```
FILE: MSG      ASSEMBLE A              VM/SP CONVERSATIONAL MONITOR SYSTEM

*                                                                      MSG01110
*  Come here if it's a connection complete interrupt (IPTYPE = X'02')  MSG01120
*                                                                      MSG01130
CONNCOMP EQU   *                                                       MSG01140
         OI    CONCOMP,X'40'      Post the CONCOMP ECB                 MSG01150
         BR    R14                AND RETURN                           MSG01160
*                                                                      MSG01170
*  This is our connect pending exit.  It should never be driven.       MSG01180
*  If it is, we'll just ignore it and return.                          MSG01190
*                                                                      MSG01200
CONPEND  EQU   *                                                       MSG01210
         BR    R14                                                     MSG01220
         EJECT                                                         MSG01230
*********************************************************************** MSG01240
*                                                                     * MSG01250
*              E Q U A T E S and C O N S T A N T S                    * MSG01260
*                                                                     * MSG01270
*********************************************************************** MSG01280
         SPACE 2                                                       MSG01290
         DS    0D                                                      MSG01300
SETMSG   DC    CL8'CP'            Parameter list to tell CP we         MSG01310
         DC    CL8'SET'              want our message via IUCV         MSG01320
         DC    CL8'MSG'                                               MSG01330
         DC    CL8'IUCV'                                              MSG01340
         DC    8X'FF'                                                 MSG01350
SETON    DC    CL8'CP'            Parameter list to tell CP we         MSG01360
         DC    CL8'SET'              want our message set back         MSG01370
         DC    CL8'MSG'              to 'ON'.                         MSG01380
         DC    CL8'ON'                                                MSG01390
         DC    8X'FF'                                                 MSG01400
IUCVPLST DC    40X'00'            Used for the IUCV parameter list     MSG01410
MSGID    DS    F                  Holds the IUCV message id            MSG01420
MSGCLASS DS    F                  Holds the IUCV message class         MSG01430
STARMSG  DC    CL8'*MSG'          The name of the IUCV Message Service MSG01440
SAM      DC    CL8'SAM'           Name which CMS will know us by       MSG01450
CONCOMP  DC    F'0'               Connection Complete ECB              MSG01460
MSGIN    DC    F'0'               Message has arrived ECB              MSG01470
PATHID   DC    H'0'               Holds the IUCV path id               MSG01480
BUFERLEN DC    H'140'             Says our buffer will be 140 chars    MSG01490
MSGAREA  DC    CL140'             Message buffer area                  MSG01500
DISABLE  DC    X'00'              Byte to disable us for interrupts    MSG01510
         REGEQU                                                        MSG01520
         NUCON                                                         MSG01530
         COPY  IPARML                                                  MSG01540
         END                                                           MSG01550
```


SESSION REPORT

SHARE

61	B638	CMS ARCHITECTURE AND INTERACTIVE COMPUTING	300
SHARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE

CMS	LARRY GRAZIOSE	BAM
PROJECT	SESSION CHAIRMAN	INST. CODE

BANK OF AMERICA, 1455 Market Street, San Francisco, CA 94103 (415) 622-1881

SESSION CHAIRMAN'S COMPANY, ADDRESS, and PHONE NUMBER

CMS ARCHITECTURE AND INTERACTIVE COMPUTING

Charles Daney
Senior Scientist
Tymshare, Inc.
20705 Valley Green Drive
Cupertino, California
Installation Code: TYM

August 22, 1983
CMS Project
Session B638

This paper contains the personal views of the author and not necessarily those of Tymshare, Inc.

l/c/pal/1

1. INTRODUCTION.

We shape our buildings:
thereafter they shape us.

Winston Churchill

Of all forms of visible otherworldliness,
it seems to me, the Gothic is at once
the most logical and the most beautiful.
It reaches up magnificently -
and a good half of it is palpably useless.

H. L. Mencken

It has become common to talk about architecture in connection with computer software. We ought to think about it and consider why. One reason, I believe, is that traditionally architecture has dealt with large-scale forms and relationships, and with multi-level structures that depend upon other similarly complex structures. And so it is with large software systems. Further, just like major public buildings, our software systems and the underlying hardware design tend to be with us for long periods of time. Finally, just as our architectural environment shapes the way we work and play, the way we interact with others, even the way we think, so also do the operating systems and large application programs that we use to animate our computer technology. Architecture has great power over our lives.

My purpose here is to talk about the architecture of CMS: what's good about it, what's bad about it, and in what directions it may, or should, be heading. At times I will be speaking about mere details and features, but always keep in mind that these are usually influenced strongly by the overall structure.

There probably hasn't been any time since CMS was conceived in 1964 that people who use CMS a lot or are responsible for its care and feeding haven't been paying attention to details in the architecture of CMS. But there are recognizable milestones in the development of CMS, such as the announcement in 1972 of VM/370. In more modern times, however, I think the most significant milestone was the announcement of the so-called BSEPP (Basic Systems Extension Program Product) version of CMS in 1979. That was the first time that IBM added major new function which clearly responded to explicit requests from the user community, and that was the first time that IBM made it clear that VM/CMS was a "strategic" product. At the same time, the 4300 series of processors was announced. The compatibility of VM/CMS and the 4300 series has contributed to the great success of both since then.

It did not take long before the leaders of the VM Group of Share recognized that IBM intended to devote substantial resources to the development and enhancement of VM/CMS. They realized that a significant effort would be required to inform IBM of the user community's needs for VM/CMS. A task force was formed in the fall of 1979 to undertake this effort, and a year later it delivered its final report. [1] I do not intend to summarize the conclusions of that report, though they are still very relevant and IBM has hardly begun to address most of them. But I will inevitably repeat many of those conclusions in delivering my own opinions.

One of the appendices in the final report of the task force set out some proposals for the restructuring of CMS. It seemed to me and several others in the CMS Project that there was a lot more that could be said in that area. So we decided in early 1981, with a naivete that is almost touching, to undertake the quixotic mission of writing a white paper on CMS restructuring. This we delivered in 1982. [2] It seemed an appropriate excuse for a session, and that, it would seem, is why I am here today. I want to acknowledge the collaboration of my co-conspirators in this endeavor: Pat Ryall and Larry Graziose. However, although many of the conclusions I shall mention today were formed in that effort, I will also be going quite a bit beyond it. Also, of course, these opinions are my own, and cannot in any way be blamed on Pat or Larry, or my installation, or anyone else.

