# An Introductory of a Window-Based Environment for Simnon on the Sun Workstation

Frederick, Dean K.

1987

*Document Version:*
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7366)/1-029/(1987)

# An Introductory Study of a Window-Based Environment for Simnon on the Sun Workstation

Dean K. Frederick

Department of Automatic Control
Lund Institute of Technology
September 1987

**Department of Automatic Control**
**Lund Institute of Technology**
P.O. Box 118
S-221 00 Lund Sweden

*Document name*
Report

*Date of issue*
September 1987

*Document Number*
CODEN:LUTFD2/(TFRT-7366)/1-029/(1987)

*Author(s)*
Dean K. Frederick

*Supervisor*

*Sponsoring organisation*
The Swedish Board for Technical Development
STU-project 86-4049

*Title and subtitle*
An Introductory Study of a Window-Based Environment for Simnon on the Sun Workstation

*Abstract*

An initial version of Simnon has been created that runs on the Sun 3/50 workstation and makes use of the SunView window system to provide an enhanced user interface. Commands have been implemented for: creating a number of windows for plotting response curves, editing files, obtaining directories of Simnon files and editing them, displaying the values of variables and parameters, listing numerical values of variables versus time, and providing help on the window-related commands.

The general topic of using windows to obtain an enhanced user interface for Simnon is discussed as are specific issues that had to be resolved in the course of this project. A number of suggestions are included for extending this work so as to yield a more complete window-based version of Simnon.

*Key words*
Computer Aided Control Engineering; Man-machine interaction; User interface; Graphics; Windows; Simulation; Simnon; Sun workstation

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*

*ISBN*

*Language*
English

*Number of pages*
29

*Recipient's notes*

*Security classification*

# 1. Introduction

During the past several years the engineering workstation has come of age, both in terms of its computational and graphical capabilities, and its moderate price. Furthermore, the trend in that direction shows no sign of letting up. Examples of such machines are the Sun Microsystems Model 3/50 and the DEC microVax II, both of which have large high-resolution screens, powerful central processors, and floating-point hardware. The cost of these machines has gotten to the point where it is feasible to have a machine dedicated to a single engineer, at least in many situations. This situation raises the important question of "What has happened to the user-interface capabilities of the control-related software?". As of the moment, the answer is "Not a great deal".

The basic issue in this type of work is "How can the user of the computer be given better ways of specifying the problem to be solved, controlling the actual solution process, examining the results of the solution in graphical form, and repeating these steps in an iterative fashion while keeping track of all of the models and parameters being used and the large collection of results that will be generated?".

Many of the ideas and tools that will be used in addressing this question have their roots in the pioneering work done at Xerox's Palo Alto Research Center, which influenced the developments by the people at Apple, first with the Lisa, and then with the Macintosh.

The idea of using a mouse with multiple windows, menus, and icons to make a highly interactive and hopefully user-friendly environment is not at all new. The key point as far as control engineers are concerned is that to date these features have not been built into much of the control-related software. There has always been a substantial lag between the time new terminals and display devices become available and the time that a substantial fraction of the control engineers are using them on a routine basis. For example, the Tektronix 4010 storage tube terminal and the X-Y plotter were the dominant graphical output devices for many years, to be followed by the DEC VT100 with retrographics, which provided an emulation of the Tektronix storage-tube graphics. Although more versatile terminals such as the Tektronix 4025 are in use today, the style is still basically that of the Tektronix 4010. Fortunately, we have gotten away from the teletype days, although one does still see plots that have been 'drawn' in alphanumeric form on line printers. The work described below represents one beginning of the inevitable transition to a new plateau for the user interface. The necessary tools are here, both in terms of hardware and the all-important software.

## 1.1 Previous Efforts

There have been several applications of windows, mouse-selectable menus, and large-screen high-resolution graphics to date in the control system area. Probably the most advanced of these is the second-generation of System-Build by Integrated Systems that provides a model-building front end to their MATRIX$_X$ program (Shah et al., 1985). When run on a workstation such as the

Sun, this software allows the user great freedom and flexibility in generating interconnecting blocks of subsystems in order to produce the overall model for analysis and/or design. Pro-Matlab by The Mathworks (Little and Moler, 1987) makes limited use of windows by producing its graphical output in a window that is separate from that in which the program is running. However, the current implementation appears to support only a single graphical window, which means that when a second plot command is given, the plot in the existing window is replaced by the new one and thus is lost.

John Edmunds of the Control System Centre at UMIST has ported his comprehensive control analysis and design program CSS to run on the Sun workstation, in addition to a variety of less capable terminals. However, he has written his own windowing system rather than using that of Sun so as to gain portability. A new analysis package from Lawrence Livermore Laboratory, called Eagles/Control (Lawver and Poggio, 1985; Gavel et al., 1986), has been designed to run on a microVAX workstation, in addition to the usual VT100 with retrographics. Some work done by the author while at UMIST in 1985 and expanded since then at Rensselaer Polytechnic Institute allows a portion of the UMIST Control Design Suite (Munro and Bowland, 1984) to be run in a window environment with prespecified windows for specific purposes.

## 1.2 The Necessary Ingredients

In the course of reading this report it should become apparent that the process of converting a large conventional Fortran program to a window environment requires a variety of resources and talents. First, one needs the appropriate hardware, including a large high-resolution screen with roughly one million pixels, a powerful processor that can support multitasking, and a capability for doing floating-point arithmetic at a rate and precision suitable for demanding control-type calculations. Perhaps less obvious are the needs for software and an understanding of how to use the various software components effectively.

The key software ingredient that has not been encountered in the traditional Fortran-style programming is the window-related software and whatever programming language that the software developer must use in order to interface with it. In the case of the Sun we are talking about the SunView software (Sun, 1986b) and the C language. An additional software ingredient is the graphics language. In the case of the Sun, this may be the Graphical Kernel Standard (GKS) (Enderle et al. 1984; Hopgood et al., 1983) ACM SIGGRAPH Core System (Sun, 1986a), or ANSI Computer Graphics Interface (CGI). A possible alternative would be Programmer's Hierarchical Interactive Graphics System (PHIGS) (SIS 1985; Shuey et al., 1986; Brown, 1985), although it is not part of the Sun software at the moment.

The key point is that when one couples the very wide range of software associated with the windowing system and graphics software with a knowledge of the Unix operating system and an intimate knowledge of the application programs (that are most likely written in Fortran), one has an extremely broad-band software environment within which to work. The probability of finding a single person with the full range of required talents is not great. Hence, it is likely that a certain amount of teamwork will be required for the development of window-based software on a timely basis.

## 1.3 Outline of the Report

This report describes some initial efforts that have been taken to develop a modern work-station-based version of the simulation program Simnon that is designed to run on a Sun Microsystems 3/50 workstation. Because only two months have been devoted to the effort the achievements to date are only a fraction of what the potential for this type of environment is. However, the directions that must be taken to achieve a window-based version of Simnon and the potential benefits of doing this are quite clear at this point.

In the next chapter we will look at some of the ways in which windows can be used to advantage with Simnon. In Chapter 3 the accomplishments of the past two months will be described. Extensions to the work to date are described in Chapter 4. The report concludes with a summary of the activities to date and some thoughts on what the future might bring.

# 2. Simnon with Windows

In this chapter we will discuss ways in which the technology of the engineering workstation described in the previous chapter can be applied to the conventional Fortran simulation program Simnon (Åström, 1985; Elmqvist et al., 1986) Although the discussion will be related to this particular program it should be kept in mind that much of what will be said will apply equally well to a wide variety of the programs that are commonly used for computer-aided control system design (CACSD), both existing and yet to be written.

