The Evolution of the SSSP Score Editing Tools

William Buxton, Richard Sniderman,
William Reeves, Sanand Patel
and Ronald Baecker

Structured Sound Synthesis Project
Computer Systems Research Group
University of Toronto
Toronto, Ontario
Canada
M5S 1A1

Introduction

This paper traces the evolution of score editing tools developed by the Structured Sound Synthesis Project (SSSP) at the University of Toronto. The focus is on the use of interactive computer graphics to assist a composer in the editing of a file of information detailing a musical score. In this context we examine some of the issues involved in designing user interfaces involving interactive graphics systems. An outline of an overall design strategy is presented which has been used successfully in SSSP hardware and software design.

Background

The SSSP is an interdisciplinary project whose aim is to conduct research into problems and benefits arising from the use of computers in musical composition (Buxton: 1978; Buxton and Fedorkow: 1978; Buxton, Fedorkow, Baecker, Reeves, Smith, Ciamaga and Mezei: 1978). This research can be considered in terms of two main areas: the investigation of musical data and processes, and the study of musician-machine communication.

In designing the system, we decided early on to adopt a highly interactive approach to the design of the human interface. Batch processing as in Music V (Mathews: 1969) is an alternative, but one which widely separates the composer and the program, causing serious delays in the feedback loop. We feel a score editor must be interactive because there are facets of the task which demand control and aesthetic judgement by the composer in an interactive and exploratory manner. Several modes of interaction have previously been used in music systems, such as alphanumeric text as in MUS10 (Smith: 1978), voice recognition (Tucker, Bates, Frykberg, Howrath, Kennedy, Lamb, Vaughan: 1977), and piano-type keyboards (New England Digital Corp.: 1978). In our work we have adopted a bias towards graphics-based interaction (Baecker: 1979; Newman and Sproull: 1979) in the belief that this approach can make a significant contribution towards an effective human interface. First, music lends itself well to representations in the visual domain. Second, the task of editing music is complex in the sense that there are many parameters and commands to be manipulated and controlled; this complexity can be reduced by the graphic representation of information. Third, previous work (Pulfer: 1972; Tanner: 1972; Vercoe: 1975) indicates that more congenial interfaces can be constructed using dynamic graphics techniques.

The hardware environment centers on a PDP-11/45 running under the UNIX time-sharing operating system (Ritchie and Thompson: 1974) and a digital sound synthesizer (Buxton, Fogels, Fedorkow, Sasaki and Smith: 1978) with its own LSI-11. The graphics hardware includes a refresh, vector-drawing graphics display, a digitizing tablet with accompanying cursor box, and a slider box. Several terminals with alphanumeric keyboards are also available for non-graphical processing.

3. The Evolution of Score Editing Tools

3.1 Overview

The main area of SSSP activity has been the development of high level "front-end" programs which would give the musically sophisticated (but technologically naive) user a high degree of access to the potential offered by the computer-synthesizer combination. Thus, the development of score editing tools was initiated even before such tools could be "hooked up" to the synthesizer, and has proceeded continuously since. Figure 1 charts this development and the following sections detail the salient features of each stage.

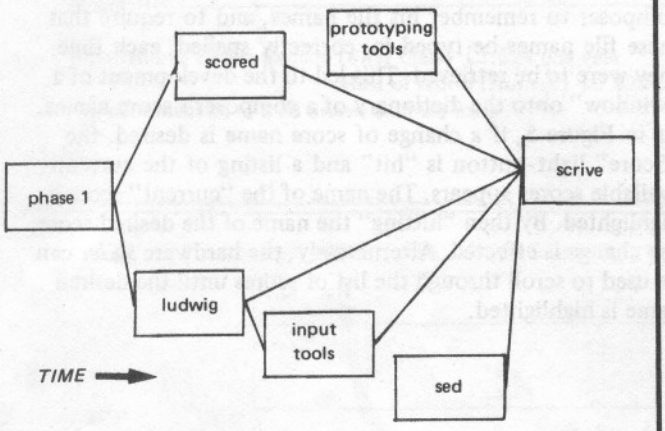

Figure 1. Evolution of SSP score editing tools

3.2 In the beginning . . .

We first needed to quickly address some of the fundamental issues concerning graphical interaction as applied to score editing. This led to the design and implementation of an editing environment which was never used in conjunction with the synthesizer. However, it was at this early stage that techniques of graphically realizing entities of musical notation were developed – techniques that were used and built upon in subsequent work.

It was also at this stage that an interactive method of note input was devised which has remained the major note input tool to date. Figure 2a shows the note input tracking cross being positioned over the desired pitch. On depressing the button on the cursor, a "marker note" symbol appears at the indicated pitch. Concurrently, the tracker is replaced by a sequence of notes. This is shown in Figure 2b. This sequence of notes "tracks" or follows the motion of the cursor on the tablet. The roles of the tracking-cross and the menu are now reversed. Instead of the conventional stationary menu and moving pointing tool, we have a moving menu and a stationary pointer. By placing the note of the desired duration over the marker note symbol (*i.e.* moving the menu), and releasing the cursor button, a note is input. This is shown in Figure 2c. The ledger lines, tail direction, bar lines, and note spacing are automatically handled.

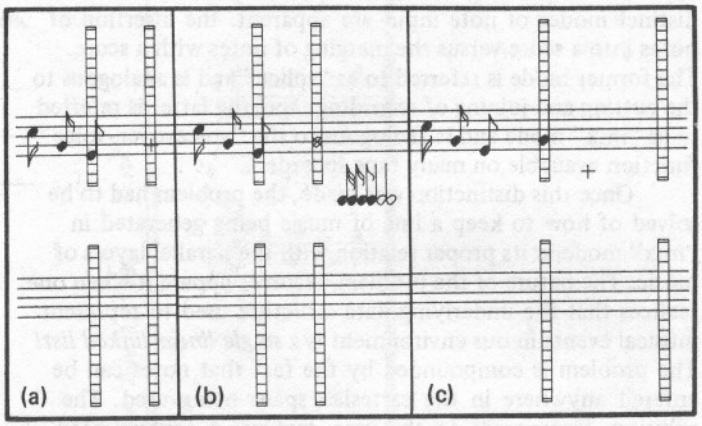

Figure 2. Input of a note

This note input tool is very flexible. By positioning the marker note in the first set of ledger lines, the new note is chorded with all notes above or below it. This allows polyphonic scores to be edited. By manipulating other input transducers, the composer can also enter ties, delete the last note and make pitch corrections on the last note. A rest input tool was also included, the protocol for which is identical to the note tool except that the sequence of notes becomes a sequence of rests.

This prototype score editor was discarded soon after completion. Valuable lessons had been learned and the cost of development was not so great as to discourage its quick replacement. This latter point serves as the cornerstone of SSSP design strategy – namely that no prototype should be so costly as to make it unfeasible to discard it upon completion. If that is the case then the time should be spent in the development of tools to assist prototype design, rather than in the development of the prototypes themselves.

3.3 ludwig

ludwig (Reeves, Buxton, Pike and Baecker: 1978) was the first score editing program to be used by composers in conjunction with the synthesizer. Although both it and its precursor use common music notation (CMN) to represent musical events, this does not imply an affinity or a bias towards CMN. Rather, CMN was used to present the new computer-related concepts to composers in a familiar environment. It was recognized that CMN is just a special case of notation in a cartesian space, using notes and other special symbols placed in some relation to staves. In fact, as will be seen in Section 3.4, an editor using "piano-roll" notation was developed at the same time.

