IMPLICATIONS OF WINDOWING TECHNIQUES FOR CAI

Jesse M. Heines, Ed.D.
Georges G. Grinstein, Ph.D.

University of Lowell
Dept. of Computer Science
Lowell, MA 01854

paper presented at a conference on

Visions of Higher Education: Trans-National Dialogues

Stockton State College
Pomona, New Jersey

August 5, 1985

ABSTRACT

Windowing is a technique that allows a single computer terminal to act as either multiple output devices for a single computer program, or a single output device for multiple computer programs. This paper discusses the use of windowing in computer-assisted instruction (CAI) programs to allow independent control of functional areas in complex CAI displays and simultaneous display of output from a running computer program and coordinated instructional material.

INTRODUCTION

One of the basic fixtures of the tomorrow's learning environment for higher education will be instructional programs delivered by computers. The vision of highly interactive and adaptive teaching machines has been around for decades, but the move to developing such systems always seems to stall before it gets up a significant head of steam. Two of the obstacles to widespread use of computer-assisted instruction (CAI) in computer science courses are:

- the need to display a large amount of information on the screen at one time, and
- the need to either simulate sophisticated computer processes so they can be demonstrated from within a running CAI program or to exit the CAI program so students can try out the concepts being taught.

This paper examines the implications of using windowing techniques to address both these problems, so that more effective computer science CAI programs may be developed in the future.

USING WINDOWS

Windowing is a technique that allows a single terminal to act as either:

- multiple output devices for a single computer program, or
- a single output device for multiple computer programs.

The availability of this technique has a number of implications for CAI, where screen display space is often at a premium. The two main implications we examine are:

- the independent control of functional areas in complex CAI displays, and
- simultaneous display of output from a running computer program and coordinated instructional material.

The first part of our effort is to specify a virtual windowing system by defining the features we would like to see in a computer science CAI windowing system. The second part is to examine how such a system might be implemented in current environments, concentrating on the trade-offs that might be made to enhance simplicity and performance.

THE VIRTUAL SYSTEM

CAI screens are often complex, combining explanatory text and graphics with representations of the subject matter such as

simulated computer screens. Heines (1984) has pointed out the advantages of establishing discrete functional areas for various screen components. Windowing can be used to make control of these areas truly independent, greatly simplifying coding and debugging. Rudimentary windowing is already available in some CAI authoring systems, but these implementations can only handle non-overlaid windows. We know of no CAI authoring system that currently provides windows of arbitrary dimensions and overlay levels, dealing fully with the problems of occlusion and restoration.

Even in more sophisticated windowing systems, the missing feature, from a CAI point of view, is the ability for a process running in one window to "filter" a process running in another window. One often finds that sophisticated CAI programs contain simulators for software (such as command interpreters or compilers) that already exist on the system but that are inaccessible from inside an executing image. It is certainly true that most of today's major operating systems allow a main process to spawn a child process, pass control to it, and then analyze its results when the subprocess is terminated by the user. These systems do not, however, generally allow the parent process to "eavesdrop" on the child, analyze its results as the student works, and interrupt as soon as an error is spotted. ("Shelley," a new CAI authoring system for the IBM PC that was demonstrated at Data Training's Computer-Based Training Conference in March, 1985, is a notable exception.)

The functionality described above may, at first reading, seem to have little relevance to windows. It is, however, extremely relevant because it governs the types of interactions that one can implement in a windowing system. Without the ability to "filter" processes in the manner described above, windows supply little more than a convenient technique for the control of functional areas. While this minimum functionality is still useful, it leaves the contribution of windowing techniques to CAI far below its promise.

Sample Applications

During the 1984 spring term, Heines worked with Karen Smith at Brown University to explore the development of a CAI course with the features described above. Brown makes extensive use of Apollo Domain systems in its introductory Pascal programming course, and these systems have a number of hardware features specifically designed to simplify the implementation of windowing systems. To this hardware Brown's programmers have added BALSA,

the Brown Algorithm Simulator and Animator, a system that provides software interfaces to the windowing hardware along with other, more sophisticated features. (For a fuller description of BALSA, see Brown and Sedgewick, 1984.)