The plan, then, is first to survey a few of the factors that have made CMS successful and popular. Then I will go into a much longer discussion of what I see as some of the most noteworthy deficiencies of CMS at present. I will conclude by looking at architectures for interactive computing in general, and possible implications for CMS.

2. STRENGTHS OF CMS.

No architecture is so haughty
as that which is simple.

John Ruskin

Lovely promise and quick ruin
are seen nowhere better than in Gothic architecture.

George Santayana

I'm going to spend some time going over what I consider to be the strengths of CMS, because I want to put on record those characteristics of CMS which are meritorious and should be preserved in extension and successor interactive systems. Many of these observations can be found in the CMS Task Force report itself.

The very most important strength, to my mind, of CMS is the excellence of its system metaphor or conceptual model for programmers. The notion of the "methaphor" embodied in an interactive computer system has been popularized by recent personal workstations like Xerox's Star and Apple's Lisa. The metaphor is simply the concept or idea which the system's designers intended for the system's users to employ in organizing their knowledge of the system. In the case of Lisa, for example, that metaphor is the office desktop.

For CMS the metaphor is a virtual machine. CMS commands are conceived and named appropriately for dealing with concepts pertaining to real (mainframe) computer systems: disks, tapes, readers, printers, computer files, and so forth. In this context, the command names and parameters are very mnemonic; they are very easily learned and understood by programmers. Software developers have access to essentially all the facilities of a real computer. Consequently, it is no surprise that CMS has been so popular with computer professionals.

At the same time, it comes as no surprise that CMS is difficult to learn and use by people who are not familiar with computers. Still, I stress the existence of the metaphor as a strength, because there are any number of other interactive systems which do not embrace a metaphor of any kind.

Despite some notable lapses, CMS command syntax is fairly simple and consistent. File names and parameters tend to be specified in the same way to all commands. Defaults tend to work as expected. However, this consistency is almost an accident, since there are no system facilities that promote it.

Another strength of CMS is its relative simplicity of

implementation. This is a consequence of several factors, such as the fact that CMS is a single-user, single-task system, that many functions essential in a complete interactive system (like paging, spooling, physical I/O, etc) are handled elsewhere in the system hardware and software, and, lastly, that, until recently, CMS has flourished under a regime of benign neglect by the vendor.

From this simplicity of implementation flow several other agreeable characteristics. CMS is generally efficient in its use of resources and hence provides good response time. (There are exceptions, of course, and this generalization is threatened by various poorly designed recent "enhancements"). Because of its simplicity, it is easy for system programmers to learn and understand its internals. This in turn allows for relative ease of debugging, repair, modification, and enhancement. And, in spite of the haphazard, even chaotic, fashion in which CMS has evolved, its basic simplicity has preserved whatever decent level of robustness and reliability CMS still possesses.

The CMS file system deserves special mention for its qualities of simplicity, effectiveness, reliability, and device independence. These characteristics are shared by both the original CMS file system and the newer "EDF" file system introduced a few years ago. In fact, the EDF file system merits substantial praise for perserving compatibility with, and the best characteristics of, the old file system while introducing more functionality and better performance. Incidentally, my own 191 disk is still in the old format. I haven't lost data due to a file system error in many years, and EDF was far from perfect for awhile.

The CMS file system is simple and easy to use both for application programmers and end users. One never has to even think about different device types. Things like BUFNO, TRK, ABSTR, RLSE, CONTIG, MXIG, ALX, SPLIT, BFTEK, OPTCD, DSORG, EROPT, NCP, etc. never arise to trouble the mind. Available space management is automatic.

As a consequence of both its simplicity and its computer-oriented metaphor, CMS is flexible, modifiable, tailorable, and extensible - by anyone from casual users to systems programmers, although in different degree. Old commands can be given synonyms or entirely replaced. New commands can be added by anyone who can write a program or an Exec.

The command executors, of which there are now three, contribute heavily to this tailorability of CMS. They permit both slight modification of the syntax and semantics of existing commands to suit individual preferences as well as the construction of powerful and complex application systems. Most importantly, the availability of a relatively good executor has contributed substantially to the popularity of a building-block approach to application development under CMS, through the ease of combining numerous commands and tools into integrated applications. The

development of the tools themselves was stimulated in the first place by the programmer-friendly nature of CMS.

In summary, my advice to designers and developers working on extensions to CMS would be as follows:

(1) Keep it SIMPLE.

(2) Keep it FAST.

(3) Keep it CONSISTENT.

(4) Keep it FLEXIBLE.

(5) Keep it NATIVE.

3. WEAKNESSES OF CMS.

The genius of architecture seems to have shed
its maledictions over this land.

Thomas Jefferson

If the design of a building be originally bad,
the only virtue it can ever possess
will be signs of antiquity.

John Ruskin

Clearly, CMS presents different faces to different groups of users. Another way of looking at this is in terms of the diverse "environments" in which various classes of users do their work. For each type of user there is a different environment. The first class of users is what programmers think of as "end users"; for them the environment may be PROFS, or SQL, or word processing facilities. But a second type of end user is the professional programmer himself. His environment consists of the original, native CMS commands and utilities. A third type of "end user", albeit non-human, is the executing application program. Another way to consider the environment of this last "user" is as that faced by the professional programmer who must develop applications to execute in CMS.

My purpose is to illustrate a number of the deficiencies and weaknesses in CMS in order to motivate my enumeration of some of the ways I think that CMS must be extended or redesigned. These deficiencies turn out to be very different when viewed from the perspectives of different classes of users, i. e. in terms of the different environments.

It will turn out that many, if not most, of the strengths of CMS are also weaknesses, at least when viewed from different perspectives. This is another reason I want to insist on considering the different environments separately.

This is my justification for lumping together all end users in one category. These considerations even apply to professional programmers, though perhaps with less force, because many of the deficiencies in the end user environment have to do with one of CMS's greatest strengths: its consistent virtual machine metaphor. As we all know, professional programmers can put up with more cryptic computerese than normal people. Yet I would contend that many of these problems bedevil even the professionals - perhaps unbeknownst to them.

