A Survey of compartmental modelling packages

Olga Kroupina and Raúl Rojas
Free University of Berlin, Institute of Computer Science
Takustr. 9, D-14195 Berlin, Germany
{*krupina*|*rojas*}*@inf.fu-berlin.de*

Technical Report B-04-08

June 8, 2004

Abstract

Computer simulations allow researchers to explore processes underlying neural functions. Biologically detailed simulations of single neurons and neural networks are based on compartmental modelling. This article describes seven well-known compartmental modelling packages and compares their characteristics. The important aspects of selecting a program are discussed.

1 Introduction

Recently, there has been a dramatic increase in the number of neurobiologists using computational methods as an aid to their empirical studies. As more experimental data is accumulated, it becomes clear that detailed physiological data alone is not enough to infer how neural circuits work. A quantitative modelling approach helps to understand the functional consequences of particular neural elements. Therefore, computer simulations are an important tool for neuroscience research.

Biologically detailed simulations of single neurons and neural networks are typically based on the approach of compartmental modelling; that is, each cell is divided into many isopotential compartments, joined by conductances, and activated via simulated ion channels and current stimuli. Each compartment is then modelled with equations describing electrical currents [16]. The advantage of this approach is that it places no restrictions on the membrane properties. The influence of the diversity of activated channels

can be taken into account by including additional equations describing the channel dynamics. The mathematical result of this approach is a system of ordinary differential equations, one for either a compartment or for an ionic channel.

For modelling this on the computer, one needs to solve the equations arising from model elements in parallel.

2 General versus special-purpose simulators

Before starting the modelling process everyone has a problem of selecting the most suitable simulation system. The most important feature of each package in the diversity of existing packages is what particular physiological data can be simulated and how easily the model can be extended physiologically to specific data sets. The physiological characteristics available for modelling are the focus of attention for both authors when creating a program and modelers when using it.

By learning the widespread packages one gets to know that they have a general domain of application. The well-known packages, such as GENESIS [3] and NEURON [7], were developed as an attempt to create a "general" simulation system. That means that they must be capable of solving problems at many different levels of detail (i.e., parts of neurons to large neural systems). The problems arising from that approach range from sub-cellular components to whole networks of cells and systems-level models.

One can try to create the simulation for particular purpose. Dedicated applications, if written well, would put the computational power and memory of the computer to optimal use.

However, general simulation systems can provide many important advantages over dedicated code. A well-designed neural simulator can provide very good performance, even when compared to dedicated code. Many advantages result in a dramatic speedup in the process of building and expanding models. Almost always, it takes much more time to build a model than to learn the modelling techniques and actually simulate the problem.

In the first stage of simulation one needs time to learn how to use the programs. Despite considerable efforts to improve the user interface, the simulation system demand time to master the models. In case of the project for more than one year I would recommend to spend time in learning the packages as GENESIS [3] or NEURON [7].

The most important advantage of general simulation systems is their extensibility. Firstly, it enables the user to add new elements to the simula-

tion without the requirement to rewrite all the existing code, as is often the case with dedicated simulations. Secondly, previously developed and tested elements can be used later in other simulations. This will save lots of time developing new simulations.

As well as speeding up the development new simulations, using general simulation systems can be made more accurate by implementing specially developed methods and techniques. That includes new integration technologies or client-server architecture. Some "general" simulation systems use new simulation-related technology like parallel computing.

As mentioned above, use of well-known packages gives one the possibility to reuse previously designed simulations. It can be later developed into a form of neural database, which not only represents structural information about the nervous system, but does so in such a way that structural details are functionally organized. For example, the NEURON User's Group was established to "share useful tips with other NEURON users, learn about the latest revisions and program features". SenseLab Project was created to develop the models of neurons and neural systems, based on the model of the olfactory pathway. It consists of a database for collecting and analysing the neuroscientific information. The GENESIS users group, BABEL, maintains a database of published simulations. Thus, since general simulation systems have extensive user, and communication between those users results in the accumulation of more and more information about the nervous system.

However, the general domain of nerve simulation is still too large for any single program to optimally deal with every problem. In practice, programs have their origin in the attempt to solve restricted classes of problems. The packages for special kinds of problems exist along with "general" simulation software and can be very effective in particular cases. For example, the packages for simulation of single neurons and small networks have not lost significance and can be useful for simulation models with a detailed morphology of cells.

3 Important aspects of selecting a program

In the first step of discussing what package is appropriate for modelling, it is important to define a problem of modelling and check whether the simulation system's scope is broad enough to handle the modelling project. In the previous paragraph it was mentioned that the simulation system at first could be characterized by the physiological elements possible to simulate.

All the packages I will describe can model passive membrane models, the

difficulties can arise whenever they are able to simulate ionic channels, and extended gated channels depended on ionic concentrations, plasticity and another point processes.

Most of the "general" simulation systems are extended on the domain of simulation large neural networks, but they often require very large computational resources from the high-powered computers because of the huge number of variables included in the simulation. Most of the packages are based on compartmental modelling, which is derived from cable theory by replacing continuous equations with a set of ordinary differential equations. Thus, it is possible to accurately simulate the morphology of a cell, whereas the differential equations describing the process of simulation included also the ionic and synaptic channels.

The portability of simulation systems plays an important role. These are often Unix-based simulation packages designed to run also under X-Windows. Problems can arise already in the phase of installation for users with not enough IT-experience and then later in the process of its usage. Not every simulation system developed for Unix can be successfully run on any computer with a Unix system on it.

Along with this one should realize that most of the time spent on typical models involves constructing and adjusting the simulation process. Therefore, one of the next important features of a particular software is how the user interface is organized. Very often each package has its own interpreted scripting language, in which users define the components and running parameters of their simulations. Script interpreters offer a very flexible interface; the simulation can be started both from the command prompt and by reading the model description from scripting files. The advantage of packages with these features is that they allow interactive control the model. But these packages require learning the script language and programming skills. Another group of software reads the complete model description from files but allows little interaction during run-time.

"General" simulation systems often do not provide a Graphical User Interface (GUI) or have a very simple one and as a result they can't visually represent the simulation process. A wide spectrum of applications makes it difficult to construct a GUI for each model. The complexity of the model (sometimes thousands of compartments and channels in each cell) requires flexible modes of specifying selections of single compartments or groups of compartments for editing and display. GENESIS provides the possibility to construct the user interface using a set of graphical modules, which are the same as computational routines from the user's point of view. So one needs time and perseverance to build the appropriate interface along with

the simulation itself. NEURON provides a GUI only for primal research by setting parameters, control of voltage and current stimuli, and graphical presentation of the results as a function of time and position.

Programs developed for a particular task often have a user-friendly interface. One can try to find between packages, which offer more accessible user interfaces, this one supporting modelling the setting task.

Any simulation system assuredly should be user-extensible to allow the incorporation of new modelling efforts. It depends on the following characteristics: the modelling language, modular design, and open-ended code. Most systems use an object-oriented approach, which enables the user to easily add new modules to extend the system for a particular application. The modular design means that there are libraries or databases of standard simulation components, which can be chosen and used for quickly constructing new simulations.

