Simulation Software for the Macintosh

DANIEL K. BOGEN

Numerical simulation can be used to predict and understand diverse phenomena, from predicting the number of cases of a sexually transmitted disease in a population, to studying the interaction between coupled biological oscillators. Writing a computer program to carry out such simulations, however, can be daunting for those without experience in programming and numerical methods; even for those with experience, the drudgery of writing data entry and graphical output routines, and the usual exasperation of debugging programs, may be enough to scuttle a simulation project. Fortunately, two software packages for the Macintosh, STELLA (*1*) and Extend (*2*), make simulations not only easier but fun, in that the Macintosh graphical interface allows the user to work with diagrams of complex systems and not just the equations.

To understand what simulation software is, it is important to understand that simulation software is different from equation-solving software. Simulation software allows the user to describe a complex system in bits and pieces, to draw a block diagram of the system on the computer screen, to enter in the mathematical description of each block (the relation between "inputs" and "outputs"), and to observe how the variables of the model change over time, but it does not require that the entire set of model equations be written down in one place. Simulation software is used to study a system or process by allowing the user to observe the effect of parameter changes on the different variables in the model. For instance, one could ask, "How is the incidence of sexually transmitted disease altered by increasing the cure rate," or, "What coupling strength is required to synchronize oscillators of differing natural frequencies?"

STELLA for Compartmental Models

One common type of mathematical model that can be used to describe chemical, ecological, pharmacological, demographic, and epidemiologic systems in particular and transport systems in general is the compartmental model. In such a model substances or entities move about or change state (conceptually at least) according to the concentration, prevalence, or amount of that substance in the system. Systems described by rate processes (chemical reactions, epidemics, transport of molecules across membranes, and so forth) yield compartmental models. [For an overview of the use of such models in biology, see (*3*) and (*4*).]

STELLA is a software package for handling compartmental models (or models that can be visualized in compartmental terms). Version 2.1 of STELLA for Education is available for the Macintosh 512KE, Plus, SE, and II models (*5*). The price of STELLA is $200 when purchased by those associated with an academic institution; otherwise the cost is $450. Two versions of STELLA, a Macintosh Plus/SE version and a version optimized for the Macintosh II, are shipped with each order. All versions run on the Macintosh II, but only the latter takes full advantage of the numerical coprocessor. The Macintosh II version also supports a color display. The accompanying 392-page manual, *An Academic User's Guide to STELLA*, not only gives a step by step description of how to use STELLA, but gives many illustrative examples, taken from physics, biology, sociology, and economics. The manual also explains what one does with a simulation once it is running on the computer and thus provides a fairly complete tutorial on compartmental modeling.

The best way to describe what STELLA does is to start with a simple example: the absorption of a drug from the gastrointestinal (GI) tract into the blood and excretion of the drug from the blood. In terms of compartmental modeling, the drug moves from the GI compartment into the BLOOD compartment, and then is eliminated from the BLOOD compartment, as diagrammed in Fig. 1A. In this case there are two transport coefficients K_u and K_e, which describe the rate of uptake and excretion of the drug, respectively. (In this example, flux is unidirectional.)

Without simulation software, the standard approach to this problem would be to write the transport equations for each compartment (*3, 4*). If X_1 is the amount of drug in the GI compartment and X_2 is the amount in the blood, and if it can be assumed that absorption depends only on the amount in the gut and that excretion depends only on the amount in the blood, a system of coupled differential equations results:

$$dX_1/dt = -K_u X_1$$

$$dX_2/dt = K_u X_1 - K_e X_2$$

With STELLA, however, most of the programming is performed simply by drawing the diagram. STELLA provides a number of mouse-driven drawing tools (*5*). One of these places a rectangle, which represents a compartment (called a "stock" in STELLA parlance), anywhere on the screen. Another tool draws pipes between rectangles (compartments) to indicate fluxes from one compartment to another. Each of these pipes (called "flows" in STELLA) has a symbol placed along its length that denotes the presence of a valve that modulates the flow through the pipe. This valve contains what might be called the "flow rule" that governs the flow from one compartment to another. The user supplies the flow rule, and STELLA understands that the flow depletes one compartment and augments the other. STELLA allows any flow rule that results from the algebraic combination of quantities in any of the compartments in the model, along with basic mathematical functions and constants. The flow rule is entered by clicking on the flow symbol and then typing in the rule, or by using mouse clicks from a list of options. Any compartment that enters into the flow rule must be connected to the corresponding flow symbol in the STELLA diagram. This connection is made by another drawing tool, which draws a connecting arrow from compartment to flow (a "connector" in STELLA terminology). Thus a brief glance at the diagram indicates which variables influence flow between which compartments. One final element is the "converter," indicated by a circle. This element has many uses, but is commonly used as a convenient graphical handle on important parameters in the model that may need to be changed from time to time. Clicking