One of the applications that Heines and Smith built was an instructional game to help students master the concepts and syntax of Pascal passing parameters by value and by reference. The game was called "The Parameter Mystery", and its initial scenario was simply that someone had been murdered. The students' task was to determine who murdered whom with what. To play the game, students typed Pascal assignment statements and procedure calls as if they were writing a program. They could assign values to seven predeclared variables and call seven predefined procedures. Each procedure required certain information to be passed to it and returned certain information in turn. The students' entries were evaluated interactively, with detailed error messages for incorrect statements. Procedures called correctly with the appropriate parameters yielded clues to the mystery, from which students could eventually deduce an answer. (For a more complete description of "The Parameter Mystery," see Smith, 1985. Extensions of this work are now being pursued by Heines at The University of Lowell under a grant from Digital Equipment Corporation.)

Insert Figure 1 about here.

The initial screen for "The Parameter Mystery" is shown in Figure 1. This figure shows the BALSA logo and basic screen layout with five windows:

1. The first window is the topmost line of the screen in which the message "Type your first entry" appears. This is where a student's input appears as s/he types it.

2. The second window is the large (and currently empty) rectangle taking up most of the left-hand side of the screen. This is where feedback on the student's input will appear. This feedback will be either an explanatory error message (see Figure 2), a confirming message for an assignment statement (Figure 3), or a clue for a valid procedure call (Figure 4).

Insert Figures 2, 3, & 4 about here.

3. The third window at the top of the right-hand column of rectangles contains a standard help message on additional commands the student may enter. These are "quit" to terminate the program, "scalars" to display all known scalar values, and "formals" to display the predefined procedure names and their formal parameter lists.

4. The fourth window is the narrow one-line message that displays the number of statements already entered.

5. The fifth window at the bottom right of the screen is a dynamic display of the values of all user-assignable variables. At the beginning of the program, all of these values are undefined (see Figure 5).

Insert Figure 5 about here.

Each student entry caused information to be updated in four of the five windows. Note, however, that none of these windows overlapped. The use of windowing in this instance therefore did not result in an increased display space, but rather simplified the display of updated information and the overall management of this rather complex screen.

A more sophisticated windowing application can be seen in a complementary application built by Smith and Heines. This application reinforced material presented in lectures and prepared students for playing "The Parameter Mystery" by visually demonstrating actual programs that make use of procedures and parameters. (This application is also described more fully in Smith, 1985.)

Insert Figure 6 about here.

Figure 6 shows one screen from this application, containing four windows.

1. The first window is again the topmost line of the screen in which the message "Press Return To Continue" appears. This is where program output appears and where a student's input appears as s/he types it.

2. The second window is the short, wide rectangle below the logo that displays explanatory messages.

3. The third window is the large rectangle at the lower left of the screen that contains the program code being demonstrated.

4. The fourth window is the narrow rectangle at the lower right of the screen that displays parameters and their values.

As each line of the program in the third window is executed, BALSA encloses it in a narrow box (see the fourth line from the bottom in Figure 7). When a procedure is called, the actual parameters are displayed in Window #4 (Figure 7). The called procedure is then overlaid on top of Window #3, occluding the calling code and the formal parameter identifiers are displayed in Window #4 (Figure 8). The actual parameter values are then shown to be assigned to the formal parameters in an animated fashion to drive home the concept of parameter passing (see the time exposure in Figure 9). When the called procedure terminates, the overlaid code is removed and the screen in Figure 7 is automatically restored by BALSA. We believe that the visual power of this system is unprecedented in demonstrating the relationship between actual parameters in a calling procedure and formal parameters in the procedure being called.

Insert Figures 7, 8, & 9 about here.

THE PHYSICAL SYSTEM

While the advantages of windowing systems are easy to conceive, the implementation of these capabilities is complex. Even in systems that handle only non-occluded windows, keeping track of which window you are addressing and where positions in that window are located in its own relative coordinate system can be mind-boggling for the applications programmer. PC Pilot offers such a capability, and Heines' experience is that the task is still complex even though PC Pilot windows only support text and the language provides macro commands for switching from one window to another (see Using IBM Pilot, 1984).

When each window contains a separate process, the system must also handle interwindow communication. Since most modern operating systems provide some capability of interprocess communication, the problem can be reduced to template matching: one window to one process. Interwindow/interprocess communication can then be implemented in several ways:

- set up a parent process that periodically monitors its child processes,
- set up a parent process that continually monitors its child processes,
- set up a parent process that only responds upon an interrupt from its child processes, and
- set up a parent process that periodically interrupts its child processes.

