M and Robotics

by Max Rivers, James Poitras, and Pablo Halpern

Abstract

Besides being an excellent database and graphical user interface tool, M has proven itself in the development of commercial and medical robotics. This article describes two commercial projects and one medical robotics project. The first commercial project uses M as a front end to a PASCAL-driven robot; the second commercial product involves direct control of hardware from M. Both of these systems dispense precise amounts of colorants through pumps to generate paint mixtures according to manufacturers' formulas. End users will be hardware store clerks who custom mix paint for contractors and other customers.

In the third project, a medical application, a robotic pill dispenser is connected to a pharmacy prescription package, so that the pills are counted and dispensed, and a label is printed as the prescription is filed into the database.

Introduction

For years, the M programming language has been inaccurately typecast as just a "medical data management" language. M Technology has come a long way from its exclusive use as a medical database language. In the commercial applications described in this article, M is a natural choice because of its power as a database manager. The two commercial systems documented here require searching and updating large databases by unsophisticated end users, so we decided to use a graphical user interface (GUI) for the front end, another application for which M is well suited.

In a matter of weeks we developed a 4GL screen generator (fourth generation programs produce M code from data—in this case screen definitions). These screens are then driven by a generic front end that is independent of the content of the application. This is desirable because each colorant manufacturer designs its formularies differently, and wants to have a unique look and feel to its end product.

As a side note, this front end required about three thousand lines of M code and six weeks of one programmer's time. This replaced forty thousand lines of PASCAL code, which took more than two years to program.

Another interesting side note about these projects is that because the customer base is worldwide, all text displayed had to be language-independent. To accomplish this, all text written to the screen goes through a translator. By setting a software switch (which can be password- or function-key-activated) the entire system instantly translates between languages.

Commercial Application Using Interrupts and TSRs

In the first project, the hardware communicates to the computer through a proprietary circuit board, which requires software written in a compiled language (because of timing issues). The PASCAL driver designed for this purpose took the form of a "terminate and stay resident" (TSR) process. On booting the machine, the TSR loads into RAM (random access memory), with its beginning address stored into the Interrupt Table. Each entry in the table has a number assigned to it, which becomes its interrupt identification. DOS reserves some of the interrupt addresses for things like disk I/O and keyclicks, while others are left open for use by software, such as this TSR. Once loaded, the TSR becomes dormant (but stays ready—with its territory in RAM remaining reserved for its code) so that another process can run—in our case the M system.

When a user presses a key or when there is an internal clock tick (which happens 18.2 times per second), a hardware interrupt is generated, signalling DOS to get the starting address from its Interrupt Table and run the code at that address. That process then executes to completion, at which time control returns to the next instruction on the process that was interrupted. Interrupt modules tend to be coded very compactly so that the interruptions are as short as possible.

In the case of a software interrupt, the software voluntarily interrupts itself, almost like calling a subroutine, except that using an interrupt allows for the call to be programming-language independent.

