Department of Computer and Information Science
University of Massachusetts
Amherst, Massachusetts 01003

Integrating Scanning Software: Training and Vocational Uses for the Nonvocal

Stan Kulikowski II *

COINS Technical Report 85-7
March 1985

1

* The work reported in this paper is the result of cooperative effort with Bruce Hawkins of Smith College and David Drake, Scott Conti, and Thomas Gruber here at U.Mass. We were supported in part by grants from the Massachusetts Rehabilitation Commission and the Raytheon Corporation. An earlier version of this paper was presented to the IEEE Third Annual Workshop on Computing and the Handicapped, 8 Nov 1984.

ABSTRACT

Human thought and intelligence is a primary resource in the use of information systems, and computer technology has led in the development of utilities to enhance and amplify these human abilities. Profound physical impairment should not prevent a person from using computer environments; indeed, such facilities have great value to someone whose mind is their primary asset. The ongoing expansion of dedicated personal computers has stimulated the development of scanning systems for prosthetic communication with some of the most disabled members of our society. For several years our working group in computer science (in collaboration with Smith College) has studied first-generation software prosthetic systems and have specified uniform scanning utilities. We interviewed primary service agencies for the nonvocal, formulated a model of distinct user subclasses for information systems, and created a field-testing group for implementation and maintenance of programs we create.

Integrated software refers to the ability of a program or its data to operate with other programs. We here introduce the application of scanners which integrate with standard computer operating systems for training and vocational use with nonvocal people. One program (ready for public distribution) optimizes output into a popular microcomputer environment (the Apple II family) and others which integrate into the mainframe operating system of our computer reseach facilities (the VAX/VMS environment). We shall demonstrate the principles of software integration which minimizes the requirements of specialized hardware.

0.0 INTRODUCTION

Profound physical impairment is one of the extremes in the human condition. Our medical profession has developed techniques for stabilizing the incident condition which produced the impairment in many cases. Whether the impairment is present at birth or acquired through trauma, the medical concerns are foremost during the first year or so, often leaving decades of longterm case management to nonmedical facilities. The prevalence of these conditions are surprisingly high. Experts have estimated from 500,000 to over a million people live in this country who have communication needs which are unexpressed due to physical impairments. While the physical needs of these people are understood well enough to support their continued existence, expression of the intellectual needs of the paralyzed or nearly paralyzed are left unexpressed. We have come to refer to this population as nonvocal, and their situation will continue until direct repair of the central system is possible.

During the last several decades, computer facilities have been developing tools for enhancing human abilities to process large amounts of information. Computer systems have become commonplace in high technology applications, but only recently the emergence of small personal computers have expanded applications into the daily life of millions. As soon as these microcomputers arrived in the public sector, many programmers familiar with biomedical and engineering approaches to nonvocal communication began to realize the potential of these small systems to enhancing the lives of nearly paralyzed people. If computer utlilities could amplify the expression of human intelligence, what better application than to apply these utilities to people who have active minds in bodies which are largely inert?

Today, we are familiar with the first generation of scanning systems written for nonvocal people. Most of these systems have the characteristics of the first-generation of other computer applications. They are originally written for application with a single client; and, since they have not had the advantage of comparing alternative solutions to the task, they often arrive at partial sets of utilities. Some systems will have very desirable features which are completely missing in others. Second-generation scanning systems will soon be available which have the advantage of collecting the best solutions of previous systems into standard sets of utilities based on classes of users rather than individual client cases.

At the University of Massachusetts in collaboration with Smith College, we have for several years been researching the requirements of scanning systems for nonvocal communication with people who are nearly paralyzed. We have reviewed many early first-generation scanners (and written a few ourselves). From this experience we have completed the specifications for such scanning utilities

based on the four subclasses of prosthetic communication users: the congenital cases; the traumatically injured; the temporary class; and the prescriptive specialists. We have formed a case study group based on this model in which we are field testing software and scanning utilities.

In this paper, we are pleased to describe a series of scanning systems developed for vocational and training uses in controlling standard computing utilities. These systems, by their nature, integrate at a low level with machine-dependent routines.

