Orchestrating Remote and Local Databases

by August M. Turano

Introduction

Database technology represents a key software methodology that drives data acquisition and retrieval in the information age. Databases exist on virtually all types of hardware and software environments, from the smallest personal computer (PC) to the largest mainframes. Until recently, databases have been separate and distinct units that might, but usually did not have to, interact with front-end application programs. Although distributed database software is still not a popular technology, activity in this area is growing.

This article demonstrates a set of prototype applications in which local PC databases are used by GUI (graphical user interface) front-end programs, working in concert with local X-base databases and a large distributed M database on a remote machine, to perform various tasks. Sophisticated modern interfaces that can be designed by screen-oriented click-and-drag windows programs are particularly appealing to systems analysts and designers who must reengineer older applications or create new applications to compete against contemporary windows-oriented interfaces.

Modern Interfaces

The traditional roll-and-scroll interfaces are a thing of the past. Users now demand windows with color, pop-up menus, and variable screen-control elements. These elements must react appropriately to many different forms of screen input. Graphical systems, such as X Window and Motif, have been used but often require special terminals and can be very computer-intensive.[1] The PC-based Microsoft Windows is extremely popular: many users enjoy its consistent user interface display in various environments.[2] The cost of PC MIPS (million instructions per second) is roughly one-tenth that of mainframe MIPS. Therefore, if PCs can validate, capture screen data, and obey screen-oriented user directives, it makes sense to use them rather than place the load on the mainframe. Great cost savings can be realized by employing cooperative processing using both PCs and larger machines connected by networks. The relatively low cost of PC hardware and most related software encourages experimentation and development. Operating systems, application interfaces, programming languages, statistical packages, graphics packages, etc., cost hundreds of dollars for the PC, while the same functionality on a mini or mainframe computer may run into thousands or tens of thousands of dollars.

Event-driven computing has now become the de facto standard for most commercial applications. This programming paradigm is not difficult to learn, it just takes a while to adjust to it. Traditional programming usually begins with declarations, then proceeds to input, computation, and finally output. This activity usually is performed from the top down. In the event-driven environment, the user drives the application by clicking on a screen control element. The user decides the program path, and the application designer merely enforces the rules that govern the activity taken. A control item might be a command button (FILE/QUIT), a text box, a radio button (buttons that are mutually exclusive, such as "sex: male or female"), or a list box. Potential activities depend on the type of control: each control object has specific properties associated with it. The programming environment should limit what operations can be performed on each. The old bottom-of-screen line with a warning and/or error message is gone.

Creating user-friendly software for a front-end application, such as a data-entry transaction, can reduce employee turnover, since a better user interface for critical applications is important in the overall business application. Again, the reasonable cost of the software and hardware makes tools and experimentation in this computing area both exciting and worthwhile.

Combining a Remote M Database with Local PC X-Base Technology

Using what is defined more traditionally as client/server computing, the PC and a remote network or large mainframe can work together. Think of the PC as the front-end client responding to the user and sending requests for information to the remote database server. Validating input and responding to mouse clicks or commands when the transaction is ready to be filed is a local (PC) function. Forwarding the accumulated

information to the host machine or network to store the data in a separate repository is highly desirable and more secure. The remote database system can be connected via Ethernet or even serial lines depending on the speed and the amount of communication needed between the PC client and the host server.

. . . a central data repository makes more sense and builds confidence in the application system.

In the popular vernacular, the operating environment between the PC and the host is utilizing Dynamic Data Exchange (DDE). "DDE is an open, language-independent, message based protocol that lets applications exchange data or commands in any mutually agreed on format. The basic concept is that of conversation between client and server, with the client being the initiator".[3] It is the message, not the medium, that is important in moving data between independent computer systems.

The client software performs a variety of tasks, such as forms presentation, data acquisition, and validation. Data captured by the presentation layer can also be used for data manipulation, queries, etc. Using the local PC for validation and program execution minimizes the need for excessive data requests on a network: data are requested only as needed. Searches and sorts can be executed locally, even if the data must be retrieved from the host server. Report generators and statistical and graphical packages can produce output either directly from the database or indirectly from secondary files that might have been created from a search or sort function.

