THE GMIS 2 ARCHITECTURE

Jerry Lamb, Robert Selinger, & Chat Yu Lam

Energy Laboratory Working Paper MIT-EL 77-014WP

June 1977

Abstract

This document describes the software details of GMIS 2 as it is implemented on the VM/370 system.

Table of Contents

I. INTRODUCTION

This document is a detailed description of the GMIS 2 communication and system operation routines. It is intended for the programmer who desires an in-depth description of procedures used for multiple virtual machine communication among the GMIS 2 virtual machines. Readers should be familiar with basic GMIS operation and additional reading material is listed in the bibliography.

Section II presents an overview of the GMIS configuration and describes the various functions of the different virtual machines. These virtual machines are: the manager virtual machine, user virtual machines, communication virtual machines, and the data-base virtual machines. Section III presents a summary of the Virtual Machine Communication Facility (VMCF) as utilized by the GMIS routines. Section IV presents a detailed view of the GMIS System including a functional flow diagram of the system operation. Section V describes the various utility routines used by all GMIS 2 virtual machines. These routines are either used for inter-virtual machine communications or for performing special tasks. Section VI describes the special GMIS system programs unique to each type of virtual machine.

In describing each program we emphasize the associated data structures and mechanics of its operation, rather than elaborate on its logic which can be obtained from the program listing. This document is intended to provide a guide for the GMIS system programmer to update, maintain, and change the software as the need arises.

II. OVERVIEW OF THE GMIS CONFIGURATION

The configuration of virtual machines used in the present GMIS is depicted in Figure 2.1, where each box denotes a separate virtual machine. The blocks across the top of the page represent different user-oriented programs (modeling and analytical systems, editors, etc.) and the blocks across the bottom of the page denote different data management systems, each running on its own virtual machine (DBVM). A user may access any modeling system and request a connection to any virtual machine. A communications virtual machine (CVM) associated with the user's machine (UVM) provides the necessary communications interface between the user's analytical capability and the desired database system. With this configuration, it is possible for a user to access the modeling or analytical capability with which he is most familiar, even though it may be running under an operating system different from the other available modeling or analytical capabilities. Thus, the user is not required to learn new analytical capabilities.

In addition, since each virtual machine may run any existing model or program under its normal operating system, such a configuration eliminates the need to devote resources to translating application packages and programs between operating systems.

Furthermore, the GMIS configuration permits interaction between aplication languages and programs not originally envisioned by their developers. For example, an analytical package is greatly enhanced by having its data management capabilities extended. Hence, a user of the APL/EPLAN analytical capabilities, for example, may request data that is stored and managed by SEQUEL database management capabilities.