1.0 INTEGRATION OF SCANNING SYSTEMS

The principles of software integration are deceptively simple. A program is called integrated if it operates upon another program or its data. The simplest form of integration is to write several programs which operate upon a common data base. One program is used to create a stored file of data; another manipulates and displays this data; and perhaps a third which formats reports based upon the data. This is the form which common business software takes. A software house will write a spreadsheet to enter numerical information and display in a form commonly used by accountants. Another database program is written to to search and retreive the information from separate storage files, and a third program is written to use word processing and graphics displays to report on the information. We say this task is relatively simple because the teams of programmers within the software houses have complete and detailed source code and data specifications to work with. The end user does not have these and has to use the integrated software as a package. A word processor or graphics utility preferred from one company cannot operate upon the spreadsheet data created by another firm. Sometimes data can be transferred into common form (like files of standard ASCII characters or integers) then these can be processed into another system. These steps are often difficult and sometimes deliberately made impossible by firms who want the user to employ all of their components rather choosing the ones with the most desirable features.

The more difficult form of integration is to require a program to operate with other software that is hardware-compatible, but whose internal specifications are not known when the system is designed.

The one of the tasks we have been researching is scanning systems to replace the standard keyboard input for the most disabled people in our society. Many of the standard computing utilities which have been developed to enhance human intelligence are in their third or fourth generation, while our scanning utilities for the nonvocal are barely completing their first. Clearly, everyone involved in the biomedical applications for prosthetic communication would prefer to have

powerful scanners which integrate with the most uptodate computing utilities. Rather than writing a new word processor which scans, we would rather have a scanner which operates with the best word processors available, and could move to better ones when they emerge. Writing unique programs which are accessible to the nonvocal is often like rediscovering the wheel and tends to be stigmatory. Information skills learned before injury will have to be relearned or working with systems different from those used by nondisabled colleagues can interfere in cooperative tasks.

In this paper we are reporting progress in the development of systems which provide scanning utilities for the nonvocal and which integrate with powerful programs written for normal computer use. We started this work with the cooperation of the Massachusetts Rehabilitation Commission who were interested in providing vocational and training tools for nonvocal clientele. Our primary completed piece is called TalkBASIC (Drake and Kulikowski 1984). It operates in a microcomputer (the Apple II family) which is common in schools. We shall also describe a simpler scanner, called VAXscan (Conti 1984) which operates in our research environment (VAX 11/780 minicomputers) and connects to a more powerful scanner, called SpeakEasy TALK (Gruber, Kulikowski, and Hawkins 1984) written in Bell Laboratories' C programming language (Kernigham and Ritchie 1978). This will be transferrable to a wide range of microcomputer operating systems when sufficient memory becomes available.

Most scanning systems now available to the nonvocal are designed to output text in a natural language– usually English. Alphabet scanners can, of course, be used to produce several languages if the user is competent in them, but alphabet scanning is fundamentally slow. Scanners improve when vocabulary items are available for scanning, but this lowers the expressive power of the devices by limiting the words to a particular language. Even the array of alphabet characters should be optimized for a particular text production task, and this lowers efficiency for production of other tasks. With our interests in integrating scanners with standard computing utilities and with a bias for vocational and computer training, we have focused these scanners on the production of files containing programs which execute in a standard operating environment.

1.1 MICROCOMPUTER SCANNING

In this section we will describe how TalkBASIC worms its way into the DOS 3.3 operating system (Apple Computer Inc. 1984) found in Apple IIs. Its method of operation is to integrate itself with the system's input routines. These usually reside in the Apple's ROM chips, but TalkBASIC is used to take over 16K of expansion memory and make a copy of the ROM logic. Our TalkBASIC program then puts its scanning routines into 4K of available memory within the ROM and then shifts control of the central processor to this scanner-mutated ROM copy in the RAMcard. The illustration Figure 1-1 shows the location of TalkBASIC, hiding like a spider in the RAMcard.

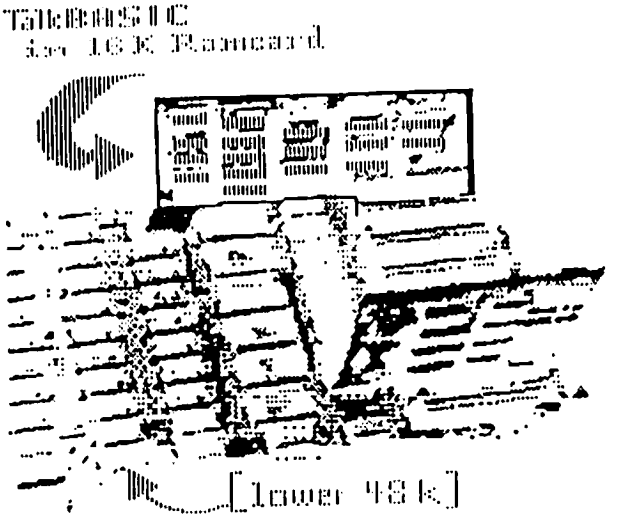

Figure 1-1. TalkBASIC in RAMcard

Once TalkBASIC has inserted its scanning routines into the ROM copy, the lower 48K of the machine operates as usual, except that it now provides scanning interpretation of pushbutton responses from the game I/O.