To begin with, there is the command language. It is better, more English-like, more mnemonic than that of other interactive systems, perhaps. But still it has its warts. There are multiple

commands to do essentially the same thing, such as sending a "spool" file to another user (assuming you even know what a spool file is). And different commands have different options to accomplish the same result - the famous TERM/TYPE dilemma. Further, for the non-computer-oriented user, many commands are not the least bit mnemonic.

Although part of the problem with the command language is the underlying metaphor, I am persuaded that the larger part is with the nature of command languages in general. Command languages are specialized languages that depart markedly from the idioms of everyday speech. Nor would the availability of a command language indistinguishable from natural language be fully satisfactory. Natural language is too verbose and imprecise, even for non-computer-oriented users. Worst of all, it requires a lot of typing.

The traditional alternative to command languages is menus. Yet both computer professionals and everyone else quickly tire of conventional menus. The problem, I think, is not with menus per se, but rather with current display technology. After all, when you have just 24 lines of 80 characters to deal with, you have to dedicate most or all of a screen to the menu instead of the data, with the consequence that you may have to negotiate several levels of menus in order to accomplish some desired effect, and all the while your text or graphics or data or whatever is really important to you can't be seen. By the time you finally see it again, you've forgotten what you changed. Your train of thought is certainly broken.

The deficiency, then, which I am trying to describe here is the lack of several things: ways to present the user with menus of current alternatives, ways to select quickly from those menus with a choice of input devices such as mice or function keys or touch pads, and most of all, ways of partitioning the user's screen into multiple windows so that all the currently relevant data and menus can be seen simultaneously. Just imagine, say, a list of files in one window, a couple of windows with views of a file, and one more window with help information in it.

Better terminal hardware is part of the answer, but not all of it. After all, CMS has no facilities for window management, and its whole internal structure revolves around reading a command from a typewriter terminal.

There are many other problems with the end user's environment. For example, there is no way to go back to an earlier point in the session to recall a previous error message or view more than 24 lines of output, other than the extremely awkward artifice of "spooling the console", whatever that means. Within reasonable limits, one ought to be able to scroll backward in a session just by hitting a key.

The help facilities available under CMS are not very good, but

they certainly are slow. I happen to believe that programs, whether applications or operating systems, should be "self-describing" in the sense that documentation is mostly unnecessary. For all the hue and cry about the need for high-quality system documentation, the real truth is that most end users hate to read documentation. "Documentation" is synonymous with drudgery and boredom. What users really want is to be able to sit down at the terminal and figure the program out for themselves, and the quicker the better.

I don't think that the Apple Lisa approach with heavy use of icons is the only way to accomplish this. You should be aware that Apple's claims that Lisa can be learned very quickly just by sitting down at the keyboard are substantially true, but icons also turn a lot of people off. There are other ways to implement self-describing programs. I think the fundamentals are as follows:

(1) Provide a clear and relatively complete basic help facility with short, easily searchable items that is available at all times.

(2) Provide longer, interactive tutorials with lots of executable examples for the more complex parts of a system. Make the program capable of teaching its own use.

(3) Use metaphors appropriate to the task at hand. Use descriptive language on the screen that refers to these metaphors. Keep the program simple and consistent with its metaphor.

(4) Provide both a simple command language tied to the metaphors and menus from which selection can be made with a variety of input devices.

(5) When prompting for input or decisions, tell clearly what is being requested, what the alternatives are, and what the defaults are.

For the professional programmer as end user there is an additional set of deficiencies in CMS. It is appropriate to single these out for special mention, because CMS is first, and still foremost, a tool for professional programmers to do their jobs. In light of this fact, the existing problems are really quite surprising.

The most astonishing gap in the environment that CMS provides the professional programmer is the lack of tools for source code management. True, there is the UPDATE command. And except for facilities of ISPF, there are no code libraries, no tools for version control, no easy means for teams of programmers to work on the same systems of code.

I must confess my unfamiliarity with ISPF. Although it may have

answers to some of these problems, I would like to point out that the implementation of hierarchical structure in the CMS file system would be a conceptually simpler way of solving many of these problems. Having separate subdirectories is a very natural way to manage different versions of code; it is also much more flexible than multiple minidisks. And incidentally, although hierarchy is often criticized as an inappropriate metaphor for non-computer-oriented users, it is (or should be) well understood by professional programmers, whom we are now considering.

There are many other problems in the environment that CMS provides to professional programmers. Another obvious one is the extremely poor debugging facilities. This is just incomprehensible; even TSO has a decent symbolic TEST facility. Debugging under CMS is a trial of hex calculation and the ability to follow internal control block chains. There are no control block formatting tools. CMS does not even have a trace table.

One more deficiency, just for good measure, is the inadequate support of high level languages. For example, there is the incompatiblity of the different languages. Although this is a problem which is nowhere adequately solved, it is certainly worse under CMS than TSO. Several years ago the CMS and PL/I Projects came up with a long list of problems with running PL/I under CMS, most of which have yet to be fixed. Generating executable modules of high level language programs is a complicated, error-prone process, which is subtly different for each language.

4. WEAKNESSES OF CMS: EXECUTION ENVIRONMENT.

In architecture the pride of man,
his triumph over gravitation, his will to power,
assume a visible form.
Architecture is a sort of oratory of power
by means of forms.

Friedrich Nietzsche

The third class of problems that I want to consider is the special concern of those who develop applications to run in CMS. But in terms of environments, we are really talking about the environment that is faced by the executing application rather than by its developer. Of course, the developer is the one who must contend, repeatedly, with the deficiencies in this environment.

I am going to spend relatively more time in this area than in the other two, for a couple of reasons. The first is that it is the best excuse I have to discuss problems with CMS structure and architecture, which is one of my ultimate objectives. The other is that the deficiencies in the CMS execution environment must be corrected before we can make much progress in improving the other environments of CMS.

I am going to discuss deficiencies in the CMS execution environment by using a specific application as an example, the VMSHARE conferencing facility. I am going to say a little about the internals of this system in order to make the point about how many basic operating system primitives needed by real, live applications are simply missing in our current CMS. I trust that most of my present audience knows what VMSHARE is. If not, you may think of it simply as a sophisticated electronic mail system.

As you may know, VMSHARE was originally implemented by Dave Smith, starting in 1976, as a system, a very large system, of CMS commands and Execs. In fact, VMSHARE was one of the earlier examples of developing sophisticated end-user applications using mostly Execs and native CMS facilities. That original system illustrates vividly various strengths of CMS that have already been discussed: its flexibility and tailorability, the power of the Exec processor, the usability of the file system, and the amenability of CMS to the building-block approach to application development.