## 2.1 General Comments

In the chapters that follow we will get more specific and discuss the details of various commands and program features. One other comment to be kept in mind is that we are talking about user-interface features as they can be implemented on today's engineering workstations, but are paying no regard to the question of compatibility with the large majority of terminals in use today. While the issue of downward compatibility is undoubtedly important to many people, such as those who must pay the bills for the design tools, it is being excluded from present consideration so as not to inhibit our thinking. On the other hand, every effort will be made not to get carried away. For example, there will not be any discussion of color, as the author does not consider it essential to making a significant improvement in user-interface quality for the large percentage of CACSD applications.

## 2.2 The Major Parts of Simulation

In order to provide some structure to our discussion, let's consider the major components of a simulation project. We will start at the point where the requisite analysis has been done and the user has a set of equations, parameter values, and time functions that have been expressed in reasonably correct, but not perfect Simnon syntax in a collection of '`.T`' files. The process from this point on can be divided into the following four steps:

- model setup and debugging,
- simulation setup and debugging,
- production runs, and
- evaluation of results.

Figure 2.1 shows these major simulation tasks and the typical Simnon commands that might be used in a simulation project as a function of time. It is understood that these steps will be repeated in an iterative fashion as the work proceeds and the user will most likely accumulate a large number of plots corresponding to a variety of parameter values, and will probably end up with several different versions of the model. A key point that will guide our thinking is that the user will have a great deal of information to keep track of, presumably more than can be kept in the mind at one time. Hence, there

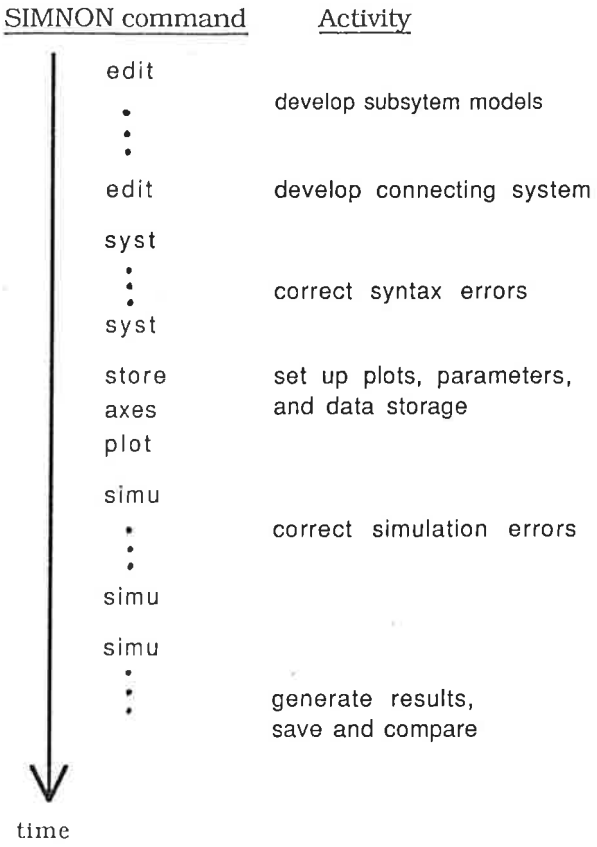


**Figure 2.1** Steps in a typical Simnon scenario

will be a constant need for a variety of information to be readily accessible, although generally not always visible. Because of the large quantity and variety of information to be accumulated, it will be important that one can readily:

- view several rather different pieces of current information, and
- have visual reminders of that information that is not currently visible.

The first of these requirements suggests that we need several windows active on our screen at any one time, perhaps with some partially obscured, but readily uncovered. The other item suggests the need for icons that will be visible without taking up much of the precious screen space, but whose image will remind the user of what specific information it represents, such as a plot for specific parameter values.

Next we will consider each of the three major simulation components in some detail, in terms of how the engineering workstation and its windows and multiprocessing capabilities can be put to use.

## 2.3 Model Setup and Debugging

Among the problems that the user faces in this phase of the simulation process are:

- keeping track of files and system names,
- responding to syntax errors, and
- editing files from within Simnon.

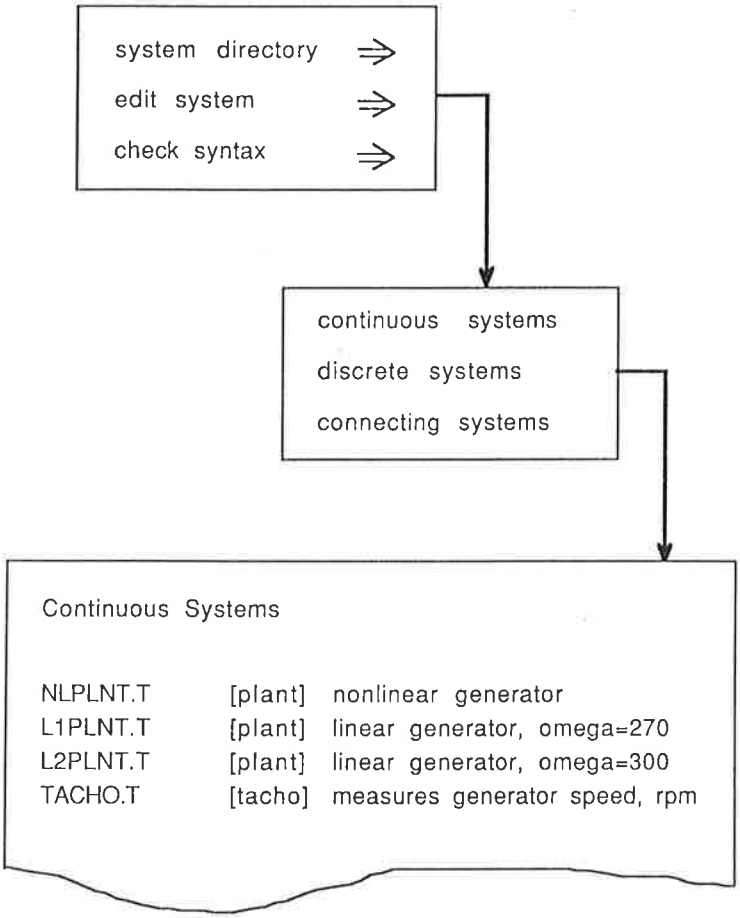


**Figure 2.2** Menus and a display for system files

Let's have a look at each of these for ways in which windows can be used.

### A subsystem directory window

A window can be created that will allow the user, via menus or some other selection tool, to obtain a listing of all '`.T`' files within the present directory, arranged by the type of system (continuous, discrete, or connecting ) and containing the file name, the system name, and a brief comment string that will remind the user about the details of the file. The macro files could also be included as a separate category. Figure 2.2 shows what the popup menus and part of one of the displays in the window might look like. According to the Sun conventions described in Appendix B of Sun (1986b), an arrow to the right of a menu entry indicates that there is a submenu corresponding to that item that will become visible if the cursor is dragged off the right edge of that menu item. As an alternative to the popup menus, one might use the SunView panel facility to create buttons on which the user can click for a selection.

### Responding to syntax errors

It is nice to be able to edit files from within Simnon and to have the editor invoked automatically when a syntax error is detected during the compilation of the code following a `SYST` command. However, the use of an editing window and the visual editor of the SunView system can provide significant improvements over the present implementation. It should be possible to have

an editing window pop up automatically with the proper Simnon file loaded and the offending line visible and highlighted. As an alternative to the Sun editor we could presumably set up its our version of Simnon to invoke our favorite editor.

A point that doesn't involve windows is that it should be possible to have a procedure for checking the syntax of the individual subsystem files prior to giving the SYST command. If there was a syntax checker to do this as the files are being created or after a specific file has been entered, many of the problems that get detected only after the SYST command has been given could be caught at an early stage. Nonexpert users could be helped greatly by having such a feature.