Of course, the available integration techniques are a significant property of simulation systems. The methods for solving neural equations describing circuit spread in the cells can range from explicit methods to highly implicit methods. The explicit forward Euler method is ease to implement and takes less computational resources, but it can be unstable in some conditions. In contrast, implicit methods take a lot of extra work but are more accurate and more stable. Most "general" simulation systems use the implicit integration method based on the one developed by Hines, which is most effective for detailed cell models that contain many compartments. The adaptive step size used in the numerical integration of the differential equations which describe the model, can have an influence on speed and accuracy of the calculations. The possibility for the user to define the step size of integration and to interactively control it could be useful in some cases.

In the case of a model with a huge number of neurons, or with a complicated structure of activated channels, the available computer resources can restrict the choice of the simulation program. Recently developed architectural techniques like client-server structure or parallel computing can be a solution for this.

The client-server architecture involves the separation of the program into two parts: an equation solver and a simulation controller. The equation solver takes virtually all the computer time and the algorithms never change within a class of problems. The simulation controller, which must manage a large number of parameters, runs the simulation, displays the results, and continuously changes during the investigation of the problem. It follows that the separation of these parts into independent blocks would be effective in using computer resources.

Some packages can be ported to "parallel" computers (computers with many processors running simultaneously) that are supposed to give a huge increase in speed over standard single-processor computer designs. Firstly, they can increase execution speed, but there is a trade-off as increasing the number of computing processors also increases the communication overhead between these processors. Secondly, because the model is distributed over a lot of processors' memory, huge models can be implemented.

4 Compartmental modelling packages

In this section I will go through those simulation packages that are extensively used by a wide groups of users, and compare their characteristics by the discussed above.

4.1 NEURON

Detailed morphology: sections versus compartments. NEURON was developed on the basis of the neural system CABLE for simulation "nerve equations with cable geometry". As a result, it has imitated the basic characteristics of CABLE [5].

The most important feature of NEURON setting it apart from other simulation systems is that it can effectively deal with problems where "the cable properties play a crucial role" [6]. Although NEURON assumes a spatial discretization as a basis as all well-known simulation systems do. It does not deal with "compartments". The main component of NEURON is the "section" of the continuous cable, anatomical and biophysical parameters being the functions of the positions along the cable. Each section is ultimately discretized into compartments (determined by the parameter *nseg*, which specifies the number of nodes at which solutions are computed). Sections are connected to form any kind of branched tree structure. The particular approach used by NEURON is special insofar as the location of the voltage is not at the edge of a segment but rather at its center and an extra voltage node having zero area is located at the end of the section.

Most cellular properties depending on the position along the length of a section are specified in terms of a continuous parameter ranging from 0 to 1 (normalized distance). Another strategy used by NEURON that helps the user define which section is intended is the new "range" variables. Properties are specified with the syntax *rangevar(xmin:xmax)=e1:e2*. The position expressions must meet the constraint *0<=xmin<=xmax<=* 1 and have values between *e1* and *e2*.

The interpreter. The user interface has the same style as the previous simulation system CABLE. Creating cable sections, specifying parameters and controling of the simulation can be performed via a C-like interpreter called HOC, which was extended by the addition of object-oriented syntax that can be used to implement abstract data types and data encapsulation. NEURON provides a built-in implementation of the microemacs text editor, as well as it can accept HOC code in form of ASCII files.

Graphical User Interface. The default graphical interface in NEURON, which is developed as a public domain of a C++ class library, is suitable for the primal research by setting parameters, control of voltage and current stimuli, and graphical presentation of the results as a function of time and position. For more general problems these operations could be effected by writing the procedural statements to an interpreter.

A practical example. Here I will show how to construct the simple neuron composed of a soma and two outgoing dendrite cables. Soma contains the ionic channels with Hodgkin-Huxley dynamics, whereas the dendrite has a passive channel. As a first step one should establish a model topology and specify the morphological properties, as in the following HOC code:

```
create soma, dendrite[2]
for i=0,1 {connect dendrite[I](0),soma(1)}
          forall Ra=35.4
                  soma {nseg=1
                        L=30       //length , micron
                        diam=30  //diam , micron
                        insert hh  //hh channel
                        gnabar_hh=0.5*0.120 }
for i=0,1 dendrite[i]  {nseg=5
                        L=100
                        diam=2
                        diam(0:1)=10:3
                        insert pas //passive channel
                        e_pas=-65 //equilibrium potential
                        g_pas=0.001
                        }
```

As was noticed above, the parts of neuron have a parameter *nseg* and all parameters are functions of x, e.g. the position of connection (dendrites are located on the left side of the soma). This example also demonstrates the

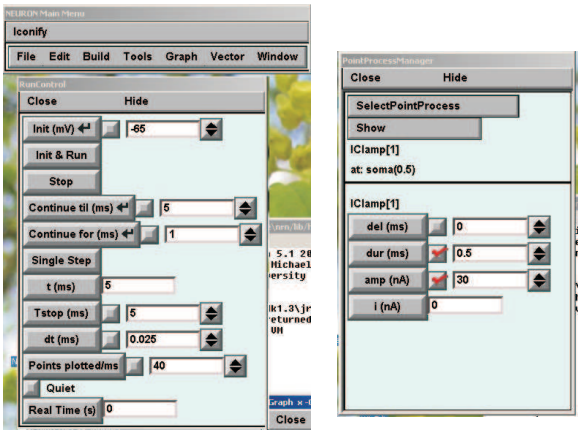

Figure 1: Run Control window of NEURON for executing and controlling the simulation; setting up VClamp from PointManager window.

use of the range variables by the definition of the diameter a of dendrite's segments.

This code can be saved in the `*.hoc` file and later be loaded via the graphical interface, which can be started as *nrngui* run library. The NEURON `Main Menu` contains functions to load data with model descriptions, edit the anatomical and biophysical parameters, and graphic control.

The `Run Control Window` that is shown in the Fig. 1 contains the menus for controlling the simulation. In this window one can repeatedly call the built-in single-step integration function and change the value of membrane parameters during a run.

Now one needs to construct an electrode to inject a stimulating current into the soma. From menu `Point Processes`, one can choose the required stimuli (IClamp in this case) and set the parameters, as illustrated in Fig. 1.

The menu `New Graph` (from the directory `MainMenu/Graph`) enables to build up the graphs for simulation control and save them later in PostScript format. To create a voltage vs. time graph, one can select a voltage axis, in which one can order to plot voltage at soma (0.5). The screenshot of this experiment is shown in the Fig. 2.

After creating several graphs, you may want to save the windows you have created (i.e., graphs and panels) to a file `*.ses` so that you can recall them at a later time. One can choose the windows for saving (from menu `Main menu/Window`) and then later open it (from menu `File/load session`).

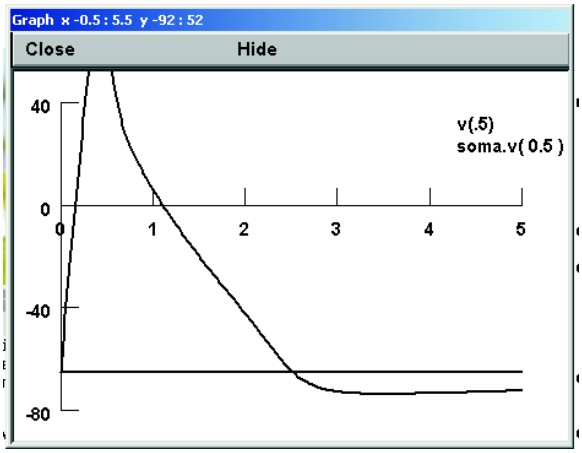

Figure 2: Plotting graph of voltage with NEURON.