Several functional objectives of VMSHARE emerged early in its development. In addition to providing a tailored environment in which the commands would be customized to the task at hand, the system also had to provide what became known as a padded-cell environment: users would be limited to only such capabilities as the system was intended to support.

This ability to create secure, limited, or "padded cell" environments is a natural requirement of many typical application systems. But it was not possible to do it in the CMS of seven years ago without system mods, and it is still impossible today. Yet it is very easy to do! Mainly, you need a way to prevent the user from issuing commands to CP - that's a relatively simple CP mod. Within CMS itself there are several very small features that are necessary. Probably the most basic of these is provision for an exit routine that can examine CMS commands before they are executed in order to be sure they are allowable. Ideally, this should handle commands from any source: the command line, the editors, the executors, special commands like FLIST, etc. Unfortunately, the lack of any centralization of the command handling function makes this capability unnecessarily difficult.

In order to mantain security, the application must also be able to retain control when an abend occurs. At Tymshare we did this with a special Exec that was run after abend cleanup was complete. It is possible that the new abend exit provided by VM/SP release 3 will satisfactorily handle this requirement, but one can't be sure until one sees how it is implemented.

The initial implementation of VMSHARE used the old CMS line editor, which needed a couple of simple modifications in order to be made "secure", so as to prevent access to unintended files. I am happy to report that XEDIT can be made secure without mods, due to its programmability with macros, but with a steep penalty in terms of performance and loss of function. Some very simple changes to XEDIT would allow more security with fewer unfortunate side effects.

Jumping ahead to 1980 in this mini-history of VMSHARE, we can observe that the database had become fairly large - over 400 MEMO and PROB files. (There are more than 1500 today.) It was becoming difficult, to say the least, to find specific information one wanted. The solution was simply an index. I decided to include in the index all but a few hundred common "noise" words, so that just by specifying the term or terms you wanted you could get a list of all files that contained them. This kind of an index is a very simple thing to implement in a higher-level language using VSAM or a similar indexed file access method. You simply provide for one record for each term to be indexed. The record is of variable length and contains a coded list of all files that reference the term.

Well, CMS "supported" VSAM, but it was not the kind of support I wanted. CMS VSAM, as you all know, uses code kludged in from DOS. It does not fit well in the CMS architecture; it is slow and inefficient; it requires one to learn a new language just to create and manage files; and it is incompatible with the CMS file system. In short, it makes the application developer's life miserable. I was not about to do what was a "for fun" project using "CMS" VSAM. Fortunately, I was not alone in my distaste for

this facility. Others at Tymshare held this view also, so we had undertaken to develop our own native CMS indexed file system, and to provide ISAM and VSAM emulation with it.

Thus, as an application developer, I was able to implement a keyword index in VMSHARE with very little fuss or bother. Indexed access methods are really nice for doing large data management tasks. IBM owes it to CMS application developers to provide a really good one. From an architectural point of view, the access method should be very well integrated into the rest of CMS and use the native CMS file system, so that it's very easy to use, like the rest of CMS is (or should be).

By 1980 VMSHARE had become fairly popular within the VM Group of Share - so much so that people were finding it difficult to login to the single userid provided for that purpose. Although there were other reasons of an economic nature for providing only one conference login, it is apparent when you think about it that there is one overwhelming technical difficulty as well. That is, CMS provides NO WAY to share ordinary CMS files in read-write mode. Never did, still doesn't. This is NOT a satisfactory situation for application developers.

This was no novel insight even 5 years ago, so we undertook to correct this deficiency too. The process of implementing CMS file sharing taught us a number of lessons about deficiencies in the internal architecture of CMS as well. A lot of people have spent a lot of time thinking about the best way to implement file sharing under VM/CMS. One alternative is to move the file system into CP. This was actually tried at Brown University under CP/67. It has the drawback that CP is NOT the right place to put an application-oriented tool like a file system, for a number of reasons.

Every other approach to file sharing in VM/CMS that I have heard of uses a service virtual machine in some way or another. Some approaches use the service virtual machine primarily for synchronization control and pass most of the data through a writable shared segment. This approach is probably viable, but very tough to implement with good data security and integrity. The easier approach is to use the service machine for everything - both synchronization and data management. I contend, and will explain later, that this approach is in an important sense the RIGHT approach, in spite of the high overhead of intermachine communication.

In any case, whether or not it is the right approach, it is the EASY way, so that's what we chose to do. In our file sharing implementation, all files to be shared are accessed and managed directly only by a service virtual machine. A new command, called CONNECT, is provided in CMS to replace the ACCESS command. CONNECT requests the service machine to grant access to a certain collection of files (residing on a normal minidisk), and to set up internal CMS control blocks as if the minidisk had been

accessed normally. Most CMS file system functions have been modified to recognize the special case of a "connected" disk and simply pass the request on to the service machine. A few functions like LISTFILE did not require modification at all, since they operate entirely off of control blocks.

One of the beauties of this scheme is that applications need very little, if any, modification in order to use it. You simply replace one ACCESS early on with a CONNECT and you're in business. Thereafter all of the shared files appear to the user as if they were present on a linked minidisk. The file server even notifies the user machine of any changes that occur to the files being shared. The proof of the relative transparency of this mechanism to applications may be found in the fact that VMSHARE required very few changes in order to use it. The changes that were necessary were almost all in the area of synchronization. Conceptually, there just isn't any way around the problem that if two different users of a file can modify it simultaneously, then they had better obey (or be constrained by the application to obey) some sort of locking protocol.

The rest is history. In February of 1981 concurrent access to the VMSHARE database became available. No changes to the user command syntax were required. Users were not even aware there had been a change. In spite of the fact that there are real performance implications of using a service machine to do data management, little if any performance degradation could be noticed by users, partly because conferencing isn't really that data-intensive an application. Mostly it's terminal I/O.

But back to CMS. Communication between user and service virtual machines is done with VMCF. (IUCV wasn't then available). It is notorious that IBM never supplied decent macro-level support for the use of VMCF in CMS, much less any higher-level language support. Finally, in Release 3 of VM/SP, IBM has provided some macro facilities for IUCV, but they fall far short of what is needed. Partly, this is because of specific architectural deficiences in CMS itself which have not been addressed.