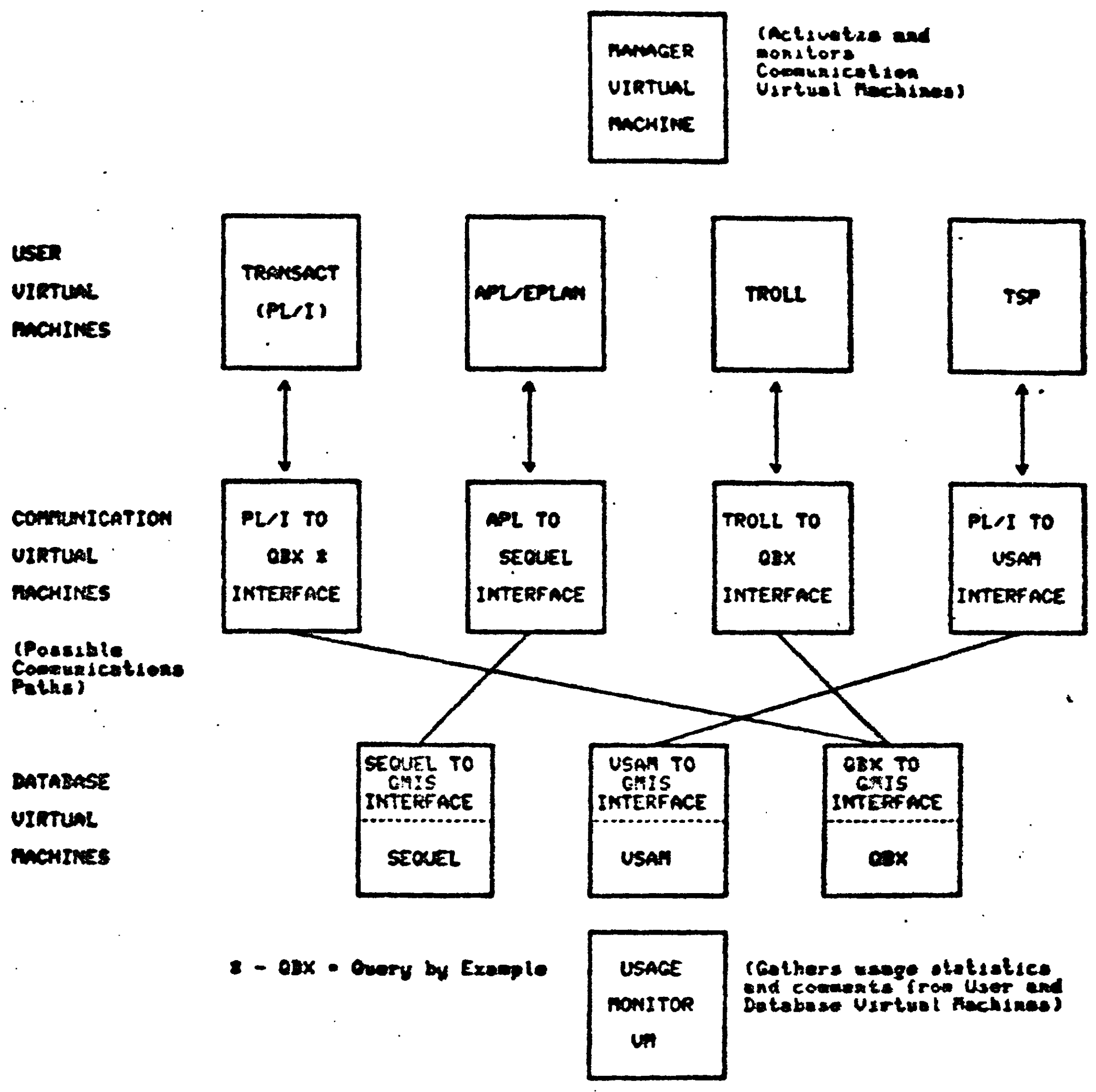


FIGURE 2.1 GMIS Architecture

Functions of the Virtual Machines

Functions of the Manager Virtual Machine

The primary function of the manager virtual machine (MVM) is to respond to user requests to create the connections between the virtual machines servicing that user by activating the necessary CVM's. The other function of the manager is to disconnect and automatically log out the appropriate CVM's once the user has finished using GMIS.

To accomplish these functions, several procedures were added to the user VM's and the manager VM. When a user logs into his user VM by sending a message to the manager VM (MGR) the message sending procedure uses the VM/370 Virtual Machine Communications Facility (VCMF).

1. He may request a communications VM
2. When the manager VM receives a message from a user, it:
    a. checks the VM ID of the send for authorization and,
    b. looks at the message sent by the sending VM.

If the message is to log in a CVM, then it will check to see if such a VM already running. If not, it automatically logs one in (note that the manager VM has Operator privileges, which permit it to log in other virtual machines). The manager also periodically checks all CVM's to see if they have "owners," i.e., if the user VM's are currently logged in. If a communications VM does not have an owner, the manager VM automatically logs off the user's CVM.

Functions of Communications Virtual Machines (CVM's)

The communications VM's provide mechanisms for user VM's to communicate with database VM's. Since each interface routine is custom-built to permit communications between a specific user environment and a specific database system, each user environment (e.g., PL/I, TROLL, APL) has a single standard communications interface written that uses the most efficient communication capabilities available in that environment. Likewise, each database system has a standard communications front end built for it which receives transactions in a format that requires a minimal amount of preprocessing on the part of the database machine. Any reformatting of transactions from the user or replies from the database system is handled by the interface routine which resides in the user's communications VM.

Functions of the Database Virtual Machines

GMIS provides users with access to an interaction relational database management system called SEQUEL, which has developed at the IBM San Jose Research Laboratory. In the near future, we will also offer the Query by Examply relational database system, which is being developed at the IBM Yorktown Heights Research Laboratory. These relational systems allow database transactions to be entered on line, and prepare replies to these transactions in the form of single valued results or tabular reports.

