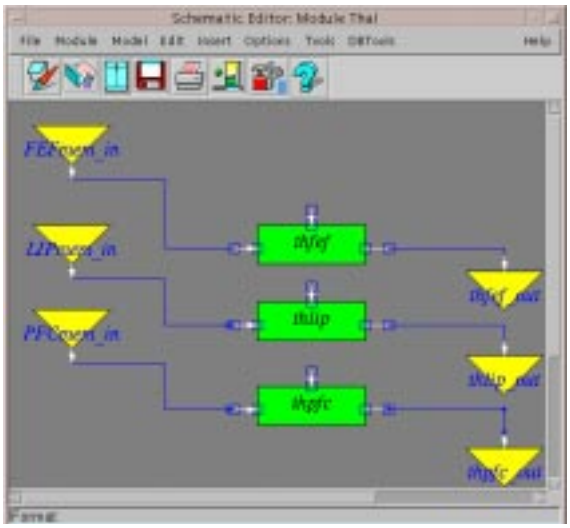

Figure 4: The Schematic Editor

•The Icon Editor is responsible for building the graphical appearance of the individual icons (see figure 5). It also reads and writes icons to the library of modules.

•The Language Sensitive Text Editor is responsible for building the module in NSLM.

•The NSLJ Generator generates NSLM code and Nsl_Link script files for compiling the model. It also calls the Consistency Checker to keep track of the model's version at the time of creation.

•The Consistency Checker is responsible for keeping track of the versions of the modules that the model contains and checking that the ports from one level match those of the next level. The Consistency Checker is called every time a model, module, or icon is saved.

•The BMW Interface is responsible for downloading models and modules over the internet, as well as, providing support for uploading and browsing (see the Database Tools Menu in figure 6).

•The SDB Interface allows the user to connect objects (modules and interconnect) on their schematics to the Summary Database (see figure 6). When in "Image Map Display" mode, selecting modules causes queries to be made to the SDB and the results to appear on the user's screen. As an example, if the user clicked on a module representing the brainstem, then all of the "summaries" supporting (or opposing) the modeler's implementation of the brainstem would be retrieved from the SDB and displayed.

Figure 5: The Icon Editor

Figure 6: The Database Tools Menu within the Schematic Capture System

MODELING IN NSL

We earlier saw that the structure of a NSL model is hierarchical (see figure 1). At the top level of the hierarchy, we have the model. At the leaf level of the hierarchy, we have modules representing neurons (see figure 7), parts of neurons, or layers of neurons (see figure 8); and in between, we have modules of modules.

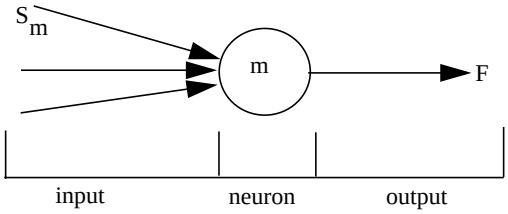

Figure 7: A single-compartment neuron, with one output and many inputs. Its internal state is described by a single scalar quantity called the membrane potential m; the output is described by another single scalar called the firing rate F.

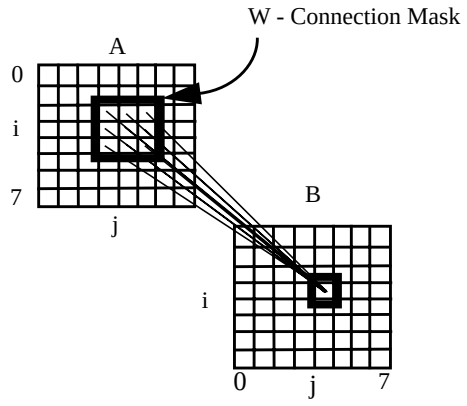

Figure 8: An example of a connection mask, here shown providing the synaptic weights for the connections from array A to a typical neuron of array B.

In many of our models we use arrays of Leak-Integrator neuron-models, interconnected by masks which represent regular patterns of neuron interconnection via synaptic weights (Weitzenfeld 1995). We typically use one array to represent the membrane potential and another to represent the average firing rate of a given set of neurons. The computational advantage of neural arrays and interconnection masks make them a clear choice for many models. Instead of describing neurons on a one by one basis, an array can be described as a single unit, while the connections between arrays can be described by a mask storing synaptic weights.

An interconnection among neurons would then be processed by computing a spatial convolution of a mask and an array. For example, as shown in figure 8, if A represents an array of outputs from one array of neurons, and B represents an array of inputs to another array, and if the mask W(k,l) (for -d<=k,l<=d) represents the synaptic weight from the A(i+k,j+l) (for -d<=k,l<=d) elements to B(i,j) element for each i and j, we then have

$$B(i, j) = \sum_{k=-d}^{d} \left(\sum_{l=-d}^{d} W(k, l) A(i + k, j + l) \right)$$

which can be described by a simple expression in the NSLM language as:

B = W@A.

The short hand notation "@" represents convolution and is easily expanded by the simulator into the more complex expression (Arbib, 1989).

THE NSL SIMULATOR

NSL is a flexible, neural network simulator that can be used to simulate a variety of biologically realistic as well as artificial neural networks. The core system provides a scheduler that allows for different time steps and hierarchical processing of modules. The core system also provides a library of common neural network math functions. The interactive system includes graphical plotting capabilities such as temporal plots, spatial plots, area level graphs, and zoom/unzoom, as well as a rich collection of input widgets that can be linked together to mimic experimental protocols. To understand the simulator better we will examine the architecture, which is made up of several processes designed and implemented in an object-oriented fashion.

The main system contains four processing threads (see figure 9):

•The Control Thread where modules are loaded and model control takes place.

