Technical Report TR-89-33†

GENERAL PURPOSE VISUAL SIMULATION SYSTEM: A FUNCTIONAL DESCRIPTION

by

John L. Bishop and Osman Balci

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0106

15 October 1989

† Cross-referenced as Technical Report SRC-89-003, Systems Research Center, VPI&SU.

ABSTRACT

The purpose of the research described herein is to develop a software system that aids a simulationist in constructing and executing a general purpose discrete event visual simulation model. A literature review has shown the need for an integrated visual simulation system that provides for the graphical definition and interactive specification of the model while maintaining application independence. The General Purpose Visual Simulation System (GPVSS) developed in this research meets this need by assisting a simulationist to: (1) graphically design a simulation model and its visualization, (2) interactively specify the model's logic, and (3) automatically generate the executable version of the model in C programming language, while maintaining domain independence. GPVSS is developed on a Sun 3/160C computer workstation using the SunView graphical interface. It consists of over 11,000 lines of documented code. GPVSS has been successfully tested in many case studies.

CR Categories and Subject Descriptors: I.6.m **[Simulation and Modeling]**: Miscellaneous; I.6.2 **[Simulation and Modeling]**: Simulation Languages

Additional Key Words and Phrases: Animation, simulation environments, simulation support systems, visual simulation

TABLE OF CONTENTS

1. INTRODUCTION

Many visual modeling tools and packages have become available in recent years. Of the software currently on the market, most tend to be "either collections of FORTRAN subroutines, such as SEE-WHY, or purpose built languages such as Inter_SIM and Xcell" [Bell and O'Keefe 1987] or extensions to existing simulation languages. Examples of this approach include CINEMA which interfaces to the SIMAN simulation language [Johnson and Poorte 1988], and TESS which interfaces with the SLAM II simulation language [Standridge 1986]. RESQME [Gordon et al. 1988] provides a full visual simulation modeling environment, but forces the user to use such primitives as "Sources", "Sinks", "Chains", "Nodes", "Queues", etc. [Sauer et al. 1982]. There is a need for an integrated visual simulation system that provides for the graphical definition and specification of the model while maintaining application independence.

The purpose of the research described herein is to develop a software system that aids a simulationist in constructing and executing a general purpose discrete event visual simulation model. The software system assists a simulationist to: (1) graphically design a simulation model and its visualization, (2) interactively specify the model's logic, and (3) automatically generate the executable version of the model in C programming language, while maintaining domain independence.

Section 2 provides background knowledge in visual simulation. Section 3 describes the hardware and software environment of GPVSS. The functionality and usage of GPVSS are illustrated by using an example problem in Section 4. Concluding remarks are stated in Section 5.

2. BACKGROUND

Bell and O'Keefe [1987] report that the first significant work in Visual Simulation (VS) was conducted by Hurrion at the University of Warwick in England. Hurrion was trying to simulate a job shop manufacturing system. The difficulty in accomplishing this lay in the fact that often a human scheduler had control over the system. Attempts to model the scheduling process with traditional modeling techniques were unsatisfactory. Therefore it was decided to implement a "man-in-the-loop" model. In order for the human scheduler to make realistic decisions, it was necessary for the scheduler to be able to know the current state of the model. To accomplish this, an iconic visual display with letters depicting entities was created. From this and subsequent work, Hurrion introduced the term Visual Interactive Simulation (VIS).

Since its inception, VS/VIS has become an increasingly popular method of problem solving [Paul 1989]. Graphical symbology, movement depiction, display type, and the degree of user interaction vary greatly from one system to another.

In this section, we define the related terms; describe the graphical symbology and display type; introduce the types, levels, and methods of interaction; explain the VS/VIS model construction; discuss the conceptual frameworks for VS/VIS programming; provide five perspectives on VS/VIS; and finally present the advantages and disadvantages of VS/VIS.

2.1 Definition of Terms

Recent years have seen the increasing proliferation of discrete event simulation packages that include some facility for animation, visual simulation, visual interactive simulation, visual modeling, or visual interactive modeling.

Animation refers to any graphic display of information where the information to be imparted to the viewer is conveyed by image change [Baeker 1974]. This includes many displays clearly outside the scope of simulation. Therefore it is inappropriate to restrict the definition of "animation tool" to that which provides a display "portraying the dynamic behavior of the system model" with some variable degree of user interaction [O'Keefe 1987]. The term "animation" has been used to refer to a variety of dynamic simulation displays and degrees of user interaction. According to Mathewson [1985] "animation is a particular use of the topological information in a program generator". Standridge [1986] uses the term "animation" to refer to either simulation-concurrent or post-simulation animation. A *simulation-concurrent animation* is defined as a dynamic display generated by the state of the simulation model, regardless of whether user interaction with the model is supported. *Post-simulation animation* is a dynamic display driven by a simulation trace and may or may not include facilities for user interaction.

Visual Simulation (VS) is the process of building a model which permits a visual display of its input, internal behavior, and/or output, and experimenting with this model on a computer for a specific purpose.

Visual Interactive Simulation (VIS) includes all of the capabilities of VS and adds the capability of user interaction with the running model. This interaction may be either model or user determined, depending on who initiates the interaction [O'Keefe 1987]. Prompting the user to make some sort of scheduling decision is one example of model initiated interaction. In fact, the need for a user to know the current state of the modeled system when making such a decision is considered the impetus behind the first VIS efforts [Bell and O'Keefe 1987]. User initiated interaction allows the user to change model parameters and continue execution of the model. This ability to "play" with the model can be crucial for understanding the system. For example, the TRAVIS (Traffic Intersection Visual Interactive Simulator) [White and Balci 1989] allows the user to change such parameters as traffic light sequencing, interarrival times of vehicles, and observe the effect on the modeled system

produced by these changes.

The term *Visual Modeling* (VM) has been used relatively loosely in the literature, and refers generally to the process of building any form of Visual Simulation [Paul 1989]. *Visual Interactive Modeling* (VIM) is restricted to the process of building a VIS model.

2.2 Graphical Symbology

Currently there are a variety of methods used to represent the state of a system model visually. These methods consist of keyboard characters, icons, and three dimensional rendering [Paul 1989].

Keyboard characters are the simplest method of representing model objects. The use of keyboard characters is inexpensive in both hardware and development time. Hardware costs are lower because no special graphics hardware is required to display the character image. Development time for creating the object image is lower because the object images (characters) already exist. The tradeoff is that the diversity of model object images that can be employed is extremely limited, and one may feel constrained in adequately conveying the diverse objects in a system visually. In addition, object movement representation is crude at best.

Icons are a more faithful representation of system model objects. Icon images may be pre-defined, defined by the user, or generated automatically [Paul 1989]. The use of iconic representational graphics is typically more hardware- and development-intensive due to the need for special purpose graphics hardware and the time required to create the icon image. Despite the additional overhead, the use of icons to represent system model objects greatly enhances the quality of the visual simulation.

If desired, the visual simulation may be further enhanced by the addition of animation. Animation may be in the form of icon movement, or icon change. In icon movement animation, the icon is dynamically drawn on the screen with time, position, movement, and speed attributes analogous to the object being represented in the system. Icon change animation occurs when the animation is implicit in the varying icon image. Thus it would be possible to show a customer moving from a queue to a server, and show him walking at the same time. The overhead associated with icon change animation is considerable since it requires multiple icon images for each object to be animated and requires substantially more skill and effort of the modeler.

Finally, *three dimensional rendering* can be used for maximum realism. Techniques range from simple polygon projection to photo-realism. The primary drawback of these methods is that they are currently not able to run in real time due to hardware considerations. However, with technical advances such as the Intel 80810 RISC architecture CPU, which features on-chip Phong and Gouraud three dimensional shading, the prevalence of this technique is sure to increase.

2.3 Display Type

Hurrion [1980, 1986] classifies display types into schematic, logical, and null displays. *Schematic displays* attempt to parrot the system being modeled. Typically they have a detailed blueprint or schematic diagram as a background over which icons representing objects in the system move. This approach is particularly suitable for creating dynamic displays of relationships among objects where a spatial or simple logical relationship exists.

Logical or *summary displays* take the form of bar charts, histograms, time series, etc. Use of this type of dynamic display is appropriate when relationships among system model objects are extremely complex or when use of a schematic display is inappropriate. For large complex models, it may be more useful to present the user with a logical display containing summary statistics rather than observing a schematic display of the model.

Null displays enable the end user to run the simulation without the performance degradation associated with a dynamic visual display. This method is similar to executing a conventional simulation model.

2.4 Interaction

The degree of interaction that VIS provides users is perhaps one of its most significant benefits. Within VIS, interaction varies greatly in the type, level and methods for incorporating interaction.

There are two means by which a user may interact with a running model: model prompted and user prompted. *Model prompted interaction* occurs when the simulation itself prompts the user for input. An example of this method in Hurrion's seminal visual simulation of a job shop manufacturing system where the scheduler was prompted to make scheduling decisions. *User prompted interaction* is characterized by the user's ability to specify when the interaction is to occur. This method enables the user to change model parameters and continue execution of the model.

Hurrion and Secker [1978] categorize three levels of interaction: basic operations, priorities, and algorithms in the order of increasing levels of interaction. *Basic operations* consist of interactions that effect one-to-one changes in the system model. Any entity or any attribute of an entity on the dynamic display may be modified, deleted, moved, or replicated and the corresponding modification is incorporated into the system model. *Priority interactions* modify the priorities of operations, jobs, or entities. This form of interaction gives the user the capability of assigning dynamic priorities to entities for the balance of the simulation run. *Algorithm interactions* modify the various driving algorithms of the simulation model. Using algorithm interaction, an analyst would be able to create "algorithmic modules" which could be combined in various ways to produce the desired system model. Hurrion [1980] later combines priority and algorithm interaction into a new level called

structural interaction.

O'Keefe [1987] outlines three methods for handling both model- and user-initiated interaction: embedded programming, standard interactions, and stopping interpretation. The *embedded programming* method incorporates all display generating code into the model itself. This method provides a great deal of flexibility, but requires much more development effort and offers little reusability. Providing a library of *standard interactions* is a step forward. This method allows the modeler to pick and choose among a set of pre-defined interactions thereby decreasing development time. If need be, some standard interaction libraries may be extended. OPTIK is an example of such a library [O'Keefe 1987]. *Stopping interpretation* is perhaps the most powerful method by which a user may interact with the simulation model. As the name implies, the underlying code of the simulation model must be interpreted in order to utilize this method. When the interpretation is halted, it is possible to actually alter the simulation model code. While this may provide great flexibility, it does require the user to have some knowledge of the code being modified.

2.5 Construction of VS/VIS Models

Two methods of VM/VIM currently exist. The first technique separates design of the model and the dynamic display [Macintosh et al. 1984; Hurrion 1980]. Hurrion [1980] explicitly divides the VM/VIM process into construction of the simulation model and design of the display and interaction facilities. This has the effect of forcing the modeler to construct two models: a model of the system, and a visual display which is in effect a model of the model.

