Execution - Is It Always Deadly?

by Winfried Gerum

Winfried Gerum

M is the language of choice for the software professional because many features make it unique among programming systems. Some features are just better than similar features in other computer languages, but others are truly unique. Let's take a close look at some of them.

Few languages are self-referential. M is self-referential in active and passive ways. The passive feature is the $TEXT feature which allows a program to read itself (or other programs). The active features are XECUTE and indirection, which allow M code to execute new M code.

Indirection and XECUTE are two closely related concepts in M. The description in manuals and textbooks looks very simple and straightforward: Both insert data from a database, from user input or from computations on the fly, into the execution flow of a program. It is as simple as birth, and quite as fascinating.

A textbook will tell you that you may replace many, but not all, morsels of code by something created on the fly.

There are four types of indirection: argument, name, pattern, and subscript. If that sounds all Greek to you, you're not alone. It depends on runtime values, whether

```
SET A=B+@C
```

is legal or not. If C yields a name value (e.g., "X", "A4") the expression is valid (*name indirection*). If C evaluates to "A+1" or to "5" the whole thing is invalid! But if you see

```
WRITE @X
```

all of the values "X", "A4", "A+5", "5" are legal (*argument indirection*). What makes the whole thing difficult to grasp, I think, is that indirection cannot insert just any piece of reasonable code into a line of M code. I know it can be easy to implement arbitrary replacements by indirections. But that does not help. The communists' credo has been "Marx is always right." Our guideline is that the MUMPS language standard is always right.

Since not all forms of indirection work, and since a static syntax analysis cannot detect certain flaws in conjunction with indirection, it is potentially dangerous to use indirection. Directors in charge of a large hoard of programs therefore may prohibit the use of indirection (and other nice things in M, h_ll! Why don't they do it in COBOL?).

The concepts of indirection and XECUTE are so extremely powerful that advanced programmers should not do without them.

Simply using XECUTE or indirection does not make a program a professional one. This command, similar to the use of the naked reference, indeed should be strictly forbidden for novices.

Name indirection is simple: Wherever the syntax allows for a name, you may write @expratom to replace the name by an elementary expression yielding a name. But unless the result is interpreted as some other type of indirection, it is not allowed if that expratom yields something other than a name.

Argument indirection means that one or more arguments of a command may be replaced by an expr. Again, do not assume that any kind of argument indirection is legal. Unfortunately, it also is illegal to have an argument indirection evaluate to **no** argument, even when a command may allow for an argumentless syntax.

It is meaningless to have multiple arguments on a QUIT command. Maybe for that reason a previous version of the MUMPS standard did not allow for argument indirection after the QUIT command. Thus "QUIT @X" would be legal only if X was the name of a defined variable. Fortunately, argument indirection on the QUIT command now is permitted. Take a close look at the FOR command. Argument indirection has a very different meaning here:

```
FOR @X DO SOMETHING
```

is always illegal! But

```
FOR @VAR=@FLIST
```

is legal if VAR yields a (possibly subscripted) name and FLIST yields something like "1:1:10" or "1,2,5:10," etc.

Pattern indirection allows us to use @ on the right hand side of the pattern operator.

Index indirection is a really nifty thing. The first part of an array reference is the result of indirection and the rest becomes rather plain. The standard is very hard to read at that point (something like rgnamind ???) and so are textbooks.

It looks something like

```
@expratom@(L expr)
```

Where expratom yields the name of a subscripted or an unsubscripted variable. Then some additional subscripts are appended. Unless you have seen the first fairly good example, you do not have any idea what that is good for.

The XECUTE command executes the value of an expression as a piece of M code. As with almost all commands of the MUMPS language it may have a postconditional on the command. The XECUTE command is one of the few commands that may have a postconditional on the argument (the other such commands are DO and GOTO). Simple as M is, I became aware of the latter fact just five years after starting working with M. Many will warn you that indirection and XECUTE will impair performance and reduce readability of M code.

I have seen many examples that might strongly support such statements. But such examples are like looking at cars crashed in various accidents. Regardless of how terrible an example might be, it does **not** prove that the whole concept should be discarded. Unless you understand the implications of your code completely you should not distribute it to the rest of mankind.