Changing the model parameters via a graphical interface will evoke generating appropriate `hoc` code, as it has happened for a stimuli:

```
objref stim
soma stim = new Iclamp(0.5)
```

The statement `objref` assigns point mechanisms (as synapses and electrodes), which are described in terms of localized absolute current and conductance. Point process elements are always referenced by any object element (segment).

NEURON also allows the incorporation of standard density mechanisms, which are described in terms of current and conductance per unit area; examples include voltage-gated ion channels.

Moreover, users can define density mechanisms and point processes for each individual model, which can be linked into NEURON using the model description language NMODL [9]. Model equations written using NMODL are independent on the numerical methods used to solve them and do not require changing as the methods change or as the interface specification of NEURON is changing. Model description files can be checked for consistency of units using a unit checker program.

One of NEURON's advantages is that enables the user to specify a 3D-topology, which is useful when the model is based on the anatomical reconstruction of data or if 3-D visualisation is principal. The approach keeps the anatomical data in a list of "points", from which one defines the area, diameter, and resistance of each segment.

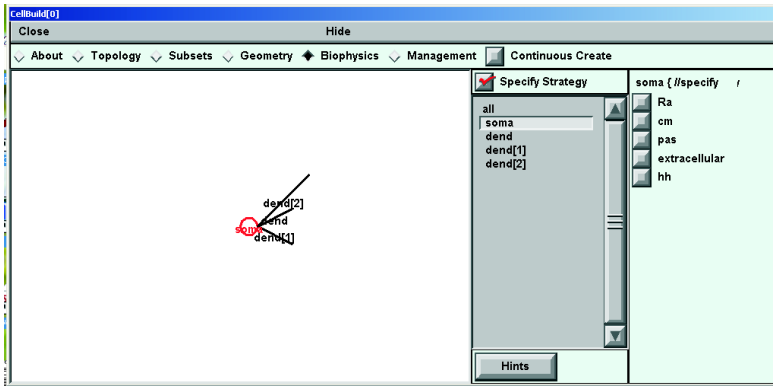

Figure 3: Constructing and managing models of single neurons with Cell Builder.

Cell Builder and Network Builder. One can use Cell Builder to enter the specifications for the model cell without having to write any hoc code oneself. It allows setting up branched pattern, assigning biophysical and anatomical parameters for groups of sections (see Fig. 3).

Network Builder provides a GUI for setting up the architecture of a network containing either artificial or "real" neurons and assigning their properties and their connections (see Fig. 4) [11].

Integration methods. NEURON provides a choice between a first-order fully implicit integration method (backward Euler) and a more accurate second- order variant of the Crank-Nicholson time step [4]. Hines, one of the NEURON authors, has presented and implemented the improvements in computation speed for simulations of arbitrarily branched cables with Hodgkin-Huxley kinetics. This improvement takes advantage of the essentially tridiagonal character of the matrix equation for each branch of a "tree" network and solves the equations as efficiently as for an unbranched cable.

Operating system and available documentation. Initially developed in the Unix environment, NEURON was ported to Windows and MacOS. The Unix/Linux distributions include full source code; the Windows and MacOS distributions employ an identical computational engine, GUI and NMODL definitions were implemented with `hoc` code. NEURON is available free of charge from the NEURON web site, along with extensive

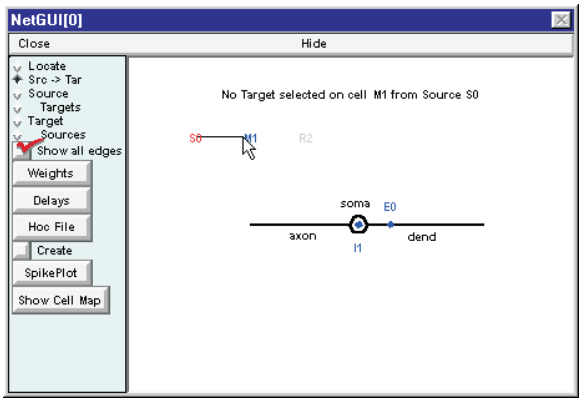

Figure 4: Modelling networks with Network Builder.

documentation and tutorials, e.g. [8] [7] [10]. From this page one can sign up to the NEURON User's group, which is organized as a mailing list, for getting announcements about program updates and sharing useful tips.

4.2 GENESIS

The GENESIS object-oriented approach. Because GENESIS was created as "general" simulation system, it is based on the "object-oriented" approach [13]. The extensibility and generality enable users to exchange and reuse models and their components. GENESIS provides the basic components as compartments, various types of channels which may be added, axonal connections to synapses as "building blocks" which are used to construct the simulation. Each element contains data fields of the values of parameters used by the element. The elements are connected by a system of links called "messages". For example, the following code constructs soma-like compartment and sets the parameters:

```
create neurtral /cell
create compartment /cell/soma
setfield /cell/soma Em {Erest}//volts
Rm {RM/area}  //Ohms
Cm {CM*area}  //Farads
Ra  {RA*length/xarea}  //Ohms
```

To connect two compartments, a "primed" compartment (dendrite) that needs to send both its axial resistance and its membrane potential at the

previous step, it only needs to receive soma's membrane potential to execute the integration process.

```
addmsg cell/dend cell/soma RAXIAL Ra previous_state
addmsg cell/soma cell/dend AXIAL previous_state
```

Script language interpreter and graphical user interface. The commands in the above example can be invoked either interactively from a command prompt or from files containing simulation scripts. The user interface incorporates the Script Language Interpreter (SLI), which is analogous to the Unix shell commands and also conatins the built-in set of commands for constructing and controlling simulations. Along with the simulation elements there are also the graphical objects linking with the scripting language.

That way, graphical objects constitute XODUS (the X-windows Output and Display Utility for Simulations), that provides the graphical user interface (GUI). For example, graphical "windows" for simulation of a two-cell network in a feedback configuration can be created as in Fig. 5. The simulation contains two neurons, each of them is composed of two compartments corresponding to a soma and a dendrite. A dendrite has synaptically activated channels while a soma contains ionic channels with Hodgkin-Huxley dynamics. The advantage of this approach is that one can build the GUI specially for each simulation model and then interactively change simulation parameters; but it takes additional time to master the visualization process.

Operating system and implementation language. GENESIS and its graphical interface are implemented in C and run on graphical workstations under Unix. There is also a parallel version of GENESIS (called PGENESIS) that can be started on parallel computers and would be used effectively for large network models with a large number of neurons [15].

GENESIS object libraries and extensibility. GENESIS object libraries include the spherical and cylindrical compartments from which the physical structure of neurons is constructed, voltage and/or concentration-activated channels, dendro-dendritic channels, and synaptically-activated channels with synapses of several types including Hebbian and facilitating synapses. In addition, there are objects for computing intracellular ionic concentrations from channel currents, for modelling the diffusion of ions within cells, and for allowing ligand gating of ion channels. There are also a number of "device objects" that may be interfaced to the simulation to

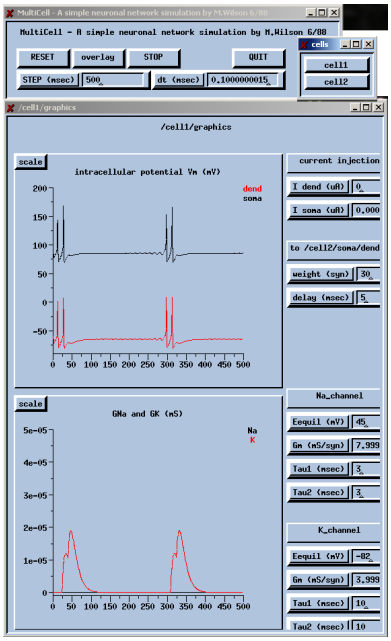

Figure 5: The GUI constructed with GENESIS shows a simulation of a two-cell network in a feedback configuration.

provide various types of input to the simulation (pulse and spike generators, voltage clamp circuitry, etc.).