A database VM, regardless of the database management system running on it, receives transactions from the communications virtual machines belonging to different users, and stacks these requests in the order that they are received. Each request is processed (one at a time) by the database management facility, and the reply is passed back to the communications VM which sent the transaction request. After each reply is sent, the database machine selects the next request from the stack, identifies the sending communications VM, and processes the transaction.

III. Virtual Machine Communication Facility

This section contains a brief explanation of what VMCF does and a description of its use by GMIS 2. For GMIS 2, an assembly language program was written to interface PL/I to VMCF. The protocols for each entry point are independent of the messages or data that are being transferred and its interpretation. Unless there is an error in VMCF, the actual assembly language implmentation is invisible to the calling program.

For user environments and database systems which can be interfaced through communication modules written in PL/I or FORTRAN, the VMCF inter-machine communications mechanism is used for highly efficient signalling and message sending. Some of the user environments, such as TROLL and APL, do not normally permit the user to directly call external routines, and thus one must use a spooling mechanism utilizing virtual card readers and punches for sending and receiving messages.

VMCF Overview

The Virtual Machine Communication Facility (VMCF) is part of the CP component of VM/370. VCMF provides virtual machines with the ability to send data to and receive data from any other virtual machine.

Messages and data are directed to other virtual machines logically via the userid. The amount of data that can be moved in a single transfer is limited only by the sizes of virtual machine storage of the respective virtual machines.

VMCF is implemented by means of subfunctions invoked using the DIAGNOSE instruction with a code of X'68' and a special parameter list called VMCPARM. A VMCF subfunction is indicated by a particular subfunction code in the VMCFUNC field in the parameter list. A virtual machine must invoke the AUTHORIZE subfunction before it is allowed to use any of the other subfunctions. Once this is done it can send and receive data to and from other virtual machines which have also authorized VMCF.

A special external interrupt (code X'4001') is used to notify one virtual machine of a pending transfer of data. This interrupt is also used to synchronize sending and receiving of data. Along with this interrupt, the virtual machine receives a message header that is logged into a preassigned virtual storage area. This message header is used to define the type of request and to provide data transfer information, such as length of data. The message header is also used to notify the originator of a transaction of the success or failure of the transaction. In this case, the message header includes such information as residual counts and data transfer return codes.

Figure 3.1 lists the VMCF subfunctions and gives a brief description of each.

IXEVMC

To facilitate data transfer and signalling, the VMCF functions used by GMIS 2 are invoked via a high-level interface, IXEVMC. This interface is written in assembly language and communicates directly with CP and VMC via the Diagnose instruction, and contains the external interrupt handlers. (For a description of the parameters required by CP for VMCF (VMCPARM), see the VM/370 System Programmer's Guide.) The entry points in IXEVMC are callable from PL/I programs, require only a few parameters, and are described in Section V.

VMCF messages for GMIS 2 are composed of two parts, a header indicating type of message and a text section. The header consists of a message type and a code. The code is presently used to indicate to the CVM which module is desired. The text portion is used to pass instructions, queries, and data between VM's (Future plans call for the header to be included as an integral part of the VMCF's communication parameters.)

Function	Code[1]	Comments
AUTHORIZE	Control	Initializes VMCF for a given virtual machine. Once AUTHORIZE is executed, the virtual machine can execute other VMCF subfunctions and receive messages or requests from other users.
UNAUTHORIZE	Control	Terminates VMCF activity.
SEND	Data	Directs a message or block of data to another virtual machine.
SEND/RECV	Data	Directs a message or block of data to another virtual machine, and requests notification of a reply.
SENDX	Data	Directs data to another virtual machine on a faster but more restrictive protocol than the SEND subfunction.
RECEIVE	Data	Allows you to accept selective messages or data sent via a SEND or SEND/RECV subfunction.
CANCEL	Control	Cancels a message or data transfer directed to another user but not yet accepted by that user.
REPLY	Data	Allows you to direct data back to the originator of a SEND/RECV subfunction, simulating full duplex communication.
QUIESCE	Control	Temporarily rejects further SEND, SENDX, SEND/RECV, or IDENTIFY requests from other users.
RESUME	Control	Resets the status set by the QUIESCE subfunction and allows execution of subsequent requests from other users.
IDENTIFY	Control	Notifies another user that your virtual machine is available for VMCF communication.
REJECT	Control	Allows you to reject specific SEND or SEND/RECV requests pending for your virtual machine.