Various issues were addressed for the first time during the course of *ludwig*'s development. One of the more important ones was motivated by the rather facile observation that music is not usually monolinear. Take for example any Bach *Two-Part Invention.*

One does not think of the composition as a linear sequence of notes, but rather as a layering of parallel voices (see Figure 3). In notating the *Invention* a more reasonable approach would be to notate each voice separately. Thus two distinct modes of note input are apparent: the insertion of notes into a score versus the merging of notes with a score. The former mode is referred to as "splice" and is analogous to the cutting and joining of recording tape; the latter is referred to as "mix" mode and is analogous to the "voice-over-voice" function available on many tape recorders.

Once this distinction was made, the problem had to be solved of how to keep a line of music being generated in "mix" mode in its proper relation with the parallel layers of music. The nature of the problem becomes apparent when one realizes that the underlying data structure used to represent musical events in our environment is a *single linear linked list!* The problem is compounded by the fact that notes can be entered anywhere in the cartesian space being used. The solution, transparent to the user, involves a sophisticated manipulation of the pointers which constitute the linkages of the list.

Another issue considered in the *ludwig* design was that of *score navigation*—how to enable the composer to "get around" the score, the notes of which may or may not be currently displayed on the screen. Two major techniques were developed. The first involves *scrolling* the score across the screen in "real-time" at the touch of one of the hardware *sliders*. The second involves pointing at the light-button "Search," and pressing a cursor button. As seen in Figure 4 a time-line representing the entire duration of the score appears on the screen. Two angle brackets indicate the portion of the score currently visible. Thus the user can readily answer the question, "Where in the score am I?". By placing the tracking-cross anywhere on the time-line and depressing the cursor button, the corresponding portion of the score will become visible.

The problem of simplifying the composer's task in inputting and outputting score files was tackled in *ludwig's* design. Computer science-related nomenclature (read/write) was initially used for these operations but was not understood readily by the composers. More user-oriented nomenclature (retrieve/save) met with far greater success. It was also apparent that the burden should not be placed on the composer to remember his file names, and to require that these file names be typed in, correctly spelled, each time they were to be retrieved. This led to the development of a "window" onto the dictionary of a composer's score names. As in Figure 5, if a change of score name is desired, the "Score" light-button is "hit" and a listing of the currently available scores appears. The name of the "current" score is highlighted. By then "hitting" the name of the desired score, the change is effected. Alternatively, the hardware *slider* can be used to scroll through the list of scores until the desired name is highlighted.