The kinetics library supports kinetic-level modelling of biochemical pathways.

Extended objects can be created by using GENESIS script language. One should notice, however, that a new element would be created from existing GENESIS objects which have at least some of the same properties. New fields, message definitions and actions may be added to the root element before it is converted to an extended object.

GENESIS script libraries. For large simulation models GENESIS has the ability to construct simulations using information from data files and from pre-compiled GENESIS object libraries. The object libraries, which are available within the GENESIS distribution, are the result of communication between members of the GENESIS Users Group.

The channel library contains models for different types of potassium channels, including several types of calcium-dependent channels, as well as sodium channels. The available single-cell models include cerebral cortical pyramidal cells, hippocampal pyramidal cells, cerebellar Purkinje cells, mitral, granule, and tufted cells from the olfactory bulb, a hippocampal granule cell model, a thalamic relay cell, and an *Aplysia* R15 bursting pacemaker cell.

So, the cell reader allows building multi-compartmental neurons by reading "cell parameters" from a cell descriptor file. The cell reader expects to find the library of prototype elements in the `neurokit/prototypes` directory, and then builds the cell by making copies of them, replacing the default parameter fields with values from the cell descriptor file (`‘‘*.p’’`). The Neurokit simulation, which can be found in the `Scripts/neurokit` directory, provides a graphical interface for the cell reader (see Fig. 6). Simulation in Neurokit is possible without knowledge of the GENESIS script language.

This approach enables the user to construct networks connections after defining all necessary elements in the "prototypes" library (as arrays of definite kinds of cells) and then specify connections by synapses individually or in groups. The commands available in GENESIS enable the user to configure the parameters of connections in flexible ways.

Integration methods used in GENESIS. Integration methods provided by GENESIS vary from explicit methods, which require more compu-

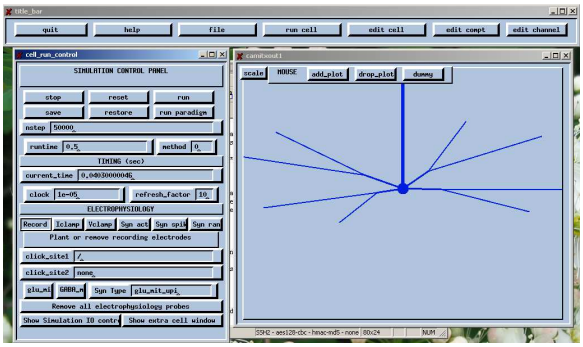

Figure 6: Neurokit simulation.

tational power, to more accurate implicit integration methods.

As a default method, GENESIS uses the exponential Euler method because of the form that have the equations typically encounter in the modelling. Other explicit methods range from forward Euler different orders of Adams-Bashforth methods (2nd, 3rd, 4th, 5th order). As the implicit method GENESIS uses a version of Hines method [4], which was implemented as a special object, `hsolve`. This object can be used in conjunction with the backward Euler and Crank-Nicholson methods only for elements describing branched neuronal structures.

Special techniques to speed up GENESIS simulations. In addition to "parallel" computations possible with GENESIS and faster integration techniques, I found "table lookup" very useful. For example, using the `tabchannel` object instead of `hh_channel`, the rate parameters can be specified by a table instead of fitting them into one of the three functional forms. Of course, evaluation of the functional forms takes additional computational time. In this way more general types of channels can be implemented.

Another element `table` may be used in many situations where there is no GENESIS object that performs integration calculations. As an example of simulated channels, implementing a "fast" gate, which has activation dependent only on voltage, leads to speedup of calculations.

Using "multiple clocks" for elements whose behavior is described by variables that involve different time scales is also an important advantage. It gives more control over factors which affect speedup and accuracy of the simulation.

GENESIS distribution, documentation, and tutorials. The distribution code of GENESIS includes full the source code together with XODUS and extensive tutorials and a manual with descriptions of basic elements and simulation routines.

Although the source code is available with the distribution, users trying to understand its logical structure will come across several shortcomings. Details of implementation are hidden; algorithms and data structures are closely related. The integration algorithms do not start with calling some function, but rather directly from the object reference itself. It makes data and algorithms more dependent on one another. Therefore, trying to write elements of one's own and integrate them with GENESIS elements is almost impossible.

The authors of GENESIS have published "The Book of Genesis" [14], which can be used both as documentation for GENESIS and as a text book for modelling techniques designed as two parts. The first part consists of the introduction into different neurobiology topics, which was illustrated by appropriate GENESIS tutorials. The second part offers a user guide to GENESIS and explain the basic features of GENESIS as well as the process of creating a GENESIS simulation.

As mentioned above, many simulations, object libraries, documentation material and tutorials were developed by GENESIS users. The GENESIS User's group, BABEL, contains mainly of researchers making serious use of GENESIS. BABEL's users have a right to have access to the BABEL directories and email newsgroup. It is used as a tool to exchange new simulations, post questions and hints for setting up GENESIS simulations. Improvements and new development are discussed by e-mails sent to all members.

The history of the GENESIS development. There is the prejudice among neurobiologists that GENESIS is most suitable for simulation of single-cell models that has its origin in the well-known models first simulated with GENESIS. The earliest GENESIS simulations were biologically realistic large-scale simulations of cortical networks [22]. The De Schutter and Bower [19], [20] cerebellar Purkinje cell model is typical of a large detailed single-cell model, with 4550 compartments and 8021 ionic conductances.

Although GENESIS continues to be widely used for single-cell modelling and for modelling small networks, in recent years it has become increasingly popular to use GENESIS for large network models (which has become possible as a result of the latest developments in GENESIS).

4.3 SNNAP

SNNAP graphical interface. The SNNAP simulator can be a good choice for modellers having little or no programming skills, because one of the important and distinguishing features of SNNAP is a quite well-developed graphical user interface (GUI). SNNAP provides editors embodied into graphical interface editors for specifying the anatomical and biophysical properties of neurons and the structure of networks, saving results of simulation to files and control of simulation results [12].

When a modeller does not have much time for learning the simulation techniques, SNNAP can be used as "a tool for the rapid development and simulation of realistic models of single neurons and small neural networks", because along with ease of use it incorporates the common elements encountered in realistic neurobiological modelling.

Operating system. SNNAP was implemented as a Java application and can run on any computer system. SNNAP is distributed as Java `*.jar` files necessary to run the application and two subdirectories of tutorials and examples. The installation of SNNAP requires a functional version of Java on the user's computer. One need only unzip the SNNAP data and put them in the appropriate directory.

Modular organization of input files. The graphical editors of SNNAP generate the input files, which have a hierarchical structure. All equations describing ionic channels, neuronal coupling and inserting mechanisms together with parameters describing the process of simulation can be read with appropriate SNNAP editors.