Figure 1-2 illustrates the TalkBASIC hierarchical structure. The scanning routines start on the MAIN level and the user selects transitions to the other levels. These scanners appear as windows which pull down over the top half of the usual text screen whenever pushbutton input is present.

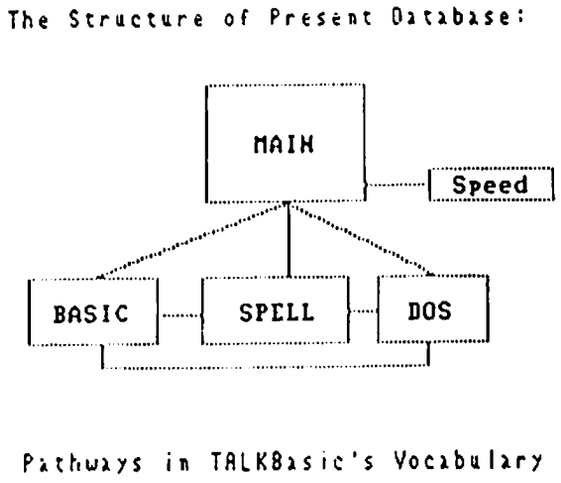

Figure 1-2. TalkBASIC Scanning Network

All of the TalkBASIC scanning windows appear with the name of the present window in upper leftmost position. Items contained with square brackets are nonterminal items: they perform a transition to some part of the system. When terminal items are selected they appear at the DOS]-prompt on the Apple screen. These items are interpreted by the CPU exactly like keyboard input.

The origin of the scanner cursor is the item which appears next to the the window's name item. This is called the home position of the scanner. The primary cycle of the cursor moves vertically for row selection. We have reserved the topmost row for network transitions to other scanner windows and the [FULL] function (which pulls the scanner window up for viewing the entire Apple text screen). The second row is reserved for characters used by all scanning levels: <SPC>, the spacebar character; [UNDO] which removes the last selected item; and <CR>, the carriage return. After the user has selected a row during the primary cursor cycle, the name item of the window becomes part of the secondary cursor cycle for item selection. The name item is a breakout which homes the cursor back to primary cycle if the user has entered the wrong row.

The SPELL screen (illustrated in Figure 1-3) is the heart of the system. We believe that primitive alphabet spellers should be optimized for the language

that is to be produced. The _ETN order is familiar for prosthetic scanners which produce English, but in our application we expect the user to be producing the Applesoft dialect of BASIC within the DOS operating system. We made initial studies of BASIC programs to arrive at the order of the SPELL character arrays. These studies were not exhaustive– only about 40,000 characters of total programming texts– but they did set a nonarbitrary procedure for establishing the order to our primitives.

[SPELL]	[BASIC]	[DOS]	[MAIN]	[FULL]
	<SPC>	[UNDO]	<CR>	->
	E N	O "	: $	) (
	T I	A C	L ,	> <
	0 R	M =	K ;	. !
	1 2	S B	U &	#
	P 3	4 5	6 7	8 9
	– F	+ /	?	[ ]
	H D	* ^X	Y _	'
	G %	<– J	Z	
	W QU	V X	^C	

Figure 1-3. SPELL scanning window.

The BASIC and DOS windows contain frequent vocabulary items used in Applesoft programs and the operating system. Here we arrived at a dilemma because of our dual purposes of vocational versus educational uses of the scanner. The most frequent vocabulary items used by BASIC programmers would establish an optimal order something like PRINT, REM, POKE, PEEK(, and so forth. Our field studies of people just learning the language, however, showed that they preferred these items listed in the familiar ABCD alphabet order. After preliminary rewriting in our laboratory, we were faced with encroaching programming limits since there were only 4K memory available for the scanning routines to fit within the ROM logic. The result was somewhat inelegant– we had to append small lists of additional commands to bottom of the alphabetized items.

```
[BASIC] [SPELL] [DOS]    [MAIN]    [FULL]
        <SPC>   [UNDO]   <CR>      ASC(
        CHR$(   COLOR=   DATA      END
        FLASH   FOR      GET       GOSUB
        GOTO    GR       HCOLOR=   HGR
        HPLOT   HTAB     IF        INPUT
        INVERSE LEFT$    LEN(      MID$(
        NEXT    NORMAL   ON        PEEK(
        PLOT    POKE     PRINT     REM
        RESTORE RETURN   RIGHT$(   RND(
        STEP    THEN     TO        VTAB
```