Terminal independence is an additional requirement of the implementation, as it is not reasonable to expect student or faculty usage to be restricted to a single terminal type. In addition, the window manager should conform to international graphics standards, and most of these require device independence as a primary feature. Given the diverse university environment and the large number of possible extensions to this work, we feel that tools development is as important as the final implementation itself. A large amount of time has therefore been spent developing tools that will be used to implement interwindow communication.

Implementation Under UNIX

The Apollo Domain systems used to implement the sample applications discussed above ran a version of UNIX. UNIX is well-known for its sophisticated interprocess communication capabilities via "pipes," and BALSA handled virtually all of the required window management tasks. The master CAI program for both of the sample applications was actually run as a subprocess with BALSA as the controlling process. Each windowing operation was coded in the CAI program as an "interesting event" that signalled an interrupt to BALSA. A second set of procedures (written in C and Pascal) told BALSA what to do at each interrupt. Such procedures included updating variables, displaying data in one of the windows, demonstrating a Pascal program, and accepting keyboard and mouse input from the user. The program ran quite quickly, since each Apollo system was an semi-independent workstation.

The combination of UNIX and BALSA therefore proved to be a highly functional, albeit somewhat opaque, environment for implementing the sample applications. However, this software was intimately tied to the Apollo systems, particularly the high resolution Domain display. We therefore began experimenting with implementing similar functionality under VAX/VMS.

Implementation Under VAX/VMS

The run time library supplied with VAX/VMS Version 4.1 includes a number of screen management functions that provide the primitives needed for windowing. Such primitives include defining a pasteboard (a physical screen area) and defining windows (virtual screen areas) on this pasteboard. The run time library also provides utilities to delete a window, make a window visible or invisible, move windows, change priorities, accept input from a window, etc. These operations are somewhat terminal independent. We have tested them on a VT100, a GIGI, and a VT240. While we have encountered some problems running applications on more than one terminal type, we are not sure at this point whether those problems lie in our software or the run time library.

When we looked at the possibility of using interprocess communication with different windows, we ran into great difficulty. First, as is usual, we found that the system documentation is written for very sophisticated applications programmers and leaves a great deal unsaid. Second, we found that the system overhead involved in the creation of internal "mailboxes" for interprocess communication was large and caused the application to run very slowly.

When our prototype windowing communication software ran with one user there was a noticeable set up time, but the windows communicated with reasonable speed. When several users were on the system, interprocess communication was slow. The most successful implementation we have accomplished to date involved spawning a number of processes, some to handle interprocess communication and others to handle output to the different windows.

We succeeded in setting up a parent that continually monitored its child processes and in setting up a parent process that only responds upon an interrupt from its child processes. We are still looking at ways of implementing the other two approaches under VAX/VMS.

CONCLUSIONS

We have demonstrated that windowing is a highly desirable CAI feature, but implementation has proven difficult. BALSA and UNIX provide most of the needed capabilities on Apollo Domain systems, but this software system is difficult to transport to other systems. VAX/VMS screen management utilities are in some ways

more device independent, but they provide considerably less functionality. In addition, the high overhead of interprocess communication on VAX/VMS makes use of these routines untenable under normal system loads.

In some ways, our work thus far has provided more questions than answers. The prototype applications discussed above have significantly helped us to organize our thoughts about asynchronous processes in the CAI environment, but at this time more practical approaches still need to be devised.

REFERENCES CITED

Brown, Marc H., and Robert Sedgewick, 1984. A system for algorithm animation. Brown University Dept. of Computer Science, Technical Report No. CS-84-01.

Heines, Jesse M., 1984. Screen Design Strategies for Computer-Assisted Instruction. Digital Press, Burlington, MA.

Smith, Karen E., 1985. Developing and evaluating a computer-assisted instruction dialogue on parameters. Brown University Dept. of Computer Science, Technical Report No. CS-85-04.

Using IBM Pilot, 1984. IBM Corporation, Irving, TX.

Figure 1. Initial BALSA screen layout for "The Parameter Mystery."

Figure 2. Clue from a correct call to procedure "knew" in "The Parameter Mystery."