Of course, this feature allows creating libraries of SNNAP elements and exchange of new models among collaborators.

Examples demonstrating modelling with the SNNAP graphical user interface. The main SNNAP window contains buttons to open the various functional windows to edit SNNAP parameters, edit model parameters and execute simulation (see Fig. 7).

The Hodgkin-Huxley model remains the basic formula most often chosen by neuroscientists and can be used as a good example for demonstration. The simulation can be loaded from the directory `/Examples/hhModel` by opening the appropriate `*.smu` file. The results of the simulation are shown in Fig. 8.

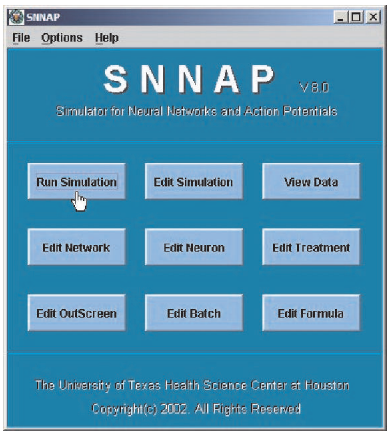

Figure 7: The main window of SNNAP.

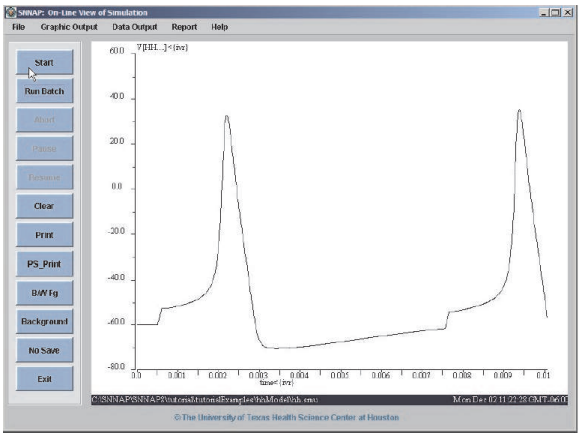

Figure 8: The SNNAP simulation window displays two action potentials.

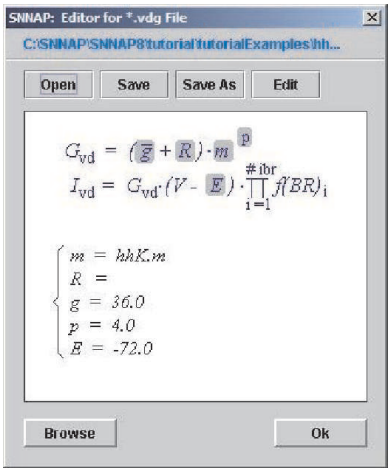

Figure 9: Editor window provided by SNNAP for voltage-dependent currents.

Changing model parameters can be done from the "Formula Editor" window . In our example, editing Hodgkin-Huxley voltage-dependent current can be implemented in the Editor for `*.vdg` files (Fig. 9).

The example `/Examples/hhNetwork` provides you with a small network having a common architecture, for an easy introduction to SNNAP. Fig. 10 shows the behavior of a three-cell network while Fig. 11 presents the network architecture. As basic elements, Hodgkin-Huxley neurons were used, connections were made with Alpha Synapses.

Physiological data to be modelled with SNNAP. SNNAP is a Simulator for Neural Networks and Action Potentials. SNNAP can simulate current flow using multicompartmental models of neurons [12]. The compartments can contain voltage- and time-dependent channels (typically Hodgkin-Huxley). The neurons can be connected with both electrical and chemical synapses.

Additionally, it is possible to express different kinds of plasticity in chemical synapses, as homo- and hetero-synaptic depression and facilitation. Examples of simulations of synaptic connections and synaptic placticity can be found in the subdirectory `/Examples/Synaptic_connections`. Fig. 12 demonstrates the results of simulation of synaptic facilitation and heterosynaptic placticity. Heterosynaptic placticity can be implemented by including a second messenger that modulates transmitter release.

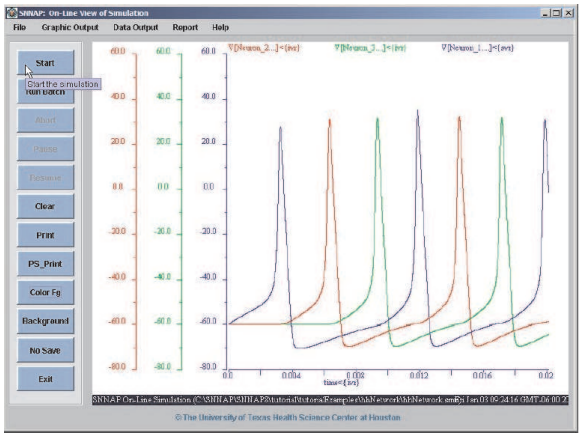

Figure 10: Results of a simulation with SNNAP of a three-neuron network.

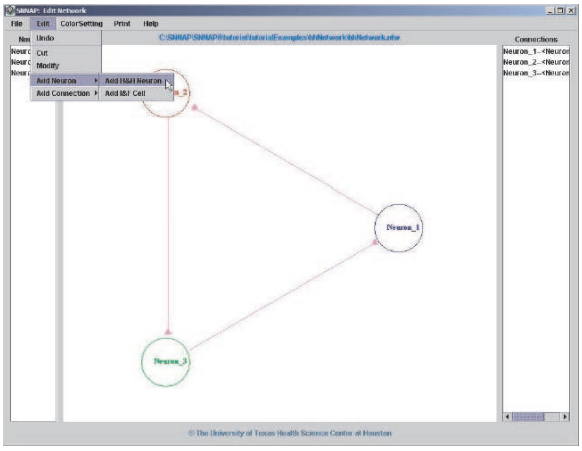

Figure 11: The SNNAP GUI showing the architecture of a three-neuron network.

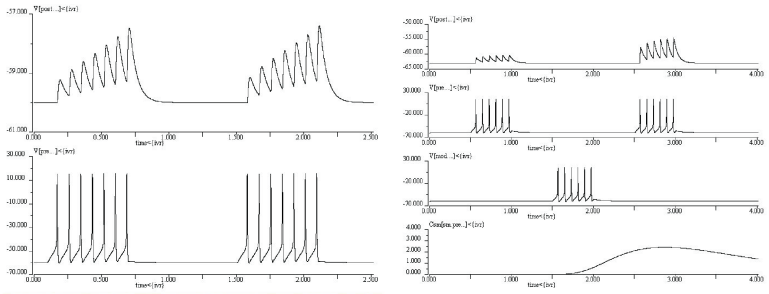

Figure 12: Results of a simulation with SNNAP of synaptic facilitation and heterosynaptic plasticity.

As a demonstration of relatively complex models, simulation of the bursting model in *Aplysia* R15 cell, which is to be found in the directory `/Examples/H_H_type_neurons/R15`, could be chosen. This model incorporates a voltage- and time-dependent conductances, an intracellular pool of calcium, an intracellular pool of second messenger, and modulatory interactions from the ion and second messenger pools and the membrane conductances. The model produces endogenous bursting activity. At $time = 60$ *sec* a modulator is applies via the treatment file. It modifies the electrical activity of the neuron.