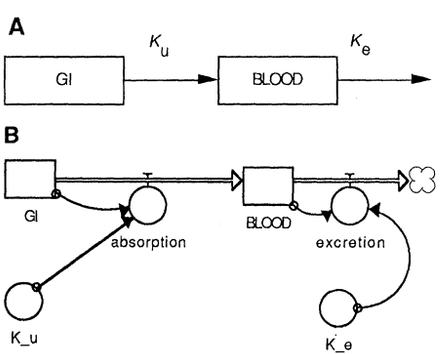
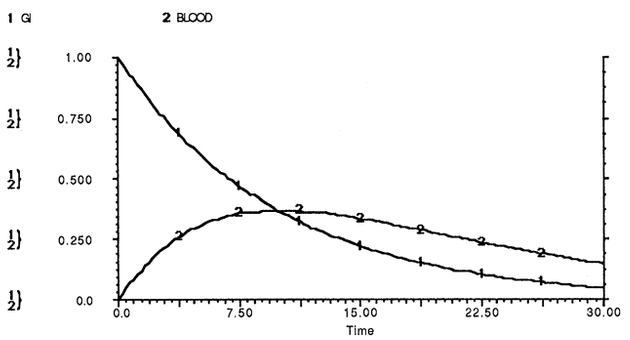


Fig. 1. (**A**) Compartmental model of drug absorption and excretion and (**B**) equivalent STELLA diagram.

Department of Bioengineering, University of Pennsylvania, Philadelphia, PA 19104.

A



B

```
BLOOD = BLOOD + dt * ( absorption - excretion )
INIT(BLOOD) = 0
GI = GI + dt * ( -absorption )
INIT(GI) = 1
absorption = K_u*GI
excretion = K_e*BLOOD
K_e = .1
K_u = .1
```

Fig. 2. (**A**) Output of STELLA simulation of simple drug absorption-excretion model. (**B**) Finite difference equations generated by STELLA model.

on a converter brings up a dialog box into which the parameter value is entered. (A dialog box is a Macintosh device where a small window pops up with edit fields, check boxes, and "buttons," which the user can alter to set values and conditions.) The converter may also be used to evaluate mathematical functions involving one or more variables of the model. The value of this function becomes a new variable. All of the symbolic elements in the STELLA diagram can be labeled, or named, as the user wishes; there are additional editing tools for deleting, copying, and pasting.

For a drug absorption model, one might build a diagram like Fig. 1B. The simulation is run by first entering values for constants and initial conditions of the compartments. This process is simply a matter of clicking on the "converters" (constants) and "stocks" (compartments) and typing in the correct values. Next one would go to the "graph pad" window and select which of the compartments (up to four) is to be graphed. Finally, one chooses the simulation time, scaling, and integration method, and then selects "Run." As the simulation proceeds, compartment output values are traced on the screen. For parameter values $K_u = 0.1$ and $K_e = 0.1$, the plot shown in Fig. 2A is obtained, which shows an exponential decay in the GI drug level along with an increasing and then decreasing blood drug level. Another way of observing the simulation is to use STELLA's "animation" feature. Here the compartments in the diagram appear to be filled with fluid; and the fluid levels rise or fall during the simulation as the values of the compartments increase or decrease.

The model diagram, graphs, and even a table of values can be cut-and-pasted from STELLA to other applications. STELLA also makes available a complete list of the equations (in finite-difference form) that define the model, even though the user has only explicitly entered in the flow rules. For the simple drug model, the equations are listed in Fig. 2B.