An alternative method is to provide the user with the capability to create the system model and specify the dynamic display at the same time, so that a one-to-one correspondence exists between objects in the display and objects in the model [Gordon and MacNair 1987; Bell and O'Keefe 1987]. Unfortunately most of these systems have suffered from a distinct lack of application independence.

2.6 Conceptual Frameworks for VS/VIS Programming

Event scheduling, activity scanning, three-phase approach, and process interaction are the commonly used conceptual frameworks (world views) for VS/VIS programming. (Balci [1988] describes how to implement a simulation model in a high-level programming language using these conceptual frameworks.)

Since the *event scheduling* conceptual framework is event (or time) based, animation of state changes in the system model is relatively straightforward. However, if user interaction with the model is desired, several difficulties arise. Since the simulation always leads the animation display by some amount, the additional burden is placed on the simulation of continuously keeping track of

the model state at the current animation time. Also, any future events that have been calculated must be re-calculated using the new system model parameters.

Hurrion [1980] advocates the *activity scanning* conceptual framework because it provides significant "advantages for interaction at run time". Provided the model is designed such that a one-to-one relationship exists between objects in the model and objects in the dynamic display, it is a relatively easy task to change the activity tests.

The *three-phase* and the *process interaction* conceptual frameworks possess the advantages of activity scanning and the disadvantages of event scheduling. Since the process interaction conceptual framework is object based, changes in attribute values of objects in the dynamic display can be reflected quite easily in changes in the appropriate system model object attributes. On the other hand, objects experiencing delays which are time-based and unconditional must be handled using a future object list [Balci 1988] which must be handled in a manner similar to that discussed for event scheduling.

Ultimately, the choice of a conceptual framework for implementing a VS/VIS model is dependent on the degree of user interaction desired and the nature of the system being modeled.

2.7 Perspectives on VS/VIS

O'Keefe [1987] defines five separate views of VS/VIS: statistical, decision support, computer aided design (CAD), gaming, and simulator.

The *statistical* or *traditional perspective* treats VS/VIS as a selling aid. Statistical experimentation is the primary means of decision support. Little or no provision is made for the user to interact with the system model as it is running. This perspective is almost mandated by the use of post-simulation animation, since any significant degree of user interaction is ruled out. *Decision support perspective* views VS/VIS as a decision support tool to solve unstructured problems. Consequently much emphasis is placed on providing the user with facilities to interact with the system model. Use of a standard set of interactions may limit the usefulness of a VS/VIS system under this perspective. The *CAD perspective* perceives VS/VIS as a tool for designing a system by combining either pre- or user-defined parts. The CAD perspective is particularly appropriate when an object-oriented approach is used. Users can create instances of objects and place them in the system model. The CAD perspective is prevalent with VS/VIS users involved in manufacturing system design. *Gaming perspective* views VS/VIS as a tool for learning as opposed to analysis of results. The *simulator perspective* incorporates VS/VIS as a rudimentary simulator with the user as the "man-in-the-loop".

The perspectives presented above are sufficient to describe the domain of capabilities of

VS/VIS when taken as a whole. However, no single perspective currently exists to adequately encompass VS/VIS.

2.8 Advantages of VS/VIS

Many benefits are associated with VS/VIS [Bell and O'Keefe 1987; O'Keefe 1987]. O'Keefe [1987] states that VS/VIS gives the client and developer three additional capabilities not found in traditional simulation methods in the areas of selling, gaming, and learning.

VS/VIS augments *selling* by providing both a communication and a presentation medium. During model development the dynamic display becomes a communication medium that enables the modeler and the client to discuss model validation, development, and experimentation. The dynamic display becomes a presentation medium for the presentation of model results obtained using VS/VIS or by traditional statistical experimentation.

VS/VIS enhances *gaming* capabilities by making it possible to use gaming with relatively complex system models. The need for user interaction with the running system model was the impetus behind Hurrion's original job shop visual simulation.

Learning capabilities are enhanced because the interactive capabilities of VIS allow the client to "play" at managing the system. This allows the client to gain knowledge pertaining to management of the system.

In addition to these augmented capabilities, the following are generally accepted advantages that VS/VIS provides.

The dynamic display associated with VS/VIS augments model credibility through the enhanced presentation of simulation results. This is due to the fact that the client need not have an extensive simulation background to understand model results [O'Keefe 1987]. In a survey by Bell and Kirkpatrick [1986] seventy percent of the respondents felt that their decisions were implemented more often when VS/VIS was used. Although Bell and Kirkpatrick [1986] themselves admit the validity of their survey is in question, clearly VS/VIS has had some beneficial effect on model credibility for some.

VS/VIS enhances model verification, validation, and testing. This is because the analyst can easily see the effect of incorrect behavior when the system model is not working correctly [Paul 1989]. In addition, the communication medium that VS/VIS provides enables the client to participate to a greater degree in establishing the simulation model credibility. In the same survey by Bell and Kirkpatrick [1986] eighty-eight percent of the respondents said that validation was either "faster" or "much faster" when using a VS/VIS system.

One additional benefit of VS/VIS is that the client can be involved in model development at a

much earlier stage than is otherwise possible. This allows the developer to tailor the model much closer to the clients needs, and to possibly avoid costly misunderstandings. In addition, the client may soon know enough about the system to ask questions that would not have previously been thought of [Palme 1977].

2.9 Disadvantages of VS/VIS

VS/VIS has definitely advanced the state of the art in discrete event simulation. However, VS/VIS does have some undesirable characteristics that the analyst and client should be aware of before choosing to use VS/VIS.

It is difficult to estimate the cost/benefit ratio of developing a VS/VIS simulation [Bell 1985]. While increased costs are certain, the benefits of implementing VS/VIS are much less tangible.

VS/VIS tends to be hardware specific [Bell 1985] due to graphics requirements. Furthermore, the choice of hardware determines the graphics capabilities available to the modeler. While increased complexity may or may not be desirable, this does have the effect of placing a limit on the maximum complexity of the model display.

The use of VS/VIS introduces further requirements for the construction and presentation of model results that would not be present for conventional simulation techniques. In addition to constructing the system model, the analyst is now responsible for creating a dynamic display. Currently, a good screen design methodology has yet to be developed within the simulation community. Consequently, the modeler must also be substantially familiar with graphics design and programming. Some aspects of model behavior may be difficult or impossible to portray visually. Paul [1989] presents an example of a port model that easily portrays incoming ships entering their berths. However, the assignment of a given ship to a berth, the primary impetus behind the simulation, is dependent upon many different attributes such as cargo, berth capabilities, and ship type. Paul [1989] states "Any attempt to represent visually such rules would probably be self-defeating" due to the resultant complexity in the dynamic display.

The use of a dynamic display as a presentation medium may require the modeler to add more detail to the system model than actually required to obtain the desired statistical results. Bell [1985] cites an example of a truck dispatching system model. If the system road network is dense, it may not be relevant which route a truck actually takes to arrive at a given point. The client, however, may not fully accept the model as credible without actually seeing which route the truck takes.

A dynamic display is an aid in model verification/validation, not a means in itself. Paul [1989] makes the observation that "undue confidence" may be placed in the model merely because it "looks alright". Bell [1985] warns that since VS/VIS displays the model transient period in great detail, and

since the client can observe this transient behavior in the system every day, there is a distinct possibility that the client may assess model credibility based on the transient period rather than steady state.

The client must be aware that interacting with the running system model by changing model parameters invalidates the usual assumptions associated with steady-state analysis of model output. The danger exists that the client views a "snapshot" of model behavior and assumes that the system always exhibits these characteristics [Paul 1989]. The use of a dynamic display does not abrogate the need of the modeler to instruct the client about the nature of statistical methods.

Paul [1989] addresses one area that has been distinctly lacking in VS/VIS literature – human factors. The expressions "seeing is believing" and "a picture is worth a thousand words" immediately come to mind when discussing VS/VIS. Yet within the legal community eyewitness evidence is considered the least reliable form of evidence. Paul [1989] states that "dreams, wishes, desires and thoughts" can affect an individual's perceptions. Another human factors consideration is the choice of colors used in the dynamic display. Red and green are two of the most widely used colors in VS/VIS but roughly ten percent of the total population is unable to distinguish these due to color blindness [Paul 1989]. Thus, considerable attention must be paid by the modeler to established human factors guidelines when designing a VS/VIS dynamic display. As the quality and quantity of VS/VIS systems increase, the impact of human factors considerations is increasingly to be felt. The onus is on the modeler to determine when, where and how to implement VS/VIS.

3. HARDWARE AND SOFTWARE ENVIRONMENT

GPVSS is implemented on a SUN 3/160C color computer workstation running the SUN UNIX operating system. The SUN workstation consists of a 16.67-MHz MC68020 microprocessor with a 16.67-MHz MC68881 floating-point co-processor, a 380-MB Fujitsu Eagle disk subsystem, a 1/4-inch cartridge tape subsystem, 8-MB of main memory, a 19-inch color monitor with a resolution of 1152 X 900 pixels, a pointing device called a mouse, an Imagen laser printer, and a connection to the Ethernet network which enables high speed file transfer to and from other University computing systems.

The most significant element of the user interface is the 19-inch bit-mapped display screen. This display technology offers excellent graphics capabilities. The user communicates with the SUN via keyboard input and the mouse. The mouse may be used to point to or select locations on the screen. The system provides feedback on the current mouse location by continuously updating the position of the mouse pointer. Virtually any material that is displayed on the screen can be pointed to and treated as input.

The windows are the basic building block of the user interface and are roughly analogous to sheets of paper on a desktop. Windows may be resized, overlapped, closed, or quit. Resizing a window allows the user to effectively increase the available workspace. Overlapping is useful when the user needs to interact with multiple windows, but there is not enough room to display all of them. Closing reduces a window to a small 64 X 64 bit image known as an "icon". Icons are useful when a window contains a process that does not require immediate user interaction, but it is still desirable to keep that process running (shell, for example). Quitting a window destroys the window entirely when there is no further use for it.

Menus and buttons are perhaps the most important window features in terms of the user interface. Menus provide a user with the ability to select an option from a finite list of choices. Menus may be "pop-up", requiring the user to depress a mouse button before they are active. Buttons allow the user to interact in a more limited way with the running program. Selecting a button requires the user to position the mouse cursor over the button image and depress a mouse button. This action executes a C function that has been associated with the button by the programmer.

The GPVSS code consists of approximately 11,000 lines of documented SunView, C, and EQUEL/C code. All window features are programmed using the SunView application package which consists of high-level object-oriented routines to create a graphical user interface [Sun Microsystems 1988]. All storage and retrieval of model components and data are achieved by the use of INGRES relational database management system [Sun Microsystems 1986]. The use of INGRES is completely hidden from the user of GPVSS. EQUEL (Embedded QUEry Language)/C is used to access INGRES within SunView programs. Although many modern computer systems have the hardware capability to support GPVSS, the SunView interface technology has proved invaluable for its rapid development.

4. FUNCTIONAL DESCRIPTION

A problem is defined in Section 4.1 for use as an example to illustrate the functionality and usage of GPVSS. For the example problem, we show how to develop and execute a visual simulation model step by step.