[1] The word "Data" in this column indicates a data transfer subfunction whereas the word "Control" indicates a VMCF control subfunction.

Figure 4.1 Virtual Machine Communication Facility (VMCF) Subfunctions

There are three message types:

M = message text follows

Z = log off associated CVM (manager specific)

X = switch to indicated interface module (CVM specific)

IV FUNCTIONAL DESCRIPTION OF GMIS

Each GMIS virtual machine communicates with other VM's through the VMCF interfaces described in Section III. Several routines, common to all GMIS VM's, are used to accomplish the sending and receiving of messages. Figure 4.1 shows the routines and data areas associated with this software. In addition, each machine has GMIS system programs specific to its functions. Figure 4.2 shows the entire GMIS communication configuration. (Note: All possible programs are shown, not core configuration at any specific time.)

The basic functions of each communication module are:

VMCINIT - Initialize the VM for VMCF communications

VMCTIME - Initialize timer interrupt - done only once

VMCWAIT - Place machine in a wait state for external communication or timer interrupt

VMCSEND - Send a message through VMCF

VMCRECE - Process a message received from VMCF after a call to VMCWAIT

VMCTERM - Terminate VMCF communications

IXEUVM - Returns a UVM ID to a calling CVM. The ID is read from disk as shown in Figure 4.1. The CVM needs UVMID for communication.

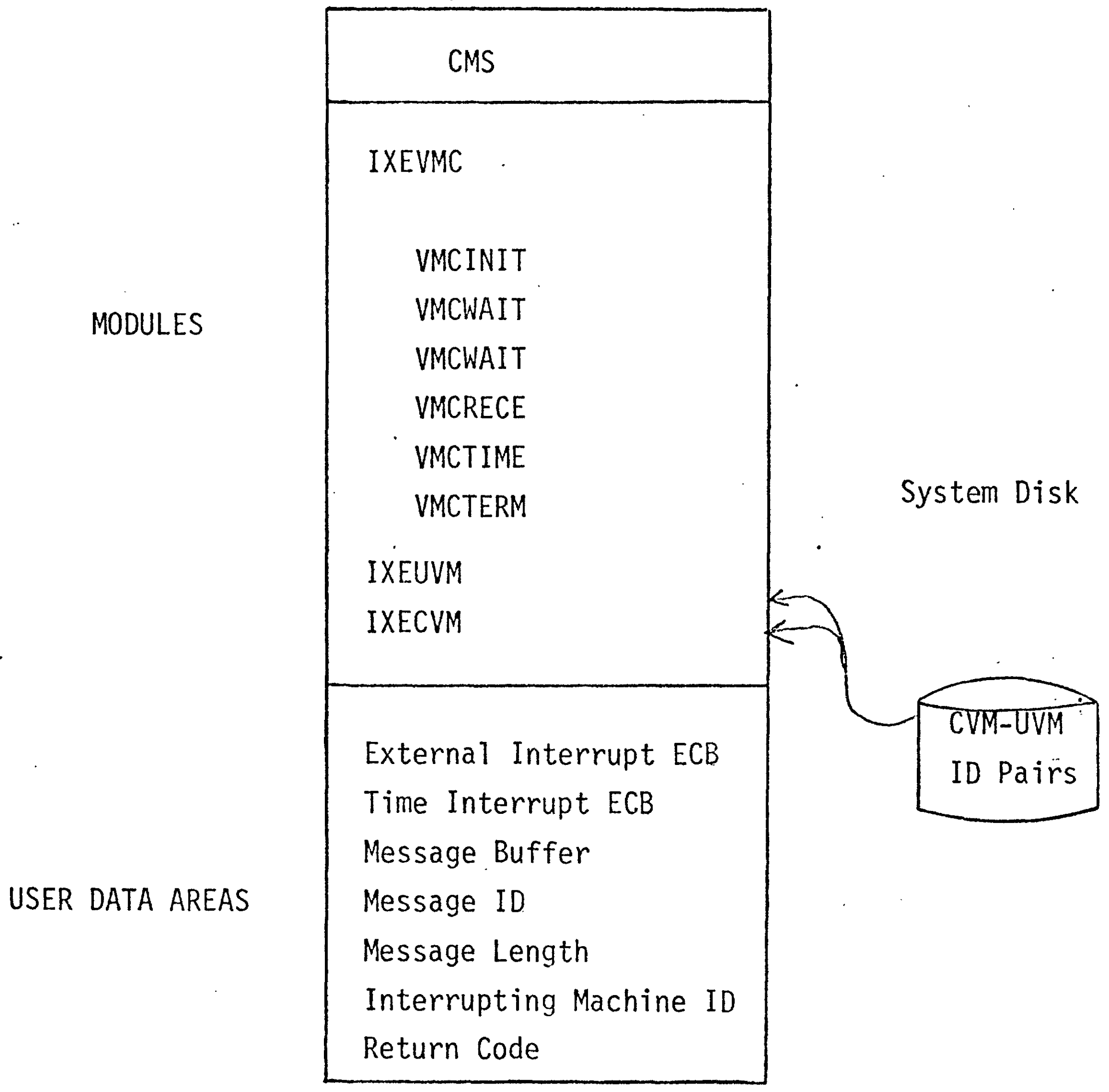
IXECVM - Analogous routine for a UVM to get a CMVID.