The problem given above is simple enough that it can be solved analytically, without resort to simulation. A more complicated example that better demonstrates the capabilities of STELLA is the epidemiology of a sexually transmitted disease (STD). In this model, based on an analysis by Frauenthal (7), there are four compartments: uninfected males, infected males, uninfected females, and infected females. Males "flow" from the uninfected to the infected compartment according to the flow rule (here, actually an "infection rule"), which gives an infection rate equal to the product of uninfected males, infected females, and male "susceptibility." The female infection rule is similar in form. Males also flow from the infected compartment to the uninfected compartment by another flow rule, which we might call the "cure rule," which describes the cure rate of the disease. Female cures would follow a similar rule. A STELLA diagram for this looks like Fig. 3.

Once the STELLA model is up and running, the user may want to explore what happens when different parameter values in the model are changed. If there are many parameters to be tested, this exploration process can become a bit tedious, as each element in the model to be changed is "clicked" and new numerical values are typed in. A solution to this problem is provided by a companion program to STELLA, called STELLAStack. This program produces a HyperCard interface to the STELLA model, which allows the user to construct a HyperCard stack that supplies the parameter and initial values to the simulation. The stack can place all of the parameter entry information on one screen display. A stack can even be constructed that has "sliders," on-screen control knobs which control the magnitude of parameters. With such an interface the user could set the parameters in the model much in the same way that the volume, bass, treble, balance, and so forth are set in a hi-fi system. This powerful interface does not come without a small price, however, and that is that the user must perform some HyperCard programming. HyperCard programming (or "scripting" as it is called in the HyperCard vernacular) is not difficult, but it does take some time and effort. Fortunately, STELLAStack is an additional interface that can be added on anytime after the STELLA model has been completed. STELLAStack is available to academic users for $150 and to nonacademic users for $295.

Extend for Block Diagram Models

Extend is a versatile software package that can simulate a wide variety of systems, including those which either cannot be described as compartmental systems or which are more clearly conceptualized in another form. Version 1.1 of Extend runs on Macintosh Plus, Macintosh SE, and Macintosh II computers and is priced at $297 for academic users and $495 for nonacademic users. Both color display and numerical coprocessor-assisted computations are supported on the Macintosh II, but the program is not specifically optimized for the coprocessor. Extend is shipped with a 200-page manual that contains tutorial and reference information.

Just as the basis of STELLA is the compartmental model, the basis of Extend is the block diagram. Block diagrams (featured on blackboards around the world) consist of various symbols (circles, rectangles, triangles, or pictographs) interconnected by lines, often with arrows. Each symbol represents a specific type of process or activity, having inputs and outputs that carry information, energy, or material of some form. Extend allows the user to build a simulation

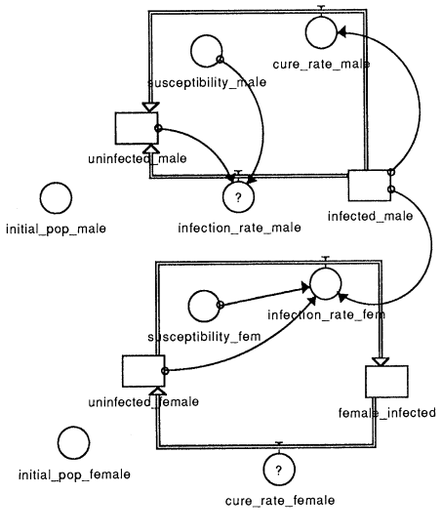


Fig. 3. Compartmental model of sexually transmitted disease, created with STELLA.

directly from such a block diagram. The computer screen acts as the blackboard, and a set of drawing tools allows the user to construct icons, or pictographs, representing the different blocks. For instance, one might use a picture of a test tube to denote a chemical reaction, a flame to represent combustion and so forth. These icons can be copied and pasted from other Macintosh picture-making applications and even from scanned images. The blocks can be moved about on screen with the mouse; blocks that need to be repeated in different locations are generated with cut-and-paste operations. Once the blocks are in place, lines are drawn between them with a penlike drawing tool to indicate that the outputs of one block acts as inputs to the other blocks. Each line represents a signal within the system, which can either be a regulating signal (such as a voltage, a hormone concentration, or the incidence of a disease) or a flow signal, like the flows used in compartmental modeling. To make clear what is an output and what is an input, each block icon includes one or more "connectors," or small square tabs; inputs are denoted by empty squares, and outputs are denoted by solid squares. Blocks and signals may be given names, or labels, to maintain clarity.