4.1 Definition of an Example Problem

The operating system of a computer installation with a single central processing unit (CPU) controls the processing of jobs sent by three independent terminals in a round-robin manner. The user of each terminal "thinks" for an amount of time and sends a job to the computer system for execution. Think times are exponentially distributed with mean 250 milliseconds. The user is

inactive until the computer's response starts appearing on the terminal. Thereupon, the user goes through a "thinking" process again and sends the next job. The arriving jobs join a single queue, with first come first served discipline on CPU access. Repeated usage of the CPU is accomplished in a round-robin manner. The CPU allocates to each job a maximum service quantum of length QUANTUM (100) milliseconds (not including overhead). If the job is finished within this service time period, it spends a fixed overhead time OVERHEAD (15) milliseconds at the CPU after which the job's output is sent to the originating terminal. If the job is not finished within QUANTUM, its remaining service time is decremented by QUANTUM and it is placed at the end of the queue after spending a fixed overhead time of OVERHEAD. Job service times are exponentially distributed with mean 500 milliseconds. All transmission times between the terminals and the host are assumed to be negligible. (See Figure 5 for a graphical description of the problem.)

4.2 Model Generator

Typing GPVSS activates the GPVSS and displays the top-level window shown in Figure 1. GPVSS has three major components: Model Generator, Model Translator, and Visual Simulator. This section describes the first component.

With the use of the Model Generator component of GPVSS, we graphically design the model and its visualization, define submodels, interactively specify the logic of each submodel, and define and specify objects and attributes. When the Model Generator icon is selected, GPVSS pops up a menu for the user to input the name of the model to construct as depicted in Figure 1. Upon the entry of model name, GPVSS displays the image editor window which can be resized to the size of the monitor as shown in Figure 2.

4.2.1 Drawing Background and Dynamic Object Images

The first step in constructing the model is to draw the images that are used in the dynamic display. Two types of images must be created: the model background images and the Dynamic Object (DO) images.

The drawing is achieved by using the pen, line, rectangle, and select modes of the Image Editor. Pen (default) mode is used to draw freehand using the left mouse button depressed. Line mode is entered by selecting the "Line" button on the Image Editor control panel and is used to draw lines on the canvas between two points. Rectangle mode is entered by selecting the "Rect" button and is used to draw rectangles. The select mode is provided so that the user can select any portion of the drawing for duplication on the canvas or storing on disk. The "Load" and "Save" buttons on the control panel are used to load and save background and object images. The Image Editor window is scrollable in x

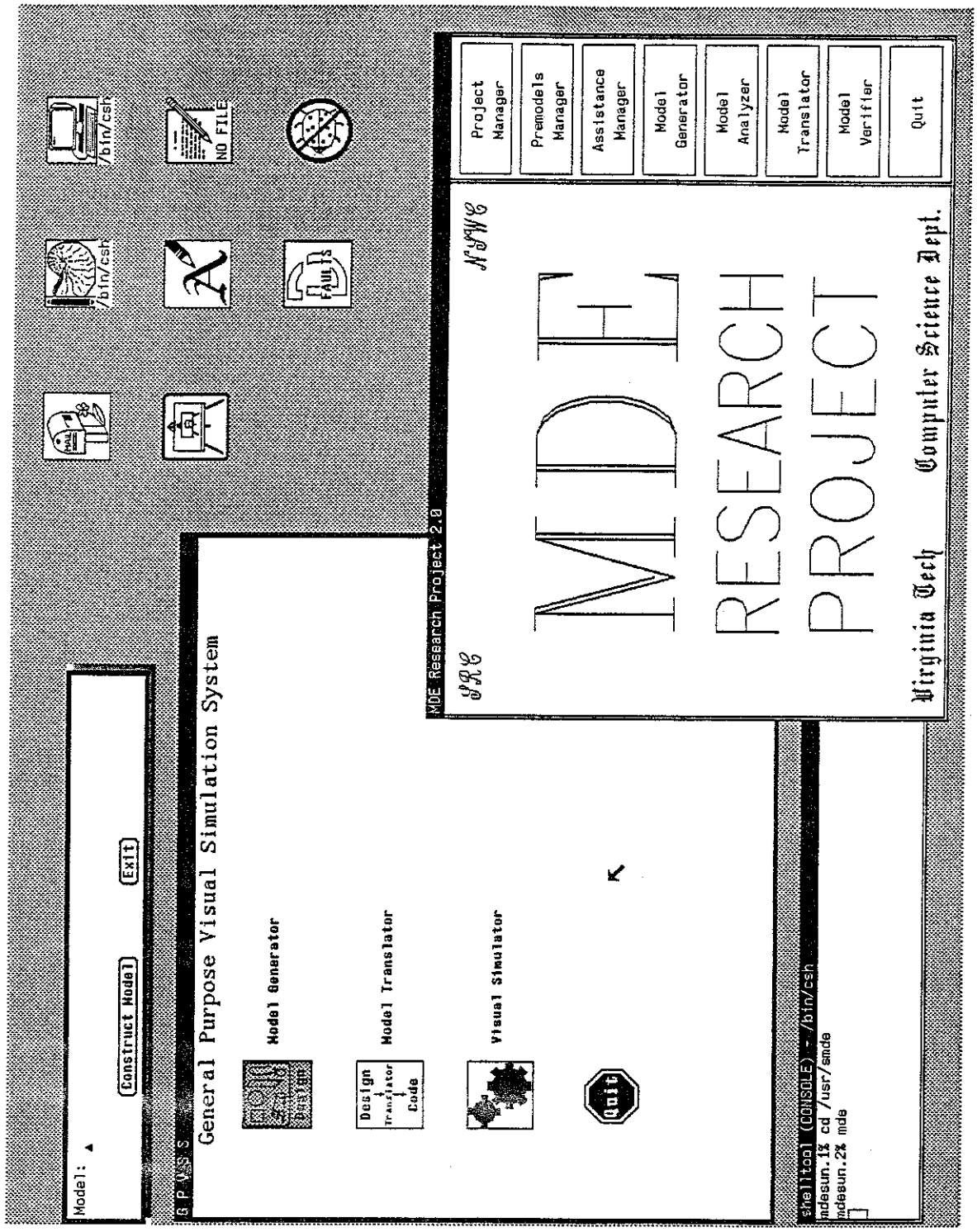

Figure 1. GPVSS Top-Level Window and Activation of Model Generator

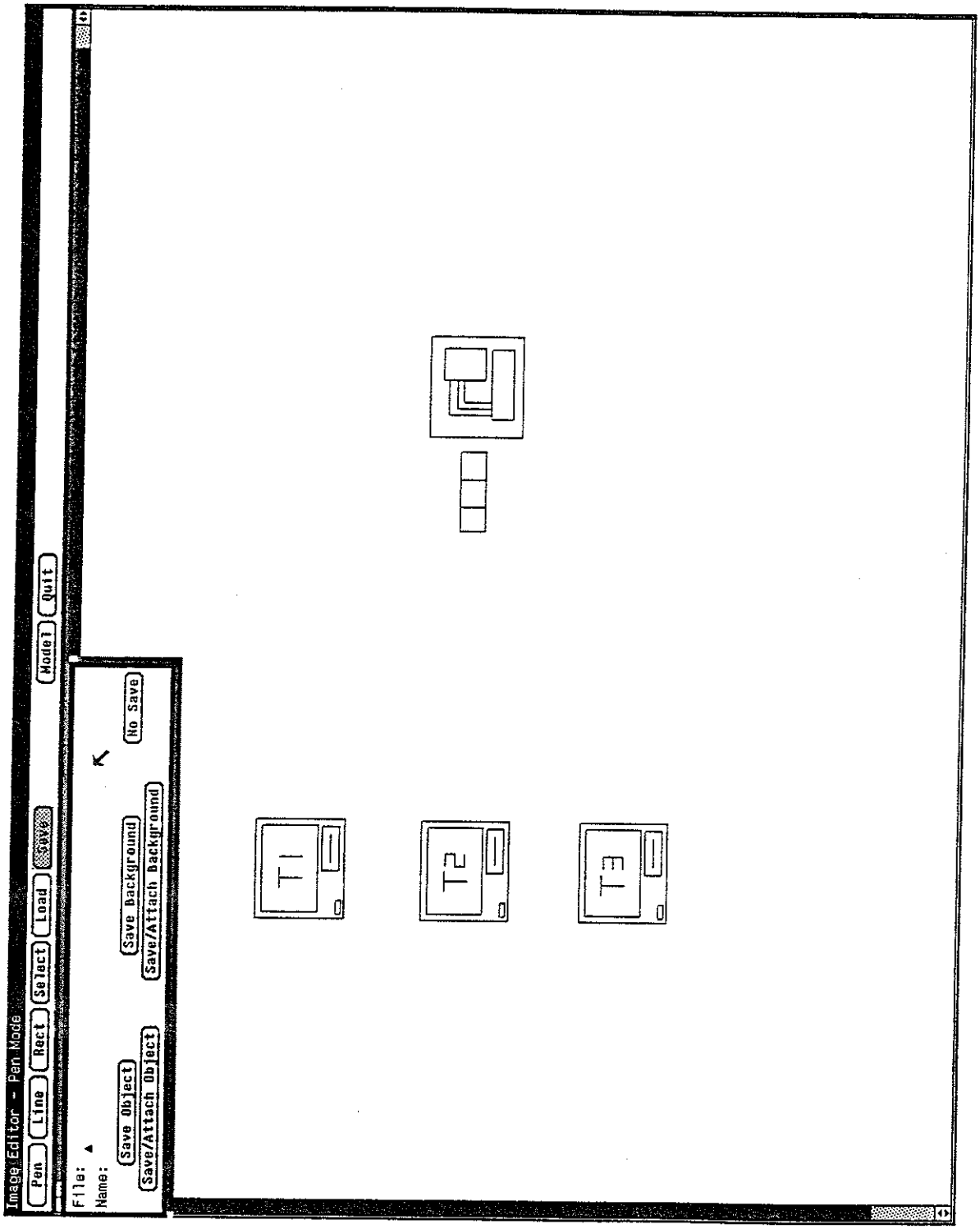

Figure 2. The Model Generator's Image Editor Mode and Drawing and Saving Background Images

and y directions as indicated by the arrows in the scroll bars in Figure 2.

For the example problem, the background images are drawn as shown in Figure 2. The quality of background images is limited only by the patience and artistic capability of the modeler. To associate the background images with the model, we select the "Save" button on the Image Editor control panel. This brings up a blocking requester for the filename (see Figure 2). The file path name is entered at the "File:" item and the "Save/Attach Background" button is selected. This saves the background in the specified file and places an entry in the INGRES relational database associating this file with the background images of the model.