The basic functions of the programs specific to certain VM's are:

Manager VM VMMGR - logon on CVM when requested by UVM; terminate any unused CVM

User VM GMISINT - get special GMIS message for user and print it, get the ID of the CVM and request the MGR to initialize it, request CVM to load desired interface module (APL - SEQ or Pl/I-SEQ)

Communications VM CVMINIT - load appropriate interface module after initializing CVM


CMS
MODULES
IXEVMC
VMCINIT
VMCWAIT
VMCWAIT
VMCRECE
VMCTIME
VMCTERM
IXEUVM
IXECVM
System Disk
CVM-UVM
ID Pairs
USER DATA AREAS
External Interrupt ECB
Time Interrupt ECB
Message Buffer
Message ID
Message Length
Interrupting Machine ID
Return Code

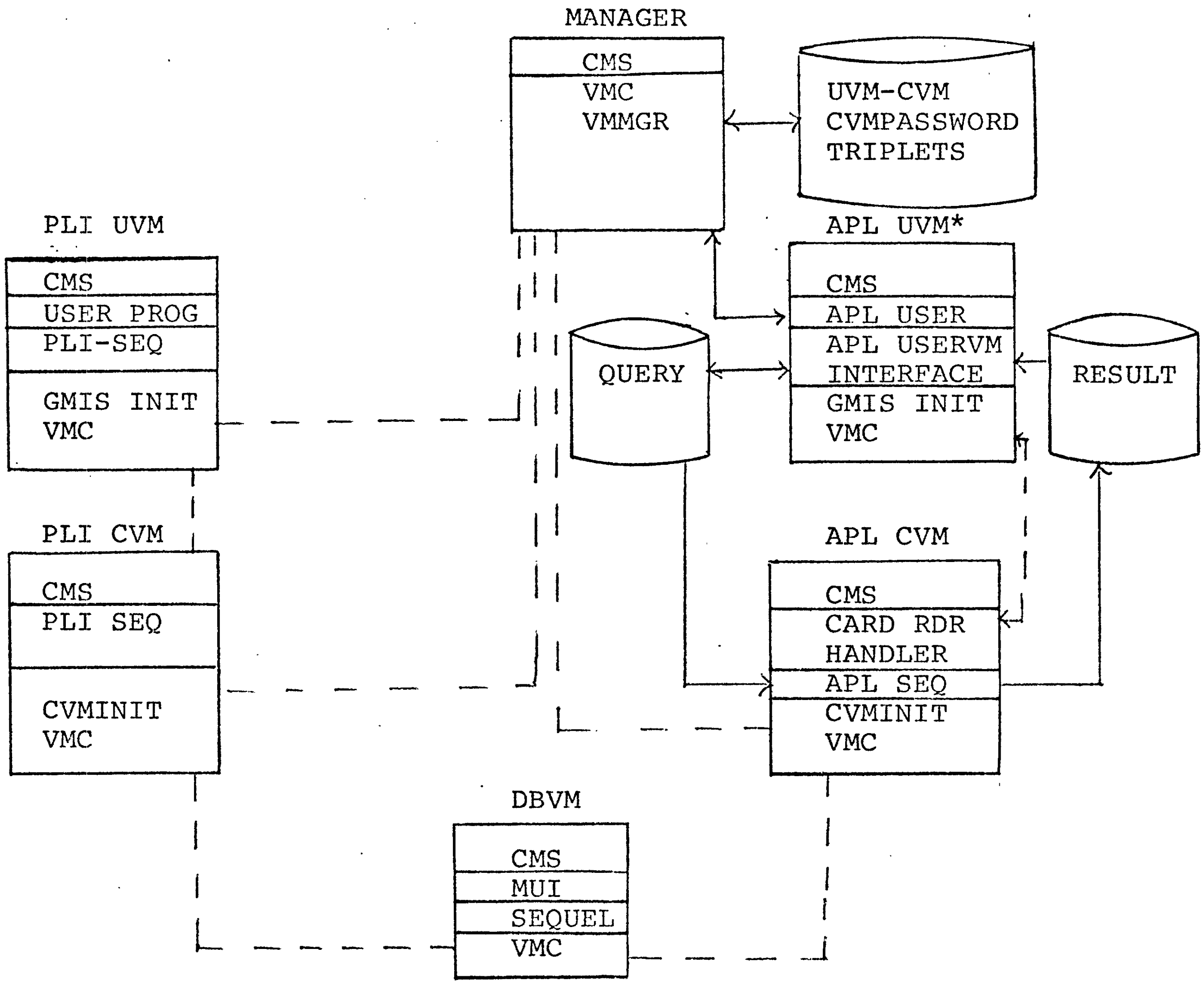


FIGURE 4.2 GMIS Configuration

————————	Non VMCF
— — — — —	VMCF

Figure 4.3 shows the sequence of operations, routines called, and messages passed to start up a PL/I UVM - CVM - DBVM link. PL/I was chosen because it is the general case that will be used for future expansion. The figure shows messages as

type, code   TEXT

Routines are assumed to perform the functions above unless specific functions are noted. Time flows generally from top to bottom in Figure 4.3.

```
MGR            UVM                CVM                 DBVM

LOGGED IN AT                                          LOGGED IN
IPL                                                   MANUALLY
VMMGR                                                 DBEXEC
  read triplet                                        VMCINIT
VMCINIT                                               VMCWAIT
VMCTIME
VMCWAIT
               LOG IN
               GMISINIT
                print msg
               IXECVM
               VMCSEND(MGR)
                m,0,'initial z'
               CP SL 5 SEC
VMRECE
  autolog cvm
                                  LOGGED
                                  CVMEXEC
                                   CVMINIT
                                   VMCWAIT
               CP QUERY CVM
                 fail=stop
               PUNCH SWITCH *
               VMCSEND(CVM)
                 X,2
                                  PUR RDR ALL
                                  VMCRECE
                                   load PLI-SEQ
                                  CP MSG CVM 'ready'C
                                  VMCWAIT
               USER PROG QUERY
               VMCSEND(CVM)
                M,B,'DBVMID QUERY'
