Changes in the World and Working Environment of the M Programmer

by Kate Schell

The world of the M programmer is changing. It is changing quickly, and on multiple fronts. The past ten years have seen serious acceleration in user sophistication and hardware capabilities. The cost of disk space continues to plummet, the size of personal-computer (PC) networks continues to grow. Some hardware capabilities have brought about new system architecture requirements. Coding for a graphical user interface (GUI) or client-server system requires an isolation of database-access code from user-interface code. The M standard, which is turning into a suite of standards, continues to evolve thanks to the work of the MUMPS Development Committee (MDC) and contributions from the M community. The computer industry continues to discuss reusable code, but the favorite model is now object-oriented, not just structured. Finally, there are tools and systems available to the M programmer today that were not around ten years ago or even four years ago. These tools can help build faster, nicer-looking systems; access data in legacy databases; build reports that once had to be coded by a programmer; or assist in the maintenance, enhancement, and documentation of established M systems.

The Computer User

The profile of the average computer user has changed seriously in the past decade. Once most people accessed a mainframe or minicomputer from a terminal on their desk, or in a central computer lab. Output went onto noisy printers loaded with zebra paper in a remote room. Word processors were stand-alone, dedicated machines. Spreadsheets were just beginning to come into use. Computer games were frowned upon not because they wasted worker time, but because they squandered precious CPU (central processing unit) cycles and connect time. Computers were, for the most part, largish machines sequestered in air-conditioned rooms. Now, most common household appliances and motor vehicles contain computers. Bank tellers have been replaced by computer terminals. Computers are used to ensure timely delivery of pizza, as well as to help pilot space shuttles. Today many people have personal computers on their desks at work; a substantial number of those people also have computers in their homes. They use IBM PC clones, or Macintoshes, with modems, sound blasters, CD ROM, and optical scanners. In short, the machines have become ubiquitous.

Many computer users have taken the time to learn how to install hardware themselves. Their children study computer architecture in school, and spend their afternoons with a little machine known as Sega. Today's computer users run word-processing packages, spreadsheets, database management systems, games, and communications packages that help them navigate the Internet, manage e-mail, send and receive facsimiles, effect banking transactions, and submit tax returns. These systems come with a GUI, hypertext help, and documentation that is colorful, well organized, nicely bound, and professionally indexed. The typical PC software package costs between $50 and $400 for a single user. People expect to be able to sit down at a new application and get through it without extensive training. The more complex systems come with on-line tutorials built into them. The most recent user aid is the "wizard," an interactive tool that directs a user's work while that person does real, constructive work, such as building a complex document.

Users of computers have become more familiar with the capabilities of computer systems, and in the process, they have become more demanding. Programmers are expected to know about the aesthetics of screen layout as well as the flawless algorithm. We are expected to guide the user of a computer system in successful use of that system: listing only menu choices available to the user, displaying lists of possibilities, or making them available through help. The "look and feel" is supposed to be familiar, the chances of success are supposed to be high.

The GUI has become the norm for most of the software available on PCs. Vendors of systems written in M will tell you that they have a very difficult time selling systems without GUIs, even though a GUI-based system requires a larger investment in hardware. People who are computer knowledgeable look askance at "dumb" or "graphically challenged" terminals. They want to be able to call up their M system while they are using a word processor or spreadsheet, and then be able to return seamlessly to the M system; in some cases, they want to be able to extract data from one system and paste it into another. The progress that has been made on GUI development has been critical to continued sales of M systems.

Developments in the M Standard

The M standard continues to evolve. This year a contract with a customer who was still using M/11 required me to go back to coding without extrinsic functions or parameter passing. I found that I could still do the work I needed to do, but it was much harder than it should have been. All in all, I'm glad that the 1990 M standard supplanted the previous standard. There is a new M language standard currently in canvass that will alter seriously the way in which we handle errors, retrieve and file data, deal with character sets, and communicate with non-M systems. The MERGE command copies an entire global or global subtree, eliminating the need for those "tree walk" subroutines we all learned to write. SSVNs (structured system variables) will make more of the M system information available to programmers in a standard fashion. In addition to the language standard, we have a new suite of proposed standards dealing with data access between M systems called the Open M[UMPS] Interconnect (OMI), a platform-independent M Windowing Application Program Interface (MWAPI), and bindings to the Graphics Kernel System and X. Object-oriented extensions, and new programming and data structures, as well as new networking protocols and revisions to the MWAPI, are under discussion.