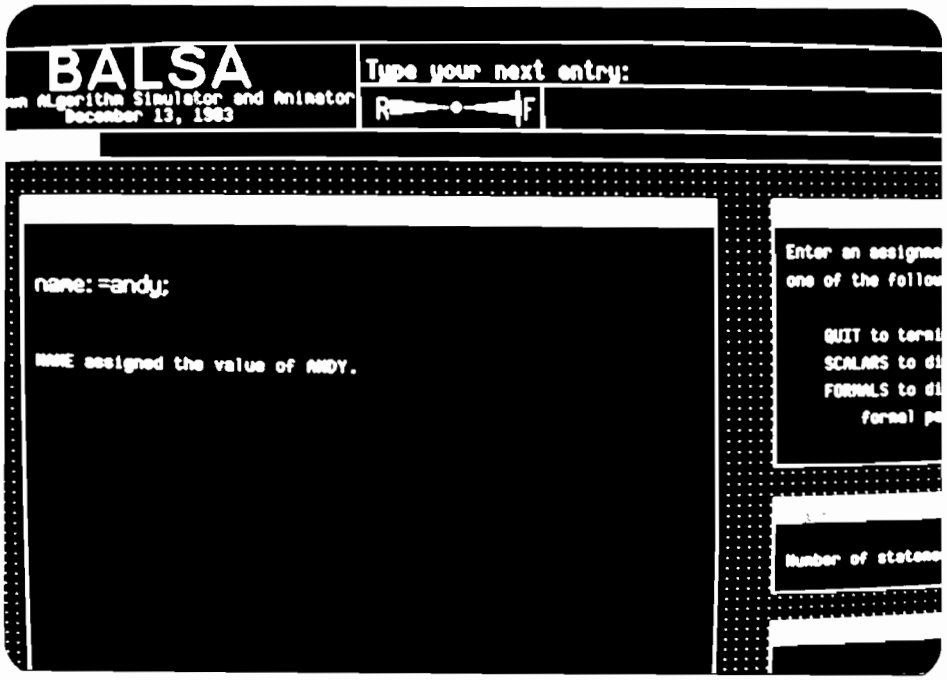

Figure 3. Confirmation of a correct assignment to variable "name" in "The Parameter Mystery."

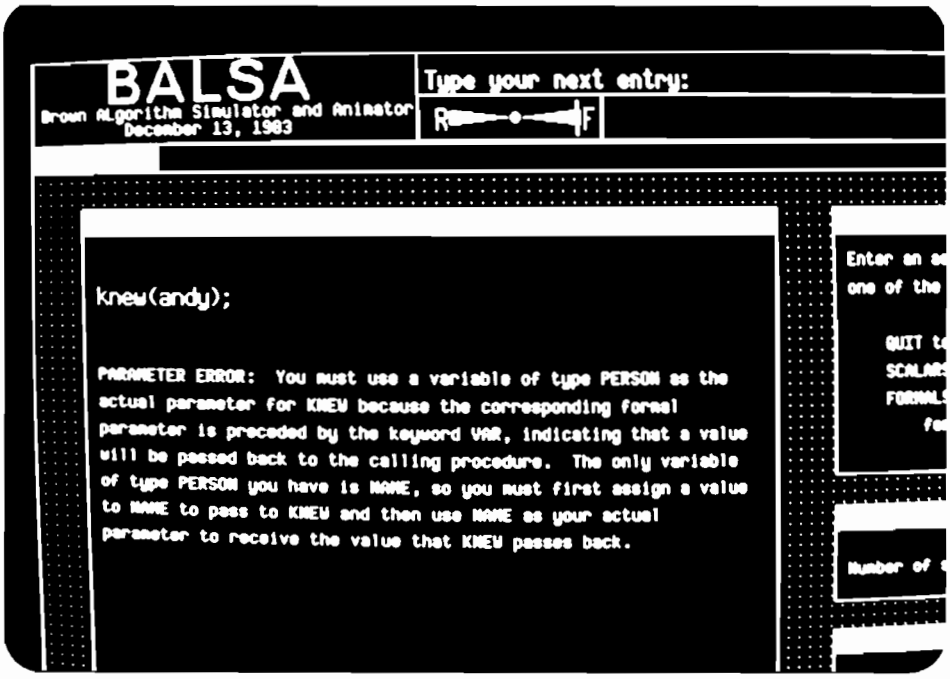

Figure 4. Error message for an incorrect call to procedure "knew" in "The Parameter Mystery."

Figure 5. Windows showing the number of statements already entered and the values of all user-assignable variables in "The Parameter Mystery."