The construction of the on-screen block diagram gives the overall structure of the system, but what allows it to be used in simulation is that the definition of a block includes the mathematical relation between inputs and outputs. This relation is described by a user-written program, or script, in which a C-like programming language called ModL is used. In a block script the input and output connectors to that block are given names that are then used as variables within the ModL script. That is, if there are two inputs and one output, named "Summer1In," "Summer2In, and "SummerOut," respectively, a block that sums the two inputs would contain the line "SummerOut = Summer1In + Summer2In." Thus the signal on the output connector would be equal to the sum of the signals on the two input connectors, regardless of where the input signals come from. Of course, more complicated input-output relations can be programmed. ModL includes the standard C structure and operators, but also contains an extensive library that includes trigonometric and hyperbolic functions, integration, statistical distribution and statistics, complex variables, and matrix operations (including those on complex matrices). Indeed, matrices can be passed between blocks as signals. Also included in ModL are functions of file input-output, serial port operations (communicating to other devices), plotting, and even synthesized speech.

The ModL script is entered by using an integrated text editor with the usual Macintosh interface. After the text of the script has been entered, it may be saved, but before it can actually be used in simulation it must be compiled. When there are bugs in the script, the ModL compiler produces an alert and leaves the offending portion of script highlighted.

Values of variables within the ModL script, typically those used as parameters of the model, are set with a dialog box. Extend provides a simple means of creating a user-defined dialog box for each block. Double-clicking on the block brings the dialog box up, and the user enters the parameter values.

Once a block is defined, it can be duplicated and used over and over again in a single block diagram, or it can be saved and used in other simulations. The block is customized for a particular process by setting the relevant parameters in the dialog box. Extend is shipped with several example simulations and libraries of predefined blocks which can be appropriated, with or without modification, for the user's own simulations. One block generally required is the "Plotter block," a block with four input connections that operates like a four-pen plotter and is the basic graphics display for Extend.

To perform the simulation, the user chooses "Run Simulation" from the menu, enters the starting and ending times of the simulation along with the number of time steps, and then clicks on "OK." If the plotter block has been included, up to four variables will be traced and the output data recorded in tabular form. To observe additional variables, more plotter blocks can be added.

The best way to get a feeling for Extend is to follow an example. Consider a system of coupled oscillators, which might be a useful description of biological rhythms, such as the circadian, respiratory, and cardiac cycles. Such a system can be investigated with Extend by constructing a model for a single oscillator. Drawing tools are used to form a symbol for the oscillator (say, an arrowhead, as in Fig. 4A), and then the short ModL script is written to describe the relation between the input variable (the forcing signal) and the output variable (the state of the oscillator). On the oscillator symbol the input and output connectors are also drawn. With this graphical representation of the coupled oscillator system, any signal connected to the input or output connector automatically becomes the input or output of that particular oscillator.

Generating the relation between input and output begins with the mathematical statement. The present example is for the nonlinear Van de Pol oscillator, which has been used recently for modeling circadian rhythms (*8, 9*).

$$\ddot{x} + b(1 - x^2)\dot{x} + kx = y$$

where y is the forcing function. Essentially this is a harmonic oscillator with an amplitude-dependent damping term that can provide positive or negative damping to maintain the oscillation. A fragment of the ModL script for the Van de Pol oscillator is shown is Fig. 4A. In this script, descriptive names have been used. Hence, the variable **Stiffness** is attached to the "stiffness constant," **k**; **Damping** is attached to the "damping constant," **b**; and **Forcing** is attached to **y**. The state of the oscillator remains **x**, and the time derivative of **x** is **xprime**. Two other features have also been added to the simulation: (i) a "forcing threshold," indicated by the **Threshold** variable, which eliminates the effect of forcing signals whose magnitude is less than the **Threshold** variable; and (ii) a coupling strength between oscillators, indicated by the **Coupling** variable, which multiplies the input signal to the oscillator. **OscIn** is the signal applied to the input connector, and **OscOut** is the signal leaving the output connector. To set model parameters, such as the **Stiffness** and **Damping** variables, an Extend dialog box is constructed, as shown in Fig. 4B.

