Multiuser and Multidevice M Training Techniques

by Valerie J. Harvey, Ph.D. and Lynne R. Cuda, M.C.S.

Multiuser Learning and Dynamic Interaction

Although information systems served by M Technology are typically multiuser, multidevice installations, most instructional techniques for those learning M are oriented toward a single-user experience. There are a number of properties of M Technology which are difficult to demonstrate in a single-user learning situation, such as locking, the shared nature of global arrays, traversal of database structures under conditions of active database change, control of multiple devices, and multitasking. The techniques in this tutorial have been used successfully in training personnel with little or no background in M.

Using this approach, the learner spends time in dynamic interaction with a multiuser environment from the very beginning. This interaction helps the learner develop a clear sense of the multiuser environment and thus apply and serve the M Technology system more effectively. Suggested exercises and lessons deal with shared data, concurrency control, application design, background jobs, device control, distributed processing, and the practical requirements of teamwork in information technology.

This approach can be particularly effective in re-training programmers who have experience with other programming languages, since it emphasizes some properties of the M environment that are valuable and convenient to use (See Chapter 19 of Walters, sections on features of M present in a few, but not many, other high-level languages and features rarely found in other programming languages). Practices similar to particular techniques presented in this article were incorporated into certain M-user organization employee training programs years ago, but were not combined into an overall multiuser, multitasking instructional strategy. For example, Kent Frazer's design for Rubicon employee training in the early 1980s included a final exercise in which all the trainees worked online together to produce a single application. References in this paper are keyed to the major resources currently available and published textbooks up-to-date with at least the 1995 M standard.

Sandwiching - Two Modes of Learning at Once with Multiple Threads

The technique of "sandwiching" (from the instructional viewpoint) is proposed here—addressing multiple threads of objectives simultaneously. For example, while particular exercises and structures or syntax are being treated explicitly, the environments are being used to provide learning about multiuser environments (concurrency control), alternative models of multiuser system implementations, device and network properties, teamwork in development, testing, and implementations, etc.

The experiential learning is accomplished by inserting, at intervals and in conjunction with the activities planned, explanations of what to observe or what should have been observed and the reason for the importance of this experience. Thus the learner experience seems natural, flows well, benefits from repetition, and does not seem more stressful, demanding, or compressed than would be required for a single-thread presentation mode. Simple requirements of advanced topics are introduced at the earliest possible point. For example, a number of simple exercises are designed to terminate by checking a particular global for a value, rather than by prompting the user for input. Thus learners gain familiarity with a control strategy needed for use of background jobs. When background jobs are used, they will already know how to assign global values in order to terminate or otherwise control a background job. They will also have examples of how `$GET` can be used to check for a value while using a reference to a global variable that might not be defined. This manner of terminating routines also encourages use of the two-workstation setup, since the "message" to stop iteration can be "sent" from one station while the routine is executing on the other.

Exercise and demonstration routines provide the learn-

ers with examples of certain kinds of M programming, ranging from use of the $ORDER function to variation of natural language output. Demonstration routines are purposely kept as simple as possible to invite easy inspection to see how syntax is used and how the routines work. Learners should also be encouraged to use the $DATA function to check for the existence of variables in appropriate circumstances (For a good treatment of the $DATA function, see Gerum's article). This action can be prompted by questions from the instructor like, "Do we know whether the variable ^A has been defined?" A learner using one station can check a global variable in use by a partner at the companion station.

Implementation

This approach to learning M requires simultaneous access to two complete workstations (two keyboard /screen combinations). Implementation can be carried out in various ways:

- A single PC with dumb terminal on serial port
- Two PCs connected serially, one serving as a terminal
- A client/server system
- Two workstations (terminals) on a multiuser system

There should be access to at least two UCIs (User Class Identifiers or M Directories) on each console or terminal. Certain of the exercises commonly used for the multiuser approach can be simulated using a single screen, especially in a Windows environment, but the impact seems to be greatest with simultaneous use of two keyboards for most exercises described here.

Single-UCI and Multiple-UCI Models

Elementary objectives can be handled effectively while using only a single UCI, while more advanced learning situations can exploit access to multiple UCIs. The following categories of exercises can be addressed with access to the same UCI on both screens:

• Multiuser access to globals

In order to demonstrate the shared property of global variables, a single global variable, such as ^VALUE, could receive values on one workstation while a simple loop displays the current value of ^VALUE on the other screen. Such informal practice might involve execution of the following:

```
(Console)FOR WRITE !,"ENTER VALUE: " READ
^VALUE
(Terminal)FOR WRITE ^VALUE," "
```

After the entry loop has been run a few times, and while the display loop continues to run, the learner could be asked to interrupt the loop and kill ^VALUE. The display loop will, of course, be interrupted with a sudden "undefined" error. Experienced programmers, accustomed to dimensioned, dense arrays in other languages, benefit from practice with manipulation and traversal of subscript sequences in M arrays. Several properties of arrays and their associated traversal functions need attention: the array as a sparse matrix representation, hierarchical structure, undeclared bounding under dynamic conditions, treatment of the set of subscripts at a given level in an array as a linked list, the standard subscript collating sequence, the use of <null> ("" or null-string) as a conventional header pointer for the dynamic linked list representation and how to express design of a global array for M database purposes. A series of demonstration routines is made available to learners for open-ended practice as follows.