Next we need to construct the DO images. To clear the screen, select the "Quit" button and re-enter the Image Editor. The image of the DO representing the job originating from Terminal 1 is drawn as shown in Figure 3. To save this DO image, enter the "Select" mode and select it. Then click on the "Save" button to bring up the pop up menu shown in Figure 3. Fill in the appropriate file path name and object name. Select the "Save/Attach Object" button to store the DO image in the specified file and to place an entry in the INGRES database so that the DO image may be referenced by name. Repeat this process for the DOs originating from Terminal 2 and Terminal 3 using "job2" and "job3" as the DO names.

Upon constructing and saving the background and DO images, we enter the Model Editor. First erase the DO images and re-load the background image. Select "Load" button from the menu, then select the "Load Attached Background" button. This results in the restoration of the previously saved background images. Note that the editor is still in select mode, but any previously selected image has been destroyed by the image loading process. Now select the "Model" button to enter the Model Editor.

4.2.2 Defining Submodels and Paths

For the example problem, we wish to define five submodels: Terminal1, Terminal2, Terminal3, CPU_QUEUE, and CPU. To define the "Terminal1" submodel select the "SubMod" button on the Model Editor control panel. This brings up a blocking requester for the input of the submodel name (see Figure 4). Enter "Terminal1" at the prompt and click on "Define Sub-Model". The editor is now ready to draw a rectangle to define the boundaries of the "Terminal1" submodel for logic specification and path definition purposes. This process is identical to drawing a rectangle. A submodel definition can be deleted by depressing the middle mouse button. This feature becomes inactive if another editor mode is entered. This process is repeated to define the other submodels.

The paths connecting the submodels should now be defined. Paths are uni-directional. If a DO moves both ways between two submodels, two paths must be specified. Not all submodels must be

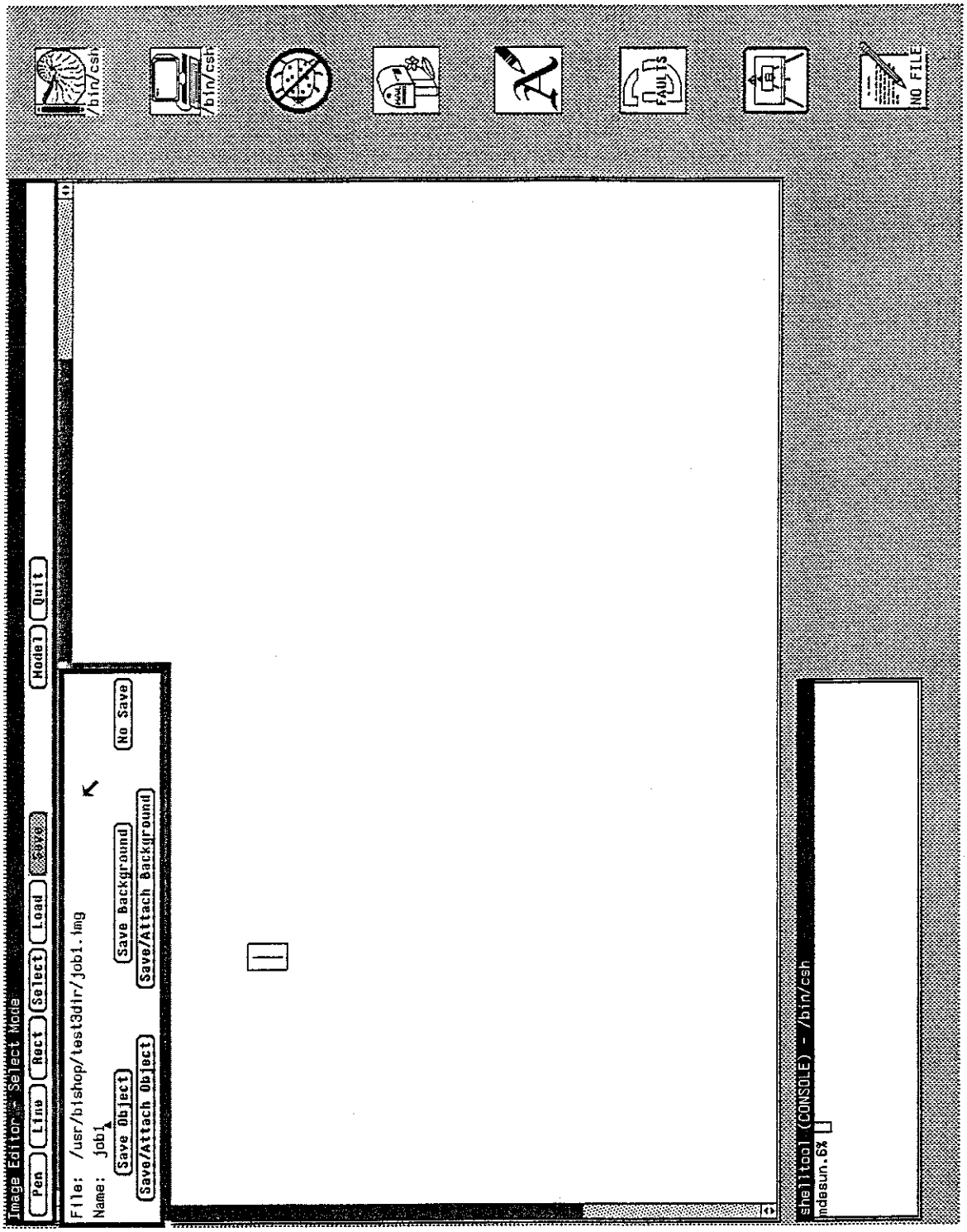

Figure 3. Drawing and Saving Dynamic Object Images

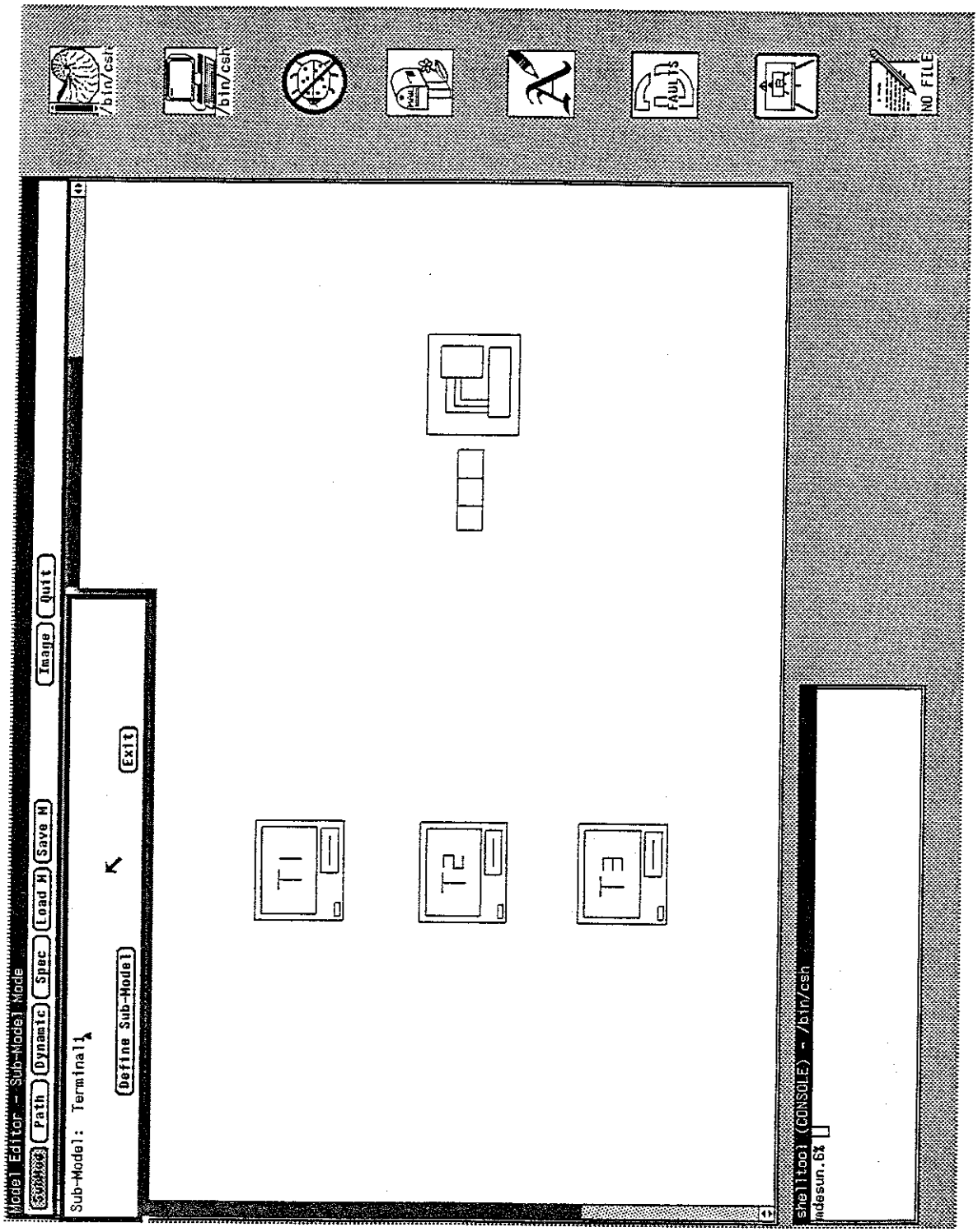

Figure 4. Submodel Definition

interconnected, but there must be a path between any two submodels in the same direction a DO could possibly travel. If a DO tries to move to and from submodels that are not connected by a path, an error occurs.

To define a path, first select the "Path" button on the Model Editor control panel. This brings up a blocking requester for the path name. After entering the path name select the "Define Path" button to start the path definition.

Locate the mouse cursor at the point the path is to begin. This point must begin within a submodel boundary (see Figure 5). Press the left mouse button to select the first point on the path. If the mouse cursor is outside a submodel, this has no effect. Subsequent intermediate points may be specified regardless of submodel boundaries by depressing the left mouse button. The path termination point must be in a submodel and is selected with the middle mouse button. Repeat the above process until all paths are defined. The completed submodel and path definitions are shown in Figure 5.

To save the completed submodel and path definitions, select the "Save M" button on the Model Editor control panel. This brings up a blocking requester for a file name (Figure 5). Enter the desired file name and select "Save/Attach Model". This stores the model and makes an appropriate entry in the INGRES database. For future model saves, the file name defaults to the last one entered if nothing is typed at the prompt, including blanks. Saved models may be loaded by selecting the "Load" button.

4.2.3 Specifying Submodels and Their Attributes

Select the "Spec" button on the Model Editor control panel to enter specification mode. A pop-up menu is activated by depressing the right mouse button while within a submodel boundary (see Figure 6). Depressing the right mouse button outside of a submodel boundary has no effect. Select the "Sub-Model Logic" menu item while within "Terminal1" submodel to specify its internal logic. This brings up a blocking requester for the path name of the directory in which all created files are stored. This path should only be entered once for all submodels. Subsequent specifications should merely select the "Specification Path" button without typing anything at the prompt, including blanks. Selection of the "Specification Path" brings up the submodel specification window.