Having built the single oscillator, one can then construct the multioscillator system. This is quite simple, in that Macintosh copy-and-paste routines can be used to place multiple copies of the oscillator symbol on the screen. The signal-drawing tool can be used to make line connections between the inputs and outputs of the oscillators, as shown in Fig. 5A. The outputs of the oscillators can be viewed by placing a plotter block in the diagram and connecting it to the output signals from the oscillators. After

A

```
real    forcing,x, xprime, a[4],b[4];
on simulate
{
real sig1;
        Forcing = Coupling*max2(OscIn,-OscIn)*(realabs(OscIn)-Threshold);
        sig1 = Forcing - ((x*x-1.0)*xprime*Damping + x*Stiffness);
        xprime = integrateTrap(a, sig1, deltaTime);
        x = integrateTrap(b, xprime, deltaTime);
        OscOut = x;}
```

B

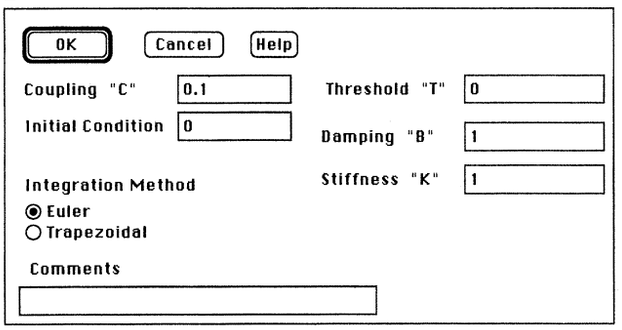


Fig. 4. (**A**) Extend simulation of coupled oscillators. An abbreviated ModL script (Extend programming language) is listed that describes a Van de Pol oscillator. The full script has additional sections for initialization and error checking. (**B**) Dialog box for an oscillator block from the Extend simulation. Clicking on any Van de Pol oscillator block in the model brings up this dialog box, which contains fields into which the initial condition or oscillator parameters can be entered.

setting parameters for each oscillator, simulation results such as those shown in Fig. 5B can be obtained.

Advantages and Limitations

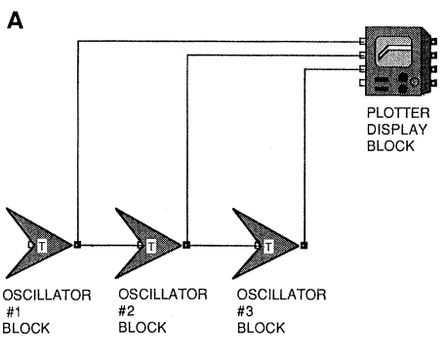
How can STELLA and Extend be compared? These are very different kinds of software and do not really compete against each other. STELLA's limitation is its strong roots in compartmental analysis. Although general systems described by differential equations, including higher order differential equations, can be modeled with STELLA, this use of STELLA is less intuitive than its applicaiton to compartmental modeling. STELLA's advantage, however, is that it carries out the latter with extreme ease, and with essentially no programming. Extend's limitation is that it does require some programming and requires more initial effort to get a model running. Extend's advantage is that it permits a diverse variety of simulations; once the subsystems have been programmed, models can be reconfigured, and parameters can be changed extremely rapidly.

Are there any drawbacks in using the either of these programs, rather than using a standard programming language? The answer is, there are essentially none, because STELLA and Extend models can be transferred to standard programming languages. The difficulty with writing simulations from scratch is that there is an enormous amount of programming overhead, such as reading parameters, preparing output files, and plotting data. In fact, in terms of programming, most of the work may be spent in handling these tasks, whereas the core of the model can sometimes be handled in a few lines of code. Simulation programs, however, already have these "overhead" functions built in, so the modeler can concentrate on the model and not on the programming. One other advantage of simulation software is that it allows the user to change the model without causing a complete jumble of computer code. If, for instance, oscillator-1 is to be connected to oscillator-3 instead of to oscillator-2, the modeler merely redraws the connections between the oscillators. There is no need to rewrite equations and risk a programming error. Related to this convenience is model documentation. If the modeler cannot remember what is connected to what in the latest version of the model, one needs only to look at the diagram. It will not be necessary to read through the computer code to find out. (Even better, it is unnecessary to read through someone else's computer code to find out exactly what has been done.)