In the application described here, the M front end interacts with the user, who selects a particular color. M then gets the formula for that color from its database, and writes the colorant amounts to be dispensed into the memory registers of the CPU (central processing unit). It then triggers the TSR's software interrupt. (See code sample in figure 1.) The interrupt activates the TSR, which reads the data from the registers,

```
MOTORINTERRUPT(FORMULA,STROKE,POSITION)
     +1   SET CONTROLIRQ=96
     +2   ;    96 (Hex) is the Interrupt Table entry assigned to this TSR
     +3   SET VELOCITY=113858,ACCELERATION=66976
     +4   ;    Parameters used to drive the motor
     +5   SET AX=32769
     +6   ;    Register AX tells the TSR which function to run, in
     +7   ;    this case the function 32769 means "Motor GoTo
     +8   ;    Absolute Location" stored in variable POSITION which
     +9   ;    is calculated based on the amount of colorant needed
     +10  SET
DATA=$ZLC(POSITION)_$ZLC(VELOCITY)_$ZLC(ACCELERATION)
     +11  ;    $ZLC function converts ASCII into the Long Integer
     +12  ;    format expected by the TSR (Extension to standard M)
     +13  ZC HCOLLECT
     +14  ;    Hold RAM memory stable (Extension to standard M)
     +15  SET PTR=$v(15,DATA,-15)
     +16  ;    PTR is the Pointer address of the variable DATA which is
     +17  ;    a comma delimited string that contains the amounts of
     +18  ;    each colorant to be dispensed
     +19  SET BX=PTR\65536,CX=PTR#65536
     +20  ;    Registers BX=Segment address, CX=Offset address
     +21  SET REGISTERS=$ZWC(AX,BX,CX,0,0,0,0,0,0)
     +22  ;    $ZWC does what $C does, except that it puts Binary values
     +23  ;    into RAM instead of the ASCII equivalents
     +24  ZC #INTCALL(REGISTERS,CONTROLIRQ,0)
     +25  ;    This is the Interrupt! (Extension to standard M)
     +26  ;
RTN       ;What did we get back?
     +1   ;REGISTERS=loAX_hiAX_loBX_hiBX_loCX_hiCX_loDX_hiDX...
     +2   ;    Returns error boolean in loDX (DL), 0=no error, 1=error
     +3   SET ERR=$EXTRACT(REGISTERS,7)
     +4   IF ERR QUIT
     +5   SET PERCENT=$EXTRACT(REGISTERS)
     +6   IF PERCENT=100 QUIT
     +7   ;    loAX has the percent done. When it equals 100,
     +8   ;    the motor has arrived at the ending location
     +9   ;    passed to the TSR in variable POSITION
     +10  DO SHOWPERCENT
     +11  ;    subroutine (not shown) to draw bar graph on screen
     +12  GO RTN
     +13  ;    Continue to loop until error or done
```

Figure 1. Sample (simplified) code to instruct the TSR to move the pump motor.

and performs one discrete part of the requested process at each clock tick: dispensing exact amounts of each colorant. It then puts a number into a register indicating the percentage done, resets the interrupt to go off at the next clock tick and becomes dormant, allowing M and DOS to continue until the next tick. This simulates a multitasking system under DOS. After dispensing all the colorants, the TSR returns either an error or completion code to the appropriate register and turns the interrupt off.

While this processing goes on in the background, M uses the time in between clock ticks to draw a percent-completion bar chart on the screen (based on the information passed through the register from the TSR) to give the user visual feedback about how much of the process is done. When M receives a completion message from the TSR, it passes this on to the end user in graphical form—either as an error message with explanation (obtained from a table) or with a message about successful completion.

This application employs many of the known strengths of the M programming language: strong database functions, powerful user interfacing, and fast code development. But instead of thinking of M in its strictest, and earliest, use as a communicator of medical information between a user and a database (as might have been the case a few years ago), we added a unique third party—a TSR written in PASCAL. In this case, the PASCAL was necessary because the controller board required timing intervals from the software that only a fully compiled language could offer. In the next application to be discussed, this requirement is not present, and M interfaces directly with the hardware.

Commercial Application with Direct Control

In the second project, the board specifications were coded directly in M. All communication to the circuit board is done through a one-byte register on the board. Each of the eight bits signifies one mechanical function of the robot. Position one, for instance, signifies the direction of the pump, either up (1) or down (0); position two turns the pump on (1) or off

(0), etc. In this way it is possible to send single or multiple commands such that, for instance, "00000011" would tell the robot to turn the pump on, moving downward.

The programmer must be careful, however, to preserve the current state of the registers. If the current state is 00001100, then sending 00000011 not only turns positions one and two on, it also turns three and four off. As a safeguard, we use a local variable, SHADOW, to keep a record of the current state. Doing a mathematical "OR" with the current state of the register, adds new commands to the SHADOW state, without turning the existing ones off (00001100 "OR" 00000011 yields 0001111). To turn a particular bit off, we use the mathematical "AND" with a zero value, because only 1 "AND" 1 equal 1. All other combinations yield zero. (See sample code in figure 2 for example.)