The specification of the "Terminal1" submodel is shown in Figure 7. The Entrance Condition (EC) for "Terminal1" is true. This means that DOs can always get into this submodel. Once the DO has entered the submodel, its origin is set to 1, and rservice_time (remaining service time) is sampled from an exponential distribution with mean 500. "Origin" and "rservice_time" are DO attributes that

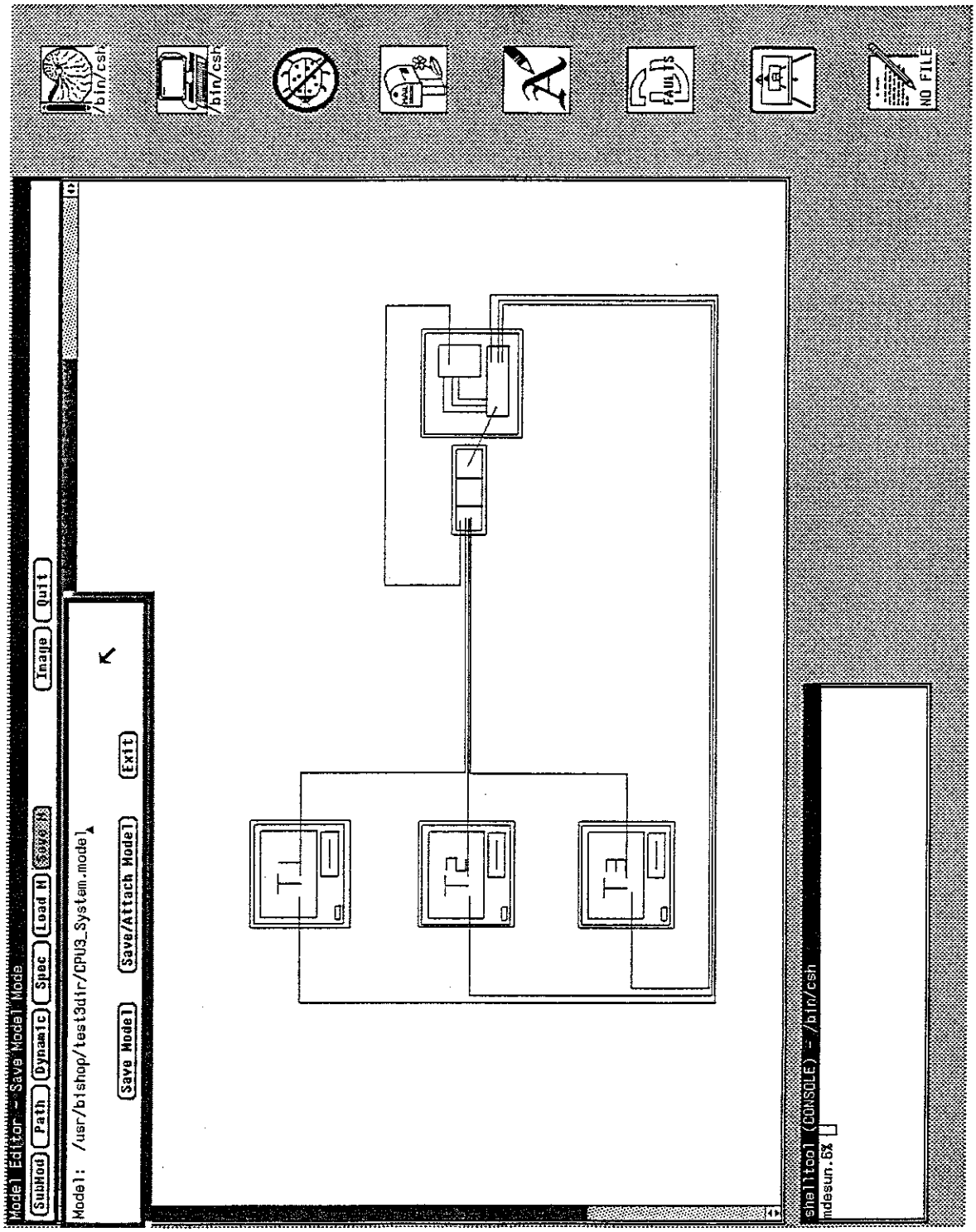

Figure 5. Path Definition

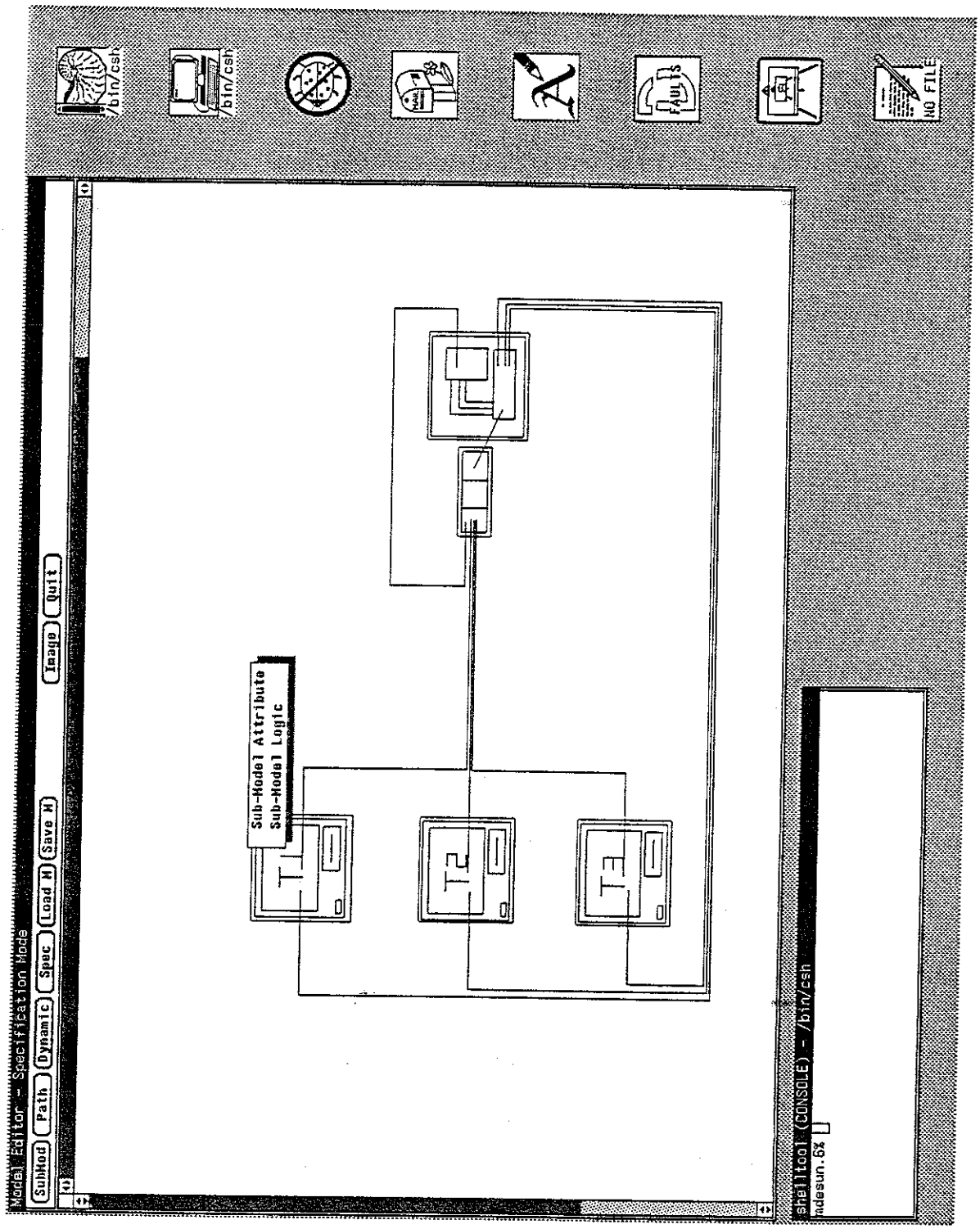

Figure 6. The Model Editor's Specification Mode

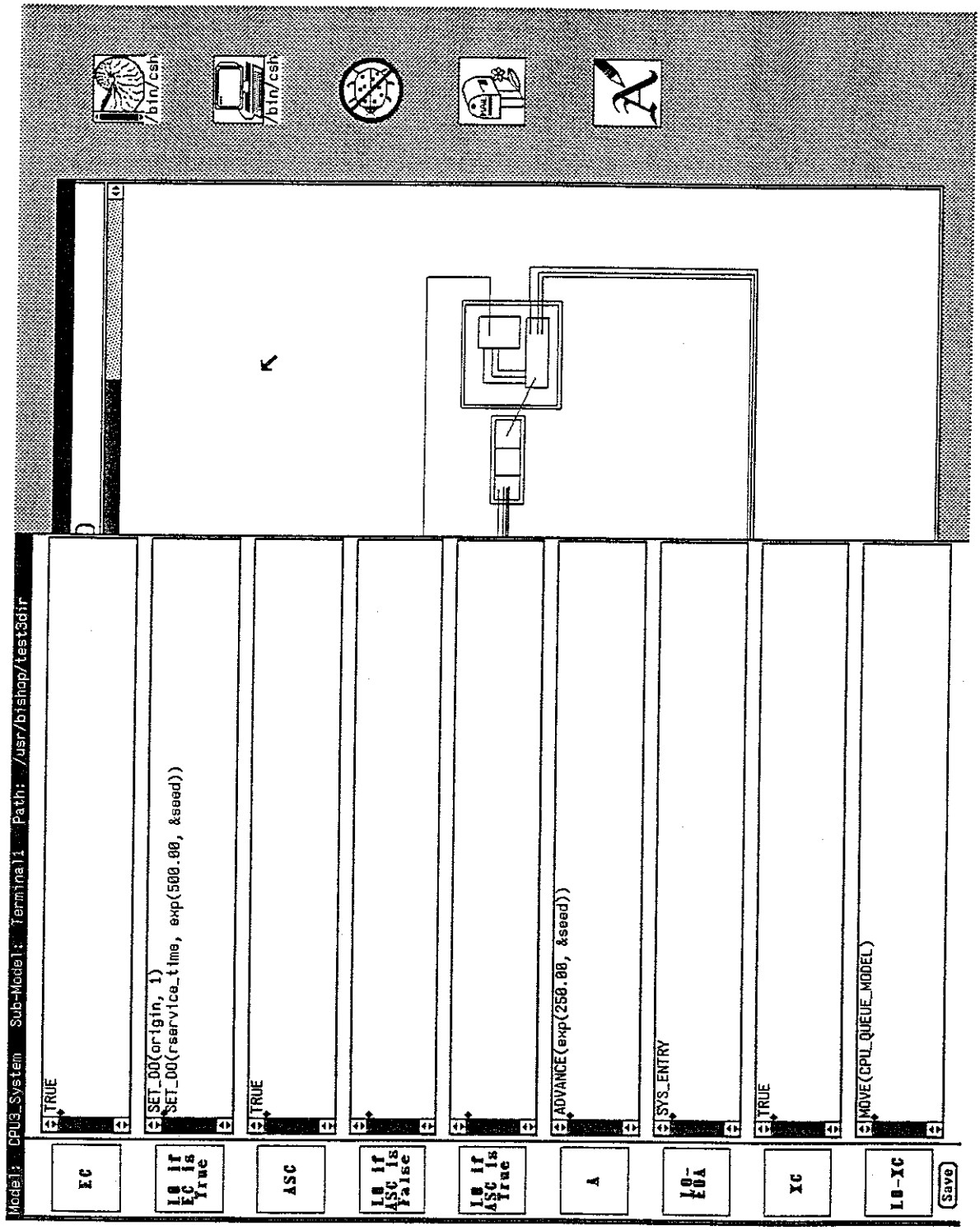

Figure 7. "Terminal1" Submodel Logic Specification

must be defined and specified along with the DOs. For now it is best to keep a list of attributes used and define them all at once later. "Seed" is a pre-defined integer seed for the random number generator. The two SET_DO statements constitute the Logical Operations (LO) if the EC holds true. Note that all subwindows in Figure 7 are scrollable up and down as indicated by the arrows in the scroll bars.