(1) The simplest routine LOWHIGH checks the bounds of subscripts at the first subscripting level of global ^Z through the expressions $ORDER(^Z("")) and $ORDER(^Z(""),-1). Possible example responses are:

(a) with no subscripts:

```
"THE LOWEST SUBSCRIPT IN ^Z IS: ARRAY EMPTY"
"THE HIGHEST SUBSCRIPT IN ^Z IS: ARRAY EMPTY"
```

(b) with one subscript 16:

```
"THE LOWEST SUBSCRIPT IN ^Z IS: 16"
"THE HIGHEST SUBSCRIPT IN ^Z IS: 16"
```

(c) with more than one subscript, as 77 89 201 900:

```
"THE LOWEST SUBSCRIPT IN ^Z IS: 77"
"THE HIGHEST SUBSCRIPT IN ^Z IS: 900"
```

The impact of SETs (such as SET ^Z(6)="") or KILLs (such as KILL ^Z(21)) at one station can be observed on the other display. If an instructor is present, subscript values can be suggested to guide the practice, extend the bounds of the set of values, and provide insights regarding the collating sequence. Exercise suggestions can be provided for learner use. Although loops or routines can be used to control the set of subscripts, it is likely better for the learner to become accustomed to the M SET and KILL command syntax and the impact of execution of these commands. Of course, as in (1b), with only one subscript, the values given for lowest and highest are the same.

(2) The next routine in the set is LOMIDHI, which reports as follows:

(a) with no subscripts
```
"ARRAY ^Z IS EMPTY - NO SUBSCRIPTS"
```

(b) with one subscript 35
```
"THE ONLY SUBSCRIPT IN ^Z IS 35"
```

(c) with two subscripts -4 and 701
```
"THE LOWEST SUBSCRIPT IN ^Z IS -4"
"THE HIGHEST SUBSCRIPT IN ^Z IS 701"
```

(d) with more than two subscripts, such as 17 62 133
```
"THE LOWEST SUBSCRIPT IN ^Z IS 17"
"THERE IS AT LEAST 1 SUBSCRIPT BETWEEN 17 and
133"
"THE HIGHEST SUBSCRIPT IN ^Z IS 133"
```

(3) The third routine in this set is `LOMIDHI2`, which differs from `LOWHIGH` in the case where more than two subscripts exist in the set:
```
"THERE ARE 2 SUBSCRIPTS BETWEEN 65 AND 1004"
```

This routine counts the subscripts between the lowest and highest values and also varies the natural language (IS vs. ARE and singular vs. plural), depending on whether one or several subscripts are present between the bounds. After "play" with arrays and these routines, learners should examine the routines and make sure they understand how the routines work and ask about any syntax for which the meaning is not clear.

• Local vs. global variables
The difference between local and global variables can also be emphasized by having a pair of participants check local and global variables at their respective stations.

• Multiuser application with two different parts of application active at the same time (on the 2 screens)
A pair of demonstration routines `RETAIL` and `RETAILM` are used for this purpose. `RETAIL` provides the opportunity to specify an account number and enter charges or payments. `RETAILM` serves as a monitor that reports on transactions (as in a central office) that are occurring. Thus `RETAILM` reacts to use of `RETAIL`. Learners see how two different parts of an application execute concurrently, performing their respective functions with respect to the same global array. This application incorporates a logging of all `RETAIL` transactions. Although invoked as a separate routine, the structure of `RETAILM` also anticipates the design of background jobs.

• Multiuser application with locking
The `RETAIL` routine demonstrates locking and can be executed concurrently on both stations for this purpose. Unlike a production routine, which endeavors to lock resources for the shortest time possible, this routine is designed to make the impact of locking obvious. Two partners working together are encouraged to access the same account number at the same time. One receives a notice that the account is "in use." An action (such as entering a charge or payment) at the station which acquires and then holds the lock releases the lock on the shared global, and a lock on that same global is then immediately acquired at the other station, where the routine has tried to lock that same global. At the instant that the charge or payment is entered on one screen, there is a reaction on both screens, and the new balance shows up on the other screen. Learners are encouraged to play with this routine and examine its code. With regard to locking, see Walters (pp. 254-259, 272), de Moel (pp. 45-46), Marshall (pp. 46-47), and the Melnick/Burack article.