### Editing

General editing and that which must be done in response to error messages should be done in an editing window that can be left in its iconic state when not in use or which can be made to pop up when needed. This editing window might have a panel with some buttons for actions such as *load*, *save*, and *directory*. It could be used in conjunction with the subsystem directory window described above, or as an alternative the functions of the two windows could be combined into one. The key point is that the editing should be done visually, with directory information on the various .T files being available at the same time.

## 2.4 Simulation Setup and Debugging

Once the model files are set and the SYST command has been given there are a number of details regarding the simulation and the form of the graphical output that must be specified. For many of these defaults will be used, with or without the awareness of the user. We will now look at some ways in which windows can be used to assist the user in the setup process and to keep track of their values once they have been established.

### Algorithm selection and error bounds

By issuing the ALGOR or ERROR command there could be a window displayed that has a panel with buttons or a popup menu with algorithm selections. A help item could give a brief discussion about the characteristics of the various algorithms and some guidance in the selection, such as the advantages of using a fixed-step method for multivalued nonlinearities.

### Plotting setup

At present one sets up the plots by issuing a sequence of commands such as SPLIT, AXES, SHOW, ASHOW, and AREA and waits to see the results in graphical form. It should be possible to use the graphical capabilities of the Sun to provide a graphical means of setting up the plots, rather than through typed commands. Fig. 2.3 shows a possible layout for a window that will have four plots, with the variables and axes as shown. Where values are not given for the coordinate limits, it would be understood that the scaling should be done automatically by Simnon, i.e., the equivalent of the ASHOW command. Ideally, the data could be entered on the setup display by using the mouse to let the user specify the locations of the entries. To reduce the typing of known items

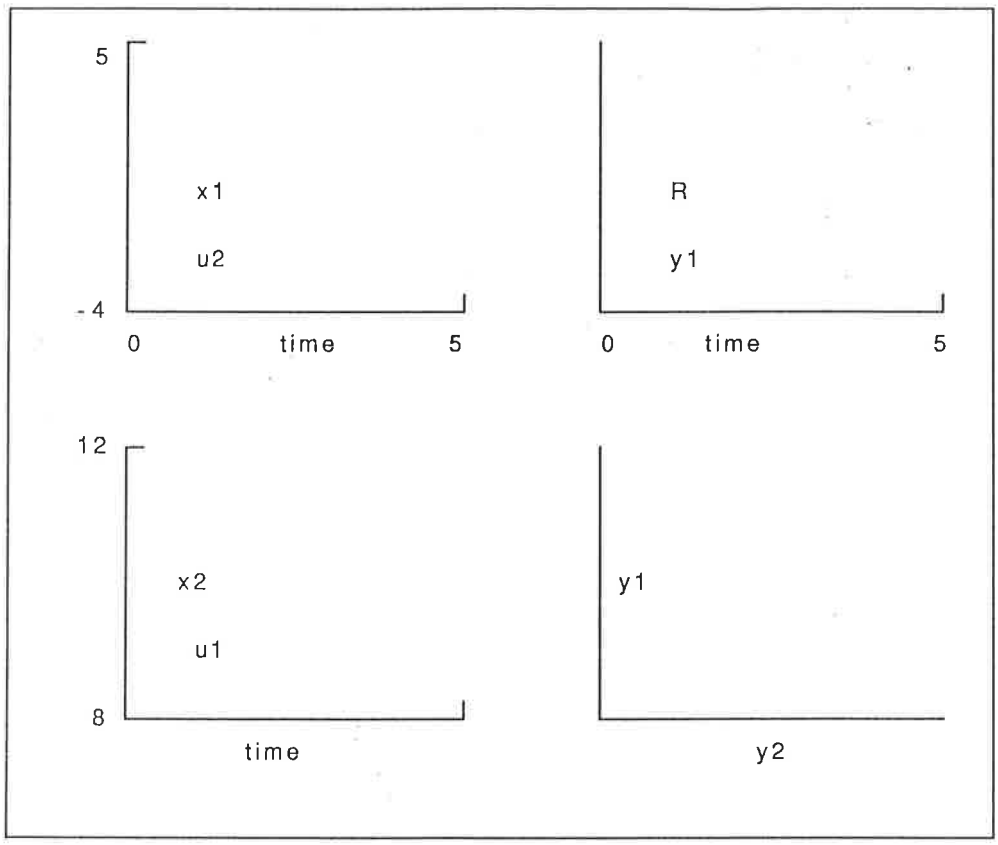


**Figure 2.3** Graphical plot layout

there could be a list of all of the variables that are available for plotting, and the user could use the mouse to select the variables for the individual axes by pointing and clicking with the mouse. If the names are all short as in this figure and there are only a few of them, such a feature would not save much time or typing. However, with a number of variables the list would serve as a reminder and would assure that typing errors were not introduced if the actual words were selected from the list and transferred by the computer.

### Status information

Lists are given in Fig. 2.4 of a number of the Simnon commands and of the information that is associated with them, either as specified by the user or as created by default. We see that there is a substantial amount of such information and it is likely that the user will have difficulty keeping track of all of it, and may even be unaware of some of it. It would be possible to have a status window in which the current values of such information are displayed and updated automatically. In normal operation the user might not wish to devote screen space to such a display so the window could be closed and represented by its icon. However, the information would be updated automatically so as to keep it current. Hence, it would take only a click of the mouse on the icon to transform it into the status display with the values assured to be current.

| Command | Information |
|---|---|
| ALGOR | algorithm type |
| ERROR | error bound |
| INIT | initial values |
| PAR | parameter values |
| PLOT | variable names |
| SAVE | file name & values |
| SIMU | start & stop times, time increment, and file name & storage increment |
| STORE | variable names |
| SYST | system file names |
| SWITCH | various settings |
| AXES, AREA, SPLIT, SHOW | plot characteristics |

**Figure 2.4** Information associated with Simnon commands

## 2.5 Output from Production Runs

Once the model has been prepared and the simulation and plotting parameters have been established, our user is ready to begin generating numerical and graphical data at a high rate. At this point the workstation and its windows can be invaluable in organizing the data for display as generated and for retrieval at a later time. Whereas the use of conventional programs and terminals requires that the user make hard copies or put the results into a file, we should be able to draw a number of plots and have them saved for later recall in graphics windows. There must obviously be some limits on the numbers of windows that can be created and that can be active at one time. However, we now have the necessary ability to save results, particularly graphical results, in a form that can be recalled in an instant.

Also it should be relatively straightforward to allow for graphical processing of the plots in their display windows by doing such things as changing line types, scaling, and adding labels and text. Part of the text added to a figure could be used in the icon of the window so as to give the icon a distinctive name.

## 2.6 Generation of Connecting Systems

A somewhat more advanced area in which the workstation's capabilities could possibly be used is in the development of the connecting system file. The usual way that this task is done is to create the appropriate text file with an editor, probably working from a block diagram of the system that shows the desired interconnections.

An alternative approach would be to have the user create the files for the continuous and discrete systems and then issue a variation of the `SYST` command. This command would allow the computer to generate lists of all of the block inputs, outputs, and the input signals to the system, in a form such as that shown in Fig. 2.5. With the mouse the user would select the

| Outputs | Inputs |
|---|---|
| [plant] y1<br>y2 | [plant] u1<br>u2 |
| [snsra] y | [snsrb] u |
| [snsrb] y | [snsrb] u |
| [ctrlr] y1<br>y2 | [ctrlr] u1<br>u2 |
| ref1<br>ref2<br>dist1<br>dist2 | out1<br>out2 |

**Figure 2.5** Input/output lists for generating the interconnecting system file

desired interconnections, thereby obtaining a text file that would serve as a template of the final connecting-system file. The next task would be to complete the work with the text editor, inserting such items as gains, signs, summers, and specifying the input functions. One might also have a library of input functions, represented by buttons or appearing on a menu, that would generate the appropriate code in the file. Such a facility could be considered as a step between straight text editing and a full block-diagram-based graphical editor.