Moreover, SNNAP includes the possibility to simulate common experimental techniques. For example, SNNAP can simulate injection of external currents into multiple cells, removal of individual conductances to simulate pharmacological agents, modulation of membrane currents via application of modulatory transmitters and voltage-clamping cells. The simulations in the directory `/Example/HH_type_neuron/Biophysics_01` show the use of voltage-clamping. The simulation shown in Fig. 13 simulates a voltage-clamp experiment of squid giant axon as held at $V = -60\ mV$ and stepped to $V = -10\ mV$ and display the total membrane current (black), sodium current (red) and potassium current (blue) that were elicited during the step are illustrated.

Numerical integration method used in SNNAP. SNNAP uses forward Euler method with a fixed time step for numerical integration of differential equations arising in modelling. The forward-Euler rule is fast, but it can be stable if the time step is too large. One should always remember that the integration time step must be small enough to ensure stability of

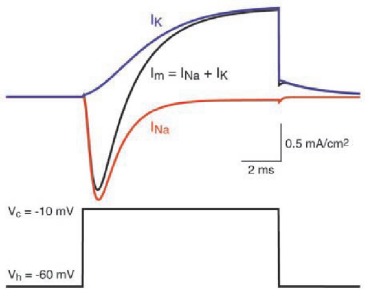

Figure 13: Voltage-clamping experiment with Hodgkin-Huxley model of a squid axon implemented with SNNAP.

the fastest process (i.e., the smallest time constant). This problem is known as the problem of stiffness.

The network of the three-cell oscillator that is included in the directory `/Examples/Neural_Networks` shows the problem of instability by choosing a time step of 1 ms. The results of instability are shown in Fig. 14.

4.4 Nodus

Nodus distribution, operating system, and programming language. The Nodus simulator was implemented as a tool for simulation of the electrical behavior of neurons and small networks. Users of Nodus are presumed to have limited computer experience. All steps of working with Nodus can be performed via a user-friendly graphical interface.

Nodus runs on Apple Macintosh computers. The distribution of Nodus includes the compiled program, example files, and the manual. The source code is not available because, as the authors emphasize, "the average Nodus user is not a programmer" [18]. Nodus was written in Fortran.

Nodus user interface and modelling elements Model descriptions are saved in the "simulation database", which contains 3 different types of input files: conductance definition files, neuron definition files, and (optionally) network definition files, which are linked together in the model with the "top" file. Each of these settings can be defined in a separate window [18].

Initial values, of parameters are saved in the simulation data file. That enables the user to start the simulation with parameters obtained from a previous simulation. These parameters can be viewed and changed during

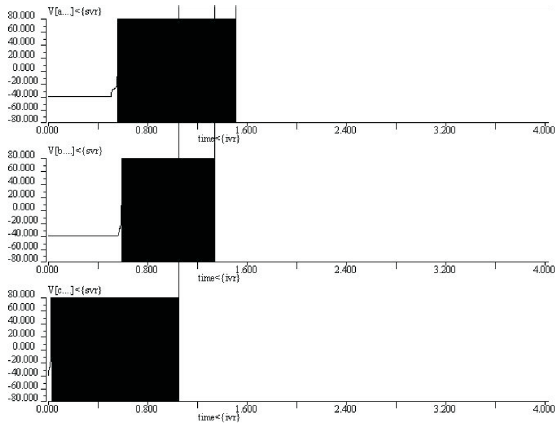

Figure 14: The simulation with SNNAP of a three-cell oscillator with time step of 1 ms.

the simulation.

The modelling in Nodus is based on compartmental modelling. Neuron definition files include information about morphological and anatomical parameters: size and cable parameters, connections and synaptic parameters. In the compartment dialog windows, the user can then select via popup menus one or several of the mechanisms to be inserted into the compartment (Fig. 15). If no elements are selected, a compartment is passive [18].

The compartment structure was implemented as an approach what the equivalent circuits of compartments are placed at their center. Standard symmetric connections between compartments (end to front) and asymmetric connections (center to front) are available. Neurons can contain ionic currents, whose description is saved in the conductance definition files. Formulation of the equations describing voltage-gated channels is fairly easy, because plotting of conductances, activation, and rate factors is provided.

Simulations of some of the standard electrophysiology experiments can be set up. There can be different currents (constant, repetitive pulses, ramps, sinus, noise) to be injected and added in any compartment. One or two neurons can be voltage-clamped (Fig. 17).

The results of a simulation can be saved on disk and plotted on the following axes: conductances, currents, and voltages (Fig. 16).

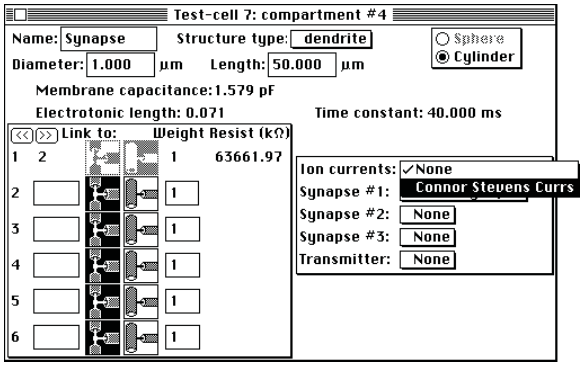

Figure 15: A Nodus window showing the construction of a compartment that can be passive as well as have activated channels .

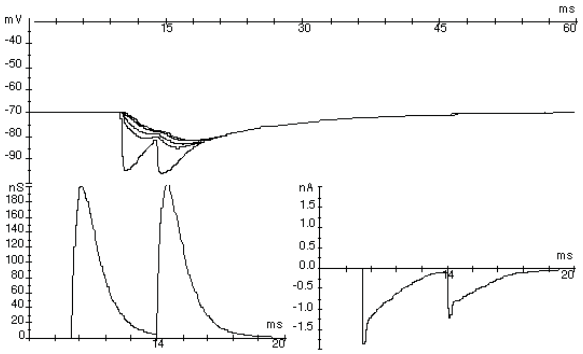

Figure 16: Graphic output of simulation results: membrane potential, synaptic conductances, synaptic currents (modelled with the Nodus simulator).

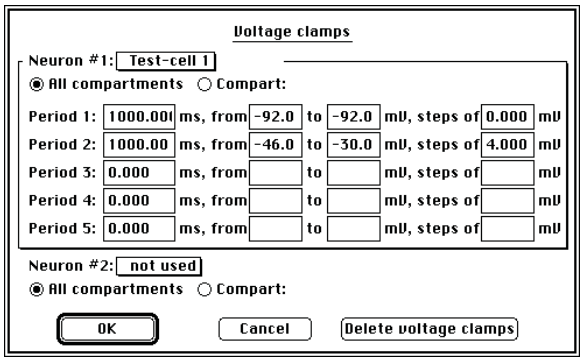

Figure 17: Nodus dialog window used to set up voltage clamps.

Integration methods Two explicit integration methods are available for simulations: an accurate Fehlberg method (fifth-order Runge-Kutta) and a forward Euler method, both with variable time steps. These methods are easier to implement, but in some cases the program can be slower than other simulation systems, but Nodus was realized with the goal of ease of use.