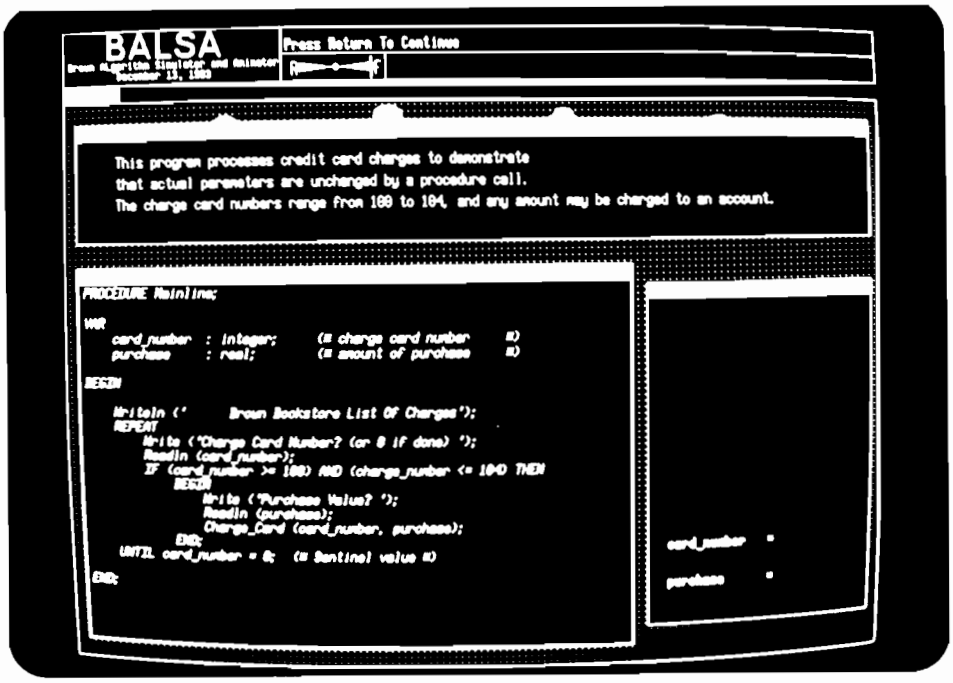

Figure 6. Initial BALSA screen layout for a CAI application that uses occluded windows.

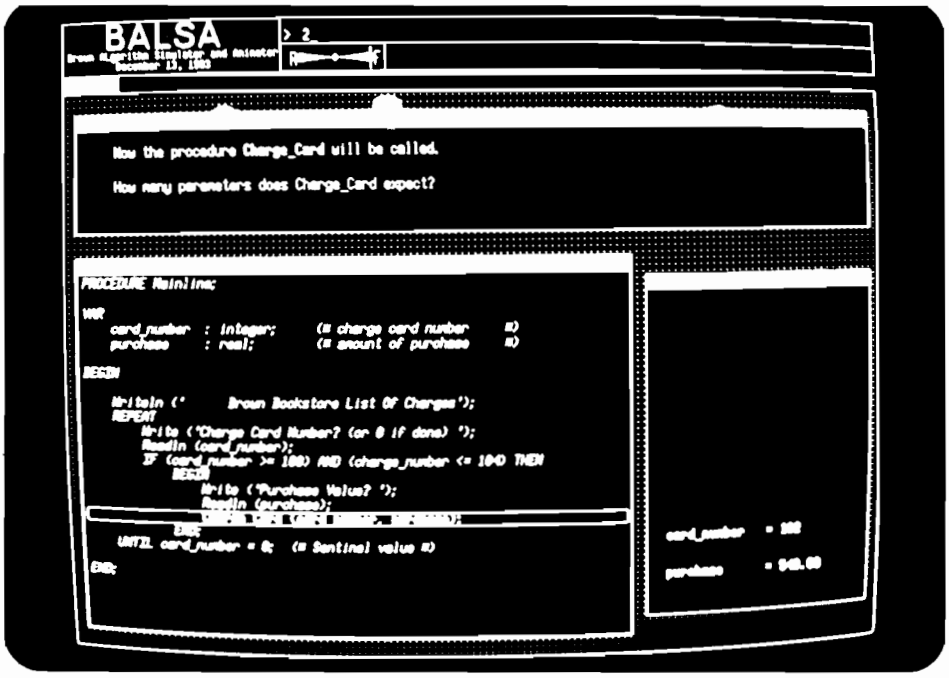

Figure 7. Highlighted procedure call and appearance of actual parameters in adjacent window.