## 2.7 Accessing Other Programs

When reviewing the results of a simulation run one often would like to perform some calculations on the variables as they exist at that point in time. The conventional way to do this is to use a calculator, keying in the particular variables, perhaps making some mistakes along the way. A better way, that becomes possible with the multiprocessing capability of the workstation, is to start up a new window with a program such as Matlab running in it which has direct access to the variables and parameters in the Simnon data base. This means that one could just type in the desired expressions to be evaluated, without having to transfer the numerical values of the variables. Also the expressions entered could be saved and reused with data from later runs.

## 2.8 SunView Conventions

As we have mentioned before, there are several different ways to get the same result, as far as the user interface goes. For example, the SunView software has a panel subwindow that allows for a variety of formats for the entry of information. These include buttons, choices, and sliders. Also everyone is familiar with the popup menu that is used extensively for control of the window system by the user. Such menus and submenus can be constructed for use with the application program and represent an alternative to the panel approach.

If one wanted to get fancy and incur the necessary overhead, it should be possible to allow the user to specify a panel or a menu version of the window tools in a setup file. One advantage of buttons is that the user sees the possible

choices on the screen at all times. Alternatively, with popup menus valuable screen space is not taken up by the menu until it is requested by clicking with the mouse. It might be said that the panels approach favors the less experienced user while popup menus are more suitable for experienced users.

It is useful to note that there is material in Appendix B of Sun (1986b) that describes what the people at Sun Microsystems consider to be good and consistent usage. The creators of the SunView software have used these conventions in their work and it would appear to be a good idea to stay very close to them, at least until considerable experience has been gained with windows. By using some of the tools that have been provided, such as the window-based debugger *dbxtool* and the icon editor, and by running the sample programs that are given in the SunView manual, one can rapidly get a feel for what is certainly an acceptable style for the user-computer interaction.

# 3. Progress to Date

In the first two chapters we have discussed the general characteristics of the engineering workstation and windows that make the combination such a powerful tool for analysis and design. Also we have described in general terms some ways that this tool can be applied to Simnon, and other CACSD packages. Now we will review the progress to date with Simnon. In order that this report may be of more value to those who are embarking on such developments in the near future we will also mention some of those things that didn't work, at least at the time they were tried. In reading this material one should keep in mind that to date the experience base with these tools is still small and consequently some of the details necessary for successful implementations are not well understood at this time.

## 3.1 Window-Related Commands

Commands have been added to Simnon that automatically generate windows that have been tailored to the specific objective of the command. The commands that have been added as of this date and the types of windows that they produce are:

- **mkwin** a graphics window for plotting
- **wedit** an editing window
- **wfdir** a directory facility for the .T files, with editing
- **wdisp** a window for the Simnon DISP command
- **wprint** a window for the PRINT command
- **whelp** help on the window-related commands

The operation of each of these commands will be described briefly below. In each case the command is issued from the keyboard, when Simnon is displaying its prompt. When the Sun version of Simnon begins it will not prompt the user for the type of terminal. The assumption is made that the code is running on a Sun workstation with the high-resolution monochromatic display.

**mkwin**

A window such as that shown in Fig. 3.1 will be produced that contains a canvas for drawing graphical output from Simnon and a set of buttons. If a Simnon command that generates graphical output is given before any **mkwin** command has been given, Simnon will automatically perform a **mkwin** operation to create a plot window. If a subsequent **mkwin** command is given a separate window will be created with an identical canvas for drawing response curves. The first window can be left active on the screen, closed to become an icon, or deleted. Based on empirical evidence there is a limit of about five such windows at present. The exact cause of this limit and its precise value are not known at this time, but it is suspected that certain resources are exhausted at some point as the number of graphical windows increases, thereby resulting in the crashing of Simnon. All of the usual Simnon graphics-related commands, e.g., SPLIT and AREA, can be used with their usual arguments.

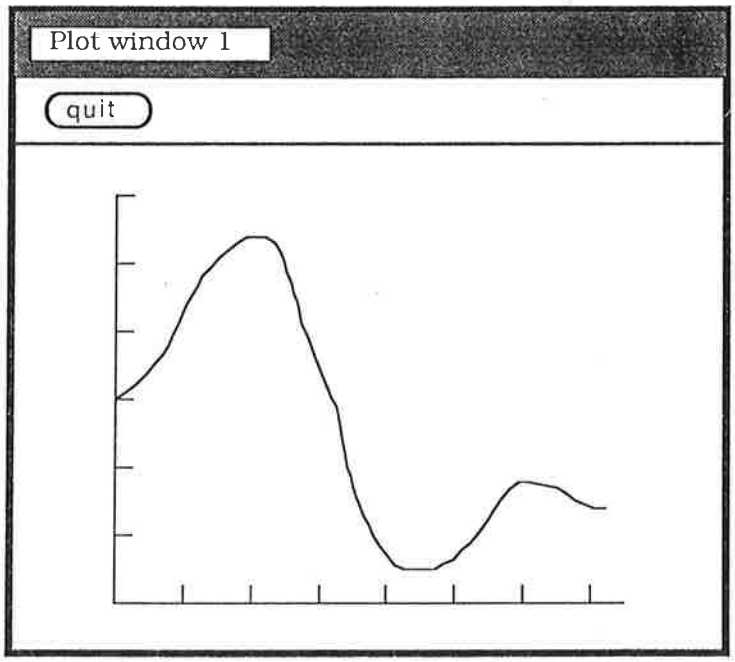


**Figure 3.1** Plot window

`wedit`

A window, as shown in Fig. 3.2, is produced for editing `.T` files by issuing the `wedit` command. It consists of a small subwindow from which Unix shell commands can be issued (referred to as a *tty* subwindow by SunView), a panel with buttons for loading and saving files, for doing a directory of the `.T` files, and for quitting the window. There will be only one such editing window. Should the user issue the command while the allowed one is in existence, an appropriate message will be issued. At present syntax errors found during compilation following a `SYST` command do not result in the editing window being activated or loaded with the offending file. It is expected that this feature will be added in the future.

`wfdir`

The window created by this command contains several buttons for generating special directories of subsets of the Simnon `.T` files, as depicted in Fig. 3.3. Specifically, one gets information similar to that described in Fig. 2.2 for the following categories:

- all `.T` files,
- continuous systems,
- discrete systems,
- connecting systems, and
- macros

There is also an editing button and clicking on it will cause a popup edit window to appear within which a selected file can be edited. Also this window can be used in conjunction with the editing window created by the `wedit` command as a means of specifying the file to be edited. To do this the `wfdir` command is given if the window has not yet been created and the user clicks