4.5 SURF-HIPPO

The models to be constructed with SURF-HIPPO. The SURF-HIPPO simulator is used to model cell models having a 3-dimensional geometry [2]. SURF-HIPPO allows construction of multiple cells from various file formats (Neurolucida, NTS and others), which can describe complicated dendritic trees in 3-dimensional space with distributed non-linearities and synaptic contacts between cells. Cell geometries may also be defined directly on the screen, using the mouse. A graphical user interface is provided, including menus, 3D graphics of dendritic trees, and data plotting (Fig. 18). Graphical output can be saved in the Postscript files. Data files may also be saved for analysis with other programs [1].

Integration method. For integrating the circuit equations, Surf-Hippo uses a variant of the Crank-Nicholson method implemented by Hines [4]. A major difference in the method used by Surf-Hippo is a variable time step option, where step size is adjusted according to an estimate of the linear truncation error for all state variables (e.g. node voltages, channel particles). The adaptive time step can give much faster run times for typical simulations, with the option of verifying selected results using the more

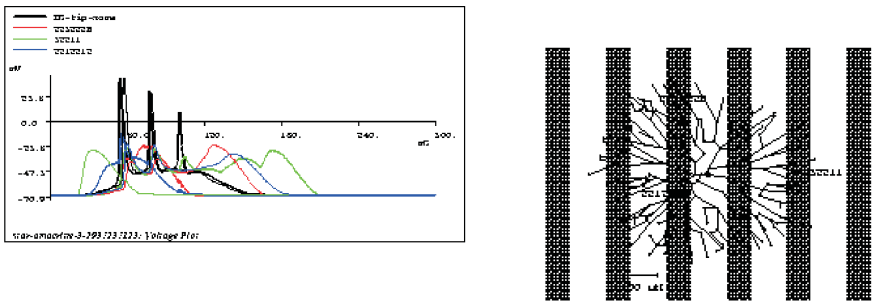

Figure 18: Graphical output with SURF-HIPPO.

conservative fixed time step integration [1].

Implementation language. SURF-HIPPO is a public domain package written in Lisp and runs under Unix and Linux. The choice of Lisp is convenient, because it has numerical performance similar to C or Fortran. SURF-HIPPO is configured to run using the public domain CMU Common Lisp and Garnet packages, which are integrated directly into the Lisp interpreter. It provides complete access to all components of a simulation directly from the SURF-HIPPO graphical interface. Another advantage of Lisp is that all functions defined in the system may be executed from the interpreter either individually or within scripts, which makes for a very flexible working environment. New code (including bug fixes) may be compiled and used as needed, without recompiling the entire executable [1].

That makes SURF-HIPPO very flexible. Knowledge of Lisp and programming experience can considerably reduce the time needed to learn and use SURF-HIPPO.

4.6 NeuronC

Basic features of NeuronC. NeuronC was developed as a "simulation language" and based on the approach of "general" simulators, the concept of a "language" in which elements of the model can be described and "run" [21].

NeuronC contains a subset of the C programming language, with standard features such as variables, assignment statements, mathematical operators, conditional and loop statements, and subroutines. NeuronC is an

interpreted language, similar to other simulation languages.

NeuronC was originally designed to perform experiments on vision, which can contain real networks with a huge number of neurons. Therefore, the main goal of simulating of such a network is to provide the possibility to solve problems at many levels of detail. Consequently, the basic idea of NeuronC is the same as for "general" simulation systems.

"However, NeuronC contains several special features: 1) it can construct and run models as large as 50,000 compartments using a conventional workstation, with run times on the order of hours (proportionately less for smaller models); 2) the models can be constructed hierarchically as a set of "conceptual modules", and this allows parameters and connections of a module or entire neural circuit to be comprehended and easily altered; 3) the models can contain two-dimensional light stimulus and photoreceptors, that allows simulating a visual physiology experiment" [21].

Neural elements and their representation. The network is organized as a set of "conceptual modules", each one "constructed" from lower-level components. The most useful types of module seem to be familiar, such as "membrane channel", "synapse", "dendrite", "synaptic interconnected pattern", and "array of neurons". Each module is simulated with an appropriate precompiled algorithm. New elements can be added in NeuronC by setting the appropriate code and recompiling [21].

After a model's neural elements have been specified, the NeuronC simulator translates them into a compartmental model. A "sphere" (usually a soma) is translated into one compartment, which represents an isopotential region enclosed by a membrane (without axial resistance). In contrast, NeuronC translates a "cable" into a series of isopotential compartments, each representing a small patch of membrane and a small volume of cytoplasm [17]. The compartments and their resistive interconnections are created from the neural circuit description by a cable-segmentation algorithm [5].

NeuronC creates 2 levels of information, "high-" and "low-level", to represent the neural circuit at different levels of abstraction. High-level data structures store the information directly from the experiment description in the input file. They describe stimulus, record, node, and neural element (cable, sphere, synapse, sensory transduction element, etc.). At runtime, this high-level information is translated by the simulator program to low-level information, stored in different data structures. Low-level data structures store information specifically for use by the simulator during compu-

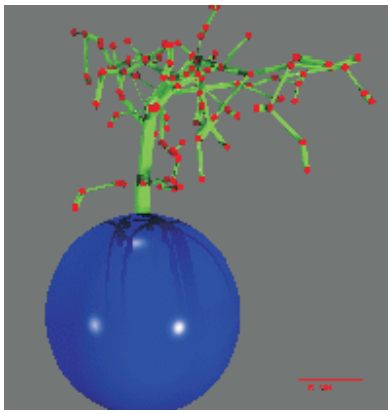

Figure 19: Morphology and connectivity of a 3-dimensional network constructed with NeuronC.

tation [21].

Integration methods. At the second level of abstraction, the functions of the electronic circuits are simulated using a set of differential equations evaluated with the appropriate numerical integration method, by default a variation of the Crank-Nicholson implicit method, developed by Hines [4]. NeuronC also includes the backward Euler implicit method and the forward Euler explicit integration method.

Graphics. NeuronC can portray the morphology and connectivity of a 3-dimensional network with a "display" statement. This statement allows selective display and exclusion of parts of a neural circuit (see Fig. 19). NeuronC also can plot voltage, current as a function of time, and in addition can parametrically graph any 2 variables (one as a function of the other).

Operation system. The distribution of NeuronC is provided with the source code and runs on most UNIX systems.

4.7 HHSim

HHSim, in contrast with "general simulation systems", such as GENESIS and NEURON, which provide more modelling options for researchers, is implemented for teaching purposes. It provides modelling a section of a

Figure 20: HHSim main window for controlling simulation results.

membrane with traditional Hodgkin-Huxley voltage-gated channels. HHSim can be used as a good demonstrative tool in neurophysiology courses.

HHSim was written on Matlab. The installation of HHSim does not require a Matlab licence and is distributed as an executable file. Executable files for Windows, Unix and MacOs are available. The version with source code is also available, requiring Matlab version 6 or higher.

Graphical User Interface. The main window of HHSim, which is illustrated in Fig. 20, provides the ability to plot membrane voltage and Hodgkin-Huxley gated parameters m, h, and n as default. The information to be plotted, such as currents or conductances, can also be selected. The data can be saved in Postscript format for printing or in ASCII data for estimation. The program provides access to the channel's Hodgkin-Huxley parameters, membrane parameters, stimulus parameters, and ion concentrations (see Fig. 21). The Drugs window allows application of three drugs: TTX, which inhibits the sodium current; TEA, which inhibits the potassium current; and pronase, which eliminates sodium-channel inactivation (see Fig. 22).