• Single application controls 2 screens
Demonstration routines and open-ended practice can be used to gain understanding of how to control two or more devices within a single thread of execution. This exercise category requires making one of the stations available to be acquired as a device by the other. Learners find out that when a user is logged in at the other station, it cannot be acquired as a device, and that once acquired, the other keyboard is inactive (unless the programming at the first station offers read access). The use of multiple `$IO` values, `OPEN`, `USE`, and `CLOSE` can be practiced through interactive programming as well as by having the learners design their own applications. Demonstration applications serve as models for learner-designed applications. For device handling, see Walters, Chapter 13.

• Background job control where the background job runs a screen
A pair of demonstration routines (`SCHDCNTL` and `SCHDDISP`) simulate an airline schedule display system. `SCHDCNTL`, executed at one station, controls the display on the screen of the other station, starting and stopping it, and providing for entry and updating of flights to be displayed. The display routine `SCHDDISP` is invoked with the `JOB` command as a background job, and thus can continue to run even if `SCHDCNTL` is stopped (there is a provision not to halt the display) and the station is used for other purposes. This application can serve as a model for a number of learner-designed applications. With regard to multitasking, see Walters (pp. 253-254), and de Moel (pp. 44-45).

• Debugging and global access
Learners working singly or in pairs can look at a global on

one screen as a routine is being debugged at the other station.

• **Monitoring system to check transactions or changes in a global**
This was incorporated in the `RETAIL` application described above.

The following categories of exercises can be addressed with access to a different UCI on each screen:
• Practice with global protection and access, using utilities provided by the respective system vendor
• Practice using extended syntax for globals (syntax which specifies environment); see Walters (p. 250), de Moel (pp. 247, 295), and Kirsten (Section 5.3.2, pp. 105-108)
• Simulate journaling or logging in a different UCI (see Kirsten, Section 6.5.1, pp. 141-143)
• Practice with replication and translation; see articles listed below for additional information
• Practice with fragmentation of global (as in distributed environment; see A. Turano's article on this and also Kirsten, Section 6.5.2, p. 144)
• Simulate distributed processing

Examples and Summary

Exercises should be designed to encourage the learners to exploit the multiuser M Technology. Here are some example exercises:
• Enter on one screen and display results on the other screen (minimums, maximums, averages)
• Voting system, with control and results on one screen voting booth on the other screen
• Point of sale (grocery store, hardware store, department store)
• Banking (ATM) simulation
• Various kinds of display systems, along the lines of the airport flight display system described above

With appropriate models at hand, multiuser applications can be readily designed. Structured learning experiences alternate with open, unstructured practice in which the learner answers individual questions with guidance and support from the instructor(s). Learners with prior experience using other languages, who might otherwise focus on differences in syntax and just on programming, readily acquire a sense of M as a programming system and of its database properties. They gain facility in using the multiuser capabilities of M systems to help them solve practical problems of application development and debugging. After developing a better intuitive sense of the global array and the role it plays in M Technology, they are ready to address global design issues which are vital for effective and productive use of the technology. Work in pairs, especially for beginners with no prior programming experience, facilitates progress in most cases. In general, this approach encourages open exploration of the M environment. M

References

de Moel, Ed. *M[UMPS] by Example.* MT, 1997.

Emery, M. A. and J. A. Pierson. "Using Global Replication in Application Development." *MUG Quarterly* 18, 1 (1988): 67-71.

Gerum, Winfried, "Do You Know All About $DATA?" *MUMPS Computing* 22, 4 (1992): 18-20.

Gould, David Allen. "DSM Global Translation." *MUG Quarterly* 20, 1 (1990): 94-99.

Kirsten, Wolfgang. *Von ANS MUMPS zu ISO M Technologie: Fortgestrittenes Programmieren in M.* Epsilon, 1993.

Marshall, Rick. *The 1995 Standard M Pocket Guide.* MTA, 1996.

Melnick, Jerry, and Ruth-Ellen Burack, "Avoiding Database Contention," *MUG Quarterly* 20, 1 (1990): 87-93.

Turano, August M., "Managing Large Global Structures Through Segmentation," *MUG Quarterly* 19, 3 (Fall 1989): 35-39.

Tweed, R. L., and J. Milan, "A New Approach to Global Replication Between Processors," *Proceedings of MUG-Europe* (1984): 43-47.

Walters, Richard F., *M Programming: A Comprehensive Guide.* Digital Press, 1997.

Valerie J. Harvey, Ph.D., is a professor at Robert Morris College in Pennsylvania and the executive editor of M Computing. *During this sabbatical semester, Dr. Harvey is a visiting scientist in Dynamic Systems at the Software Engineering Institute of Carnegie Mellon University. Write to her in care of MTA's managing editor or email to: harvey@.robert.morris.edu*

Lynne R. Cuda has a masters degree in computer science form Texas A & M University and is Coordinator of Computer Applications for the University of Florida Faculty Group Practice (FGP). She manages applications, support, and programming for the FGP billing function.