The PC, generally operated by a single user, offers very little data security; message protocols and passwords are necessary to implement proper user access to shared data. System maintenance is also difficult. If data are stored on various PC databases it is difficult to ensure proper backups or maintain an intact local database file structure or routine database maintenance. At best, it is extremely difficult to manage important data in this manner, so a central data repository makes more sense and builds confidence in the application system. This data repository is the remote database server.

The database server handles data storage and backups, provides access and security mechanisms, and produces data via database "seeks and gets," using the correct respective indexes to those data. The functions of the front-end client and the database server can be summarized as shown in figure 1.[4]

GUI Front End	Database Back End- The Database Server
• Forms presentation • Data capture • Data validation • Application logic • Report tools • Menus • Data manipulation	• Security/access • Backups • Archival activities • Indexes • Data retrieval • Data storage

Figure 1. Front-end/Database-server functions.

In coupling the PC programs with the remote database, display speed and data accuracy are critical. Sufficient speed requires minimal delay in an application when either local or remote lookups are in process. Fast disks for data inquiries are essential in either case. Because access time using the network data usually takes longer than a local database query, local databases should be used as much as possible.

Ease of Learning with X-Base Database and dBase

When trying new technology, using resources with a low learning curve works best. The X-Base database standard for local database construction is a common and very familiar format taught in almost every introductory database course. X-Base is extremely popular and powerful, in light of the abundant third-party software utilities and libraries available for just about any kind of data manipulation. X-Base provides several basic data types, including:

- CHARACTER - alphanumeric text;
- MEMO - longer text files;
- DATE - a valid date field;
- NUMERIC - numbers where width and decimal places are specified; and
- LOGICAL - yes or no.

In dBase, data are stored in files composed of individual data fields. A programmer selects a file with the appropriate index, then queries the file to access any of the data. The dBase program code accesses individual fields much like a global reference with data in various piece positions, so that each element can be manipulated.

In X-Base form, quick database lookups can be performed using indexed files, where B-tree type file construction provides fast access to specific data. Fields can be combined to form combination indexes (client number + patient ID) when the need for more involved lookups is a factor. Clipper and dBase are easy-to-learn programming languages for accessing any database file created in that environment.[5,6] The advantage of using a product like Clipper is that it is a complete but portable system; the end user does not have to own Clipper. Clipper produces executable files that are C-extensible (can call C routines directly) and can be distributed royalty-free. A large site with many PCs can use the application software for just the development cost; no additional royalty payments or licensing are needed.

M Database Technology Combined with Clipper

Clipper and/or dBase can provide a good database environment, but for reasons discussed earlier, database administration and system functions are difficult to manage. Combining both local and remote databases to be used at will is optimal: this minimizes data traffic on the network and offers the advantages of the PC programming environment combined with the security and database administration associated with a larger remote database. An application designer knows which files are static (such as religion, race, states), and these files can be kept locally on the PC. Large pools of information such as patient names, IDs, daily transactions, prescriptions, and bank account transactions are kept remotely. This design generated the first prototype applications involving patient queries and a patient-registration filing application.

This design generated the first prototype applications involving patient queries and a patient-registration filing application.

Clipper was used to design several local databases, yet the PC has access to all patient data stored on the remote M database. The remote database in our clinical environment consists of six gigabytes of data distributed on four 486/33 machines running MSM-MUMPS 3.0.12.[7] The patient-registration application necessitated the creation of several Clipper utility programs to allow re-creation of `SET $PIECE` and normal `$PIECE` extraction functions. We wrote a set of library routines to allow Clipper application code to make direct M calls in a noncryptic and straightforward way. This library works by preparing data and acquiring results via Clipper in a function called MCMD, whose syntax is simply MCMD ("MUMPS command here"). Digital Equipment Corporation has a DOS product, DSM DDP DOS, for accessing DSM-DDP protocol.[8] This terminate-and-stay-resident (TSR) product provides access using Ethernet directly from the DOS environment. A resident TSR assembles and disassembles packets for the Ethernet network. Each M command has a specific format that must be constructed before the interrupt call activating the TSR can be made. This syntax, while effective, was tedious to code in application software. We developed a clean way to allow an M programmer access to this functionality with a familiar M syntax. Support for `SET`, `KILL`, `LOCK`, `$ORDER`, `$GET`, `$QUERY`, volume-set selection, and network open and close utilities provided by the TSR were adapted to a more conventional syntax. In this way, Clipper and M syntaxes worked together to design the application.