It isn't possible to WRITE to the circuit board, as if it were a screen or a printer, because it is not a standard output device. Instead, we used a DataTree MUMPS extension to the M language called ZOUT. ZOUT "writes" out to a DOS-level address. It takes three arguments: the address of the circuit board, the value (in binary) that is being written, and an argument that tells the system if the value is in long (16 bits) or short (8 bits) word form.

This second application, through a few extensions to the M programming language, demonstrates the potential for a whole new arena for M systems. Many implementations of M are now layered over other operating systems. Accessing software (as in the case of the TSR) and hardware (as in this second case) at the operating-system level extends the reach of developers of M far beyond database management and report writing. This "reaching out" into the physical world has application in many fields, including medical, as this next application demonstrates.

Medical Application Using Software Interaction

In this medical robotics application, we were required to connect a hospital pharmacy system to an automated pill dispenser, called a Baker Cell Counter. This particular machine has sixty-four pill containers (eight rows by eight columns), each of which has the ability to count and dispense pills and send them down a chute into a holding place, where they remain until a technician puts a bottle under the chute and lifts a gate, releasing the pills into the bottle.

This particular pharmacy package has the capacity to route the printing of a label to a printer in the pharmacy from any doctor's office in the hospital. This project added the automated dispensing of pills to this process. When a prescription is filed, besides generating a request to print the label, an entry is made in a global queue for the Baker Cell Counter, which contains the medicine's identification and the number of pills to dispense.

This addition to the Baker Cell file triggers a routine that looks to see if that drug is one of the sixty-four currently in the robot. If not, the request prints on a CRT next to the robot, and a technician manually fills the prescription after the label prints out. If the drug is in the Baker Cell's database, the software opens the port connected to the robot, and outputs a command with the cell number (gotten from the database under the drug's identification) and the number of pills, and then waits for a response from the Baker Cell. If the cell counts and dispenses the pills successfully, a record comes back reporting the success. If an error occurs (not enough pills in the hopper, a crunched pill jammed the mechanism, etc.), the record contains a code that, to the best of the machine's ability, describes the problem, and then the software displays an error message on the CRT beside the machine. The message indicates which cell has the problem and what the problem might be. The program then waits for either a "try again" or "cancel" response from the technician.

This unusual M application, even in its customary medical environment, points to new and exciting areas that hospitals and their information systems departments could consider for more fully exploiting the power of M Technology. We have been involved in such far-reaching medical applications in M as speech recognition systems in operating rooms to aid in updating scheduling information in real time. Another project developed a system to telephone patients at home and generate spoken reminders each day to take their medications. The system then waited for a touch-tone telephone response indicating compliance.

With the expanded view of M that these kinds of applications inspire—that is, a system that can reach out into the physical world of the hospital and beyond—it becomes clear that there are many unexplored and untapped possibilities for aiding in the care of patients that extend far beyond medical record keeping.

