A Successful Alternative Approach to MWAPI

by Luiz Carlos Lobo

Introduction

The M language offers a set of features that distinguish it from other types of programming languages. These features are responsible for its longevity and programmers' loyalty. First is the simplicity: It takes care of the complicated part of programming—such as dealing with files, input-output, and data management—leading to clearer and shorter code. Second is the the lightness (speed): It delivers better performance than languages that provide database control, considering the same platform configuration. The statement "M lets you develop and run applications faster than normal languages" reflects M community expectations, and those should be the most important considerations for implementers whenever new M features are being conceived.

When our development team at Extensão first studied the MS-Windows interface, there was a big challenge to port SuperMUMPS, the company's M language implementation. At issue was the transformation of a very complicated programming environment into something that M programmers could code instinctively. Extensão decided to adhere to the MWAPI specification. As in other M Windows implementations, great portions of the SuperMUMPS for the MS-DOS product code were used to build a dynamic link between the MS-DOS and the Windows interface. A prototype was made available to some of our experienced beta-tester customers. The feedback we got from them was very loud and clear: "Is this the evolution of M? Too complicated! We are seriously considering Visual Basic!"

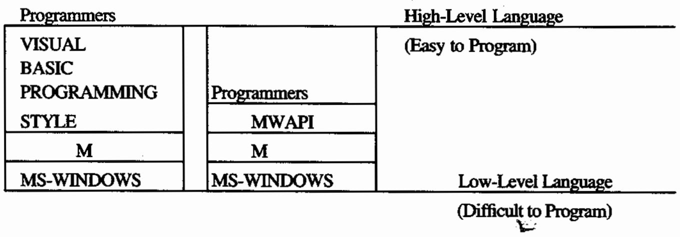
This was not the response the company expected. The reaction at Extensão went something like this: "Well, let's review the project and plan it all again." Two things came to our minds. First, that Microsoft had brought BASIC back to life when it created a great hybrid *event-* and *nonevent*-driven language. Second, that by using the MWAPI programming style, we could generate lots of work for programmers and affect M environment performance. These ideas were completely the opposite of the basic guidelines for M mentioned above. We knew we were headed in the wrong direction!

This article introduces the alternative that we ultimately developed. The article does not present an in-depth comparison between the alternative and MWAPI.

If BASIC Got There, Why Not M?

We decided to implement the Visual Basic programming style in the M language. The only way to do that was to rewrite the code from scratch, but this time we did it in strict conformance to MS-Windows architecture and treating it as a new operating system, not as an MS-DOS extension. This allowed us to use all the power Windows can offer at a high-performance level.

High- or Low-Level Language?



Extensão also had expected an explosive growth of the Windows and Windows NT market in Brazil. We knew that this new approach would demand extra effort from the company and that it could take too long to offer the product. We introduced this product as Visual M to the U.S. market at the 1994 MTA-NA Annual Meeting in Reno. Actually, the product is named VISUAL TECH, due to copyright misunderstandings.

All the effort has been worth it. We showed a beta version of VISUAL TECH at FENASOFT in 1993, a huge Brazilian COMDEX-type show. At that time, VISUAL TECH already had embedded SQL connected to Sybase and Informix DB servers running under UNIX. It was a success! The language proved itself very easy to code and faster than we ourselves could ever imagine. The language was designed for a 386@40 Mhz PC with as little as 4 megabytes of memory. Eighty percent of the Brazilian MS-Windows installed base platform configuration matches this.

But why is this Visual Basic approach faster and easier to program than MWAPI? It is faster because it is easy to code. How does it work? The whole program is based on one command and three functions. MS-Windows has objects such as buttons, list boxes, and radio buttons. Each of these objects can receive events such as a click, a double-click, a mouse down, a key-up, etc.

How can you create a Windows form? One needs only to open a window with a single command (`ZGW Open`—Graphical Windows Open) and then insert objects (`ZGW Insert`) with their properties. (Properties refer to objects' characteristics such as identification number, foreground color, position, size, visible state, font name, size, etc.)