Figure 1-4. BASIC scanning window.

```
[DOS] [SPELL]   [BASIC]   [MAIN]   [FULL]
      <SPC>     [UNDO]    <CR>     $
      ,D1       ,D2       ,S       ,A
      BLOAD     BRUN      BSAVE    ,L
      CATALOG   DELETE    EXEC     FP
      IN#       INIT      INT      LOAD
      LOCK      MON       MXFILES  NOMON
      OPEN      PR#       READ     RENAME
      RUN       SAVE      UNLOCK   WRITE
      CALL      CONT      DIM      HOME
      LIST      NOTRACE   TEXT     TRACE
```

Figure 1-5. DOS scanning window

An additional scanning window for scrolling-process interrupt will appear if the disabled user responds during Apple output processes, such as the listing of a BASIC program. The [FULL] function removes the window to view the entire text screen. The ^C (computerese for the control C character) will break the process and the ^S will continue the scrolling process. These are standard interrupt characters used by the Apple IIs.

TalkBASIC is typically used to create or execute BASIC programs. One such program we provide in conjunction with TalkBASIC is called CONFIG. This allows the user to set parameters which control the operation of TalkBASIC. If the disabled user is capable of controlling one input switch, the CONFIG program allows the item selection function to operate from any of the three pushbutton inputs in the Apple II game I/O. When more switches are used, other TalkBASIC functions (like homing the cursor or stepping the cursor faster than present scanning speed) can be configured to the additonal pushbuttons. Other parameters are set in CONFIG which establish the default cursor speed for the next time TalkBASIC is used. The user can also inform TalkBASIC if the lower case expansion option is available in the microcomputer.

The most powerful uses of TalkBASIC involve the integration with other programming utilities. The most common form of this use is to run TalkBASIC simultaneously with a BASIC programming environment. In our laboratory and in field testing we typically boot TalkBASIC then load the Applesoft Programmer's Assistant (APA) from the DOS TOOLKIT (Apple Computer Inc. 1979). APA provides standard programming utilities like automatic line numbering, line renumbering, variable crossreferencing tables, and so forth. Accessibility to these utilities have been vital to our disabled users. In this case, the integration pattern of the software is:

TalkBASIC > APA envronment > Applesoft program.

Another powerful use of TalkBASIC is to integrate it with the assembler which was used to assemble TalkBASIC's source code. We used the Big Mac 6502 assembler (Bledon and Waddell 1982). This theoretically gives the disabled user the complete ability to alter any feature of the TalkBASIC scanner. The integration pattern of this use is:

TalkBASIC > Big Mac assembler > TalkBASIC source code.

We believe this is the first prosthetic scanner with ability to alter any detail of its own internal structure. We do not anticipate that many disabled users will make use of this integration because the source code is 6502 assembly language code, and it is very tightly written to get this much utility into the 4K available memory for the scanning routines.

Not all programs will operate with this approach to prosthetic scanning in the Apple IIs. Most programs which require the use of the 16K RAMcard will interfere with the scanner's own code which resides there. Some Applesoft programs read the keyboard directly rather than through the normal INPUT routine, and so our scanner-mutated ROM copy is not accessed in the usual manner. The diskette copying utility, COPYA, found on the DOS 3.3

System Master (Apple Computer Inc. 1980) is an example of direct keyboard reading. There is an easy fix for this which the disabled users can perform themselves from TalkBASIC. It involves changing the appropriate lines of the program to INPUT statements. For the COPYA utility, this required the change of a single line of code to make the program accessible to the disabled. A more serious problem is the use of nonstandard operating systems which publishing houses use in a foolish attempt to prevent pirate copying. Since this approach does not permit entry from the standard DOS operating system, the TalkBASIC scanner is wiped out when the system is booted to enter the commercial programs. As a result, none of the Apple II word processors we have are accessible to our disabled clients in field testing. There are undoubtably good word processors available in the public domain which permit entry from standard DOS, but we have not found the time to search for them. At the end of our public testing (in computerese, this is called beta-maintenance of a program), we intend to put the finalized form of TalkBASIC blasted into chips on its own card which may solve some of the problems encountered with nonstandard operating systems. We have not had time yet to develop TalkBASIC to operate under the ProDOS operating system.