Changes Due to New Architecture Requirements

Among the largest changes taking place in the world of computing are those in the physical scale of systems, with attendant opportunities for changes in architecture. Networks of thousands of PCs were a dream ten years ago. Today they are a reality that has changed the focus of computing. A computer system based upon one large computer (or two, or three, or four of them) was relatively easy to control. A system manager and staff knew every piece of software on the system. They were responsible for loading it, received a set of manuals, sent staff members for training in the management and usage of the system, and tightly controlled the resources allocated to it. "End-User Computing" departments were unnecessary: The end users' access to computing was controlled by the systems staff. The advent of personal computing put an end to the despotic control wielded by the system manager. The wars between Macintosh and PC, Word Perfect and MS Word, Lotus 1-2-3 and Excel, and others rage at almost every institution. No matter which software package or hardware system is installed and licensed as being the "official" and therefore a supported system, someone in the institution is probably using a competitor, even though that tool is unsupported.

On the other hand, those who are willing to use unsupported systems often have good reason for doing so, and are familiar enough with the requirements of their systems to be able to fix problems on their own, or to find someone inside or outside of the organization who will help. This independence and willingness to employ the best tool for the job creates many opportunities for small M systems.

The development of the PC, networks, and the client-server systems has led to remarkable changes in system size and cost. It is now possible to build the processing and data storage equivalent of an IBM 370 (one of the high-end machines a decade ago) using a collection of inexpensive PCs. Upgrades, when necessary, can be handled piecemeal, thereby reducing system costs, and enabling them to be budgeted across several fiscal years.

As Dr. Frederick Hiltz pointed out in the November 1994 issue of *M Computing*, client-server architecture gives rise to a modification in programming style. Routines are designed to run on a specific area of the system. There are server routines that store and retrieve data, and client routines that invoke them. The server routines are designed to return a very specific result.

Speaking of results, the results of the migration to client-server have been multiple. The cost of system hardware has been cut. The selection of products available to computer users has expanded. The power broker in the system area is now the network manager. The plethora of systems required to handle a modern-day job makes it important that a user be able to switch between systems and applications quickly. An inquiry about a record in the database may interrupt the preparation of a memo. Better yet, users would like to be able to capture information from an application and paste it into the memo in question. Users expect to switch into and out of M systems as quickly and easily as they would bring up any PC-based application, such as a DB2 database.

Tools

The number of productivity tools available to M developers continues to increase. Some of those tools are devices, such as multisession terminals, workstations, or PCs with terminal emulation. I know that there is still a large number of M programmers working at so-called dumb terminals. I wonder if their managers lack imagination, or if M shops are just as "retro" as some of our programming colleagues think we are. Poor hardware in front of an expensive resource like a programmer is a really shortsighted approach to cost containment. There are a number of software tools available to speed up development. Some of these tools are purely M, others are hybrids.

In the area of building applications, screen and GUI generators can be used to create the user interface. Case tools can help with database modeling. There are reporting tools that can move report generation from a programmer to a skilled analyst, or even out to expert end users. There are new types of tools appearing that enable M shops to control their code more closely.

Version control and reverse-engineering tools are among the most recent additions to the M programmers' toolkit. Tracking of development efforts in the past decade revealed that programmers spend the vast majority of their time creating user-interface code. Prompting for data and validating input is one of the most labor-intensive parts of the job. Screen generators have been around for more than a decade, but there are a lot of shops that don't utilize a screen-building tool. In many M shops, each programmer is allowed to create a unique user-interface style. You can tell who programmed a module without looking for initials in the routines. Products fashioned this way are difficult to maintain and enhance. They can also make a user's life miserable. Consistency of look and feel is invaluable when considering the amount of training time and documentation saved. How many M shops have a tool for building user interfaces, let alone a style sheet that describes formatting of menus, screen design, help text design, and protocols for moving forward or backward in the application? In the GUI-1 generation area, M developers have a choice of MWAPI-based tools, or interfaces to non-M products such as Visual Basic and PowerBuilder.

M programmers who use the technology need to learn the basics of GUI screen design, as well as how to link the resulting screens to the file servers. GUI-based input requires an understanding of events, and has to allow the user to move between fields in an arbitrary manner.

Validation that used to be executed at the end of entry into a field may now have to wait until the event (such as a mouse click on the Save option in the File menu) that signals that the user is done entering data, and is ready to have it filed. In many ways, this feels like a step back. We had better control with "roll and scroll" because users weren't allowed to fill in one field until the previous fields had been entered. The linear nature of the user interaction allowed more control.

Managing data entry, in particular, in an event-driven environment requires skill and care on the part of the programmer. CASE (computer-aided systems engineering) tools can help with the modeling of databases. These tools allow selection of a database description methodology, and then enforce the rules of that methodology as a programmer or database analyst/designer describes data fields and their location. They have nice reporting features that enable production of database diagrams with graphic representations of relationships. Even without using a relational database technology, these tools can teach programmers the rules and vocabulary of the relational game, and some have hierarchical models available as well. (If you have to play on the relational field, you may as well use those tools.)