The most noteworthy of these deficiencies is multi-tasking. There are those who think that implementing multi-tasking under CMS is a misguided attempt to force inappropriate MVT/MVS-ish constructs into CMS, further subverting whatever remnant is left of it's architectural purity. I hope anyone who might feel this way will reconsider. In the first place, multitasking is absolutely essential for implementing most service machine applications which should be capable of responding quickly to many simultaneous users. IBM-developed service machine applications like Smart and Passthru have had to implement their own private versions of multitasking. A large number of user-developed applications have had to do likewise, all redundantly and incompatibly. Quite simply, multitasking must be a system primitive.

But even in the normal user, as opposed to service machine, environment multitasking is necessary. It is needed in order to implement the concept of "any command from anywhare" called for in the Share VM/CMS Task Force report. That is the notion that any user of an interactive system may legitimately need to be doing several distinct things (tasks) simultaneously - like consulting an appointments calender while a compilation is under way, and trying to get HELP information too. We cannot conceal such possibilities from our users for much longer - not when they have micros on their desks like Lisa or an IBM PC with Concurrent CP/M-86.

Let us return to the discussion of deficiencies in the CMS application execution environment by picking up the thread of the history of that typical end-user application we have been following, VMSHARE. As you recall, the system was originally written in CMS Execs. Although this approach was excellent for prototyping and rapid application development, it was less than ideally efficient. Resource consumption and response time were both too large. Therefore, in order to produce a more efficient production system and to lay the foundations for future enhancements, it was decided to re-implement the conferencing system in a non-interpretive language. The language chosen was PL/I, but the story would be the same in most any other language.

There is both good news and bad news in the story. The good news is that CMS is very application-friendly in many respects. Quite a few system commands can be called directly (using a simple assembler-language interface) from application programs. Even Execs can be called from applications, provided they don't in turn invoke any of the "wrong" commands, and you watch out for a few Gotchas. CMS interfaces to the file system and the user terminal are also easy to use, again with simple assembler-language subroutines. (It is, perhaps, unfortunate that access to such operating system facilities is not in fact defined in any of the higher-level languages. But another school of thought holds that such operating system dependent functions should be confined to subroutine libraries anyway. IBM hasn't seen fit to support either approach for CMS.) The net result for VMSHARE was that it was very simple to recode the "kernel" of the application in PL/I and to continue to use the same operating system facilities in almost the same way as in the Exec version. Further, many of the parts of the older Exec version could continue to be used in the PL/I version until the developer got around to rewriting them. A sizeable number of functions have not yet been converted, nor does there seem to be much need that they should be, given a relatively low frequency of use.

On the other hand, the bad news is that CMS can also be very application-hostile in many ways. The most egregious of these ways lies in the fact that CMS support for relocatable load modules has always been very poor. Until the LKED command became available, support was really nonexistent. Today, the typical CMS command or application program in executable form is still a

core-image module. If you want to arrange for one program to call another in a separate module, you have several choices, all disagreeable. You can make one program run in the transient area and one in the user area. You can write the program to be called so that it is either adcon-free or relocates its own adcons, and then load it yourself into some handy storage you have acquired. You can talk your friendly local systems programmer into defining a new shared segment just for your application's modules. Or you can process it with the LKED command into a LOADLIB and then call it with OS simulation support. But you CANNOT do what you want to do simply by using native CMS facilities. We may well ask: just why has IBM stopped short of letting LKED produce "loose" module files and letting CMS fetch them with its standard module loading facilities?

Then there are a few other Gotchas. Like the CMS Executor which delights in releasing your application's dynamic storage or its STAX and SPIE exits. Or the various and sundry programs that have their very own STRINIT calls to reinitialise your application's storage. Then there is the lack, already alluded to, of any central command handling mechanism that provides the various services of parsing a command line, resolving aliases, determining whether to run a module or exec or nucleus command, setting up plists, calling the command, issuing pertinent error messages, and cleaning up after the command. Well, yes, SUBSET does some of that, but it can't be entered recursively and won't run a command in the user area because of the relocatability problem. Consequently every application winds up implementing its own version of this package of services, subtly different from any other implementation of course.

Naturally, at Tymshare, since we fancy ourselves to be civilized folk who are too lazy and uncreative to reinvent a system function every time we need it, we implemented relocatable module support in CMS back in 1977. We thus take for granted that if an application like VMSHARE needs to call a system utility like COPYFILE or some more esoteric processing program like Script or a spelling checker or a communications package, then VMSHARE just calls it and CMS makes the right things happen for program loading and storage management.

There are other, more subtle benefits to having relocatable module support around. Namely, it is easier to implement new system function as well as new application function. Many of the things I have already mentioned, like our indexed access method and our VMCF support, were much easier to put into CMS because they could be isolated in relocatable modules. Much easier to test too. This use of relocatable modules is very similar to the "nucleus extension" support of VM/SP Release 2. Since relocatable modules that are serially reusable are automatically retained in storage, unless explicitly purged, they are in effect permanent nucleus extensions. Except that the mechanism is all automatic, and there are no cumbersome NUCXLOADs to do.

We have all learned that the modular approach is the right way to develop large, complex applications. Developers should write small building blocks that are relatively independent, and only brought together at a very late stage, execution time being the best. This is easier on developers, as well as resulting in more powerful and more robust applications. Unfortunately, in making it difficult for one module to call another, CMS hasn't got the word yet. Let's hope it does soon.

Since I've said so much about various enhancements we've made to CMS at Tymshare, let me close this unfortunately long survey of the deficiencies in the application execution environment of CMS by mentioning something we haven't done, and what the consequence is. What we haven't done that CMS badly needs is device independent full-screen I/O. The consequence is that VMSHARE still has no full-screen mode of operation. I just haven't felt like getting down on my hands and knees and coding all the diagnose 58's that would be required, to say nothing of all the bookkeeping necessary to keep track of all the possible full-screen terminals that should be supported.

What is wanted is an interface that lets the application developer call the operating system to determine the capabilities of the terminal on which the application is executing, as well as specifying what minimal set of capabilities the application requires. A system call to define virtual screen and window sizes. A system call to update or replace the contents of a window. Perhaps even some rudimentary graphics support, like defining special characters with bit maps. (Next year I'll ask for more sophisticated graphics, so watch out.)