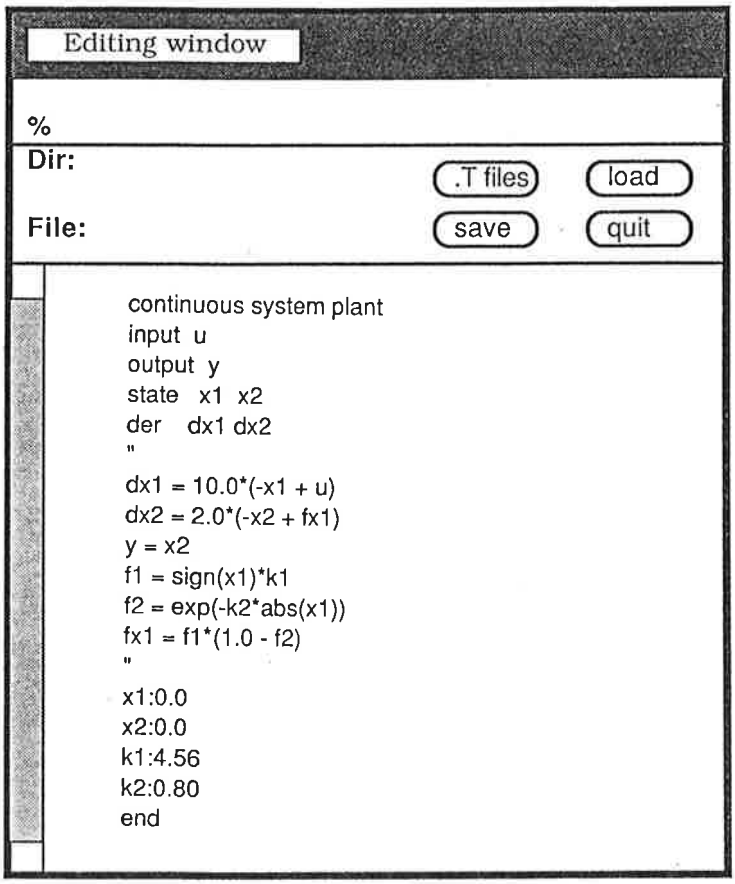


**Figure 3.2** Editing window

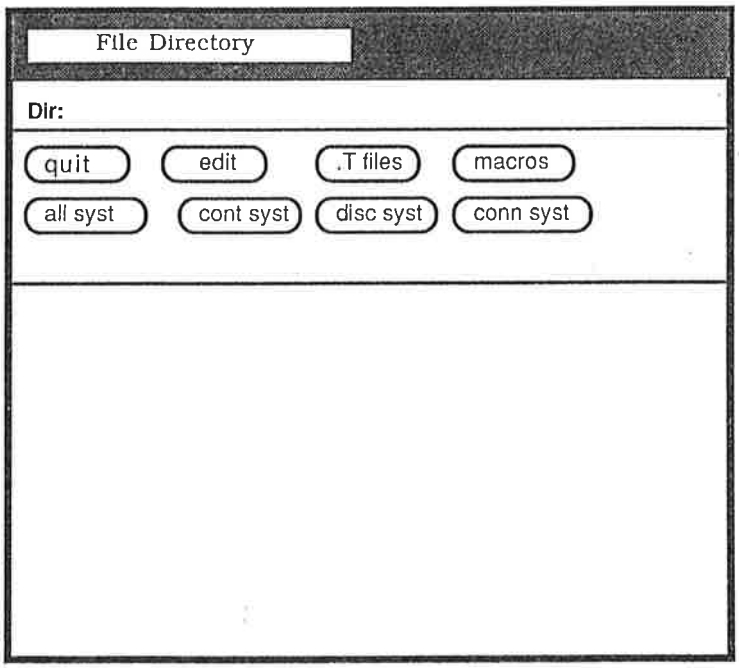


**Figure 3.3** File directory window

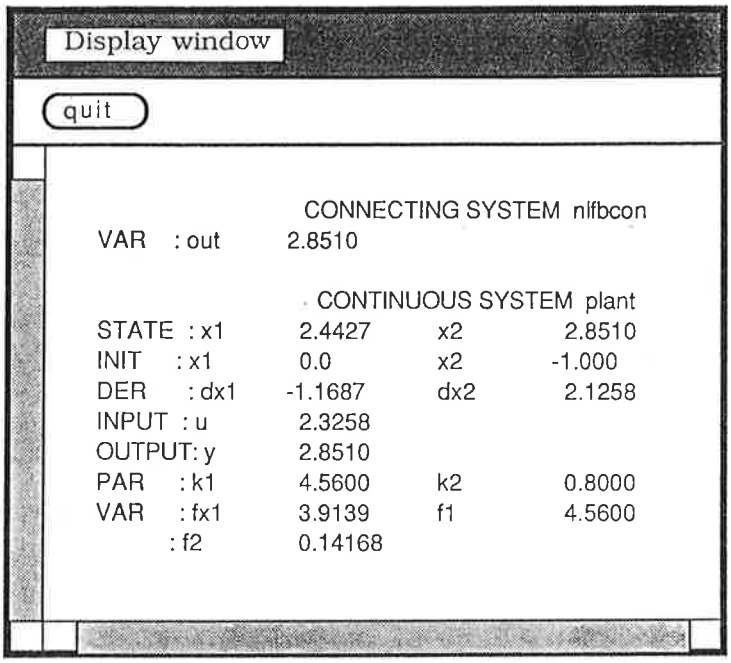


**Figure 3.4** Display window

on the button for the type of `.T` file to be displayed. Then the selection is made with the mouse and the 'put' function key (L6) is pushed, the mouse is clicked on the space to the right of the word 'File:' in the panel of the edit window, and the 'get' function key (L8) is depressed. At this point the file name should be displayed in the 'File:' slot of the edit window control panel. Finally the user should click on it to select it and then click on the 'load' button. The desired file should appear in the editing window where it can be modified using the Sun mouse-based editor. At present we are not able to distinguish between the various types of `.T` files, but the addition of such a capability is a relatively straightforward matter, not involving the windowing software.

`wdisp`

A display window is created, if one does not already exist, and the text that is produced by the Simnon `DISP` command will appear in this window. If the command `wdisp` is given a second time the information will be updated but the existing display window will be used. The window will have both vertical and horizontal scroll bars to allow the user to view large displays without the usual problem of having information roll off the viewing area and not be retrievable. An example of such a window is shown in Fig. 3.4.

`wprint`

This window is similar to the display window described above, except that the data presented are the results of the Simnon `PRINT` command, namely, the numerical values of variables for a range of time points.

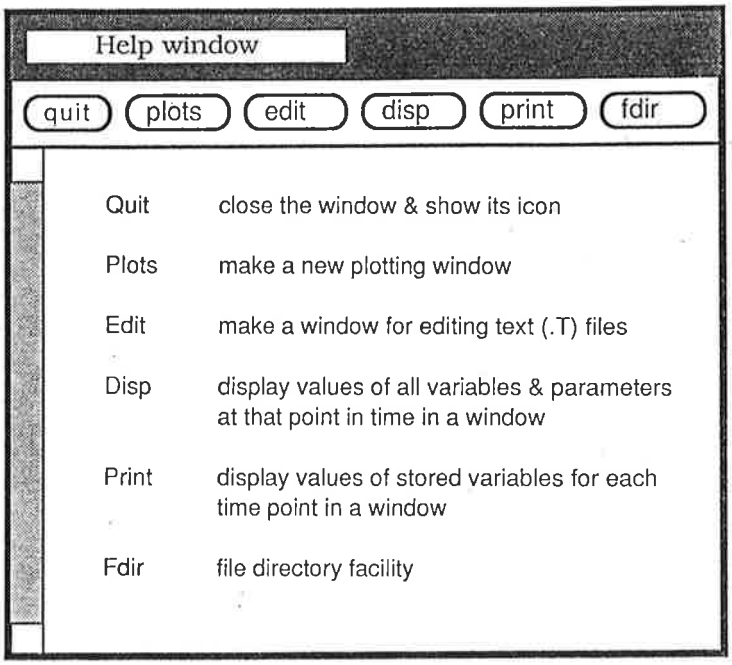


**Figure 3.5** Help window

`whelp`

A window is created and a message that provides general help on the window-related commands is displayed. The user can obtain more detailed help on specific commands by clicking on the appropriate button, as shown in Fig. 3.5. In Fig. 3.6 we see how the workstation screen might appear during a simulation study. The main window contains Simnon commands that have been typed by the user. To the right we have a plotting window showing response curves. Across the top of the screen there are a number of icons that represent windows that have been closed. From left to right, these windows are: plot windows 1 and 2, display, print, file directory, edit and help. Any of these windows can be activated by clicking on the appropriate icon with the mouse.