New products are coming to market that translate a CASE schema into a database definition usable by an M database product. In the process of creating an application, development shops with more than one programmer eventually run into conflicts over versions of routines. Somebody edits a routine and files it; ten minutes later, another programmer files the same routine, wiping out all record of the modifications made by the first programmer. The more programmers working in one area, the more likely it is that a shop will implement some form of version control. Programmers look on version control as handcuffs. In fact, version control is more like a condom: What is lost in sensitivity is gained in protection!

Some of the M tools on the market create data dictionaries that make M globals accessible to SQL inquiries. Open DataBase Connection (ODBC) access for reporting tools and selection of data from other systems is becoming a requirement for more and more systems. Users like the ability to access the data without programmer intervention. The down side of providing ODBC access is controlling it. I have experienced systems with more than fifty SQL queries running against a poorly indexed primary data file. System performance was poor, and nobody had a methodology (increase system size, limit the number of ad hoc queries, etc.) for bringing the problem under control.

The other serious issue is making sure that users understand the implications of using software that performs SQL-style searches on a database. M reports don't usually give Cartesian products of fields that have nothing in common.

Some of the most intriguing tools currently available are those that give a new perspective on existing systems. Older systems, often called "legacy" systems, frequently suffer from lack of documentation, both for program flow and for the data collected and saved into globals. RE/m, a "reverse engineering tool" for the M marketplace, visually represents routine-calling sequences and global layouts. Item ploys a static parse of the routine code, then stores the information gleaned about routines and globals in a repository. It can also produce charts indicating which routines create, update, access, or delete global data. The tool can be turned to purposes other than the visual displays that come with the system. The parsing engine runs off of a parsing tree stored in an M global. Users of the system can modify the parser to collect data, or even to rebuild code in certain ways.

Quality Training At A Fraction Of The Cost!

ESI offers a software training alternative. Computer Based Instruction is much less expensive than course fees, and eliminates travel time and expense. Perfect for individuals or groups... *the more they're used, the more you save.*

CBI's available on floppy disk and magtape:

■ Introduction to M	■ Object-Oriented Concepts
■ Intermediate M Programming	■ DSM System Management
■ Advanced M Programming	■ VMS Concepts
■ Introduction to File Manager	■ DCL Programming
■ Intermediate File Manager	■ MSM System Management
■ Advanced File Manager	■ DataTree M System Mgmt.
■ KB_SQL	■ M/SQL

NEW MWAPI CBI NOW AVAILABLE!

All trademarks are the property of the respective owners.

ESI TECHNOLOGY CORP.

Call ESI for more Details!
5 Commonwealth Road • Natick, MA 01760
Tel: (508) 651-1400 • Fax: (508) 651-0708

DOUBLE YOUR PRODUCTIVITY!

Our **MEdit™** full-screen routine editor and customizable **MShell™** toolkit will cut your development time, and make multi-platform development a snap!

We also offer expert consulting services for system management, custom software, health care, and much more!

Call 1-800-370-1935

McIntyre Consulting, Inc.
336 Baker Ave., Concord, MA 01742
(508) 371-1935 Fax: (508) 369-6693
Email: msm@mcinc.com

Another reverse engineering tool, RE/data, enables creation of a data dictionary for M systems that haven't ever had a data dictionary. This approach to older M systems can help the SQL access that is in demand today. A parser similar to the one used by RE/m examines code and identifies data elements. Identification is based upon the creation and updating of data in globals, and on data manipulation within the system. For example, RE/data will identify the year digits from a date as a separate data element if the year is manipulated or displayed as a unique element. It can isolate data that is manipulated using, for example, a $EXTRACT on a piece from a particular global node. Once the parsing is done, the results of the parse can be compared with existing information on database contents for verification.

Summary

Undertaking an overview of the challenges facing the M marketplace, and a quick look at some of the new tools and techniques available for handling these challenges, strikes me as a classic exercise in dragon-slaying. Most M shops use primarily the tools that they have developed in house. Tools vendors tell me that creating tools for the M marketplace hasn't been a financially rewarding experience. M programmers may see the value of productivity tools, but are unable to persuade managers to acquire them. For managers in shops without productivity tools: You are missing out on a chance to have systems that are more useful and acceptable to your users. You are also missing out on a major draw for new staff. Talented programmers select positions in which they will learn and grow. Offering the opportunity to learn new skills that enhance their productivity will bring in the type of motivated people who make new systems happen more quickly. If you're a programmer in a shop that could use some new tools, consider taking the initiative. You will be more marketable with current skills. Get out and take a course on your own. The MTA tutorials are a good buy. Colleges and universities have courses in design that will help you understand the issues behind GUI development.

This article is not intended to create a market for tools. This article was conceived as a quick summary about working smart, as well as working hard. M

Kate Schell is the founder of C Schell Systems, an M consulting firm in the Boston area. She chairs a subcommittee of the MUMPS Development Committee and is on the Review Board of *M Computing*. Her e-mail address is cschell@world.std.com.