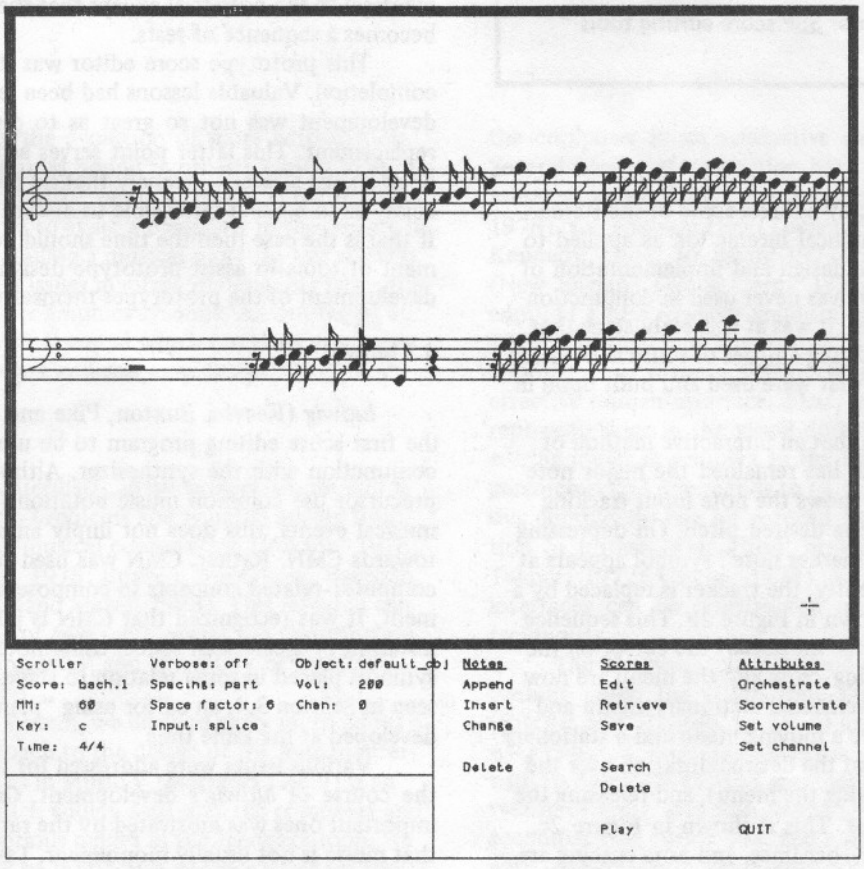

Figure 3. A Bach *Invention* notated using *ludwig*

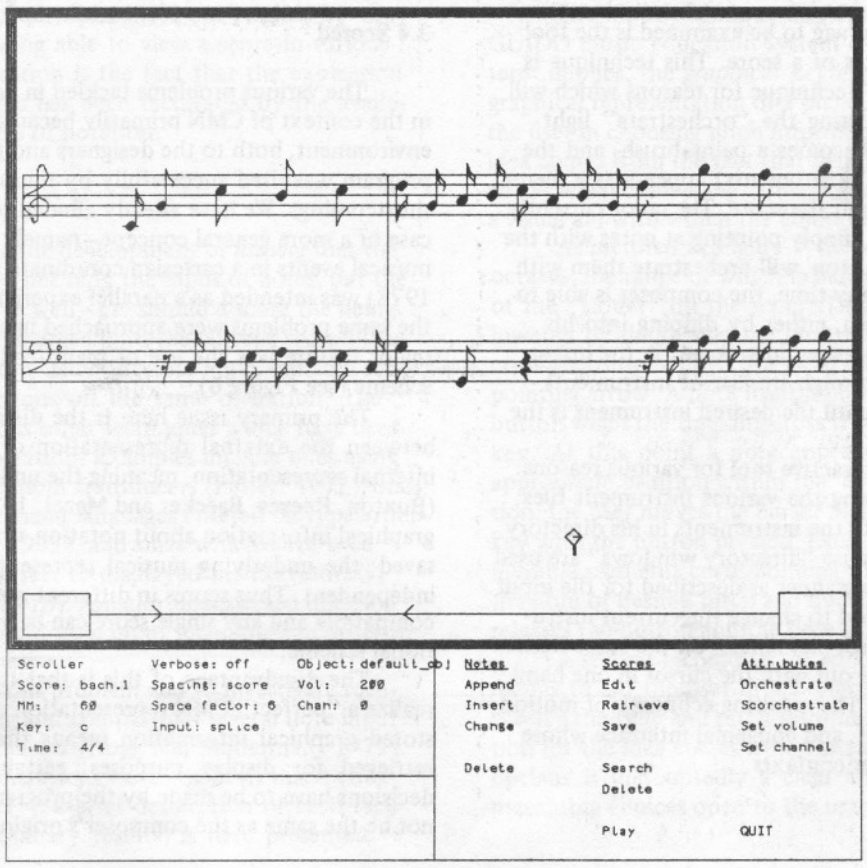

Figure 4. The *ludwig* Search command

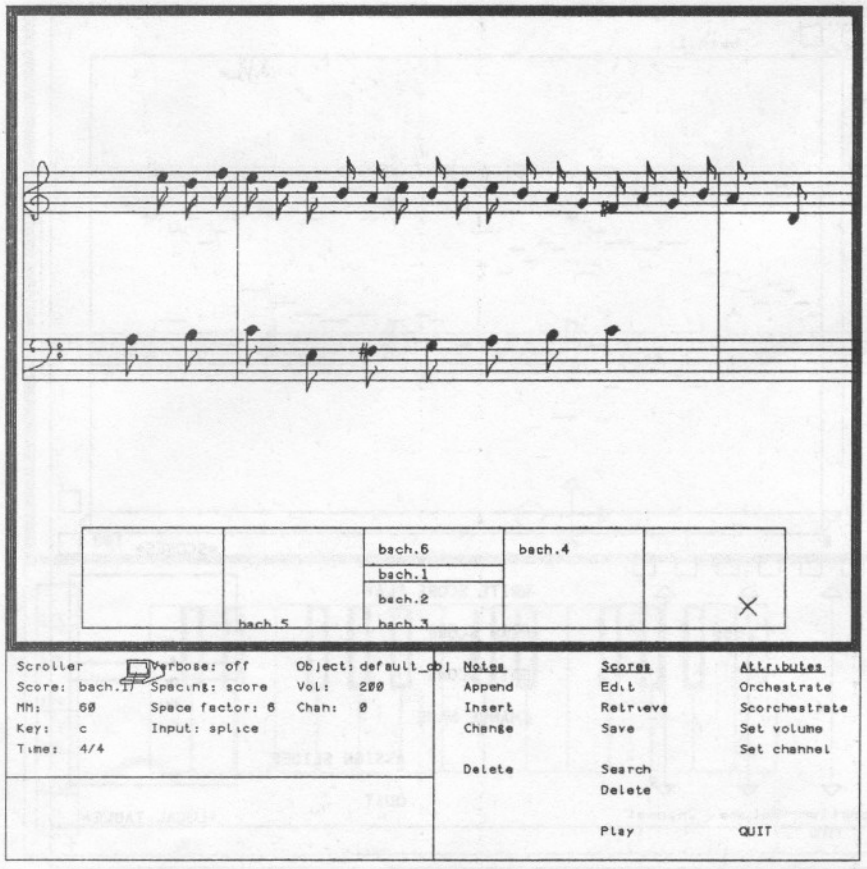

Figure 5. A directory window

The final feature of *ludwig* to be examined is the tool used to orchestrate the notes of a score. This technique is often called the "paint pot" technique for reasons which will become obvious. After selecting the "orchestrate" light-button, the tracking-cross becomes a paint-brush, and the palette of timbral "colors" (instruments) appears as a menu below the displayed portion of the score. The color currently on the brush is highlighted. Simply pointing at notes with the brush, and depressing the button, will orchestrate them with the current instrument. At any time, the composer is able to change the color of his brush, either by dipping into his palette (*i.e.* pointing at the desired instrument in the instrument menu), or scrolling through the list of instruments—using the hardware *slider*—until the desired instrument is the one that is highlighted.

This is a successful interactive tool for various reasons. *No typing* is done in accessing the various instrument files, nor must the composer recall the instruments in his directory or their spelling. This is because "directory windows" are used in orchestration in the same manner as described for file input. Also, since the sliders are used to change the current instrument in the palette, the cursor can remain on the score itself. Orchestration is thus carried out with the cursor in one hand and the sliders in the other. The resulting economy of motion results in a smooth, efficient, and congenial interface whose use can be extended to other contexts.

3.4 Scored

The various problems tackled in *ludwig* were approached in the context of CMN primarily because it offered a familiar environment, both to the designers and the users. While the program was used successfully by musicians, it had obvious shortcomings. We have already alluded to CMN being a specific case of a more general concept—namely the representation of musical events in a cartesian coordinate space. *Scored* (Hume: 1978) was intended as a parallel experience to *ludwig* in which the same problems were approached in alternative ways. The major feature was the use of piano-roll as the notational scheme (see Figure 6).

The primary issue here is the distinction being made between the external representation of a score, and the internal representation, meaning the underlying data structures (Buxton, Reeves, Baecker and Mezei: 1978). Since there is *no* graphical information about notation stored when a score is saved, the underlying musical representation is notation-independent. Thus scores in different notations are completely compatible and any single score can be viewed using any notational scheme.

The disadvantage of this is that it is not possible to realize a "perfect" CMN representation of a score. The lack of stored graphical information means that when a score is retrieved for display purposes, certain representational decisions have to be made by the program which may or may not be the same as the composer's original notation.

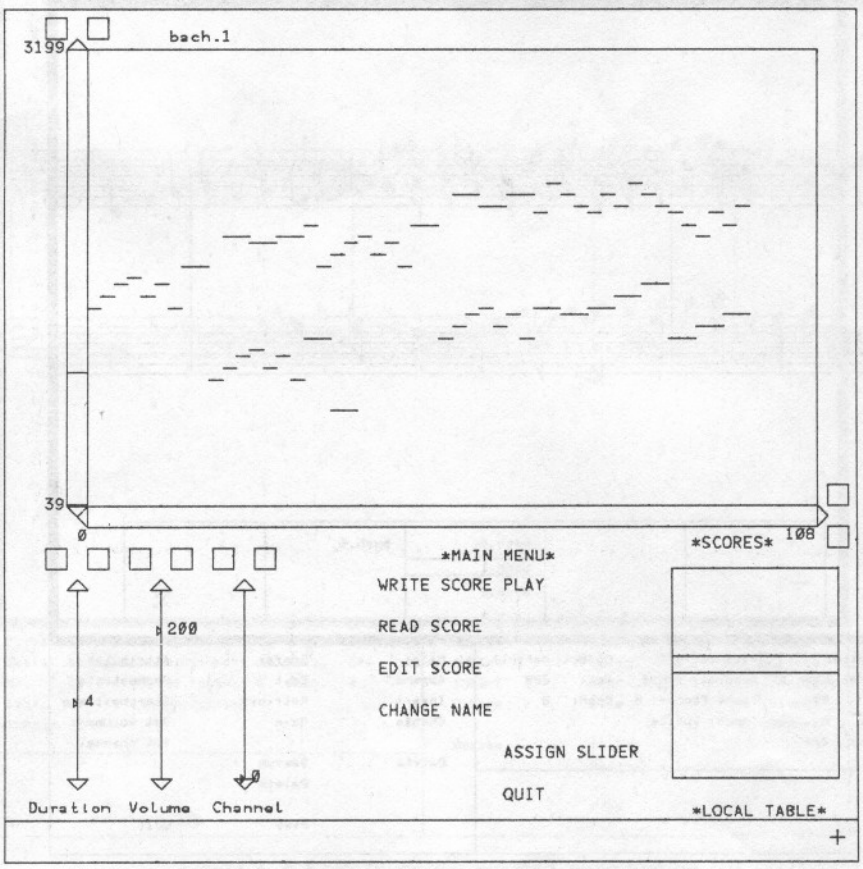

Figure 6. The Bach *Invention* of Figure 3 notated using *scored*

This drawback, in our opinion, is outweighed by the advantages gained by being able to view a score in various ways. A musical justification is the fact that the expression of different musical ideas has been facilitated by the availability of alternative forms of notation.