•The Scheduler Thread where both executable and graphical modules are executed.

•The Display Frame Window Interface where all graphics interaction takes place.

•The Script Interpreter where the NSL Script (NSLS) language is interpreted. NSLS typically defines the simulation environment, specifies the control commands, and assigns the model input and display parameters.

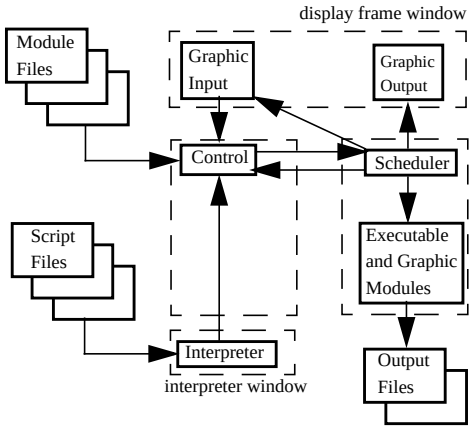

Figure 9: NSL Processor Threads: Control, Scheduler, Interpreter Window, and Display Frame Window

FUTURE WORK

Currently, the models in BMW have been generated at USC. In the coming year, we propose to open it to models developed elsewhere, as follows. Once a model has been created, a researcher can submit it to the BMW Review board. First the researcher sends e-mail to the BMW database master for a "one time" password to access the BMW ftp site. After placing it at the ftp site, it is evaluated by the board for compliance with the BMW model and documentation guidelines; as well as for originality, accuracy, and substance. Once reviewed it receives a "bronze" grade if it is downloadable and executable from the ftp site and all of the documentation exists; a "silver" grade if it is able to be stored in the Informix Database Management System with its documentation, and is executable from the Web; and a "gold" grade if it is able to be stored in the Informix Database Management System with documentation, is executable from the web, and has links from the BMW database to other USC Brain Project databases such as the Anatomy Database (NeuArt), the Collection of Experimental

Data database (TSDB), or the Summary Database (SDB). More than likely it will have links to the Summary database since the SDB provides the supporting material for the model. Once graded, the model will be placed in the appropriate location within BMW and will be accessible by the world from the Web via the Informix's Web Datablade.

CONCLUSIONS

Providing exemplary neural models of the brain to the experimentalist on a wide variety of computer platforms would not be economically possible without the World Wide Web and Java. While the Web now makes it easier for an experimentalist to find data related to their field of study, BMW is being developed to bring distant neural researchers together into one virtual laboratory allowing them to run experiments on the same model and comment on the findings. By implementing the NSL simulator in Java we were able to provide the experimentalist with almost real-time results over the Web, without having to support a different version of the simulator for every platform and operating system currently in use. But by providing databases such as BMW, NeuArt (Anatomy), the Empirical Experimental Database (TSDB), and the Summary Database (SDB), we are able to increase the time and energy an experimentalist can devote to his experiments by reducing the time required to conduct background research. The USC Brain Project is an example of how the World Wide Web and Java can be used to accelerate the advancement of a field of research by enabling better communication and collaboration within the research community.

CREDITS

NSL was first written in 1990 by Alfredo Weitzenfeld, working with Michael Arbib at the University of Southern California. The present version, NSL 3.0, has been developed so that it compiles into Java (developed at USC in the US by Amanda Alexander's team) or C++ (developed at ITAM in Mexico by Alfredo Weitzenfeld's team). NSL in Java has been developed under a grant from the NIH Human Brain Project while SCS has been developed by a grant from CONACyT and NSF. MIT Press will be publishing "The NSL Book" written by Arbib, Weitzenfeld, and Alexander spring 1999. For more information on our project please visit our web page at "http://www-hbp.usc.edu/~nsl/unprotected".

UNIVERSAL RESOURCE LOCATORS (URLS)

A postscript copy of this document exists at: "ftp://www-coffee.usc.edu/pub/nsl/websim99". A hypertext copy of this document exists at: "http://www-hbp.usc.edu/~nsl/unprotected/websim99. The root NSLC web site can be found at: "http://cannes.rhon.itam.mx/Alfredo/English/ . The root NSLJ web site can be found at: "http://www-hbp.usc.edu/~nsl/unprotected".

RECOMMENDED READING

Brown, M., etal.,1995, *Using Netscape 2*, Que Corp.

Cornell, G., and Horstmann, C., 1996, *Core Java*, SunSoft Press.

Ousterhout, J., 1994, *Tcl and the Tk Toolkit*, Addison-Wesley.Eds

Tyma, P., Torok, G., and Downing, T., 1996, *Java Primer Plus*, The Waite Group.

BIBLIOGRAPHY

Arbib, M.A., 1989, *The Metaphorical Brain 2: Neural Networks and Beyond*, Wiley Interscience, pp. 124-126.

Arbib, M.A., 1995, Brain Models on the Web, in *Computational Intelligence, A Dynamic Systems Perspective*, (M. Palaniswami, Y. Attikouzel, R.J. Marks II, D. Fogel, and T. Fukunda, Eds.), New Your: IEEE Press, pp. 219-231.

Crowley, M., 1997, *Neuromodulation in Basal Ganglia Plasticity for Visuomotor Coordination*, USC Computer Science Department, Phd Thesis.

Dominey, P. and Arbib, M., 1992, Cortico-Subcortical Model for Generation of Spatially Accurate Sequential Saccades, *Cerebral Cortex, 2:153-175.*

Weitzenfeld, A., 1995, NSL Neural Simulation Language, *The Handbook of Brain Theory and Neural Networks*, (M.A. Arbib, Ed.), Bradford Books/MIT Press, pp. 654-658.