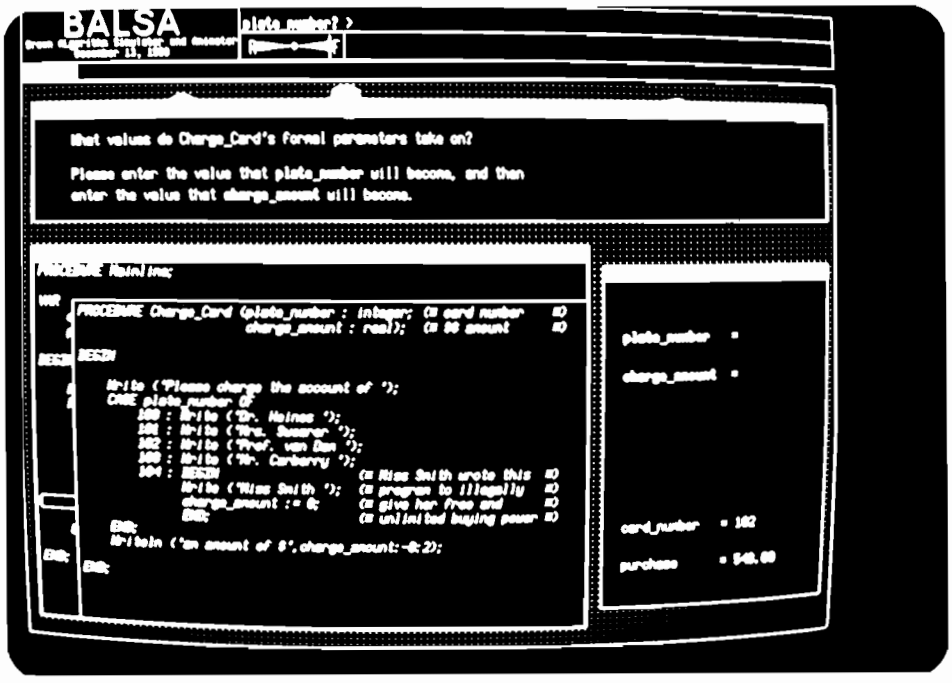

Figure 8. Overlaid code of called procedure and appearance of formal parameters in adjacent window.

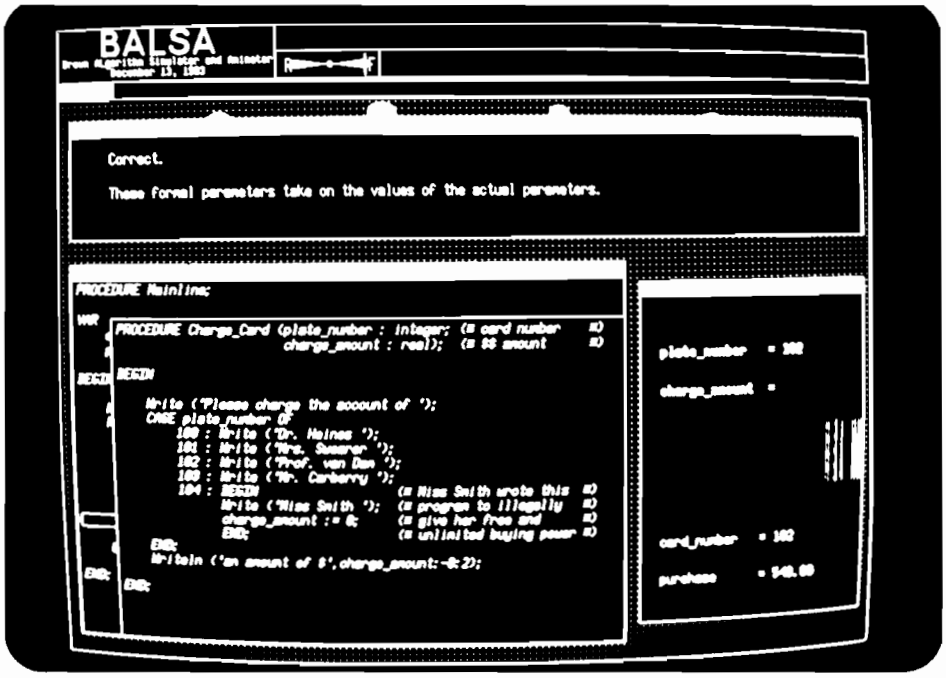

Figure 9. Movement of actual parameter values to formal parameters in an animated fashion.