3.5 Note Input Tools

It was clear during the development of *ludwig* that the mix/splice issue affected not only the input of notes, but the reading of entire scores as well—*i.e.* should a score file being input be appended to or merged with the current score? Thus it was realized that conceptually our note input tool and score reading were two variations on the same operation. The literature indicates various other techniques which have been used to perform this operation. Examples include ones using specially built hardware input transducers (Pulfer: 1972), ones using alphanumeric command languages entered on typewriter-like keyboards (Smith: 1972), and ones which have used graphical interaction *via* sets of displayed menus (Murray, Beauchamp and Loitz: 1978). A major unanswered question is how to compare different note input techniques in order to evaluate their effectiveness.

In order to study this problem and to investigate techniques for benchmarking and comparison, several note input tools have been implemented (Hogg and Sniderman: 1979). Although no formal conclusions have yet been made (this being "work in progress"), a brief description of each technique, along with preliminary results, is here presented.

One input technique implemented was based on the GUIDO music education system (Hofstetter: 1975). As the term implies, the *graphical keyboard technique* employs a graphical representation of a piano keyboard, primarily to aid the user in choosing pitch information. The motivation for the idea comes from the fact that many musicians use a piano as an aid when composing. Thus, a graphical keyboard should be a visual aid with which the composer is comfortable.

As pictured in Figure 7, the keyboard consists of two octaves, the range of which is indicated by the vertical position of the "ladder" on the score. This range can be raised or lowered by depressing the cursor button when the tracking cross is positioned over the head of the upward or downward pointing arrow. A note itself is chosen by depressing the cursor button when the tracking cross is positioned over the desired key. At this point a note appears on the score at the appropriate position within the "ladder." To choose a duration, the user moves the cursor along the length of the piano key with the button pressed. Each piano key is divided into invisible segments corresponding to different durations. When the note of desired pitch and duration is displayed, the user releases the button and the note is set in the score.

A second technique, the *total menu technique* employs menus to aid the user in making the decisions needed to specify the time and pitch information of notes. The motivation for the idea comes from the fact that a menu listing all options is undoubtedly a clear and unambiguous way of presenting choices open to the user.

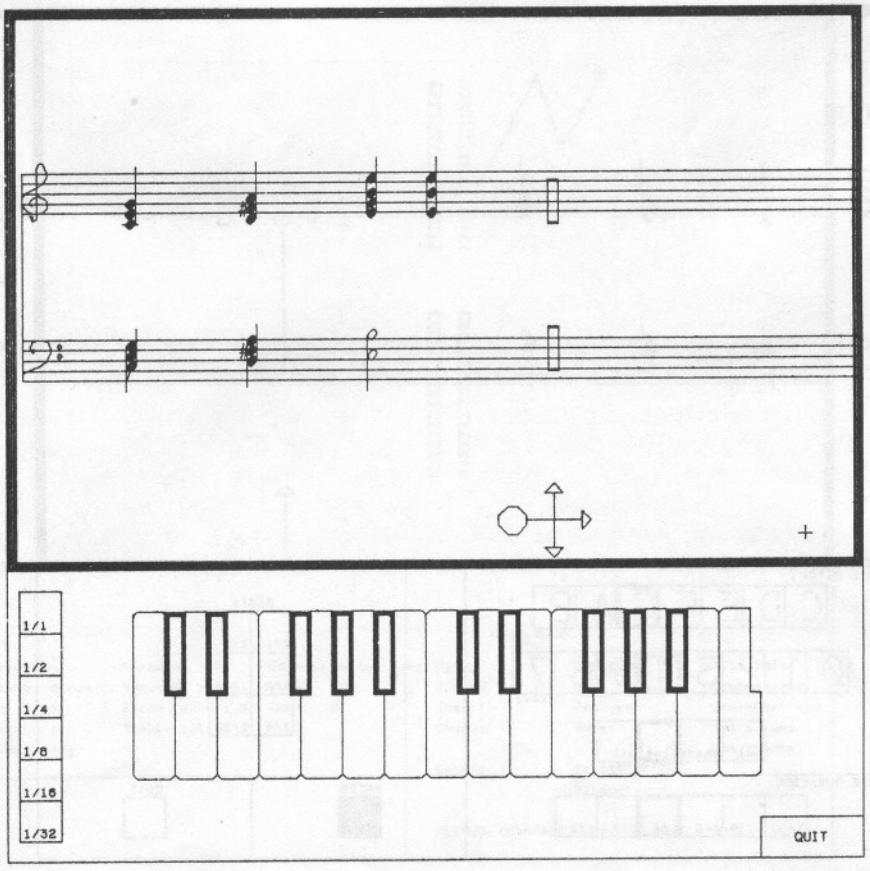

Figure 7. A typical screen layout of the *graphical keyboard technique*

Examine the menus as pictured in Figure 8. The user indicates the name, octave, accidental, and duration of a note by hitting the appropriate light buttons which are then intensified to indicate visually the choices made. When the desired choices have been made, the user hits the "ENTER" button. Contrast this system of menus with those of the PLACOMP language as shown in Figure 9 (Murray, *et al.*: 1978), from which it is derived.

Char-rec is a "short-hand" technique which employs a character recognizer to decode a set of symbols (Figure 10) designed to represent notes of varying durations. To enter a note, the tracking-cross is placed over the desired pitch on the appropriate ladder and the cursor button is depressed. An "ink trail" is laid down until the button is released during which interval the user draws the desired duration symbol. Based on the starting point of the symbol, and the shape of the symbol, the pitch and duration of the desired note are decoded. The lengths of the line segments constituting a symbol are immaterial provided they are greater than a fairly small minimal size. However, the directional changes in a symbol must be drawn distinctly to ensure correct recognition. Figure 11 shows a sixteenth note being entered.