IBM has taken note of the importance of full screen facilities in such products as ISPF. While panel managers are good for many kinds of high-level application development, they are just too slow and clumsy for programming detailed interaction between applications and screens that have to update unpredictable portions of screens in a data-dependent fashion. In effect, what you need is, at least, to be able to treat each character position as a separate "field". And this support needs to be implemented as operating system primitives that can be used easily from any application.

Enough emphasis simply cannot be laid on the observation that increasingly effective human-computer interaction depends on very high bandwidth between the computer and the terminal - and on the ability of applications to control in detail the elements of that dialog. Microcomputers with bit-mapped display devices are far out in front of maniframes with their clunky old 3270 screens in this regard. But even though the hardware technology is available (e. g. the 3290), if the operating system primitives aren't there, applications can't use it without always rebuilding from scratch.

5. THE FUTURE OF INTERACTIVE COMPUTING

The future you shall know when it has come;
before then, forget it.

Aeschylus

When half the people believe one thing,
and the other half another,
it is usually safe to accept either opinion.

Edgar Watson Howe

There is nothing
that a New Englander so nearly worships
as an argument.

Henry Ward Beecher

In order to properly assess the future of CMS and to attempt to pronounce our verdict on the direction its development ought to take, we need to go back to the essence of it: CMS is an operating system for interactive, personal computing. I take it, therefore, as an axiom that whatever directions we propose for CMS should be conceived with the objective in mind of enhancing its ability to support interactive, personal computing, rather than, say, other types of computing, such as batch or transaction processing. Some will disagree and point out that CMS can be used for these other types of computing as well. That may be true, but I will hold to my axiom on the grounds that interactive, personal computing, where CMS has been all this time, now represents the most important area of computing in general, and will only continue to increase in importance.

This being said, we must confront the fact that there is now another option for the support of interactive, personal computing, an option which did not exist, or at least did not seem real, even five years ago. I mean personal computers, of course.

There is a great debate underway, a debate which will not be resolved and which will be with us, at least through the end of the decade. That is: should interactive, personal computing in large organizations be primarily supported on large central timesharing systems, or on distributed personal workstations?

To better deal with this issue, let me be clearer about what I mean by interactive personal computing. Quite simply, I have in mind just the sorts of things CMS has always been good for, the uses which have made it so popular, the things it is still predominantly used for today: program development, text processing, electronic mail, personal electronic filing, ad hoc database inquiry, interactive data analysis, modeling, and simulation, problem solving. These are the applications that IBM salesmen rely on to sell VM/CMS computers. They are also applications that are now mostly well within the range of today's personal workstations. Although there will always be other applications, or instances of these applications, that are beyond their capabilities, for most people in most organizations, personal computers will suffice quite well.

So the question is: how should one choose? Let's look at some of the advantages and disadvantages of central vs. distributed computing.

Advantages of central computing

* Very large amounts of data may be stored.
* Much larger problems, models, etc. can be handled.
* Data is more easily shared among many users.
* Software maintenance and distribution is easier.
* User collaboration and software sharing is easier.

Advantages of distributed computing

* Users have more control, so that hardware and software can be more easily tailored to individual needs.
* Incremental expansion of capacity is easier.
* Workstations have more predictable and often better response time.
* Personal workstations have better availability.
* Personal workstations permit a higher data bandwidth between the user and the application.

Let me admit right now that I am not going to give a balanced discussion of these points. I fully concede all of the advantages I have listed for central time-sharing. I think they are obvious and speak for themselves. But I view them as challenges, as problems that computing on distributed personal workstations will gradually aspire to address. Instead I choose to elaborate on the advantages of distributed computing, because I sense they may not be as fully appreciated by my present audience. Were I addressing a microcomputer industry audience, the situation would be completely reversed, the arguments for distributed computing would seem both compelling and self-evident, and I should probably elaborate on the advantages of central computing.

Response time is clearly a factor that cuts both ways. The nature of the application will ultimately decide whether central or personal computers can deliver better response time. For trivial interactions, however, I think it is plain that personal computers have the decided advantage in that they are not burdened with time-slicing, scheduling, paging, or communication delays. For non-trivial interactions, of course, the situation is otherwise. Large mainframes can deliver 5 to 10 MIPS of instantaneous compute power, perhaps 25 to 50 times what the "average" micro can do today. This is impressive if you have access to a large mainframe that is relatively unloaded. But realistically, if you are using a shared system with an expansion factor of 20, for example, and your computation really is non-trivial so that it requires many time slices, then your advantage is only 1.25 to 2.5 times, because you have only .25 to .5 MIPS available. Or look at it this way, even more pessimistically, if you are on a 10 MIPS system with 400 other users, that's only .025 MIPS for you. Fortunately, most of those other 400 users are on a long lunch break.

Until they all come back to work at 2pm. That raises the consistency issue. The response time of a standalone micro is nothing if it's not consistent. This is not reassuring if the response time does not happen to be satisfactory to you, but if it does suffice for your particular application, it's very nice to know you'll never be caught in a computer traffic jam just when you are anxious to get your work done. And let's be humble about this: I think that most interactions of most people with their computers are fairly termed "trivial". Certainly this is true for the common activity known as editing. I'm composing these lines on my personal computer now because there's no way I can get this kind of response out of my mainframe, even now at 11pm. The power of even a personal computer for this kind of interaction is overkill - that's what's so nice about it.

What is likely to happen in the future with regard to response time? Well, we all know that the speed of large, general-purpose mainframes isn't increasing all that fast. IBM is quite frankly moving to multi-processor architectures which do not give users any improvement at all in instantaneous compute power available. And DP managers will just tend to put more users on shared systems in proportion to their total capacity. That alone would erase any possible advantages for the individual user. But even worse, existing operating systems become increasingly less efficient the more simultaneous users they have to handle. (And history teaches that future operating systems will be even less efficient.) How many times have you seen a new CPU come in that theoretically had twice the power, but could handle nowhere near twice as many users? Yet at the same time, the opportunity for price-performance improvements in microcomputers is immense. The average micro today has an 8/16-bit CPU, while full 16-bit chips are also common, 16/32-bit chips are readily available, and 32-bit CPU chips are being manufactured. Note that IBM has entered into an agreement with Carnegie-Mellon University that

calls for 32-bit workstations with .5 to 1 MIPS by 1985.