Since the Activity Start Condition (ASC) is true, the DO may engage in the activity right away. The activity duration is sampled from an exponential distribution with mean 250. This is the user "think" time. The DO remains engaged in the Activity until this time has passed.

After the activity is completed, the "SYS_ENTRY" macro (in C programming language) records the DO entering the computer system. Since the eXit Condition (XC) is true, the DO leaves the submodel and moves to the CPU_QUEUE submodel. Note the "_MODEL" extension to the CPU_QUEUE submodel name.

Upon completing the specification, select the "Save" button located at the bottom left hand side of the window to exit the window and save the specification. The other submodels are similarly specified.

The specification of submodel logic is currently in the form of C macros. A high-level extensible specification language will be developed as part of the future research.

To specify submodel attributes, activate the pop-up menu by depressing the right mouse button while within a submodel boundary (See Figure 6). Select the "Sub-Model Attribute" item from the pop-up menu. This brings up a blocking requester for the input of the attribute name. After entering the attribute name, select the "Name Attribute" button to enter the attribute specification window shown in Figure 8. In entering the brief description of the attribute, after reaching the end of the Brief Description line, the line scrolls to the left continuing to accept input. Initial attribute value can be specified as a constant, read from a file, returned from a C routine, or sampled from a probability distribution.

4.2.4 Defining and Specifying Dynamic Objects

Select the "Dynamic" button on the Model Editor control panel. This brings up a requester for the Dynamic Object name as shown in Figure 9. After entering the DO name, click on "Name Dynamic Object" to enter the DO specification window.

The DO specification window for "job1" is shown in Figure 10. A brief description of "job1" DO is entered. In entering the brief description, after reaching the end of the Brief Description line, the line scrolls to the left continuing to accept input. "Job1" is described as a job originating from Terminal 1. The name of the first submodel the DO tries to enter is "Terminal1". We input the path