Clipper to M Interface

The following example illustrates how the Clipper code performs an M function such as `$DATA`. (Calls to the Grumpfish library add sparkle to this application with special screen techniques and color.)[9] Using Clipper code to obtain `$DATA(^PA(value))` from the M database, build the full M reference as a string and pass it on to M:

```
GLOBREF:="^PA("+CHR(34)+value+CHR(34)+")"
CMDSTR:="$D("+GLOBREF+")"
RES:=MCMD(CMDSTR)
```

The result, RES, contains the value returned by `$DATA`. Note that in Clipper, the "+" is the concatenation operator when operating on strings. The actual performance of the system delivers in excess of 125 database reads per second (approximately .008 seconds per read). These numbers are, of course, dependent on the user load, network traffic, speed of the disks, and the CPU.

M, Visual Basic, and Clipper

Clipper, although extremely powerful and easy to use, still falls short in mouse support and ease in overall GUI creation. Searching for a better way to create friendly and attractive user interfaces, we turned to Visual Basic, which has been getting rave reviews for its DOS and Windows versions. Visual Basic allows a form to be created by pointing and dropping items onto the window. All items on the form are given a control name and associated with specific properties. For example, a text box can have width, border style, color descriptions for foreground and background, a tab stop number, activity for when the control is activated or deactivated, and more. Controls can be grouped together or created separately. There is a complete development environment with

menu construction, mouse support, and ease of coupling the form with specific program code. The language is easy to learn and use; it is BASIC with much greater power and control than ever before. The *Professional Edition* includes libraries for doing a variety of functions.[10]

Visual Basic puts the fun back into programming. Programs can be created for either the Windows or DOS environment, so that a nice-looking, colorful interface can be designed for any appropriate platform. The event-driven programming paradigm is a little hard at first, but after a day or two, a user can be fairly comfortable. Easily drawing the interface that the user wants is what GUI design is all about, and Visual Basic provides that. Timers can be set, graphics can be used, complete input/output (I/O) capabilities are available, and EXE files are the final output.

Giving applications a face lift benefits users and system designers alike.

The biggest drawback to Visual Basic is the ISAM (indexed sequential access method) file structure. It is awkward, not an industry standard, and is limited to 128 megabytes per file. For this reason, we chose a hybrid of Visual Basic coupled with Clipper Database file structure as the development path. A third-party vendor, Sequiter Software, has a set of library routines that allowed access to all dBase functions, such as "seek," "goto record," and "get field data."[11] This software made it possible to combine Visual Basic's programming ease with the standard dBase format and plethora of access tools and report generators available to the dBase programmer. Visual Basic now is used just as Clipper was in the initial experiments to develop PC client applications. These applications employ the same remote access TSR software that was used in the first Clipper applications. The only changes necessary were internal to the library routines to handle details concerning mixed language memory allocation and string-passing inconsistencies.

Conclusion

Redesigning the user interfaces of currently operating application code is a common and very important task. Giving applications a face lift benefits users and system designers alike. If this task is performed correctly, cooperative-processing techniques can save precious mainframe or host database CPU cycles and improve overall system performance. Using the PC to perform basic I/O and validation can significantly reduce the host processor's load. Easy-to-learn and powerful PC languages such as Clipper (X-Base) and Visual Basic make the reengineering of many applications exciting. Access to royalty-free third-party libraries permits consistency and solid software construction, affording both modularity and reusability of code. Redesigning the user interface is always worthwhile, particularly for older applications. After all, the user thinks the interface is the program.

Following is a short sample of Clipper access of M database. For a more complete understanding of the Clipper access of M database, please see the upcoming June 1993 issue of *M Computing*, which will carry a full appendix following a similar article by this author. **M**

Endnotes

1. "X Window," ANSI Technical Committee X3H3.6, and "X Window Binding," MDC Document No. X11/SC11/92-10.