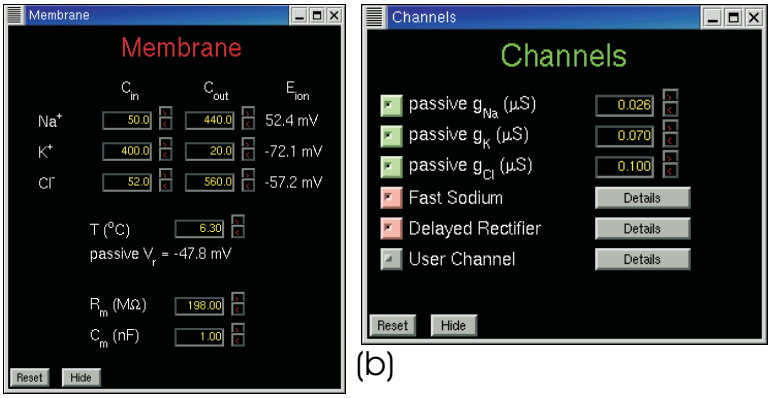

Figure 21: (a) HHSim Membrane window for adjusting of membrane parameters; (b) HHSim Channels window providing access to channel parameters.

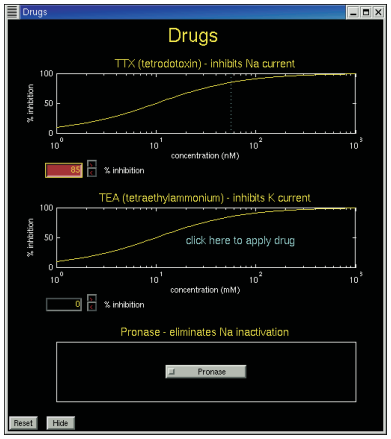

Figure 22: The HHSim Drugs window allows application of a few drugs.

	GENESIS	NEURON	SNNAP	Nodus	NeuronC	Surf-Hippo	HHSim
Oper. system	Unix	Unix, Win, MacOs	any	MacOs	Unix	Unix	Unix, Win, MacOs
Impl. language	C	C	Java	Fortran	C	Lisp	Matlab
User extensions	yes, for programmers	no	no	no	yes, for programmers	yes	no
Int. method	explicit, implicit	implicit	explicit	explicit	explicit, implicit	implicit	no information
Graphs of results	after programming	for primal research	yes	yes	yes	yes	yes
Model definition	interpreter	interpreter	editors	editors	interpreter	files	GUI
Ionic channels	yes	yes	yes	yes	yes	yes	HH
Synaptic channels	yes	yes	yes	yes	yes	yes	no
Networks	yes	yes	yes	limited	yes	yes	no

Table 1: Important properties of compartmental modelling packages

5 Conclusions

In this paper we have examined seven simulation packages for biological neural networks.

Table 1 summarizes the relevant information about the packages considered here. The characteristics given in this table have been proven useful. Table 2 presents the "best" features that can play an important role by choosing a program.

GENESIS	"general" simulation system; parameters are specified with "table lookup"; multiple time scales; extensibility; parallel computing
NEURON	"general" simulation system; numerical method developed by Hines; properties are dependent on position in the section
SNNAP	graphical interface; modular organization of input files; simulation of different types of plasticity and common experimental techniques
Nodus	graphical interface, no programming skills are necessary;hierarchical structure of model definition files
NeuronC	extensibilty; simulation experiments on vision
Surf-Hippo	3-dimensional models
HHSim	educational software

Table 2: Best features of examined packages.

References

[1] L. Borg-Graham. Additional Efficient Computation of Branched Nerve Equations: Adaptive Time Step and Ideal Voltage Clamp. *Journal of Computational Neuroscience*, 8(3):209–226, 2000.

[2] L. Borg-Graham. The Surf-Hippo Neuron Simulation System. 2003.

[3] J. Bower and D. Beeman. *The Book of Genesis: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* Springer TELOS, 1998.

[4] M. Hines. Effecient computation of branched nerve equations. *J. Biomed. Comp.*, 15:69–76, 1984.

[5] M. Hines. A program for simulation of nerve equations with branching geometries. *International Journal of Biomedical Computing*, 24:33–68, 1989.

[6] M. Hines. NEURON - A Program for Simulation of nerve Equations. In F. Eeeckman, editor, *Neural Systems: Analysis and Modelling*, pages 127–136. Kluwer Academic Publishers, 1993.

[7] M. Hines. The NEURON simulation program. In J. Skrzypek, editor, *Neural Network Simulation Environments.* Kluwer Academic Publishers, Norwell, Mass, 1993.

[8] M. L. Hines and N. T. Carnevale. The NEURON Simulation Environment. *Neural Computation*, 9:1179–1209, 1997.

[9] M. L. Hines and N. T. Carnevale. Expanding NEURON's Repertoire of Mechanisms with NMODL. *Neural Computation*, 12:995–1007, 2000.

[10] M. L. Hines and N. T. Carnevale. NEURON: a tool for neuroscientists. *The Neuroscientist*, 7:123–135, 2001.

[11] M. L. Hines and N. T. Carnevale. The NEURON Simulation Environment. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks.* MIT Press, Cambridge, MA, 2 edition, 2002.

[12] D. A. Baxter I. Ziv and J. H. Byrne. Simulatipon for neural networks and action potentials: Description and Application. *J. Neurophysiol.*, 71:294–308, 1994.

[13] J. M. Bower, D. Beeman and M. Hucka. The GENESIS Simulation System. In M.A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, Cambridge,MA, 2002.

[14] M. A. Wilson, U. S. Bhalla, J. D. Uhley and J. M. Bower. GENESIS: A system for simulating neural networks. In *Advances in Neural Information Processing Systems*, pages 348–353. Morgan Kaufman, San Mateo, 1989.

[15] M. Nelson and J. M. Bower. Simulating neurons and neuronal networks on parallel computers. In C. Koch and I. Segev, editors, *Methods in Neuronal Modeling*, chapter 12, pages 397–438. MIT Press, Cambridge, Mass, 1989.

[16] W. Rall. Branching dendritic trees and motoneuron membrane resistivity. *Exp. Neurol.*, 1:491–527, 1959.

[17] W. Rall. Theoretical significance of dendritic tree for input-output relation. In R.F.Reiss, editor, *Neural Theory and Modelling*, pages 73–97. Standford University Press, Standford, 1964.

[18] E. De Schutter. Nodus, A User Friendly Neuron Simulator for Macintosh Computers. In F. Eeeckman, editor, *Neural Systems: Analysis and Modelling*, pages 113–119. Kluwer Academic Publishers, 1993.

[19] E. De Schutter and J. M. Bower. An active membrane model of the cerebellar Purkinje cell. Simulation of current clamps in slice. *J. Neurophysiol.*, 71:375–400, 1994.

[20] E. De Schutter and J. M. Bower. An active membrane model of the cerebellar Purkinje cell. Simulation of synaptic responces. *J. Neurophysiol.*, 71:401–419, 1994.

[21] R. G. Smith. NeuronC:a computational language for investigating functional architecture of neural circuits. *J. Neurosci. Methods*, 43:83–108, 1992.

[22] M. Wilson and J. M. Bower. Simulating cerebral cortical networks: oscillations and temporal interactions in a computer simulation of piriform(olfactory) cortex. *J. Neurophysiol.*, 67:981–995, 1992.