## 3.2 Drawing with SunCore

As was mentioned earlier, there are three graphics languages that Sun supports at the moment. These are:

- ANSI Computer Graphics Interface (CGI),
- ACM SIGGRAPH Core System (SunCore), and
- Graphics Kernel Standard (GKS).

The GKS software was not available to the author at the time this work was being done so it was not considered for use. Between the other two candidates, it was decided to use the SunCore language as described in Sun (1986a) because it appears to be more versatile, although presumably at the expense of more overhead. No extensive comparisons were made in arriving at this decision and at this point it would be unfair to say that the CGI system is inappropriate. The surface on which the drawing is done is a canvas subwindow and the SunCore system allows the programmer to work in a world coordinate system of his own choosing and to specify a viewport in terms of

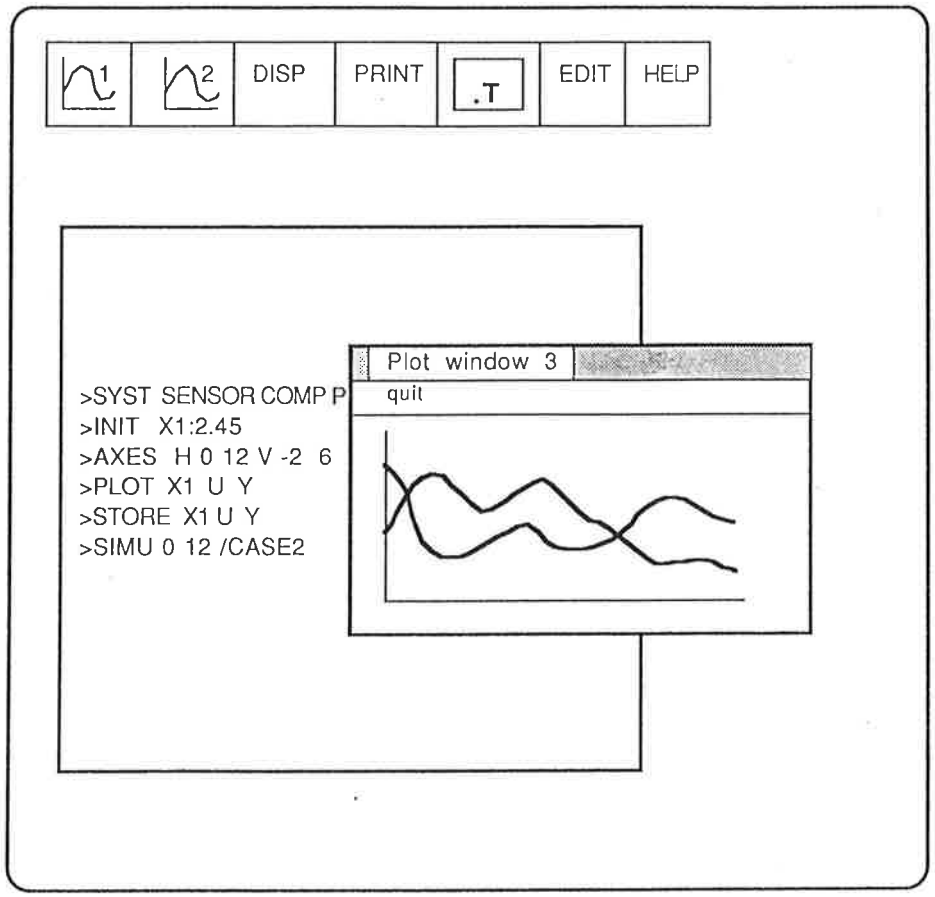


**Figure 3.6** Typical display with an active plot window

device-independent units. The world coordinate system was taken to be that of a Tektronix display, namely 4096 by 3120 units, since the Simnon graphics is set up to work with these. The viewport coordinates were taken as 1.00 by 0.75 so as to conform to the aspect ratio of the workstation display.

The Simnon code for doing the drawing was prepared by Tomas Schönthal by modifying an existing but unused graphics driver. The modifications to the code consisted of:

- creating and selecting a view surface,
- opening and closing retained segments, and
- using the SunCore commands for drawing and moving

It is also necessary to initialize the Core graphics when Simnon is being started and to terminate it when Simnon is being stopped.

There were two substantial obstacles to making all of this work. The first was to get the figures to appear on the intended window rather than on a window that was placed directly over the one in which Simnon was running. The second obstacle was avoiding having Simnon lose control and cease to execute while the graphics window was in existence. A related problem is that when multiple graphics windows exist at the same time, some coordination is required to make the plot appear in the desired window, rather than on some other one or perhaps on all of the existing graphics windows. Because of the importance of these topics to the task of porting existing code to a window environment they will be discussed in the following two sections.

## 3.3 Drawing on the Proper View Surface

Before drawing can be done with the SunCore commands there must be a view surface that has been both initialized and selected. There is a procedure called `get_view_surface` that exists in the SunCore library (Sun, 1986a). However, this procedure is set up with default values that make it grab the window in which Simnon is running and cover it with a view surface, thereby wiping out any further communication with Simnon until that view surface has been destroyed. With assistance from Gavin Bowe of UMIST and Dag Brück, this routine was modified so that it performs correctly for the intended application. Specifically, it selects the proper canvas subwindow, which is owned by the frame of the graphics window being created by the `mkwin` command, as the view surface for the subsequent drawing.

The problem of managing multiple view surfaces is handled by writing code that can keep track of which view surfaces have been created, what their 'handles' are (this is the way the window manager software refers to them), and ensuring that only the desired one is selected when the drawing is being done, with the others being in a deselected state. At the moment this code is being developed by Tomas Schönthal.

## 3.4 Keeping Simnon and the Windows Functioning

If windows are to be used with Simnon in an interactive manner, it is obvious that the window manager must look after the windows while Simnon is running and Simnon must be able to perform its calculations and respond to user commands. The windows are serviced by a notifier program that must process any accumulated inputs to a window at least four times per second (per Section 16.6 of Sun (1986b)) if the user is not to perceive delays in the response to his clicks on the mouse or movement of the cursor. On the other hand Simnon must be able to continue to do its calculations and look for and process input from the keyboard while the windows are being processed. This objective is achieved in one of two ways, depending on whether the code for the window in question is a procedure that is linked into Simnon or a separate executable program.

In the former case, which has been used for the `mkwin` command, we call the SunView C procedure `notify_do_dispatch()` that will call the window notifier program periodically provided that the program (Simnon) is doing some kind of read operation, such as looking for input while its prompt is being displayed or reading data from a file. This approach covers almost all of the Simnon commands except `ASHOW` and `SIMU`, because the execution time required is sufficiently short that the user is not likely to notice any loss of control of the windows. For the `ASHOW` command, there is no problem either because Simnon is reading data from a file while it is doing the drawing, thereby assuring that any requests for window activity will be handled without delay. Because the `SIMU` command does not involve any reading activity and can monopolize the cpu for long periods of time, it is necessary to use a different approach. There is another SunView procedure called `notify_dispatch()` which activates the window notifier routine even if no input is being done. A test was run where this routine was called from within a Fortran subroutine that is called by Simnon each time that it completes an integration step, and it did keep the windows active. This code has not been used on a regular basis

yet because the extent to which its use would load the cpu was not known. However, it clearly represents a solution to the problem and just requires some care in its implementation.