Research with the techniques described (including the one of Section 3.2) is planned along two lines. The first involves looking at their features with the aim of classifying the designs. For example, the original technique uses vertical movements of the tracking cross to make pitch choices, and horizontal movements of a menu of notes to make duration choices. The *graphical keyboard technique* on the other hand

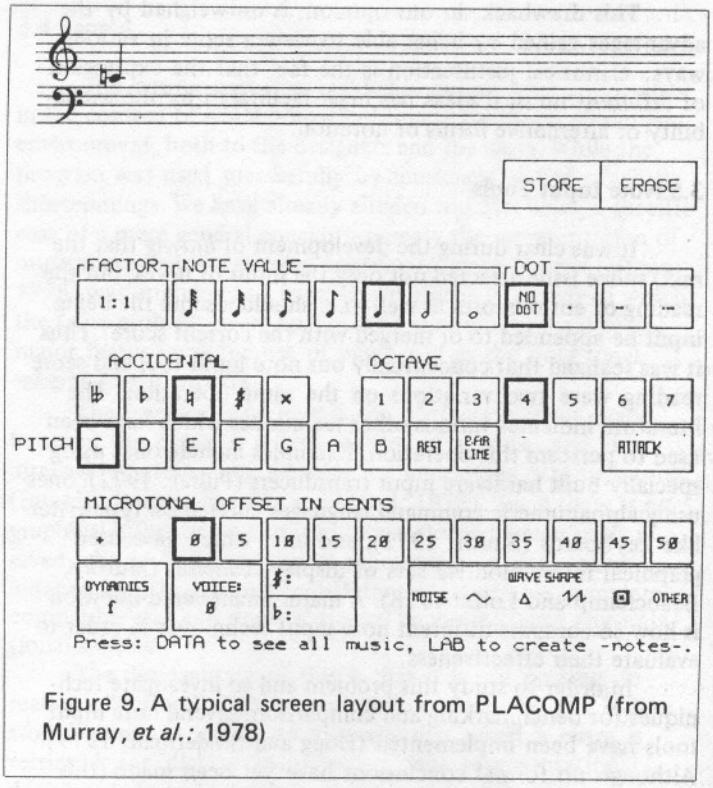

Figure 9. A typical screen layout from PLACOMP (from Murray, *et al.*: 1978)

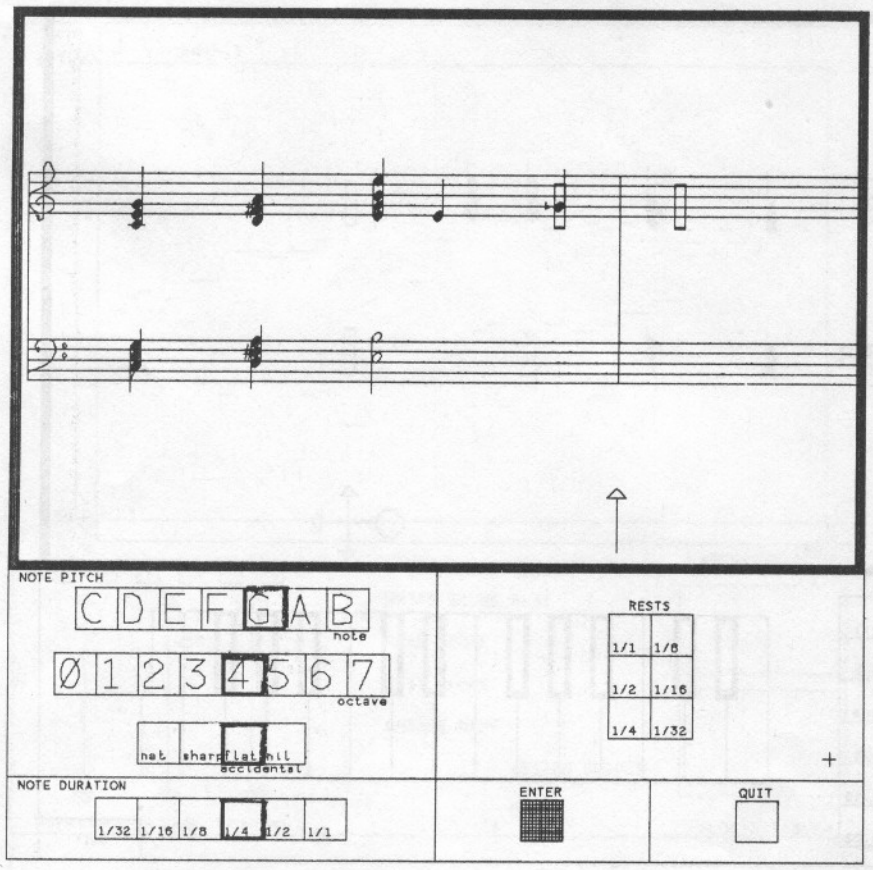

Figure 8. A typical screen layout for the *total menu technique.*

uses horizontal movements along the keyboard to make pitch choices, and vertical movements along individual keys to make duration choices. Both alternatives are natural in the context in which they are used. The *graphical keyboard technique* is similar to the *total menu technique* in that conceptually the keyboard is being used exactly as menu—each key is a light button and each key is subdivided into individual (invisible) segments forming a menu of durations.

As stated, the second line of research will involve evaluation of the ease of use of these tools. Preliminary observations indicate that the character recognizer enables a fluency of transcription that far exceeds the others—this despite the fact that it is the more difficult to learn. This difficulty arises from the fact that to use it maximally, the duration symbols must be memorized. The *total menu system* on the other hand requires no memorization since each option for every choice that must be made is clearly listed. Although it is therefore easy to learn, it is of minimal use in a composition environment, due to the number of hand motions which must be made to enter each note. It has, however, proven useful in other contexts, such as computer-aided instruction, which have different constraints than composition.

3.6 Consolidation

At this stage, focus shifted onto an evaluation of the effort being expended by the programmers implementing the score editor designs. It was becoming increasingly clear as more software was developed that various transactions were being programmed repeatedly. There was an obvious need to eliminate this duplication of effort. Some software was also being written by groups so that protocols had to be repeatedly established for interfacing their work. Thus, it was evident that before more ambitious score editors were developed it was necessary to work on a library of sophisticated prototyping tools to facilitate this development.

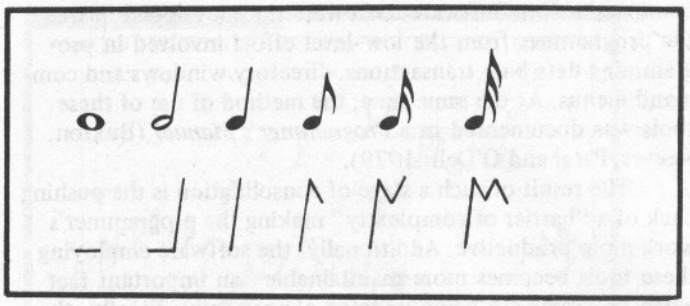

Figure 10. *Char-rec* duration symbols

The following are examples of work in this area. The first involved isolation of basic transactions being performed upon the musical data base repeatedly in all programs interfacing with the synthesizer. Another resulted from the observation that "directory windows" (as mentioned with regards to *ludwig* note input and orchestration), were being used repeatedly. Yet another resulted from the fact that all of our interactive graphics programs involved at least a one-level