Some problems can hardly be solved without the help of XECUTE and indirection. Some problems may be coded for better readability or maintainability with indirection.

The use of table-driven routines is a powerful technique that relies heavily on indirection and XECUTE.

Let us look at some examples in action.

A very frequent problem is to do a screen mask to enter a set of data: A consistent behavior of the user interface is an absolute must. Therefore it is usually not a good idea to hand code the dialogue. A (simple) description should be given for each field. In our example the definition is part of a routine, but it might as well be stored in a global. The definition then can be interpreted by a relatively small procedure:

```
+1        ;Definition of a screen mask
+2  MENU  ;YPos=1;XPos=1;Prompt=">";Variable="X";Pattern=".E"
+3        ;5;10;"Name           ";NAME;1U1.L
+4        ;6;*;"First Name     ";FSTNAME;1U1.L
+5        ;7;*;"Date of Birth ";DOB;2N1"/"2N1"/"4N
+6        ;9;20;"Phone          ";PHONE;3.N
+7        ;END
+8        ;routine to interpret that definition
+9  ST    NEW ABORT,PREV
+10       DO INIT
+11       NEW DONE,F,I,LINE,LNBR,MDEF,X
+12       SET MDEF=$TEXT(MENU) ;Description of Menu fields
+13       ;init Variables for Menu fields
+14       SET I=0 ;will be field# of Variable
+15       FOR F=2:1:$LENGTH(MDEF,";") DO QUIT:X="" NEW @$PIECE(X,"=") SET @X
+16       .SET X=$PIECE(MDEF,";",F)
+17       .SET:$PIECE(X,"=")="Variable" I=F
+18       I I FOR LNBR=1:1 DO QUIT:LINE?." "1";END" NEW @$PIECE(LINE,";",I)
+19       .SET LINE=$TEXT(MENU+LNBR)
+20       ;F now is the number of fields in a description line
+21 AGAIN SET LNBR=1 ;we start with the first line
+22       SET DONE=0 ;we still expect something to be done
+23       FOR  DO  QUIT:DONE
+24       .SET LINE=$TEXT(MENU+LNBR)
+25       .IF LINE?." "1";END" SET DONE=1 QUIT
+26       .FOR I=2:1:F SET X=$PIECE(LINE,";",I) DO
+27       ..XECUTE:X'="*" "SET "_$PIECE($PIECE(MDEF,";",I),"=")
         _"=X"
+28 REPEAT .DO  GOTO @X
+29       ..WRITE @("/CUP("_YPos_","_XPos_")"),@Prompt
+30       ..READ X
+31       ..IF X=ABORT SET X="ABORT" QUIT
+32       ..IF X=PREV SET X="PREV" QUIT
+33       ..IF X?@Pattern SET @Variable=X,X="NEXT" QUIT
+34       ..SET X="REPEAT"
+35       ..DO ERRMES("Acceptable input matches ?"_Pattern)
+36 NEXT  .SET LNBR=LNBR+1 QUIT  ;continue with next question
+37 ABORT .SET DONE=2 QUIT
+38 PREV  .IF LNBR>1 SET LNBR=LNBR-1 QUIT  ;to previous question
+39       .DO ERRMES("Type "_ABORT_" to abort") QUIT
+40       QUIT:DONE=2  ;there was an abort
+41       GOTO:'$$DONE AGAIN
+42       ;>>>here some code using the input is appropriate<<<
+43       QUIT
+44 INIT  ;General Input tokens:
+45       SET ABORT="^^"
+46       SET PREV="^"
+47       QUIT
+48 DONE  NEW X
+49       FOR  W /CUP(20,35),"Form complete? " READ X DO QUIT:X?1N
+50       .SET X=$TR(X,"yesno","YESNO")
+51       .IF X=""
+52       .ELSE  IF $EXTRACT("YES",1,$LENGTH(X))[X SET X=1
+53       .ELSE  IF $EXTRACT("NO",1,$LENGTH(X))[X SET X=0
+54       .ELSE  DO ERRMES("Please type 'Yes' or 'No'") SET X=""
+55       QUIT X
+56       ;-handle all error messages in the same way
+57 ERRMES(TXT) WRITE /CUP(24),TXT,/EL
+58       QUIT
```

Figure 1. A simple dialogue driver.