               VMCWAIT            VMCRECE
                                  VMCSEND(DBVM)
                                   M,B,'QUERY'
                                  VMCWAIT             VMCRECE
                                                       process query
                                                      VMCSEND(CVM)
                                                       M,B,'result'
                                                      VMCWAIT

               * Both are sent in case CVM is in APL mode
```

FIGURE 4.3 Example of a GMIS Communication

V. Detailed Module Description of GMIS 2 Communication Routines

Communication Routines

Most system programs in GMIS are coded in PL1. The purpose of the Kernal routines is to provide efficient functions for interfaces to PL1 and basic VM system functions. All Virtual Machine communication routines are contained in one program - IXEVMC. These entry points are callable from PL1 programs to perform initialization of the communication environment and for the sending and receiving messages, etc. The detailed mechanics of each routine are described below.

VMCINIT

The function of VMCINIT is to initialize the virtual machine communication environment. It involves invoking the authorize subfunction of the VMCF; and setting up the external interrupt handler for intercepting virtual machine communication external interrupts. A timer interrupt handler may also be set up so that the virtual machine will get interrupted every t seconds, where t is a constant in the data area of the IXEVMC program. For virtual machines that do not require this facility, one can simply set the timer off, masking any timer interrupt.

The following example will illustrate the usage of VMCINIT:

```
DCL VMCINIT EXT ENTRY;
CALL VMCINIT (MYVMID);
```

EXTERNAL INTERRUPT HANDLER

As a result of an external interrupt, the external interrupt handler receives control. If the external interrupt code is X'4001' indicating a VMC external interrupt, the resultant action of the interrupt handler is to post an event control block (a 4 byte storage area for synchronizing events). Since the ECB is the same one that VMCWAIT (see below) uses in putting the virtual machine in a wait state pending an external interrupt, the effect of the posting is a wake up of the waiting process. The ECB is in the data area of IXEVMC which both VMCWAIT and the external interrupt handler share.