It is also a mistake to put much weight on the computational speed of the either of these programs, although the examples shown here took only a few seconds to run. The time it takes to perform a simulation actually includes the time to necessary to set up the model in the computer, to set the parameters, to perform the calculations, and (generally) to construct graphs of the calculated values.

Another concern is that after the model is set and running, the investigator may want an extensive study of the effects of changing parameter values. If there are many parameters, this may require dozens, perhaps thousands, of simulation runs. In a programmed simulation it is possible to generate an entire file of parameter combinations to be used in the simulation and then have the program use each combination, one after another. This procedure is not possible with STELLA since the user needs the mouse to readjust parameters. However, a sample application which comes with STELLAStack does allow the user to enter a series of values for one parameter; and a good HyperCard programmer could expand these capabilities to include more than one parameter. Extend also allows a series of simulations to be run from a parameter file. The only weakness here is that completely automating a series of simulations—running the simulations, and printing out data and plots—while the computer is unattended is a bit awkward using STELLA/STELLAStack or Extend. This should not be a major disincentive for either of these programs, however. Since STELLA and Extend allow a model to be constructed so very quickly, these programs might be used in the initial stages of building and testing a model that ultimately is to be reprogrammed in a standard language. This reprogramming is not a formidable as

A



B

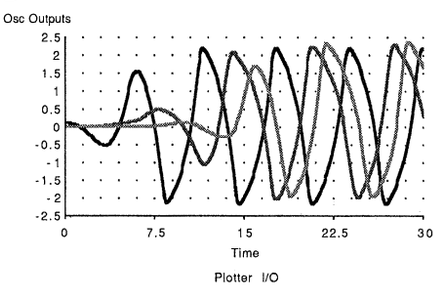


Fig. 5. (**A**) System of coupled oscillators created with Extend. The arrow-like symbol represents a single Van de Pol oscillator. The oscillator output is at the point of the arrow. The forcing signal is at the angle. (**B**) Results of Extend simulation for system of coupled oscillators diagrammed in (A). In this example the oscillators have different natural periods, but appear to move toward synchronization.

it first sounds. STELLA already generates finite-difference equations on which a "built-from-scratch" simulation program could be formulated, and Extend uses a C-like language that could be used as the basis for a simulation written in C.

Although STELLA and Extend are easy to use, there are a number of potential pitfalls for the numerically naïve. These pitfalls apply to simulation software in general. STELLA essentially solves systems of coupled first-order differential equations by numerical integration. For this process the user must select, or accept, the time-step size used in the numerical integration. When the step size is too large, the result may be merely inaccurate, wildly incorrect, or even numerically unstable, as can occur when the numerical result oscillates even though the true solution does not. To test for this kind of numerical nastiness, the user should observe what happens when the step size is decreased, for example, by a factor of 10. If the solution is essentially unchanged, the first step size is usually appropriate. Since there is a trade-off between accuracy and the speed of computation (a smaller step size requires more steps to complete the integration), different integration algorithms have been developed to make the computation more efficient (increased speed while maintaining accuracy). STELLA therefore allows the user to select the integration method from among three methods, namely, Euler, second-order Runge-Kutta, and fourth-order Runge-Kutta. Step size is also critical for numerical integration in Extend, which offers Euler and trapezoidal (modified-Euler) integration functions.

The unwary user should be informed that, as a practical matter, Euler integration is seldom used by numerical analysts because of the limited accuracy of this method (*10*). Fourth-order Runge-Kutta integration, on the other hand, can probably be considered the "old standby" (*11*). Modified-Euler integration also provides increased accuracy, but is not the most efficient technique (*10*).