1.2 MINICOMPUTER SCANNING

The future of personal computers lies in today's minicomputers with virtual memory access. Several major companies are presently preparing personal minicomputers for home use. Bell Lab's UNIX operating system (Ritchie and Thompson 1974) is slated to be the system we will all be using within the next decade of high technology progress. In this section we shall briefly describe a couple scanning utilities our laboratory has been preparing in our research environment for the day when much larger computer systems are available for the nonvocal people.

```
   $$$ VAXSCAN $$$                          COMMAND SCREEN
================================================================
 EXECUTE  RUBOUT   COMMAND  PARAMS   SPECIAL  FILES   SPELL  QUIT
================================================================

                HELP          MONITOR       WHOIS         COPY

   VFINGER      DIRECTORY     SPAWN         READ          LOGOUT

   INFORM       SET           SEARCH        CLAIM         SET HOST

   TYPE         SET DEFAULT   PROCESS       CLOSE         APPEND

   ASSIGN       SHOW          WHAT          EOD

   DELETE       SHOW SYSTEM   OPEN          CREATE

   RUN          SHOW NETWORK  WRITE         RENAME

================================================================
```

Figure 1-6. Toplevel VAXscan scanner

The top level scanner for our minicomputer environment is called VAXscan. It is implemented in Digital Command Language (DCL) which is the proprietary system command language for VAX 11/780's. A nonvocal user can use an autodialing modem to enter their system account, and the login procedure initializes the toplevel scanner which accepts any input received as the signal for item selection. The commands you see displayed in Figure 1-6 are the most frequently used system logicals in our research environment. When the commands are constructed from the scanning network, they are executed exactly as if the input were entered through keyboard routines and then control is returned to the scanner.

From the VAXscan utility which is used for system commands, the user can enter a more powerful data-driven scanner we call SpeakEasy TALK. Early versions of this system were implemented in Apple PASCAL (Apple Computer Inc. 1980) for Apple IIs, but microcomputer limitations have slowed the development of these powerful scanners. Data-driven scanners get their name from the fact that all of their vocabulary is contained in data files rather than

written directly into the source code of the scanner. This allows the users to personalize their scanner vocabulary to their individual needs. We believe this kind of scanner utility will become standard in future generations of scanning software as it has in the evolution of most other computer utilities.

```
PROGRAM simple  (output);

BEGIN
   WRITELN;
   WRITELN (' this program writes to the screen ');
   WRITELN;
END.

=================================================================
<Program>  <identifier>  <standard-IO>  <prog body>  <statement>

PROGRAM    <spell>       (output);      BEGIN        BEGIN
                                                       VAR
                                                       WRITELN ('
                                                       WRITELN

=================================================================
*  OOPS!     CONTINUE     MORE      DONE      SPELL    OTHER    RESTART
```

Figure 1-7. TALK scanner with a PASCAL data base.

The output of the TALK scanner is a text file. In the example shown in Figure 1-7, a PASCAL data base (in the bottom half of the illustration) using vocabulary structured for the programming language has been used to create a simple program (shown in the top half). Once this file is created by the data-driven scanner, its execution from the system-level VAXscan is the next task. The programming environments of minicomputers contain many of the most advanced programming utilities and it is our intention to make these available to all users-- especially those who can barely close a single switch.

REFERENCES

D. Drake and S. Kulikowski, *TalkBASIC*, Amherst, MA: Computer and Information Science, University of Massachusetts, 1984.

S. Conti, *VAXscan*, Amherst, MA: Research Computing Facility, University of Massachusetts, 1984.

T. Gruber, S. Kulikowski, and B. Hawkins *The SpeakEasy Communication Prosthesis System*, Amherst, MA: COINS Technical Report 84-22, Computer and Information Science, University of Massachusetts, 1984.

B. Kernighan and D. Ritchie, *The C Programming Language*, Englewood Cliffs, NJ: Prentice-Hall Inc., 1978.

Apple Computer Inc., *DOS 3.3 System Master*, San Cupertino, CA: 1984.

Apple Computer Inc., *The Applesoft Programmer's Assistant*, San Cupertino, CA: 1979.

G. Bledon and D. Waddell, *Big Mac*, Puget Sound, WA: Call A.P.P.L.E., 1982.

D. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Comm. ACM*, vol. 17, no. 7, pp. 365-375, 1974.

Apple Computer Inc., *Apple II PASCAL*, San Cupertino, CA: 1980.