When the window is being run as a separate program we use a different approach. Here the SunView procedure `window_main_loop()` is invoked after the window has been created. This action then causes the notifier program to respond to inputs to that window on a continuous basis. Because this is being done in a program that is totally separate from Simnon there is no interference between the two, and each can go about its own business.

# 4. Possible Extensions

As is usually the case with projects of short duration, one would like to have more time to carry things further. With a bit more work the six types of windows that have been created so far could be expanded to add some desirable features and to make them operate in a more friendly manner than they do at present. Also the code could certainly be improved to yield greater robustness and better diagnostics. Additionally, there are some more substantial steps that can be taken to yield a better product. Some of these steps would require a nontrivial commitment of resources and time so it is not obvious that they should be pursued. In any event, extensions of both types will be described below.

## 4.1 Other Windows

In Chapter 2 we described a number of ways in which windows could be used to achieve improved user interaction with Simnon. Beginning versions of several of these have been made. Types of windows that might prove very useful but which have not been included at this point are:

- status display, and
- connecting system construction.

We will comment on both of these.

### status display

A window that performs this type of function has been incorporated in the developmental window-based pole-placement program that the author (while on leave at UMIST) and Torin Shepard (a student at Rensselaer Polytechnic) have created. A similar approach should work here. The status information, which has the characteristic that it does not change as rapidly as the variables in the simulation, is written to a file whenever there is a change in any of its elements. Then the window code is signaled about the change by sending an escape sequence that is received and decoded by a special C procedure. When this occurs, the window code knows that it should open the updated file and display its contents in the status window. At this point in time it is not clear whether or not the window code should be a separate program or a procedure linked into Simnon, so both possibilities will be mentioned.

In the former case there is the problem of having Simnon tell the window program that it should open the file and display the new contents. It might be possible to send an escape sequence to do this. Alternatively, the window program could check the time stamp of the specific file periodically, say once per second, and detect when there has been a change.

In the second case, where the window code is linked into Simnon, we encounter unexplained crashes that have been observed when a Fortran program drives a window containing a text subwindow. If this problem can be resolved in the future the communication task should be relatively straightforward, presumably using an escape sequence.

**Connecting-system construction**

The development of such a window, as described in Section 2.6, would be a fairly ambitious task, as it involves getting information from the Simnon data base, as it would exist after a SYST command had been given, and generating output in a nonstandard form. Because the SYST command requires the presence of a connecting system before it can be executed, it would take some nontrivial modifications to Simnon in order for it to be able to provide the required information about the inputs and outputs of the individual subsystems prior to the SYST command. Likewise, once a display such as Fig. 2.5 has been generated, it would be necessary to be able to select specific inputs and outputs and to have their names inserted into the appropriate strings so the result will be a valid Simnon interconnection statement. One would also have the task of specifying exactly how the whole process would work from the user's point of view, including the way in which external inputs would be specified, and how summing junctions and other details would be expected to behave.

## 4.2 Expanded Capabilities for the Present Windows

Because the present version of the window-based Simnon is at a very early stage of development, there are a number of ways in which the windows that have been incorporated can be improved. In general terms, the panel buttons that have been used for the most part can be replaced or supplemented by other mechanisms that are part of the SunView panel tools (see Chapter 9 of Sun (1986b)). Alternatively, one may use popup menus. It has been conjectured earlier that users may have preferences regarding these matters and there is the factor of utilization of the screen space. In any event, the popup menu is a key part of the Sun window system and it seems appropriate that it should be employed with window-based Simnon. One consideration is that the greater the flexibility the user has, the greater the overhead and consumption of resources is likely to be.

Another feature that has been used in the file directory window but not in any others is the popup editing window. Here the editing window is not seen until the user indicates by mouse clicks that a specific file is to be edited. When this happens the subwindow in which the editing is to be done pops up automatically, with the specified file displayed. When the editing operations have been completed and the file saved, the user can click on the frame of the editing subwindow and select the item *done*, which will cause the subwindow to disappear, leaving just the original window. Obviously this feature can be used to save the valuable screen area until it is needed for that specific task.

## 4.3 Icons with Variable Names

An icon represents a window that has been closed, providing the user with a convenient reminder of the window's existence and an easy way for the window to be brought back to activity by clicking on the icon with the left button. Where there may be multiple versions of a particular type of window it will be important to incorporate a feature of the icon that will distinguish them. This is possible, but some care must be given to the decision as to what the distinction should be.

The most likely area where this problem will arise is the icon for graphics windows containing plots. If a string of characters is used, such as the values of one or two parameters, there will be a problem with space for the characters. If sequential numbers are used there is likely to be a problem with lack of recognition after a few windows have been converted to icons. To alleviate this difficulty one could have a window that provides a directory for the graphics windows, containing their sequential number for identification of the icon and a string of modest length that identifies the parameter values to which it corresponds. In fact, the string could be taken from the text that would have been entered by the user as part of the title or a subtitle. This directory window would have its own icon so it would not have to be visible at all times.

## 4.4 Extended Graphical Capabilities

One can imagine a wide range of graphical capabilities that could be added to those windows in which plots are produced. These would be capabilities that would allow the user to turn them into more finished products or to modify them for subsequent use. Typical types of modification would be:

- changing line types for selected curves,
- labeling the curves in an unstructured way,
- changing the scaling and having the plot redrawn,
- removing a particular curve from an existing plot, and
- transferring a curve from one plot to another.

The whole question of being able to perform interactive drawing is a matter of some complexity and one that has application well beyond Simnon. One characteristic to be kept in mind is the benefit to the user of being able to see the effects of the changes in visual form, i.e., "what you see is what you get". It would be helpful if one could perform some simple types of interactive drawing in order to prepare response plots for reports or visual presentations.

## 4.5 A Notifier-Based Simnon

It was mentioned earlier that the SunView software is configured in a notifier-based form, as described in Chapters 2 and 16 of Sun (1986b). Briefly, this means that the window software that the programmer develops consists of a number of independent components, each of which has been registered with the central notifier. When a particular component, such as a window, requires servicing because of a mouse click on its border the notifier becomes aware of this need and informs the service routine that will take care of it. In this manner a variety of clients can be handled in an efficient way, with the attention being devoted to those which need it. Likewise, clients that do not require attention at the moment do not impose any load on the notifier.

If one examines the major components of a simulation run, as we did in Section 2.2, it becomes apparent that Simnon can be thought of in terms of the following four major components:

- model setup,
- simulation run setup,
- calculation of the responses, and

- displaying the results and preparation of final copies.

Each of these components is only loosely coupled to the others, with information being transferred between them via certain memory locations or files. Although the first three must be done in the sequence shown on the first time through a simulation, it is logically possible to have two or more components being done simultaneously after that point. Also, there is no reason that two or more cannot be going on simultaneously, provided that the user can keep track of things. For example, if a lengthy simulation is in progress, the user could be examining plots from a previous run and preparing for the next run by making some modifications on one of the model files.

Given this generic decoupling of the simulation subtasks, it makes sense to consider the possibility of decoupling the Simnon code into several separate programs. One such decomposition would involve four programs, each of which performs one of the categories listed above, and a central controller that receives requests for activities from the user or one of the software components and parcels out the required tasks to the appropriate component. In addition, this controller software would oversee the flow of information between the components and the user. In a sense, the controller would look like a small operating system, coordinating the activities of the four components, monitoring the flow of information, and managing the available resources.

## 4.6 Modification of Simnon's Command Syntax

To date a total of six commands have been added to the Sun version of Simnon to support the windowing features. I seems likely that others will be added as developments progress. However, it also seems likely that once more experience is gained with the window environment that one might want to create a version of Simnon that gets most of its input from sources other than the keyboard, such as the mouse. One might make an analogy by comparing the mouse-driven editor in Sun windows with a traditional line editor where all input is via the keyboard. At this point in time it is difficult to predict how things will develop over the next couple of years, but one should certainly keep an open mind. It would be unfortunate to lose some of the power of this new environment because of thinking that is still entrenched in the keyboard-entry days.