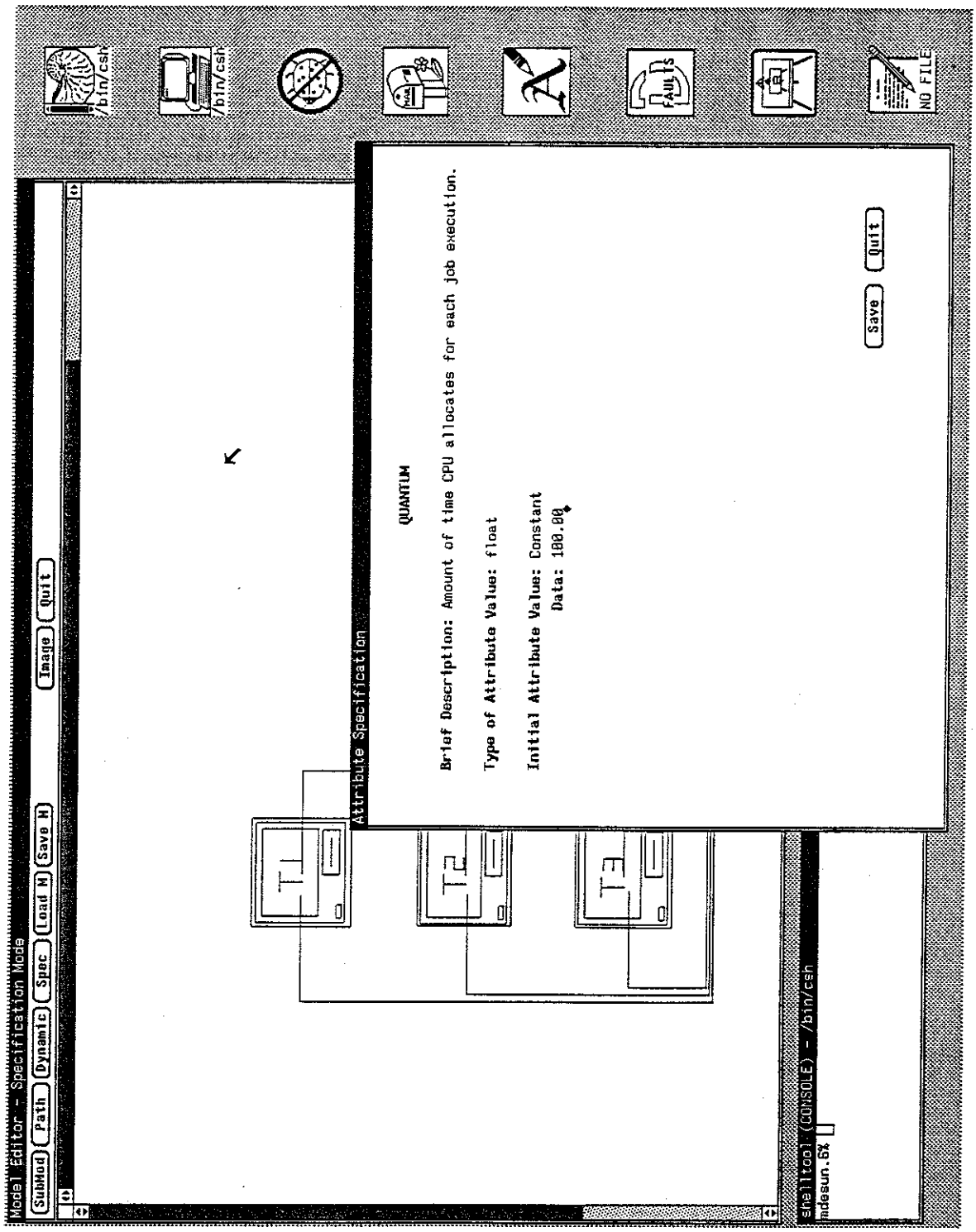

Figure 8. Submodel Attribute Specification

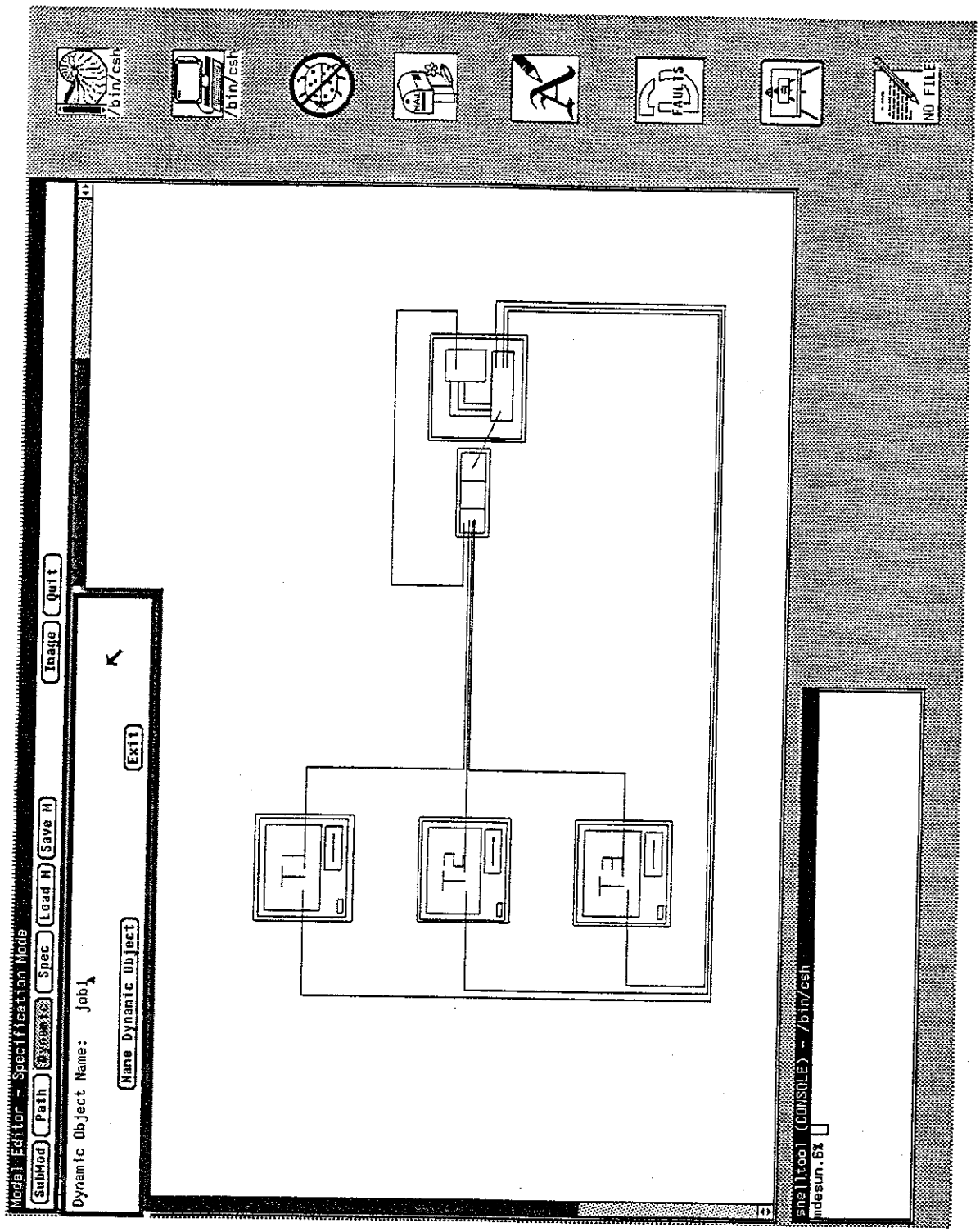

Figure 9. Dynamic Object Naming

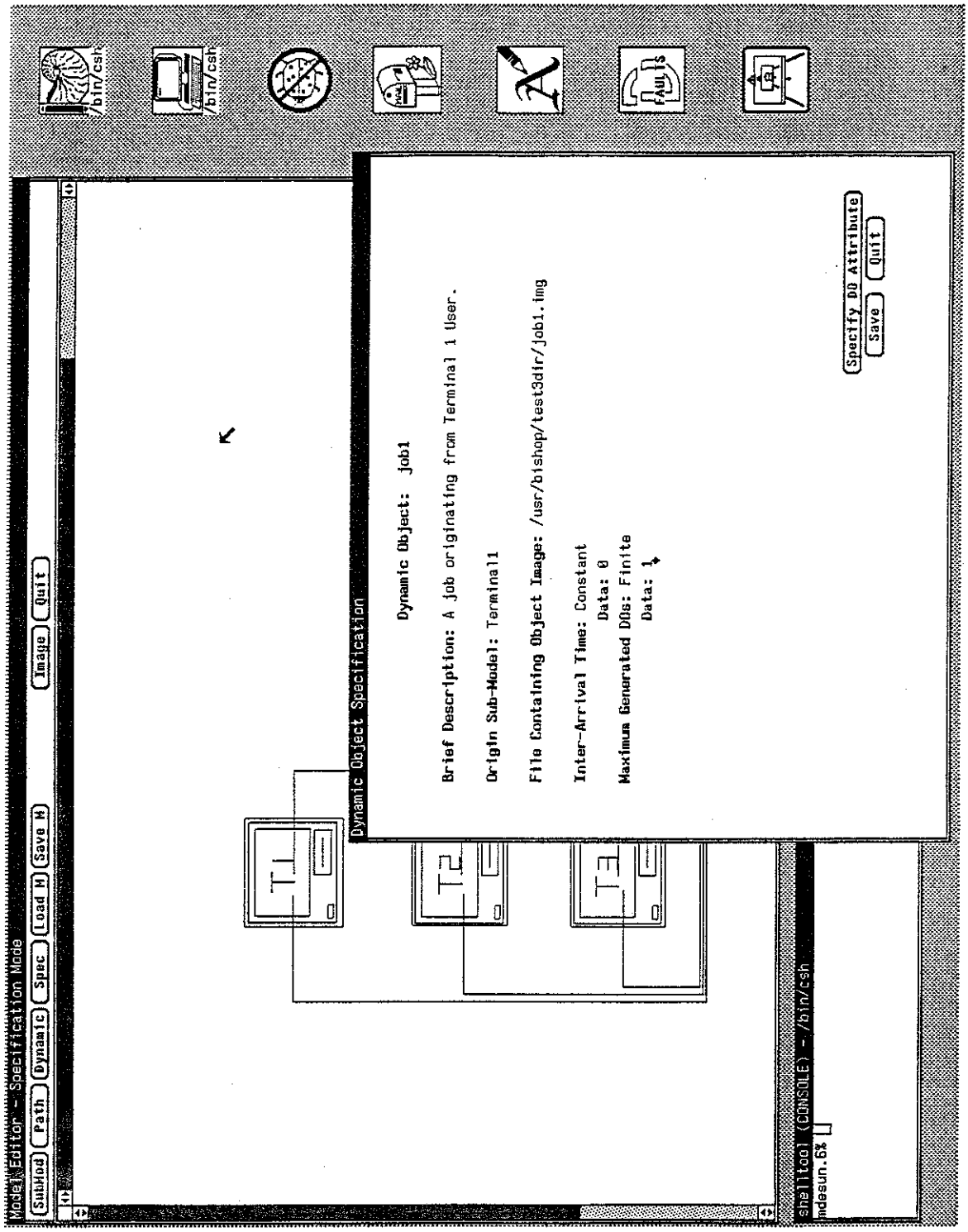

Figure 10. Dynamic Object Specification

name of the file that contains the "job1" image created earlier. Inter-Arrival Time can be specified as a constant, read from a file, returned from a C routine, or sampled from a probability distribution. The maximum number of DOs to be created of this type is one.

The DO attributes "origin" and "rservice_time" used previously in the submodel specification should now be defined and specified. DO attributes are global to all DOs. Thus a given attribute need only be defined once. Select the "Specify DO Attribute" button to bring up a blocking requester for the DO attribute name. Select the "Name Attribute" button to enter the attribute specification window. DO attribute specification is identical to submodel attribute specification.

When all DO attributes are defined, exit the DO specification window by selecting the "Save" button. This exits the program and places appropriate entries in the INGRES relational database.

4.3 Model Translator

The Model Translator component of GPVSS is activated as shown in Figure 11 and is used to transform the visual simulation model specification, created by the use of the Model Generator component, into an executable simulation. The Model Translator uses the process interaction conceptual framework [Balci 1988] for implementing the model. Select the "Model Translator" icon on the top level window to bring up a blocking requester that should already have the system model name entered. Select the "Generate Code" button to create an executable visual simulation model in C programming language. The blocking requester clears after the completion of the code generation.

4.4 Visual Simulator

The Visual Simulator component of GPVSS is used to run the simulation in visual mode or with no visualization. Select the "Visual Simulator" icon and wait for the dynamic display screen to come up (see Figure 12). Select the "Load" button to load the background image and initialize variables. Select the "Animate" button to toggle the visual simulation on and off. Selecting the "Run" button brings up the statistics panel as shown in Figure 13. Random number generator seed, number of independent replications, the length of transient (warm-up) period, and the length of steady-state period are specified. Click on the "Run Simulation" button to run the model without the dynamic display. Results of the simulation are stored in terms of confidence intervals for the performance measures of interest in a file.