2. Microsoft Windows 3.1, Microsoft Corporation, 1990-1992.

3. K. Kornfeld and K. Gilhooly, "OOPS via DDE," *Byte*, (June 1992) 145-154.

4. T. Duesher, "Selling the Database Server," *Reseller Management*, July 1991, 54-61.

5. Clipper - CA & Associates, Nantucket Corporation, 1984-1990.

6. dBase IV - Ashton-Tate, A Borland Company, Borland International, Inc., 1988-1992.

7. Micronetics MUMPS, Micronetics Design Corporation, 1992.

8. DSM-DP-DOS, Software product, post #'s. BI-PB8XA-BK and BI-PBDAA-BK.

9. G. Lief, Grumpfish Library, Grumpfish, Inc., 1988-1991.

10. Microsoft Visual Basic Professional Edition, Programming System for MS-DOS, Microsoft Corporation, 1992.

11. CodeBASIC, Database Management, Sequiter Software, Inc., 1988-1992.

YeYi Wang created vital parts of the interface library.

Dr. Turano graduated from the University of Pittsburgh in 1982 with a Ph.D. in physics. Introduced to M in 1981, he has been developing systems utilizing M in various PC languages ever since. He is director of information and communication systems at Med-Chek, a Damon Laboratory, 4900 Perry Highway, Pittsburgh, PA 15229; telephone 412-931-1281.