Conclusion

Fax and copy machines now can come fitted with devices that transmit data about how many pages have been printed, and which client should be billed. Some phone systems come with logic boards that can transmit long-distance costs directly into M billing systems in real time.

```
DISPENSE(action)        ; Instructs board to begin or end pumping of colorant
       +1       ;       The board's firmware recognizes $10 (10 in hex) as dispense
       +2       IF '$DEFINE(PORT) DO INIT
       +3       ;       Initializes addresses and arguments
       +4       IF ACTION=1 DO SETPORT(DISPBIT)
       +5       ;       Turn the first bit on, to begin dispense
       +6       IF ACTION=0 DO CLEARPORT(DISPBIT)
       +7       ;       Turn the first bit off, to end dispense
       +8       QUIT
SETPORT(BITS)   ;Output argument to Port
       +1       ;       Each of the bits represents a different command, so to change
       +2       ;       only one command at a time, it is necessary to "OR" the command
       +3       ;       with the current state of the board (stored in variable SHADOW)
       +4       ; 1111 0001 (Sample of SHADOW's value at the time of the request)
       +5       ; 0000 0010 (Command to begin dispense - value of DISPBIT)
       +6       ; 1111 0011 (Result after "OR")
       +7       SET SHADOW=$ZBITOR(SHADOW,BITS)
       +8       DO OUTPUT(SHADOW,PORT)
       +9       QUIT
CLEARPORT(BITS) ;Clear argument from Port
       +1       ;       To stop the pump, it is necessary to clear the second bit,
       +2       ;       without changing any of the others. So the command 00000010 is
       +3       ;       first reversed (NOT) to 11111101 and then this is combined with
       +4       ;       the current state of the board (stored in SHADOW) using AND:
       +5       ; 1111 0011 (Sample current state in SHADOW)
       +6       ; 1111 1101 (Reverse of DISPBIT command to stop pump)
       +7       ; 1111 0001 (Result turns off the pump because only the second
       +8       ;       position from the right is changed (to zero)
       +9       SET BITS=$ZBITNOT(BITS)
       +10      SET SHADOW=$ZBITAND(SHADOW,BITS)
       +11      DO OUTPUT(SHADOW,PORT)
       +12      QUIT
OUTPUT(VAL,PORT)        ;Actually writes to the robotic board.
       +1       ZOUT PORT:VAL:"B"
       +2       ;       This is the extension to MUMPS that "writes" to the
       +3       ;       circuit board. The 3rd arg "B" tells ZOUT to write
       +4       ;       out a byte, instead of a 2 byte word
       +5       QUIT
INIT             ;Initializes the addresses of the ports
       +1       ;       For the sake of this paper, assume that if the second bit from
       +2       ;       the right of the eight that make up the single byte of
       +3       ;       the robotics board is high (1) then the board will activate the
       +4       ;       pump and colorant will dispense. If it is low (0) then the pump
       +5       ;       will shut off.
       +6       ;       The address of the port (in HEX) is $286,
       +7       ;       which is converted to decimal 646.
       +8       SET PORT=646
       +9       SET DISPBIT=$CHAR(0)_$CHAR(2)
       +10      ;       This sets "00000010" into DISPBIT, the
       +11      ;       value that the board recognizes as the
       +12      ;       "start the pump" command
       +13      SET SHADOW=$CHAR(0)_$CHAR(0)
       +14      ;       This variable keeps track of the
       +15      ;       current binary state of the port
       +16      ;       Initially it is "00000000"
       +17      QUIT
```

Figure 2. Sample (simplified) code to demonstrate the principles involved to output directly to a robotics circuit board.

As a database-management system, M can receive automated information from external machines, or can drive other machines based on its databases, which can vary widely from prescription data to paint formulas. The only limitations are the current state of the hardware and the imaginations of the people that use it.

This article has described commercial and medical robotics projects communicating through DOS-level interrupts, directly addressing circuit boards that drive robots, and interfacing with machines that can physically manage tasks. These are only some examples of how M systems can interact directly with the world, go beyond managing medical data, and reach all the way into the physical world. **M**

Max Rivers is an expert M programmer and the CEO of Rivers M Consulting, which has specialized in unique, difficult or impossible M projects for more than fifteen years.

James W. Poitras is president and CEO of Highland Laboratories, Inc., a manufacturer of paint tinting and mixing equipment, surgical detergent dispensers, and blood donor scales. Previous experience includes sixteen years at Massachusetts General Hospital.

Pablo Halpern is the senior consultant with Halpern-Wight Software, Inc., which provides design programming and quality assurance expertise for the software industry. He has fifteen years' experience with applications and operating systems using C++, C, and PASCAL.