A routine will be ready to show the form on the screen, but there is no code associated with events at this time. Actually, the M routine will be receiving dozens of different events per second, but that is not the user's problem; it's VISUAL TECH's problem to solve. (Remember a primary characteristic of M: it takes control of the boring part of the task). The user selects the event to intercept by coding a label with its name. VISUAL TECH looks through the code labels to check if the user wants to grab the event, otherwise VISUAL TECH itself takes care of it by passing it back, treated, to MS-Windows.

So, if a programmer wants a picture to receive a click on it, he or she just has to insert a label "CLICK" concatenated with that object's ID number (e.g., label `CLICK7`) followed by the M code the programmer wants to execute for that click. A `QUIT` command will return the routine to the wait event mode automatically. The same windows command with a different argument closes the form (`ZGW close`). The programmer can accomplish all the tasks needed to build powerful and fast windows applications in minutes, with all the MS-Windows capabilities, including 100 percent multimedia control.

The sample code in figure 1 shows how a Windows form lists employees with their photographs and changes as the user selects a different employee name. Notice in the example that each object has one identification number, and when an event occurs (i.e., the user clicks the "OK" button) it generates a

```
hdemol     ;   Employee Selection Screen
           ;
+1   KILL
+2   SET v(1)="John Smith"
+3   SET v(2)="Henry Marshall"
+4   SET v(3)="Paul Stone"
+5   SET v(4)="Blair Karlson"
+6   SET v(5)="Mary Hellen"
+7   SET v(6)="Hellen Boyer"
+8   ZGWINDOW open(callback="hdemol":id=1):Captiont-"Employee Selection Screen":posx=21:posy=27:width=364:
+9   height=238:backcolor=16777215:model=1)
+10  QUIT
     ;
CREATE1    ;    Object list
+1         ZGWINDOW insert(type="PUSH":id=1:parent=1:caption="OK":
           posx=253:posy=173:width=93:height=27:fontname="System":
           fontsize=10:align=0:tabstop=1)
+2         ZGWINDOW insert(type="PUSH":id=2:parent=1:caption="Cancel":
           posx-147:posy=173:width=93:height=27:fontname="System":
           fontsize=10:align=0:tabstop=1)
+3         ZGWINDOW insert(type="LIST":id=3:parent=1:captiont-"":
           posx=13:posy=13:width=187:height=146:fontname="System":
           fontsize=10:align=0:tabstop=1)
+4         ZGWINDOW insert(type="PICTURE":id=4:parent=1:captiont="":picture="dp2.bmp":posx=213:posy=13:width
           =133:height=136:
           fontname="System":fontsize=10:align=0:tabstop=1)
           for i=1:1:6 SET x=$ZGL(addlist,1,3,v(i))
+5         SET x=$ZGL(selitem,1,3,0)
+6         ZGWINDOW focus(parent=1:id=3)
+7         QUIT
           ;    Here the user selected another employee name
CHANGE3    SET x=$ZGL (getcursel,1,3)+1
+1         SET $ZGWINDOW(picture,1,4)=x_".bmp"
+2         QUIT
           ;    click the button OK
COMMAND1   SET sel=v($ZGL(GETcursel,1,3)+1)
+1         GO END
+2         ;   click on the button CANCEL
COMMAND2   SET sel=""
+1         GO END
           ;
END        ZGWINDOW close(id=1)
+1         QUIT
           ;
```

Figure 1. Sample code for employee identification.

COMMAND event followed by the identification number of that object (e.g., COMMAND1). VISUAL TECH branches to the corresponding label and executes the code until it reaches a QUIT command. Figure 2 shows the results of the code.

Why is it faster? First, because it is specifically developed for MS-Windows rather than using adapted code, it allows the language to take advantage of all MS-Windows capabilities to make the code faster. Also, this version includes new algorithms that speeded it up. Third, the MWAPI disk and memory-intensive transferring architecture dramatically slow down an application running under the MS-Windows.