APPENDIX
Clipper Access of M Database

```
/* acn.prg program to that uses DEC-DDPDOS and CA-CLIPPER to do
         access a remote MUMPS database for lookups
         as well as data deposits */
/* Copyright 1993 A. Turano, Ph.D. */
//Necessary initialization
//The following are modules that are contained in CDDP.LIB
   EXTERNAL MCMD
   EXTERNAL OPENDDP
   EXTERNAL CLOSEDDP
   UCI="LAB"                                          //select user access account
   VOL="LPB"                                          //select volume set to access
   RES := " "                                         //Set RES (result) to character,
   ACN := " "                                         //(accession number of specimen) to character
   LOOP := 500                                        //how many times do you want to go thru the loop
                                                      //accessing remote data
//Set screen up
   SAVE DRAPE("temp.scr")                             //fancy save for DOS screen - restore on exit
   SET COLOR TO "W+/B,W/R"                            //set some screen color parameters
   SET DECIMALS TO 4                                  //set number of decimals for computations
   SET SCOREBOARD OFF                                 //don't show the bottom screen activity monitor
   CLEAR SCREEN
   //call grumpfish exploding box function
   msg="Med Chek Labs - A Damon Laboratory"
   CLRSCR(5) 'fancy screen wipe
   @23,30 SAY "Hit a key to continue...."
   FALLGUY(20,23,msg,100)                             //grumpfish library calls
   RAINBOW(msg)                                       //grumpfish library calls
//Open Channel to MUMPS system
   CLRSCR(8)
   ExpBox(1,1,22,76,1,20,"W+/BG+","Accessing MUMPS through Clipper")
   SET COLOR TO "W+/BG+"
   OK := OPENDDP(UCI,VOL)                             //open the DDP channel
   IF OK = -1
      IMPBOX(20)                                      //make an imploding box
      CLS
      @ 5,5 SAY "Bad UCI or VOLUME name!"
      RETURN
   ELSEIF OK = -2
      IMPBOX(20)
      CLS
      @ 5,5 SAY "DDP unable to start.."
      RETURN
   ENDIF
//Get ACN from user
   @ 2,5 SAY "Enter starting accession number:" GET ACN PICTURE "!9999999"
   READ
   setcolor("B/BG+")
   @ 2,5 say "MUMPS $O through Patient File, "+str(LOOP)+"Records"
   setcolor("W+/BG+")
//Build MUMPS command
   GLOBREF := "^PA("+CHR(34)+ACN+CHR(34)+")"          //construct the global reference
   CMDSTR := "$D("+GLOBREF+")"                        //make the command string to be performed
//Look at the MUMPS system and see it that ACN exists
   RES = MCMD(CMDSTR)                                 //execute a MUMPS command-remote $D
   qout("res=",RES)                                   //this is the result
   res:= VAL(res)                                     //everything from MUMPs is character MAKE numeric
   IF res = 0
      IMPBOX(20)                                      //grumpfish function
      CLS
      @ 5,5 SAY "Accession number not found!"
   ELSEIF res=10
      IMPBOX(20)
      CLS
      @5,5 SAY "Accession number not defined but has descendents."
   ELSEIF res=1
      IMPBOX(20)
      CLS
      @5,5 SAY "Accession number found and has NO descendents."
   ELSEIF res=11
      IMPBOX(20)
      CLS
      @5,5 SAY "Accession number found with descendents."
```

```
   ENDIF
   ? "Press any key to continue..."
   INKEY(0)                                          //wait for user to continue
//Found the accession #, so get the info
   CMDSTR := "$G("+GLOBREF+")"                       //construct another command string
   RES = MCMD(CMDSTR)                                //go visit MUMPS
   TARRAY := <>                                      //TARRAY is an array to hold the pieces of data
                                                     //dynamic array allocation-just like MUMPS!
   DATA := EXPIECE(RES,"^",19,TARRAY)                //Extract the 19th piece of RES
                                                     //TARRAY is just in case you want more.
//Display that patient's name+sex+age
   @ 3,5 CLEAR TO 21,75
   rollup(padr(TARRAY[14],35)+padr(TARRAY[16],10)+TARRAY[17])
//Display next LOOP patient names and time it
   STARTTIME := TIME()
   FOR I=1 TO LOOP-1                                 //DO SOME $O FUNCTIONS
      //Get the next ACN # - Perform the $O function
      CMDSTR="$O("+GLOBREF+")"
      RES = MCMD(CMDSTR)                             //go visit MUMPS
      //Get data for that ACN#
      GLOBREF="^PA("+CHR(34)+RES+CHR(34)+")"         //global reference
      CMDSTR := "$G("+GLOBREF+")"
      RES = MCMD(CMDSTR)                             //go visit MUMPS
      //Pick out the patient's name
      TARRAY := <>
      DATA = EXPIECE(RES,"^",19,TARRAY)
      //Write out PIECES 14=Name+PIECE 16=Sex+PIECE 17=DOB