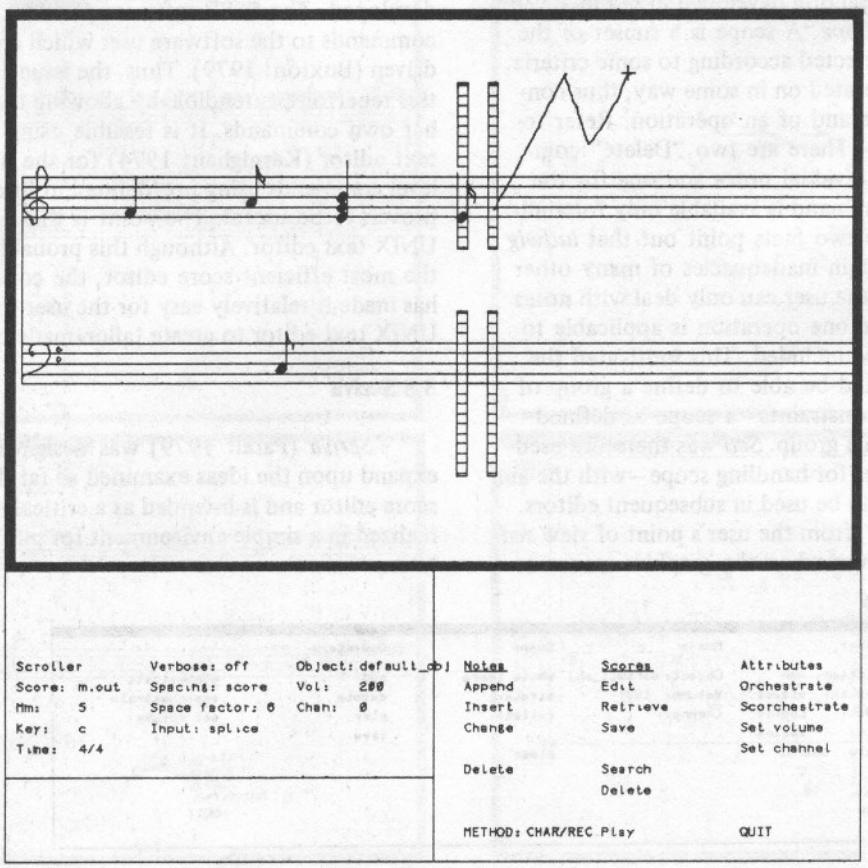

Figure 11. Appending a sixteenth note with *Char-rec*

command menu. Software tools were thus developed to free the programmer from the low-level effort involved in programming data base transactions, directory windows and command menus. At the same time, the method of use of these tools was documented in a *Programmer's Manual* (Buxton, Reeves, Patel and O'Dell: 1979).

The result of such a stage of consolidation is the pushing back of a "barrier of complexity" making the programmer's work more productive. Additionally, the software employing these tools becomes more maintainable—an important fact when programs are being updated almost daily. Finally, the design process as a whole is not only sped up, but becomes more cost effective. Less effort is expended so that the resulting programs are more expendible. When a prototype is being developed as part of an evolutionary process and not as an intended "final product," this reduction in cost is attractive.

3.7 Sed

The next score editor written was a marked departure from its predecessors in being driven alphanumerically, rather than graphically. The reason for this was twofold. First, our facilities have a number of conventional terminals with alphanumeric keyboards but very few graphical hardware resources. The availability of an alphanumeric score editor was thus necessary to increase access to the system. It also provided an environment in which to develop and test the various primitives discussed in the preceding section (and those about to be discussed) in a concise manner—one which avoided the complexity and cost of using the graphics.

One new type of transaction developed in *sed* involved the definition and use of *scope*. A scope is a subset of the musical events of a score, selected according to some criteria. This scope may then be operated on in some way, thus conceptually becoming the operand of an operation. Refer to Figure 3 illustrating *ludwig*. There are two "Delete" commands available, one for individual notes and one for the entire score. The "Play" command is available only for auditioning entire scores. These two facts point out that *ludwig* suffered from one of the main inadequacies of many other score editors—namely that the user can only deal with notes or scores. In the case where one operation is applicable to either, two versions must be included. This motivated the feeling that composers should be able to define a group of notes based on their own constraints—a scope as defined above—and operate on such a group. *Sed* was therefore used to debug and test procedures for handling scope—with the aim that the powerful tool would be used in subsequent editors.

As mentioned already, from the user's point of view *sed* allowed work to progress even when the graphics resources

```
% sed
Type 'h' for help.
?
* r bach.1
input mode: s
93
* 1,40L
1       freq: 494   dur: 1/16   obj: default_obj   vol: 200   del: 0/16
2       freq: 147   dur: 1/2    obj: default_obj   vol: 200   del: 1/16
3       freq: 262   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
4       freq: 294   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
5       freq: 330   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
6       freq: 349   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
7       freq: 294   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
8       freq: 330   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
9       freq: 262   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
10      freq: 392   dur: 1/8    obj: default_obj   vol: 200   del: 0/8
11      freq: 147   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
12      freq: 131   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
13      freq: 523   dur: 1/8    obj: default_obj   vol: 200   del: 0/8
14      freq: 147   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
15      freq: 165   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
16      freq: 494   dur: 1/8    obj: default_obj   vol: 200   del: 0/8
17      freq: 175   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
18      freq: 147   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
19      freq: 523   dur: 1/8    obj: default_obj   vol: 200   del: 0/8
20      freq: 165   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
21      freq: 131   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
22      freq: 587   dur: 1/16   obj: default_obj   vol: 200   del: 0/16
23      freq: 196   dur: 1/8    obj: default_obj   vol: 200   del: 1/16
24      freq: 392   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
25      freq: 440   dur: 1/16   obj: default_obj   vol: 200   del: 0/16
26      freq: 98    dur: 1/8    obj: default_obj   vol: 200   del: 1/16
27      freq: 494   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
28      freq: 523   dur: 1/16   obj: default_obj   vol: 200   del: 0/16
29      freq: 147   dur: 1/4    obj: default_obj   vol: 200   del: 1/16
30      freq: 440   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
31      freq: 494   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
32      freq: 392   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
33      freq: 587   dur: 1/8    obj: default_obj   vol: 200   del: 0/8
34      freq: 147   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
35      freq: 196   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
36      freq: 784   dur: 1/8    obj: default_obj   vol: 200   del: 0/8
37      freq: 220   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
38      freq: 247   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
39      freq: 698   dur: 1/8    obj: default_obj   vol: 200   del: 0/8
40      freq: 262   dur: 1/16   obj: default_obj   vol: 200   del: 1/16
* p
```

Figure 12. The score of Figures 3 and 6 in *sed* notation

were employed. However, a somewhat unexpected benefit developed. The SSSP software provides a wide repertoire of commands to the software user which are alphanumerically-driven (Buxton: 1979). Thus, the issue arises of how to make this repertoire extendible by allowing the user to define his/her own commands. It is feasible using the UNIX shell and text editor (Kernighan: 1974) for the user to build higher-level commands using pre-defined utilities; such an option has proven to be useful. The point is that *sed* is based on the UNIX text editor. Although this probably has not resulted in the most efficient score editor, the consistency of approach has made it relatively easy for the user to learn enough of the UNIX text editor to create tailor-made commands.

3.8 Scriva

Scriva (Patel: 1979) was designed to integrate and expand upon the ideas examined so far. It is again a graphical score editor and is intended as a critical mass of many ideas realized in a simple environment for purposes of experimentation.

```
Editor              Music                Scope          Operators

Notation: cmn       Object: default_obj  whole score    add          orchestrate
Display:  staves    Volume:  192         circle         delete       scorchestrate
Input:    Ludwig    Channel:   1         collect        play         set volume
Join:     splice                                       save
Score:  bach.1                         clear
Key:      C
Mm       60
Pose                                                   QUIT
```

Figure 13. *Scriva* command window layout.