To understand the difference between VISUAL TECH and MWAPI internal architecture, let's examine how an M object routine is executed in each approach.

With the MWAPI approach, during the compiling, the programmer completes the code as a normal routine. No MS-Windows optimization is done. To implement the program, the programmer requests it to run a routine. Then, the language accesses the disk and brings the object code into memory. Third, the language executes the code that launches a MERGE command. Next, the MERGE command searches the disk for defined globals and, after many disk accesses, brings data to the global buffers. Then, the language converts these data at run time to the MS-Windows internal-memory-structure format.

Next, the language tells Windows to look up these transformed data to open a Windows form. The programmer must manipulate the Windows events inside the M routine, making additional use of the CPU.

Contrast these several steps with the VISUAL TECH approach, which entails fewer steps. While compiling the routine, all needed data are present already. The language converts the data to the MS-Windows internal-memory-structure format, and stores them in the object code. Everything takes place at compiling time in just one step. To run the program, the programmer requests a routine. Second, the language accesses the disk and brings the object code into memory. Next, the language executes the code and tells Windows to look up the data prepared at compiling time to open a Windows form. Fourth, the language takes care of the Windows event for the programmers.

Conclusions

As demonstrated here, the basic concepts of the two approaches are very different. We don't dislike MWAPI at all. In fact, its good portability motivates us to offer it soon as a second optional way to access Windows from VISUAL TECH. *But how high is the price of MWAPI for the M world?* In our opinion, it is dangerous to offer such a slow (heavy) performance and complex programming style solution without checking out some other good alternatives on the market. To depend upon a configuration of 8 or 16 megabytes of memory on a 486 or Pentium machine to obtain a questionable performance doesn't seem to be a very attractive option for the customer. With all my love for M, I would still consider the choice between Visual Basic and MWAPI. The VISUAL TECH approach proves to be very competitive because it offers a faster performance (lightness) and an easy programming style (similar to Visual Basic). In Brazil, VISUAL TECH has been used extensively as the M solution for Windows. Nowadays, programmers who used Visual Basic and changed to VISUAL TECH say they find its environment to be more flexible and faster than the best-seller Visual Basic.

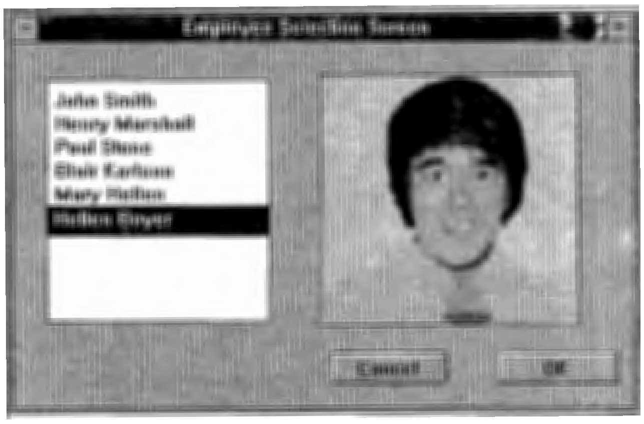


Figure 2. The employee selection screen that results from the code.

Is M for Windows competitive? You bet! **M**

Luiz Carlos Lobo is an electronic engineer and earned an M.B.A. He co-founded Extensão eight years ago. He also is former director of MTA-Brazil. He is presently offering Extensão's M languages in the U.S. market through X-TENSION Software Corporation.

If you'd like to write for *M Computing* before the close of this editorial year, there are two remaining opportunities to do so. Remember that all submissions are subject to editing. Send your ideas to the managing editor, MTA, phone: 301-431-4070; fax: 301-431-0017, or e-mail: MTA1994@aol.com.

September 1995
(publication date)
June 30, 1995 (deadline)
Tools for Development

NEW TOPIC FOR FEBRUARY 1996
November 17, 1995
Banking and Finance

November 1995
(publication date)
August 17, 1995 (deadline)
Object-Oriented Systems