Reliability and availability have always been significant issues to computer system managers and users. Great improvements have been made over the years, to the point where 99.5% availability is an attainable goal. But users' expectations have grown proportionately, as has the importance to the enterprise of continuous availability. More and more data is stored in our systems, and more and more people depend on being able to get at that data. The new users of interactive computer systems tend to be increasingly intolerant of service interruptions of even a few minutes duration. Today the average user of a shared computer system may experience a service interruption perhaps once a week - if he has a hard-wired terminal. If he should be so unfortunate as to have to access the system through a communications network of any kind, interruptions once a day are not uncommon. And the telecommunications system in the United States is quite reliable relative to the rest of the world.

By contrast, users of the present generation of personal computer may experience a service interruption only once a year, at least as far as their personal equipment is concerned. If and when an interruption does occur, the user, with any luck at all, can find an equivalent workstation somewhere else in the office that will let him get his work done, especially if it has any kind of urgency. I think that this reliability and availability of personal computers comes as a pleasant surprise in our present age of deterioration in the quality of many features of our social and material environment. It is a clear advantage of personal computers over shared systems, and it has only just recently begun to be perceived.

The last advantage of personal workstations that I want to discuss is the higher data bandwidth that is possible between the user and his application. I think it is also the decisive advantage. It is, furthermore, relatively new. Early microcomputers often used ordinary asynchronous terminals, connected at perhaps 19.6K Baud. Now bit-mapped displays are common, and they can be rewritten in as little as 1/30th of a second (depending on the software and support hardware). This makes a world of difference.

The significant advantage here doesn't lie in being able to display masses of data more rapidly. After all, most people can only read at about 800 Baud anyway. The advantage lies instead in the powerful new ways you can manipulate the screen, which is the user's principal communication path to his application. And the reason you want more control over the screen is to provide better human factors.

Imaginative use of the screen to provide a better user interface can be done in many ways. As discussed above, applications can be made to be self-describing by providing numerous cues and clues on the screen in the form of labels, prompts, and help

information. This kind of information must be changed rapidly, perhaps as often as every keystroke, to reflect the changing state of the application. Menus can be provided that scroll and change quickly, yet occupy only a portion of the screen real estate. You can offer multiple windows that allow the user to keep track of the multiple processes he may well be trying to control (remember multitasking?).

To really understand the possibilities of bit-mapped displays and character-at-a-time interaction you really need to see them in action with well-designed software. It may be as simple as a "what you see is what you get" editor, in which words are repositioned when they extend outside the margins. Or a spreadsheet program that uses highlighting to help the user locate the cursor

Not the least of the advantages of bit-mapped displays is the opportunity of communication through graphics. The picture IS worth a thousand words - the information transfer rate possible with graphics is much higher than with text, when it is appropriate for the application. In this way you overcome the objection that people can read text at only 800 Baud.

I think that we have barely begun to appreciate the possibilities that bit-mapped displays under direct CPU control offer. For example, we will see increasing use of animation, at least in limited forms, to convey information. This not only has mnemonic value, but can even serve to better hold the user's attention and make the application more "fun" to use. It has been pointed out that one of the reasons for the surprising success of Visicalc is the fact that it is just plain interesting to watch; the user receives immediate feedback and confirmation for every step in a way that supports his problem-solving. [3]

Note that it requires a lot of CPU power to take advantage of bit-mapped displays. This is a valid way to use CPU resources, and it is easily within the capacity of today's micros. But even if you had the bandwidth to drive hundreds of bit-mapped terminals from a shared system, this would impose a heavy new load on the host CPU.

Interestingly enough, a factor which does not seem to favor either personal workstations or central shared systems is cost. It has been estimated that a single user can be supported on a large, shared VM/CMS system in a research environment for about $7000 to $10000 per year, with relatively good average response time guarantees. Assuming a 3-year pay-out period, you could equip the same user with a workstation and peripherals worth $21,000 to $30,000. That would, by today's standards, be a pretty high-performance personal computer! While it would still not have the same instantaneous memory or compute power as the mainframe, it might suit the user's requirements quite well. Or, you could provide the user with a much cheaper personal computer and a certain amount of shared system resources when required for large

jobs. There just do not seem to be economies of scale in computing anymore. This may be due in part to the much greater sophistication required in the large shared system in order to handle hundreds of users, and the major inefficiencies of a multiuser operating system.

As I warned you earlier, this does not pretend to be a balanced discussion of the relative merits of central and distributed computing. Instead, my aim was to make the point that distributed computing with personal workstations is a very serious contender for the title of preferred architecture to support interactive computing. There will, to be sure, continue to be many instances when large central mainframes are required to handle heavy computing or data management demands. But in at least as many instances, distributed workstations will be the way to go.

Realistically, I am sure that both models of interactive computing will not merely coexist throughout this decade. They will be used together, as we learn how to integrate them. And gradually they will merge, until we are little aware of the distinction. The question, therefore, is: what does this have to do with the future of CMS?

6. ONE POSSIBILITY.

Future, n. That period of time in which our affairs prosper, our friends are true, and our happiness is assured.

Ambrose Bierce

People live for the morrow,
because the day after tomorrow is doubtful.

Friedrich Nietzsche

I never think of the future.
It comes soon enough.

Albert Einstein

We've all been hearing for years now, it seems, that before long we could have the power of a 4341 in a desktop computer. I think that may be somewhat of an exaggeration as far as the next year or two are concerned, but something a little less powerful is a distinct possibility. Tom Rosato told us just a year and a half ago of his experiments with the Micro CMS/370 project, in which a modified version of CMS has been run on a micro computer with a Motorola 68000 CPU and a 370 instruction set emulator. [4] Although this project was in the "experimental" category, it seems clear to me that once you have a personal computer capable of executing or emulating the 370 instruction set, then CMS is a very natural operating system to run on it. After all, CMS has been a dedicated single-user operating system ever since its earliest days in the Cambridge Scientific Center when it could run stand-alone on a 360/40. A CMS virtual machine has always been a personal computer.