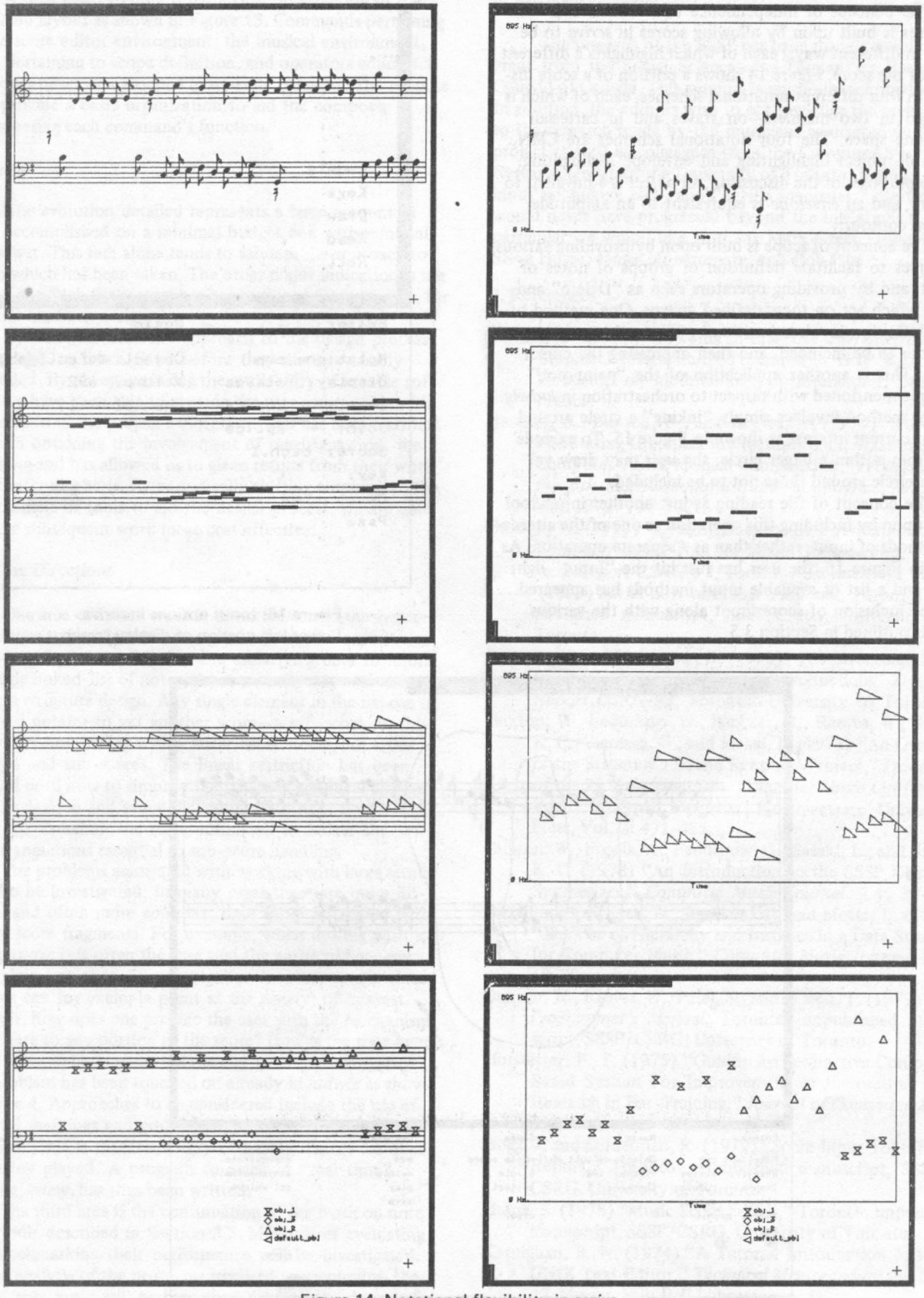

Figure 14. Notational flexibility in *scriva.*
The score display portion of eight screens.

The concept of independence of notation and data structures is built upon by allowing scores in *scriva* to be viewed in different ways, each of which highlights a different aspect of the score. Figure 14 shows a portion of a score displayed in four different notational schemes, each of which is displayed in two manners—on staves and in cartesian coordinate space. The four notational schemes are CMN, piano-roll, object highlighting and envelope highlighting. For the purposes of the discussion, an *object* is equivalent to a timbre, and an *envelope* is equivalent to an amplitude/loudness contour.

The concept of scope is built upon by providing various techniques to facilitate definition of groups of notes of interest, and by providing operators such as "Delete" and "Play" which act on these defined groups. One method of scope definition involves positioning the tracking cross over each note to be included, and then depressing the cursor button. This is another application of the "paint-pot" technique mentioned with respect to orchestration in *ludwig*. Another method involves simply "inking" a circle around notes of current interest as shown in Figure 15. To exclude notes from within a larger circle, the user may draw yet another circle around those not to be included.

The concept of file reading as just another input tool is built upon by including this operation as one of the alternative methods of input, rather than as a separate operation. As shown in Figure 16, the user has just hit the "Input" light button and a list of available input methods has appeared. Note the inclusion of score input along with the various methods outlined in Section 3.5.

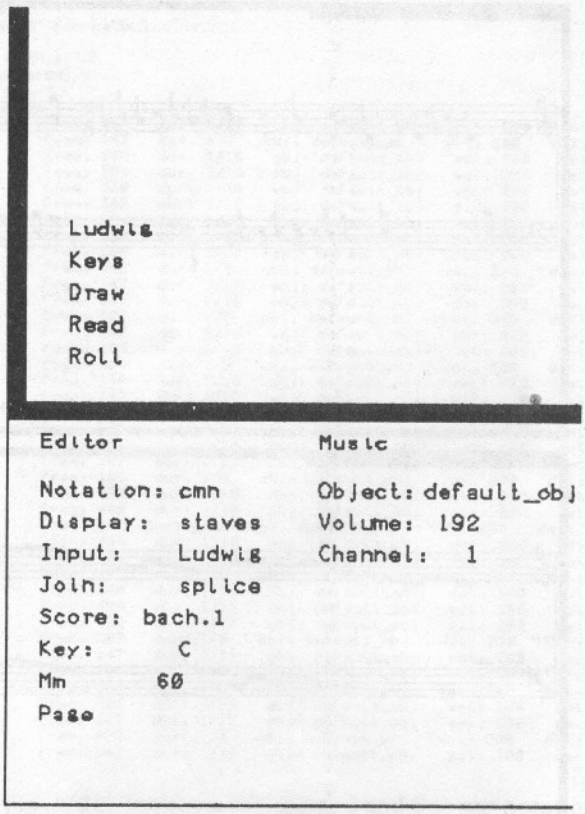

Figure 16. Input options in *scriva*.
Lower left portion of display screen.