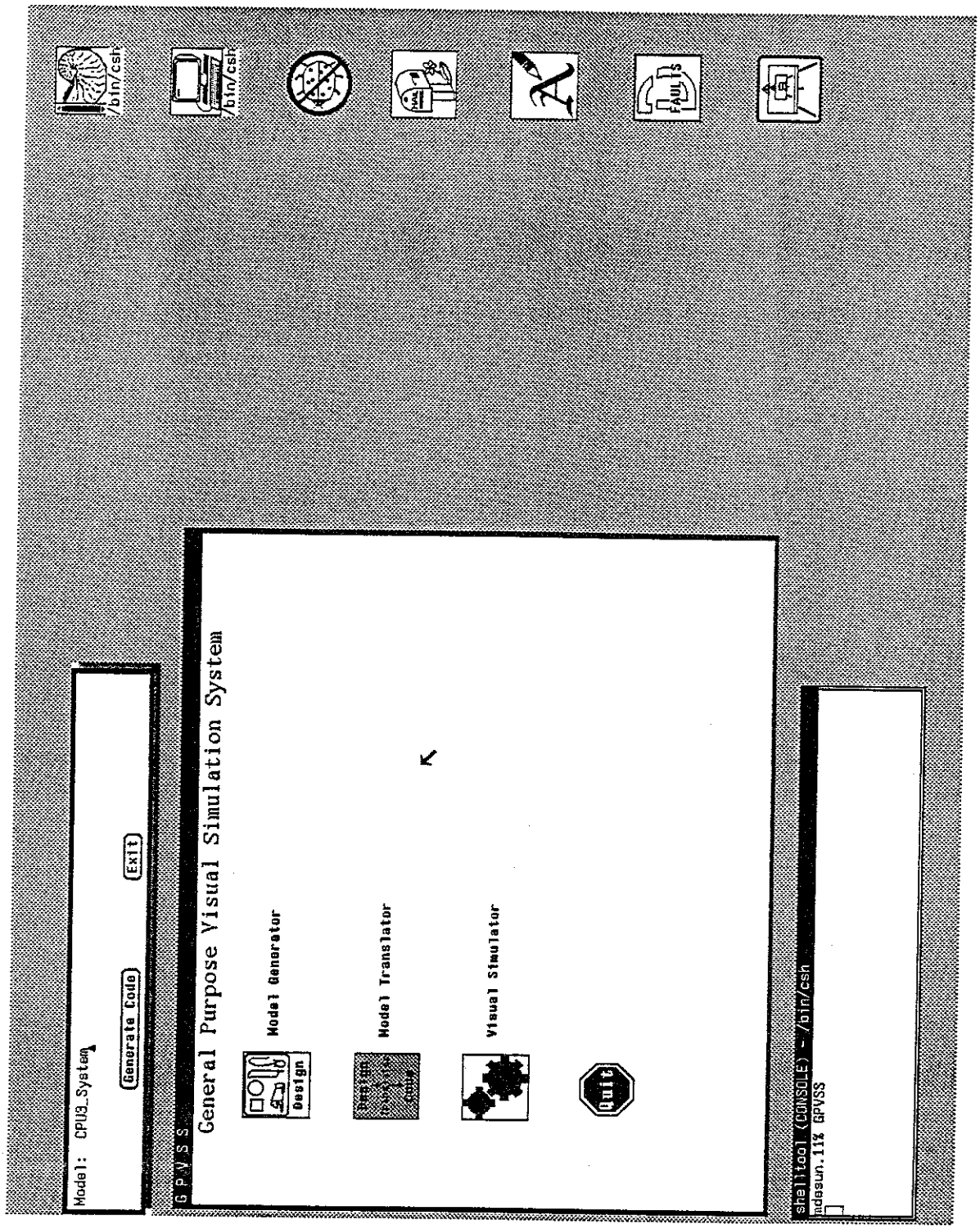

Figure 11. Activation of Model Translator and Automatic Generation of Code

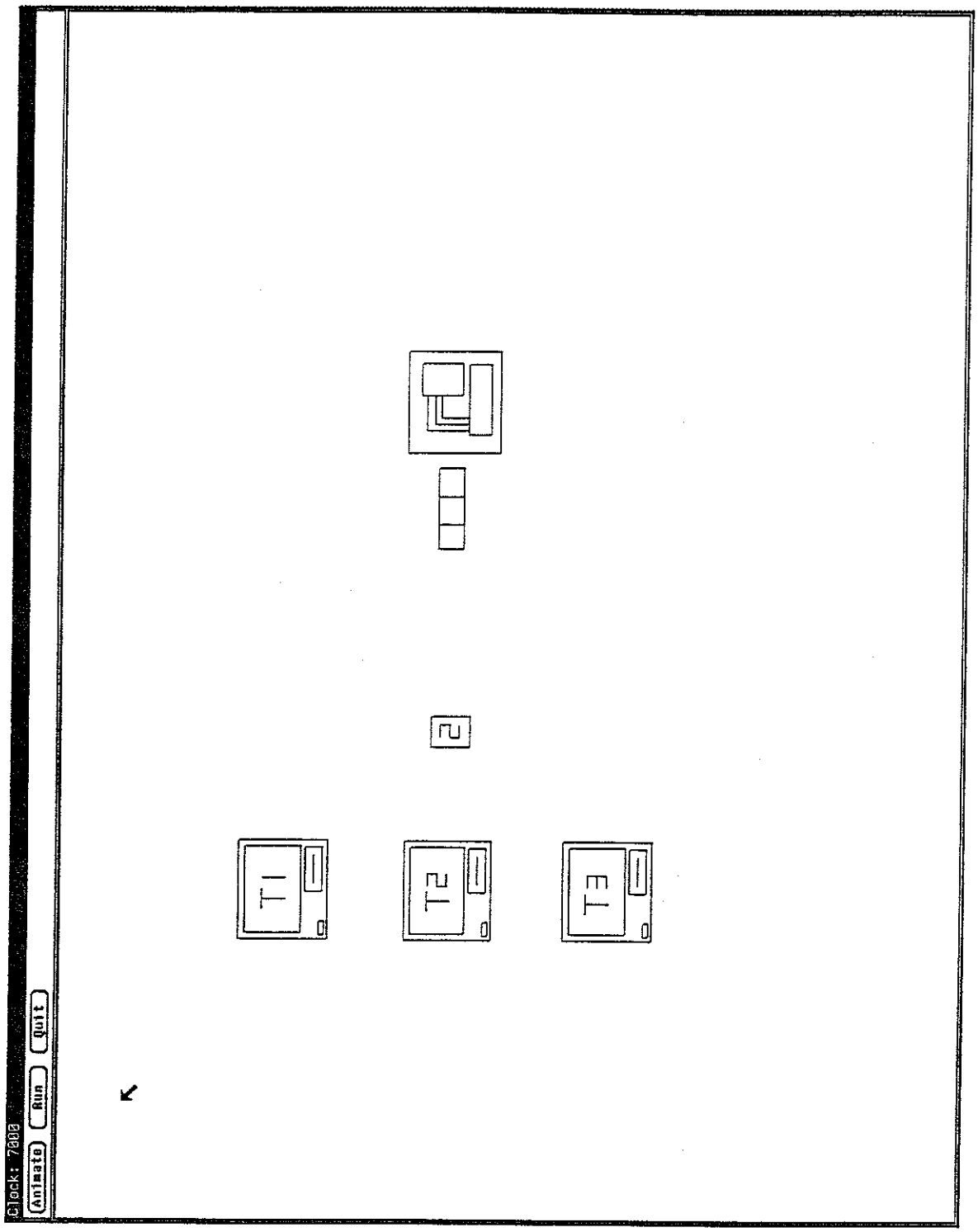

Figure 12. Visual Simulation Window

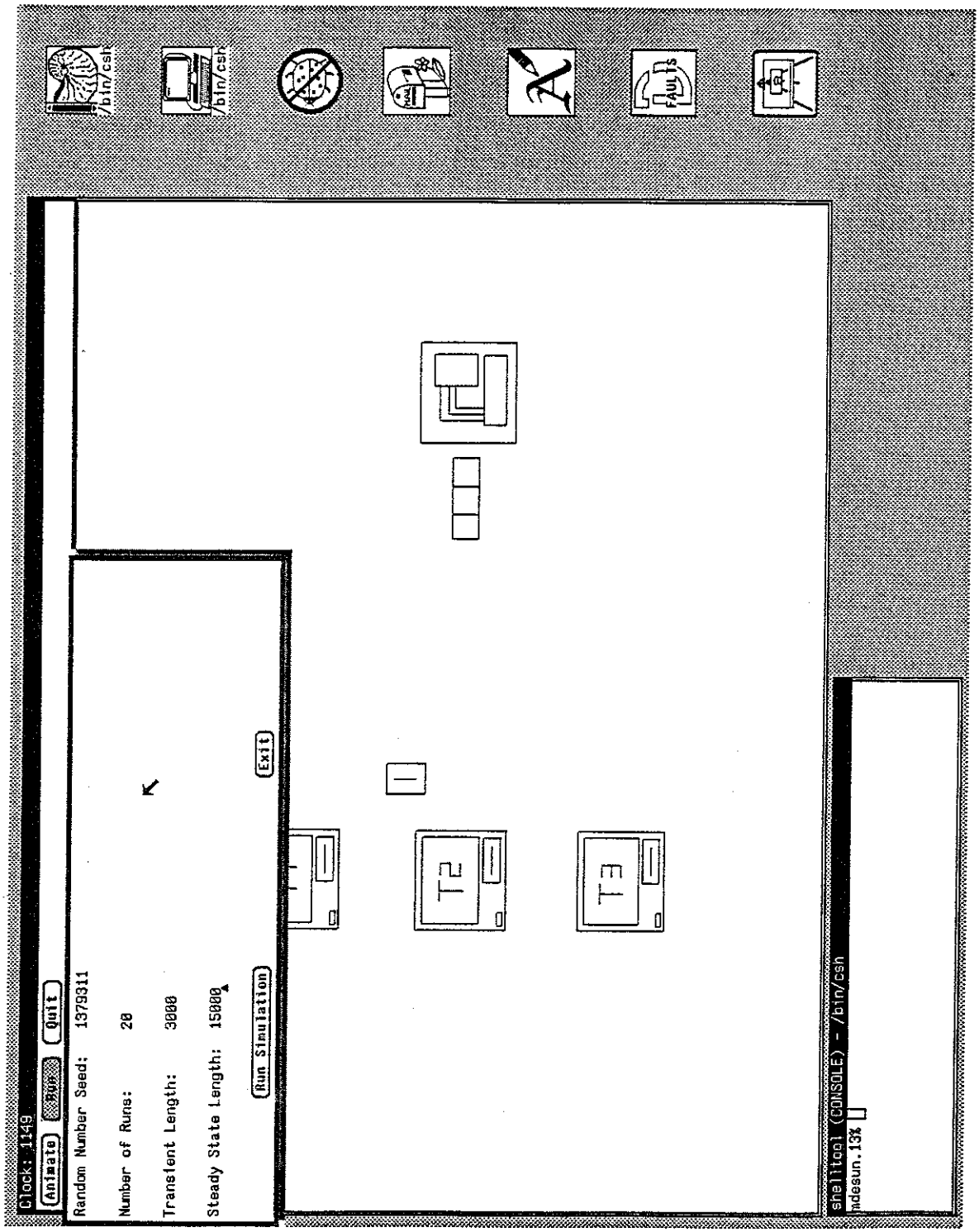

Figure 13. Running Simulation With No Visualization

5. CONCLUDING REMARKS

GPVSS is a major step in our Simulation Model Development Environments research project [Balci and Nance 1987a] in achieving the automation-based software paradigm [Balci and Nance 1987b]. With the development of GPVSS, it has been shown that a graphically described and interactively specified visual simulation model can be automatically translated into an executable C programming language code while maintaining domain independence. Our experience with GPVSS advocates the use of graphical design for developing (visual) simulation models.

ACKNOWLEDGMENT

This research was sponsored in part by the U.S. Navy under contract N60921-83-G-A165-B03 through the Systems Research Center at VPI&SU.

REFERENCES

Baeker, R.M. (1974), "Genesys Interactive Computer-Mediated Animation", In *Computer Animation*, J. Halas, Ed. Hastings House, New York, N.Y., pp. 97-115.

Balci, O. (1988), "The Implementation of Four Conceptual Frameworks for Simulation Modeling in High-Level Languages," In *Proceedings of the 1988 Winter Simulation Conference* (San Diego, Calif., Dec. 12-14). IEEE, Piscataway, N.J., pp. 287-295.

Balci, O. and R.E. Nance (1987a), "Simulation Model Development Environments: A Research Prototype," *Journal of the Operational Research Society 38*, 8 (Aug.), 753-763.

Balci, O. and R.E. Nance (1987b), "Simulation Support: Prototyping the Automation-Based Paradigm," In *Proceedings of the 1987 Winter Simulation Conference* (Atlanta, Ga., Dec. 14-16). IEEE, Piscataway, N.J., pp. 495-502.

Bell, P.C. (1985), "Visual Interactive Modeling as an Operations Research Technique," *Interfaces 15*, 4 (July-Aug.), 26-33.

Bell, P.C. and P.F. Kirkpatrick (1986), "Questionnaire on the Practice of Visual Interactive Modeling," School of Business Administration, The University of Western Ontario, London, Canada.

Bell, P.C. and R.M. O'Keefe (1987), "Visual Interactive Simulation – History, Recent Developments, and Major Issues," *Simulation 49*, 3 (Sept.), 109-115.

Gordon, R.F. and E.A. MacNair (1987), "A Visual Programming Approach to Manufacturing Modeling," In *Proceedings of the 1987 Winter Simulation Conference* (Atlanta, Ga., Dec. 14-16). IEEE, Piscataway, N.J., pp. 465-470.

Gordon, R.F., E.A. MacNair, K.J. Gordon, and J.F. Kurose (1988), "The RESearch Queuing package Modeling Environment (RESQME)," Research Report RC 13428 (#60092), Research Division, IBM, Yorktown Heights, N.Y., Jan.

Hurrion, R.D. (1980), "An Interactive Visual Simulation System for Industrial Management," *European Journal of Operational Research 5*, 2 (Aug.), 86-93.

Hurrion, R.D. (1986), "Visual Interactive Modeling," *European Journal of Operational Research 23*, 3 (Mar.), 281-287.

Hurrion, R.D. and R.J. Secker (1978), "Visual Interactive Simulation: An Aid to Decision Making," *Omega 6*, 5, 419-426.

Johnson, M.E. and J.P. Poorte (1988), "A Hierarchical Approach to Computer Animation in Simulation Modeling," *Simulation 50*, 2 (Jan.), 30-36.

Macintosh, J.B., R.W. Hawkins, and C.J. Shepard (1984), "Simulation on Microcomputers: The Development of a Visual Interactive Modelling Philosophy," In *Proceedings of the 1984 Winter Simulation Conference* (Dallas, Tex., Nov. 28-30). IEEE, Piscataway, N.J., pp. 531-537.

Mathewson, S.C. (1985), "Simulation Program Generators: Code and Animation on a P.C.," *Journal of the Operational Research Society 36*, 7 (July), 583-589.

O'Keefe, R.M. (1987), "What is Visual Interactive Simulation? (And is There a Methodology for Doing it Right?)," In *Proceedings of the 1987 Winter Simulation Conference* (Atlanta, Ga., Dec. 14-16). IEEE, Piscataway, N.J., pp. 461-464.

Palme, J. (1977), "Moving Pictures Show Simulation to User," *Simulation 29*, 6 (Dec.), 204-209.

Paul, R.J. (1989), "Visual Simulation: Seeing is Believing?" In *Impacts of Recent Computer Advances on Operations Research*, R. Sharda, B.L. Golden, E. Wasil, O. Balci, and W. Stewart, Eds. Elsevier Science Publishing, New York, N.Y., pp. 422-432.

Sauer, C.H., E.A. MacNair, and J.F. Kurose (1982), "The Research Queuing Package Version 2: Introduction and Examples," Research Report RA 138 (#41126), Research Division, IBM, Yorktown Heights, N.Y., Apr.

Standridge, C.R. (1986), "Animating Simulations Using TESS," *Computers and Industrial Engineering 10*, 2, 121-134.

Sun Microsystems (1986), SunINGRES, Volumes I, II, and III, Sun Microsystems, Inc., Mountain View, Calif., May.

Sun Microsystems (1988), SunView Programmer's Guide, Sun Microsystems, Inc., Mountain View, Calif., May.

White, R.A. and O. Balci (1989), "Traffic Intersection Visual Interactive Simulator (TRAVIS): A Functional Description," To appear.