Another consideration in the question of command-entry style is the importance of retaining the macro capability of Simnon. Under no circumstances should this feature be impaired.

## 4.7 Efficiency, Robustness and Diagnostics

The motherhood-type attributes of efficiency, robustness and diagnostics are certainly desirable in any program and considerable attention should be given to them before one considers a programming effort to be completed. Given the exploratory nature of the work reported on here, it is safe to say that major improvements can and should be made to the window-related software that is being developed for Simnon. Because there are two major software components operating in parallel, namely Simnon and the Sun windowing system, it is particularly important that they not get in each others way and that the combined package not become such a heavy consumer of resources

that response time suffers unduly. It should be kept in mind that there is always a price to be paid for the windowing features that are added.

One way to improve efficiency is to make use of spare CPU time by doing some of the tasks in background mode, perhaps at a lower priority than the main task. For example, a considerable amount of CPU activity is required to create a window before it can be displayed on the screen and used. The delay caused by the window creation process will be a function of the complexity of the window, but will typically be about four to five seconds. It seems likely that the user can be kept unaware of much of this delay by having the windows created as a background process with a low priority that is done when Simnon is running but the CPU is not particularly busy, such as when the Simnon prompt is being displayed but the user has not completed a command entry that requires processing. A number of windows that are likely to be wanted by the user can be created and placed in a closed condition, represented by the icon. Then, when the user either clicks on the icon or gives the appropriate command to Simnon, the window can appear with very little delay, ready to do its work.

It is essential that the code be prepared in a way that the user cannot easily cause the window system to get out of synchronization with Simnon, or to cause the program to crash. The window software is very complex and makes use of many of the operating-system signals. As stated in Section 16.2 of Sun (1986b), the programmer must exercise great care in dealing with any of these signals. Also, it has been observed that a variety of situations can occur that will cause Simnon to crash. Care should be taken to minimize such happenings because users will prefer to use the conventional form of Simnon if the operation of the window version is too unreliable, regardless of the potential benefits of the window-based version.

# 5. Conclusion

In this concluding chapter we will take a look back at what has been accomplished to date, and speculate a bit about what the future might bring.

## 5.1 Summary

Over the past two months a beginning has been made in the task of creating a version of Simnon that can provide the user with a substantially improved user interface by employing the hardware of a Sun Model 3/50 workstation and the accompanying SunView software tools. Although only a modest windowing capability has been created for Simnon to date, it is felt that a number of the major problems to be encountered in applying this new environment have been identified and overcome. These problems include:

- creation and selection of view surfaces for plotting responses,
- generation of windows both from Simnon and from separate programs, and
- timing and notification problems to allow Simnon and the window system to carry out their jobs in harmony.

In addition to overcoming the problems listed above, a total of six different types of windows have been incorporated in Simnon, with enough features so as to make them useful, although certainly not the last word.

## 5.2 Future Work

The existing windows can certainly be refined so as to become more useful to the user and different means of actuation can be explored, such as popup menus, buttons, and popup windows for editing.

A number of ideas have been presented for extending the existing windows that would require varying degrees of software development. Also, the techniques developed here can certainly be applied to other software packages, such as Idpac (Wieslander, 1980).

Two areas in which greater understanding is required are the causes of the unexpected crashes that occur, especially when a text subwindow is being driven from Fortran code, and the ability to communicate between Simnon and the windows. At present our understanding of the operating-system signals that are involved in the window system is minimal and it is felt that they are at the heart of the crashes.

Suggestions have been made for changes of a more fundamental nature that may become desirable, or even essential, as the usage and complexity of the windowing system expands. These include the conversion of Simnon to a notifier-based collection of programs. Additional ways in which the window-based version of Simnon could be expanded include the incorporation of external programs such as Matlab so the user can perform ancillary calculations directly upon the Simnon data base and pass the results back to Simnon or on to other targets.

# Acknowledgements

The work described in this report has been done as part of the Computer Aided Control Engineering (CACE) project at Lund Institute of Technology while the author was a visiting researcher in the Department of Automatic Control. He is most grateful to the Swedish Board for Technical Development (STU) for its support of this project under contract 86-4049 and for making his participation possible. Particular thanks are due Tomas Schönthal who did a significant portion of the programming for the window version of Simnon and provided essential assistance along the way. The author is also grateful to Sven Erik Mattsson, Dag Brück, and Leif Anderson for a variety of assistance and support. Finally, Gavin Bowe of UMIST in the UK was most helpful in sorting out some major problems in the early stages of the work.

# References

Åström, K.J. (1985): "Computer Aided Tools for Control System Design—A perspective," in M. Jamshidi and C.J. Herget (Eds.): *Computer-Aided Control Systems Engineering*, North-Holland, pp. 3–40.

Brown, M.D. (1985): *Understanding PHIGS – The Hierarchical Computer Graphics Standard*, Template, The Software Division of Megatek Corporation, San Diego, CA, USA.

Elmqvist, H., K.J. Åström and T. Schönthal (1986): *SIMNON – User's Guide for MS-DOS Computers*, Department of Automatic Control Lund Institute of Technology, Lund, Sweden.

Enderle, G., K. Kansy and G. Pfaff (1984): *Computer Graphics Programming (GKS—The Graphics Standard)*, Springer-Verlag.

Gavel, D.T., B.S. Lawver, G.P. Ledbetter, F.L. McFarland, E.G. Minor, M.E. Poggio, R.M. Shectman, J.L. Wang and J.J. Woo (1986): "EAGLES/Controls User's Manual," M-196, Lawrence Livermore National Laboratory, Livermore, California, USA.

Hopgood, F.R.A., D.A. Duce, J.R. Gallop and D.C. Sutcliffe (1983): *Introduction to the Graphical Kernel Standard (GKS)*, Academic Press.

Lawver, B. and P. Poggio (1985): "EAGLES Requirements," Computer Systems Research Group, Engineering Research Division, Lawrence Livermore National Laboratory, Livermore, California.

Little, J. and C. Moler (1987): "A Preview of MATLAB," The MathWorks, Inc., Sherborn, MA, USA.

Munro, N. and B.J. Bowland (1984): "Computer Aided Control System

Design Software – User's Guide," Control Systems Centre, UMIST, Manchester, UK.

SHAH, S.C., M.A. FLOYD AND L.L. LEHMAN (1985): "MATRIX$_X$: Control and Model Building CAE Capability," in Jamshidi, M. and C.J. Herget (Eds.): *Computer-Aided Control Systems Engineering*, North-Holland, Amsterdam, The Netherlands, pp. 181–207.

SHUEY, D., D. BAILEY and T.P. MORRISSEY (1986): "PHIGS: A Standard, Dynamic, Interactive Graphics Interface," *IEEE Computer Graphics and Applications*, **6**, No. 8, August 1986, 50–57.

SIS (1985): *Datorgrafi—PHIGS, Programmers Hierarchical Interactive Graphics Standard*, Technical report no. 306, SIS—Standardiseringskommisionen i Sverige, Sweden.

SUN (1986a): *SunCore Reference Manual*, Part No: 800-1257-03, Revision G of 17 February 1986, Sun Microsystems, Inc., Mountain View, California.

SUN (1986b): *SunView Programmer's Guide*, Part No: 800-1345-10, Revision A of 19 September 1986, Sun Microsystems, Inc., Mountain View, California.

WIESLANDER, J. (1980): "Idpac Commands – User's Guide," CODEN: LUTFD2/TFRT-3157, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.