Let us then suppose we have CMS running in a personal workstation. In what ways should CMS be enhanced for this environment? I think the answer is that we will still want remedies for all of the deficiences which we have been examining at such length. We will surely want very good facilities for handling bit-mapped displays, because it would be such a waste of good technology to live with 3270 emulation for long. We will quickly want all of the improvements to the user interface which are possible with better display technology, like multiple window management and support for self-describing programs. We will also want the structural and architectural changes in CMS that are essential, or at least very helpful, to providing the new end-user and application support: multi-tasking, relocatable modules, indexed file access method, better command handling.

What else? How about file sharing? Does that make sense in a workstation environment? Of course it does. Let's go back to the advantages we enumerated for central computing. There are really just two categories of those advantages. One category is a matter of scale, the fact that mainframes can handle more data and larger problems. The other category is sharing between users - of code, data, special peripherals, or whatever. Now, since I think that the provision of interactive computing via personal workstations is certain to occur, I also think that ways will be found to overcome as many as possible of the disadvantages this approach has relative to large shared systems.

To make a long story short, local area networks with specialized network servers to support sharing of files, peripherals, network gateways, and the like can be expected to address this requirement. In this approach, all of the participating workstations are interconnected together with high speed communications (1 megabit per second or more). Requests for a particular shared resource are passed from the individual workstation to the relevant server machine on the network. The server processes the request and replies to the requestor or returns data when it is done.

Does this sound familiar? It should, because this is exactly how our VM/CMS systems have evolved in order to provide all kinds of sophisticated services to our users. We are all familiar with how successful various products like VM/Passthru, Dirmaint, Profs, RSCS and so forth have been in implementing function in service virtual machines. We can all think of dozens of other examples in our installations and elsewhere of how service virtual machines are used to provide essential functions like mail systems, network gateways, plotter support, and file sharing. The ONLY difference is that in VM currently all of the users and service machines are virtual machines on a shared host, and they communicate via VMCF or IUCV. On a local area network the users and servers have distinct computers and communicate using the network protocols. But from the viewpoint of how these applications are implemented, the differences are almost irrelevant.

In particular, file sharing is almost the first application implemented on any local network. And the way it is implemented is precisely the same as the easiest way to do it in VM. That is, you have one or more server machines which manage the files. You either patch the user's operating system or else provide new interfaces so that all file I/O requests are routed to the server machine. The server machine interprets and performs the request, and returns the data to the user's machine. In addition, the server machine will optionally maintain a distributed file directory by informing all users when changes occur. Notice that this description applies without any differences at all to both VM and LANs. This situation is why I asserted above that service virtual machines are, conceptually, the right way to implement file sharing.

It should also be remarked that in a local network, many of the

performance penalties associated with doing file sharing under VM simply do not apply, because you do not have to worry about task scheduling overhead, and the CPU resources required to manage file sharing are not preempted from users who aren't using the facility. At the same time, the server machine can still be a bottleneck, since it's likely to be single-threaded, and you need to have bandwidth in your network in proportion to the amount of shared file usage you expect. There's no free lunch.

Another problem with LANs that I will freely admit is that as soon as you come to depend on the network and its various services the overall availability of your total interactive computing environment is much more in jeopardy. Local networking is complex and relatively new technology. There's a lot of room for failure. But I believe that improvements will be made because they have to be. And even if there is a network failure, you can still use your workstation in stand-alone mode if you are able to work on another suitable task. You are even better off if just one of the network servers goes down. Because when a central shared system is down, it's ALL down.

7. SUMMARY.

As for the Future, your task is not to foresee,
but to enable it.

Antoine de Sainte-Exupery

No person who is not a great sculptor or painter
can be an architect.
If he is not a sculptor or painter,
He can only be a builder.

John Ruskin

The brevity of human life gives a melancholy
to the profession of the architect.

Ralph Waldo Emerson

All of us know that computer technology is advancing faster now than ever before. But we tend to think of this advance in terms of faster, cheaper, larger, or denser: faster CPUs, more real memory, larger storage devices. You can now buy 5.25" Winchester disks that hold 140 megabytes of data. Next year it will be 390MB. A megabyte of RAM storage for a personal computer can be bought for less than $800, retail. You can today purchase a 32-bit workstation whose CPU chip contains 450,000 transistors. These are the quantitative changes.

However, there are equally significant qualitative changes occurring as well. Voice recognition and voice input devices usable with a personal computer will be available soon. Whole operating systems will be etched into silicon chips. Artificial intelligence programs that can process natural language and implement human expertise are commercially available. Bit-mapped CRT displays are now standard on personal computers. And they are enabling new kinds of man-machine interfaces that help their users to understand what is going on much better and hence to be much more productive. Local area networks are making it possible to entirely change the way that computing services are delivered.

This is gee-whiz stuff, but it's also real, very real. It WILL change the way we view computing.

How will CMS adapt to these changes? Or rather, simply, will CMS adapt to these changes? I'm going to pass on those questions. It's better to prophesy after the event, and the best prophets are merely the best guessers. Most importantly, what happens with CMS is largely up to its users.

Many of the things we want in CMS can be had within the current

architecture. We can have a native indexed access method, relocatable modules, padded cells, session management, good help facilties, or file sharing. I know these things are possible because they've been done. Neither do there seem to be any conceptual problems with support for bit-mapped display devices. The current architecture suffices.

But I don't think the current architecture will suffice for some other things we want as well: centralized command handling, general multitasking, hierarchical file system, support for 31-bit virtual memory. I don't think that these can be added adventitiously to the current architecture. Or at least, they shouldn't be. After nearly 20 years, it doesn't seem to be too much to ask for a thorough rewrite of CMS, from scratch if necessary, making provision for the new requirements we can perceive now. The architecture doesn't really need to change all that much. But a whole lot of its implementation certainly does.

So there are basically three alternatives: Keep adding incrementally to the present CMS and remove some of its deficiences. Rewrite CMS, bringing its architecture up to date but without drastic change, and remove most of its deficiences. Or find another interactive system.

It's up to you.

8. REFERENCES.

(1) Fred Jenkins, et al., "SHARE VM/CMS Task Force Final Report", SHARE, Inc., October, 1980.

(2) Charles Daney, Larry Graziose, Pat Ryall, "CMS Restructuring", 1982.

(3) Paul Heckel, "The Elements of Friendly Software", QuickView Systems, 1982. (Early version serialized in Infoworld, starting July 12, 1982.)

(4) Tom Rosato, "Micro CMS/370 (Cambridge nano System)", procedings of SHARE 58, 1982.