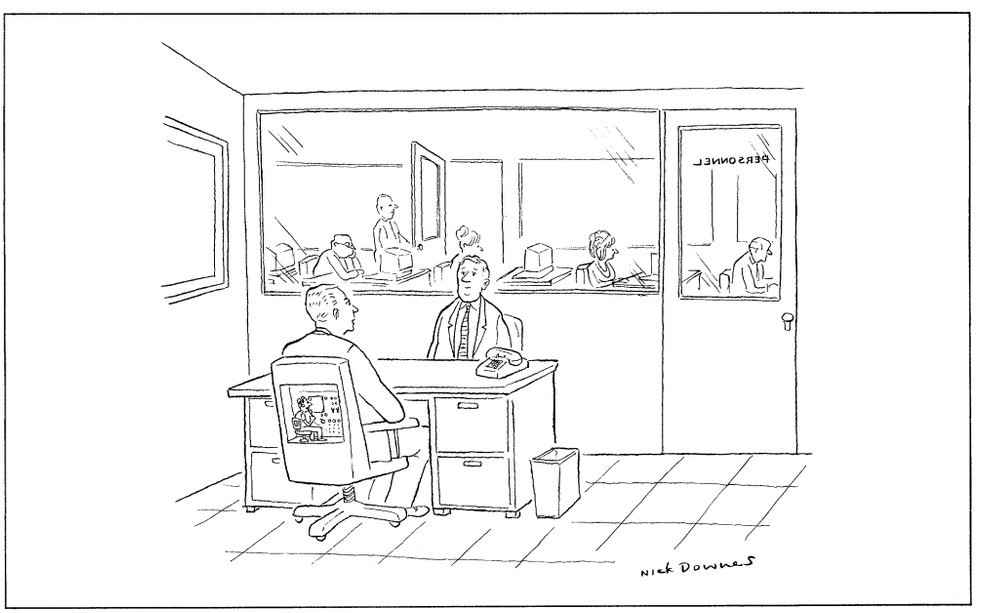
Finally, the user needs to be warned, as the Extend manual does do, that there is an additional subtlety when the simulation involves time delays. In this case the order of calculation may determine the simulation output. Extend permits sequential calculation among the blocks, from left to right, or simultaneous calculation of the blocks. The order of calculation is not so easily controlled in STELLA, however.

Approaches to Simulation

There are many kinds of software that can be used for numerical simulation. Certainly, the standard programming languages can be used for this purpose, and spreadsheet and equation-solving programs can be used as well. Also, for both the PC and Macintosh there is some special-purpose simulation software commonly used for electrical circuit design or control systems analysis, that can sometimes be used for more general-purpose simulation. The outstanding advantage of the STELLA and Extend software, however, comes from the Macintosh graphical interface. As a simulation becomes more complicated, the ability to visualize the block diagram or compartmental structure of the model ("what gets connected to what") becomes more important. In practice, simulation is seldom a matter of investigating a single model. Often, models are modified, torn apart, and rebuilt during the course of an investigation; to prevent confusion during the modeling process, the structure of the model must be clear at all times. Hence, one might say, "One simulation diagram is worth a thousand (programming) words."

REFERENCES AND NOTES

1. STELLA, High Performance Systems, 13 Dartmouth College Highway, Lyme, NH 03768.
2. Extend, Imagine That, Inc., 7109 Via Carmel, San Jose, CA 95139.
3. M. R. Cullen, *Linear Models in Biology* (Horwood, New York, 1985).
4. D. O. Cooney, *Biomedical Engineering Principles: An Introduction to Fluid, Heat, and Mass Transport Processes* (Dekker, New York, 1976).
5. The programs were tested on a Macintosh Plus and a Macintosh II (color monitor); both had 2 megabytes of random-access memory.
6. The mouse is a small, hand-operated device for inputting commands. The mouse is moved over a flat surface; a ball on the underside of the device and a set of detectors convert the motion of the mouse into the motion of a screen pointer. The screen pointer can be used for drawing or for selecting options from a menu; options are selected by pushing a button on the mouse (a process called "clicking," so named for the sound emitted).
7. J. C. Frauenthal, *Mathematical Modeling in Epidemiology* (Springer-Verlag, New York, 1980), pp. 89–98.
8. R. A. Wever, in *Mathematical Models of the Circadian Sleep-Wake Cycle*, M. C. Moore-Ede and C. A. Czeisler, Eds. (Raven, New York, 1984), pp. 17–77.
9. R. E. Kronauer, *ibid.*, pp. 105–127.
10. D. M. Young and R. T. Gregory, *A Survey of Numerical Mathematics* (Addison-Wesley, Reading, MA, 1972), pp. 422–492.
11. W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing* (Cambridge Univ. Press, New York, 1989), pp. 566–597.



"Yes, as a matter of fact, we are under micro-management."