      rollup(padr(TARRAY[14],35)+padr(TARRAY[16],10)+TARRAY[17])
   NEXT
   ENDTIME := TIME()
//Show the results and times
   @ 2,5 CLEAR TO 21,75
   @ 4,5 say "Starting Time: " + STARTTIME
   @ 5,5 say "Ending Time: " + ENDTIME
   SECS := ComputeTime(STARTTIME, ENDTIME)
   @ 7,5 say "Seconds per read, piece, and display: " + STR(SECS/LOOP)
   @ 8,5 SAY " # Reads, pieces, displays per second: " + STR(LOOP/SECS)
   @ 17,5 SAY "Press a key to CONTINUE. . ."
   INKEY(0)
//Display that patient's name
   clrscr(2)
   setcolor("B/BG+")
   @ 2,5 SAY "MUMPS $Q through Patient File, "+str(LOOP)+"Records"
   setcolor("W+/BG+")
//Display next LOOP patient names and time it
   STARTTIME := TIME()
   FOR I=1 TO LOOP                                   //DO SOME $O FUNCTIONS
      //Get the next ACN #
      CMDSTR="$Q("+GLOBREF+")"
      RES = MCMD(CMDSTR)                             //go visit MUMPS
      //Get data for that ACN #
      GLOBREF:=RES
      CMDSTR := "$Q("+GLOBREF+")"
      RES = MCMD(CMDSTR)                             //go visit MUMPS
      rollup(globref+"=")
      cmdstr="$G("+globref+")"
      RES = MCMD(cmdstr)
      rollup(res)
   NEXT
   ENDTIME := TIME()
   //Show the results
   @ 2,5 CLEAR TO 21,75
   @ 4,5 say "Starting Time: " + STARTTIME
   @ 5,5 say "Ending Time: " + ENDTIME
   SECS := ComputeTime(STARTTIME, ENDTIME)
   @ 7,5 say "Seconds per read, piece, and display: " + STR(SECS/LOOP)
   @ 8,5 SAY " # Reads, pieces, displays per second: " + STR(LOOP/SECS)
   @ 17,5 SAY "Press a key to exit. . ."
   INKEY(0)
//Measure only READ time
   @ 2,5 clear to 21,75
   setcolor("B/BG+")
   @ 7,5 SAY "Measure of RAW SPEED doing "+str(LOOP)+" $Qs:"
   setcolor("W+/BG+")
   SET CURSOR OFF
   STARTTIME := TIME()
   FOR I=1 TO LOOP                                   // DO SOME $O FUNCTIONS
      CMDSTR="$Q("+GLOBREF+")"
      RES = MCMD(CMDSTR)
      GLOBREF:=RES
      CMDSTR := "$G("+GLOBREF+")"
      RES = MCMD(CMDSTR)
```

```
   NEXT
   ENDTIME := TIME()
   SET CURSOR ON
   SECS := ComputeTime(STARTTIME, ENDTIME)
   @ 9,5 SAY "Seconds per READ: " + STR(SECS/LOOP)
   @ 10,5 SAY "READS per second: " + STR(LOOP/SECS)
//Close up
   @ 15,5 SAY "Press a key to exit. . ."
   INKEY(0)
   ImpBox(20)                                          //Close up the on-screen box
   PULL_DRAPE("TEMP.SCR",30)                           //GRUMPFISH FUNCTION RETURN
// ***************************************************************
//Mimic the $Piece function with some extras
   FUNCTION EXPIECE(mstring,delim,n,narray)
   local xstr,i && xstr=is working string, i=loop counter
                && n is the piece you want, narray =logical return
                && entire array of pieces
   IF N>256
      return "* Error piece number too large*"
   ENDIF
   xstr=mstring+delim
   FOR i=1 TO N
      pos=AT(delim,xstr) && postion of delimiter
      IF pos=0
      EXIT && exit the loop
      ENDIF
      pie=SUBSTR(xstr,1,pos-1)
      IF narray!=NIL
      AADD(narray,pie) && Build an array of pieces
      ENDIF
      IF i=n
      return pie
      ENDIF
      xstr=SUBSTR(xstr,pos+1,len(xstr)-pos)
   NEXT
   RETURN NIL && not found
/*********************************************************************/
//Mimic the set $Piece function
   FUNCTION SETPIECE(mstring,delim,N,mvalue)
   && RELEASE parray && clean up the array before starting. . .
   IF N>256
   return "* Error piece number too large*"
   ENDIF
   xstr=mstring+delim
   i=0 && counts the number of delimiters
   DO WHILE AT(delim,xstr)>0
      i=i+1
      pos=AT(delim,xstr) && postion of delimiter
      pie=SUBSTR(xstr,1,pos-1)
      AADD(parray,pie) && Build an array of pieces
      xstr=SUBSTR(xstr,pos+1,len(xstr)-pos) && shorten string by 1 delim
   ENDDO
   IF i<n && This code will add null elements to the needed depth
      FOR J=1 TO N-i
      AADD(parray,"")
      NEXT
   ENDIF && delimiter
      parray[N]=mvalue &&set the appropriate element
      mystring=""
      FOR I=1 TO LEN(PARRAY) && construct the changed delimited string
      mystring=mystring+parray[i]+delim
      NEXT
      mystring=SUBSTR(mystring,1,LEN(mystring)-1)
   RETURN mystring
function ComputeTime(STARTTIME, ENDTIME)
local MINS, SECS
   MINS := VAL(SUBSTR(ENDTIME, 4, 2)) - VAL(SUBSTR(STARTTIME, 4,2))
   SECS := VAL(SUBSTR(ENDTIME, 7, 2)) - VAL(SUBSTR(STARTTIME, 7,2))
   IF SECS < 0
   SECS := 60 + SECS
   ENDIF
   return(SECS+60*MINS)
/* **********************FUNCTION rollup ****************** */
STATIC FUNCTION rollup(mtxt)
      SCROLL(2,5,20,70,1)                              // TOP LEFT, BOTTOM RIGHT , # OF LINES TO SCROLL
      @20,5 SAY mtxt
      RETURN NIL
```

For a more complete understanding of the Clipper access of M database, please see the upcoming June 1993 issue of *M Computing*.