The above routine acts as a simple dialogue driver. It uses three out of four possible types of indirection:

- Name indirection line +15 NEW, +18 NEW
- Argument indirection line +15 SET, +28 GOTO, +29 WRITE
- Pattern indirection line +33 IF

Although this routine is fully functional, our purpose here is not to recommend it as a clever way to manage input screens. The nice thing about the code is that it can be amended in two important respects very easily: To add more input fields, just put in more lines in the menu table. To add more functionality, add more columns to the menu table (currently line 3 through 7) and insert code necessary to process those data. And it is easy to adapt the user interface to the latest fads of user interfaces.

Note that in line +27

```
X:X'="*" "S "_$P($P(X,";",I),"
    =")_"="_X
```

cannot be replaced by

```
S:X'="*" @$P($P(X,";",I),"=")=@X
```

in *general*. The latter will work if the right-hand side of the SET evaluates to a name, but it won't work in other cases.

While it is a good idea in general to write short lines of code, it is a special requirement if the code is to be printed. But there are some places where it is not possible to split code: Look at lines +15 and +18, where there is indeed a NEW in a FOR loop. This is done to initialize the variables of the menu table. The NEW cannot be moved into the block following the DO, as this would change the scope of this NEW and thereby render it useless.

As a reasonable example of subscript indirection, figure 2 is a global lister written as a five-line procedure.

Global Lister with $Order

```
GVN = Global variable name, e.g. $NA(^UTILITY)
DIR = Direction: 1 = forward listing, -1 = backward

+1 GLO(GVN,DIR) NEW X SET DIR=+$G(DIR,1) QUIT:-1'[DIR
+2      IF DIR=1 WRITE:$D(@GVN)#2 GVN,!,@GVN,!
+3      SET X="" FOR  SET X=$O(@GVN@(X),DIR) QUIT:X=""  DO
         GLO($NAME(@GVN@(X)),DIR)
+4      IF DIR=-1 WRITE:$D(@GVN)#2 GVN,!,@GVN,!
+5      QUIT
```

Figure 2. A global lister written as a five-line procedure.

It is surprisingly short and one is tempted to bet that it just won't work. The trick is that a global may be viewed as a recursively defined structure, and this procedure recursively maps such a structure. Subscript indirection is a special form of name indirection: The first part (i.e., the name and possibly some subscripts) originates by indirection, the second part comes from straightforward code. The $NAME function converts the syntactic element name to a string. It is the inverse of the name indirection operation.

One also can write a shorter and faster global lister by using $QUERY and name indirection (see figure 3):

Global Lister with $Query

```
GVN = Global variable name, e.g., $NA(^UTILITY)
forward listing only

+1 GLQ(GVN) NEW X SET X=GVN
+2          WRITE:$DATA(@GVN)#2 GVN,!,@GVN,!
+3          SET X=GVN
+4          FOR  SET X=$QUERY(@X) QUIT:X=""  W X,!,@X,!
+5          QUIT
```

Figure 3. A faster global lister.

The GLQ procedure is probably faster, since it does not have to pay for the overhead of recursion. It uses name indirection in several places, but it is not as flexible as the GLO procedure. GLO can move forward and backward. You think there is no need to do backward scanning? I recently had a project where the number and sequence of a dozen subscripts in a global structure had to be user-definable. Furthermore, the routine was supposed to traverse some levels in a forward direction and others in a backward direction at the discretion of the user. A slightly modified version of the GLO procedure did the trick. A frequent occurrence of a backward traversal occurs when the main direction is forward, but a user wants to review previously processed or used data. In this case, direction is reversed depending on user action. Note that in these cases the DIR variable should not be included in the parameter list, as the parameter list does an implicit NEW on each call.

While people tend to emphasize new features, we should not forget about the old ones that made M big in the first place.

Winfried Gerum is with Winner Software GmbH, in Röttenbach, Germany. Contact him by phone at 49-9195-940022 or by fax at 49-9195-940030.