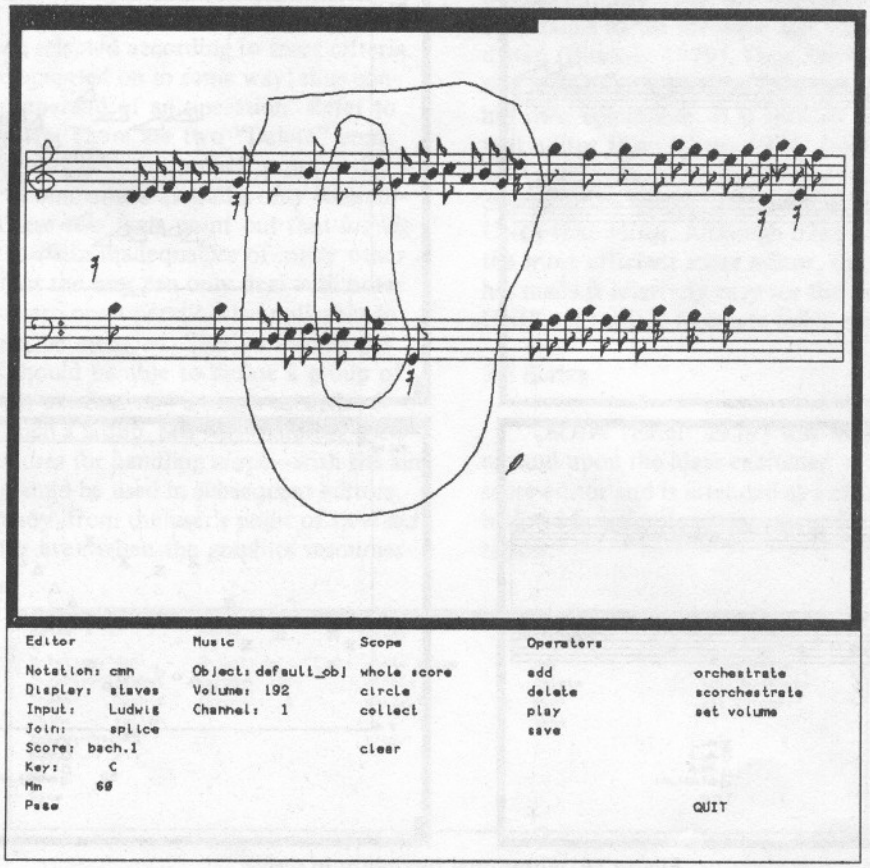

Figure 15. Scope definition by circling in *scriva*.

A final point to note is the logical organization of the command layout as shown in Figure 13. Commands pertaining to the score editor environment, the musical environment, those pertaining to scope definition, and operators which act on scopes have been grouped in separate columns. The purpose is to provide a basic organization to aid the composer in remembering each command's function.

4. Evaluation

The evolution detailed represents a large amount of work accomplished on a minimal budget and with minimal manpower. This fact alone tends to validate the approach to design which has been taken. The other major indication is the success to which the system has been utilized by musicians, for whom it has been designed.

By taking an iterative approach to the design process, we are able to catch flaws before they are irrevocably embedded. By always ensuring the availability of useable software, we have been able to provide the user community with resources from the project's inception. This has been a critical factor in obtaining the involvement of musicians from the beginning and has allowed us to glean results from their work on a continuing basis. By periodically shifting emphasis to the development of tools to aid the design process, we are able to make subsequent work more cost effective.

5. Future Directions

One area of future work is the extension of the system capabilities to handle *tree-structured* scores. Although it was mentioned in Section 3.3 that the underlying data structure is a single linked-list of notes, this is actually a special case of the data structure design. Any single element in the list can in fact be a pointer to yet another score—a sub-score. In this case we are not dealing with a linear list of notes, but a *tree* of notes and sub-scores. The linear restriction has been imposed until now to simplify the issues of graphical display, score navigation and scope definition. Our next step is thus to use the alphanumeric score editor *sed* to isolate and test those transactions essential to sub-score handling.

The problems associated with working with large scores will also be investigated. In many cases these are quite different, and often more complex, than those associated with smaller score fragments. For example, when dealing with a short passage it is often the case that the entire passage can be displayed at once. Thus navigation is rather simple since the user can for example point at the note(s) of interest. However, how does one provide the user with the mechanism to navigate to any portion of the score? How is the time length of the score to be represented visually to aid this navigation? This problem has been touched on already in *ludwig* as shown in Figure 4. Approaches to be considered include the use of rehearsal markings and grids in conjunction with time lines. A related issue is the desirability of having a score scroll as it is being played. A program to assist in "real-time" scrolling, *sview*, has thus been written.

The third area is the continuation of the work on note input tools described in Section 3.5. Methods of evaluating and benchmarking their performance will be investigated. Also an analysis of the protocols required to accomplish the task of note input will perhaps shed light on the design of a more general tool to accomplish this task.

6. Acknowledgments

The research of the SSSP has been made possible by a grant from the Social Sciences and Humanities Research Council of Canada. This support is gratefully acknowledged. In addition, the authors wish to acknowledge the contribution to the research made by the musicians associated with the project as well as colleagues from the Computer Systems Research Group of the University of Toronto. Without their input as "guinea pigs" and "devil's advocates," the work would never have progressed beyond the initial phase. In particular we would like to thank Mark Green, John Hogg, Steve Hume, James Montgomery, and Rob Pike.

7. References

Baecker, R. (1979) "Towards an Effective Characterization of Graphical Interaction," Presented at IFIP W.G. 5.2 Workshop on Methodology of Interaction, Selliac, France.

Baecker, R., Buxton, W., and Reeves, W. (1979) "Towards Facilitating Graphical Interaction: Some Examples from Computer-Aided Musical Composition," *Proceedings of the 6th Man-Communications Conference*, Ottawa, May 1979: 197-207.

Buxton, W. (1978) "Design Issues in the Foundation of a Computer-Based Tool for Music Composition," *Technical Report CSRG-97*, Toronto: University of Toronto.

----- (1979) *Music Software User's Manual*, Toronto: unpublished manuscript, SSSP/CSRG, University of Toronto.

Buxton, W. and Fedorkow, G. (1978) "The Structured Sound Synthesis Project (SSSP): an Introduction," *Technical Report CSRG-92*, Toronto: University of Toronto.

Buxton, W., Fedorkow, G., Baecker, R., Reeves, W., Smith, K. C., Ciamaga, G., and Mezei, L. (1978) "An Overview of the Structured Sound Synthesis Project," *Proceedings of the 1978 International Computer Music Conference*, Roads (Comp.), Evanston: Northwestern University Press, Vol. 2: 471-485.

Buxton, W., Fogels, A., Fedorkow, G., Sasaki, L., and Smith, K. C. (1978) "An Introduction to the SSSP Digital Synthesizer," *Computer Music Journal*, 2.4: 28-38.

Buxton, W., Reeves, W., Baecker, R., and Mezei, L. (1978) "The Use of Hierarchy and Instance in a Data Structure for Computer Music," *Computer Music Journal*, 2.4: 10-20.

Buxton, W., Reeves, W., Patel, S., and O'Dell, T. (1979) *SSSP Programmer's Manual*, Toronto: unpublished manuscript, SSSP/CSRG, University of Toronto.

Hofstetter, F. T. (1975) "Guido: An Interactive Computer-Based System For Improvement of Instruction and Research in Ear-Training," *Journal of Computer-Based Instruction*, 1.4: 100-106.

Hogg, J. and Sniderman, R. (1979) "Score Input Tools Project Report," Toronto: unpublished manuscript, SSSP/CSRG, University of Toronto.

Hume, S. (1978) "Music Score Editor," Toronto: unpublished manuscript, SSSP/CSRG, University of Toronto.

Kernighan, B. W. (1974) "A Tutorial Introduction to the UNIX Text Editor," *Technical Memorandum 74-1273-17*, Murray Hill: Bell Laboratories.

(continued on p. 60)