The result of timer interrupt is similar. In this case, a different ECB is used which is adjacent to the ECB used for external interrupts. The waiting process gets awakened resulting from either an external interrupt or a timer interrupt. Therefore, VMCWAIT will have to check the cause of the interrupt and perform the appropriate function.

VMCWAIT

This entry point can be used to put the current process into an enabled wait state pending either a VMC external interrupt, or if the timer is set on, a timer interrupt. The caller of this routine can also specify which virtual machine interrupt is to be acted on by setting appropriate arguments. VMCWAIT will wait until it is woken up whereupon it checks to see which type of interrupt caused the wake-up. The type of interrupt is indicated to the called process or routine by the return code in one of the calling parameters. For a timer wake-up, there turn code is set to a negative number; for VMC external interrupt, it is set to the return code from VMC, which is a positvie number. In the case of a VMC external interrupt, the VMID of the interrupting virtual machine as well as the source message ID and message length are copied from the data area of IXEVMC and returned via VMCWAIT.

Usage of VMCWAIT is illuastrated below:

```
        DCL VMID CHAR (8) INIT ('GMISCZ')
        DCL MSGID fixed bin (31);
        DCL MSGLEN fixed bin (31);
        DCL CODE fixed bin (31);
```

```
Call VMCWAIT (VMID, MSGID, MSGLEN, CODE);
if code¬= 0 go to Handle-timer
```

VMCSEND

This routine decodes the parameters, fills in the control information in a data area in IXEVMC used to request VMC functions and calls the VMC facility. The purpose of VMCSEND is to send a message to a specfic virtual machine. Each message has a unique message ID and this uniqueness must mb maintained. The following sequence illustrates how to send a message to another virtual machine called GMISCH.

```
        del sendvm char (8) init ('gmisch');
        del msg char (8) init ('HL WOOPZ');
        del msgadd pointer;
        del msgid fixed bin (31) init (1);
        del msglen fixed bin (31) init (8);
        del code fixed bin (31);
        msgadd = addr (msg);
        call VMCSEND (SENDVM, MSGADD, MSGLEN, MSGID, Code);
        If Code logical not = 0, then call ERR-RTN;
```

The user should check the return code. If it is not zero, then an error may have occured.

VMCRECE

This is simply the counterpart of VMCSEND. The usual sequence is to call VMCWAIT to wait for a particular virtual machine and on return , the arguements of VMCWAIT will contain the necessary information to invoke VMCRECE to receive a message. This information includes: the VMID, message id and message length.

The caller of VMCRECE has to provide a buffer with enough length into which the message received will be placed. The following will illustrate the usage of VMCRECE:

```
        Call VMCWAIT (vmid, msgid, msglen, code);
        allocate msg char (msglen);
        msgadd = addr (msg);
        call VMCRECE (vmid, msgadd, msglen, msgid, code);
```

VMCTERM

This routine is to clean up the inter virtual machine communications environment it is invoked by:

```
                    call VMCTERM;
```

The routine stops the VM from receiving or sending messages via VMCF.

A standard communication convention is used in decoding messages. Each message always consists of a 4-byte message header and a message text. The message header begins with a one byte message type, which can be the character M, indicating the message text is purely message text; or z, indicating to the VMMGR to log off the senders CVM; or X, indicating to a CVM to switch load to another interface module.

Following the message type byte is one currently unused byte, then there is a 2-byte message code. Currently the code can be fixed binary 1 indicating APLSEQ module is requested or fixed binary 2 indicating the PL1SEQ moduel is requested. Note that this code has meaning only when the message type byte has a 'X' character. The interpretation of message text will depend on the communicating agents and the communication environment. For example, VMMGR will only accept the character string M initial z in the message text from a UVM.

Utility Routines

These routines are efficient subroutines to perform very special functions. These functions include reading from a particular file, getting the time stamp.

IXEUVM

This routine is called by a CVM to obtain the VMID of the corresponding UVM. It reads a file which contains 16 byte records consisting of one UVMID -CVMID pair and returns the corresponding UVMID for the calling CVM.

IXECVM

This routine is very similar to IXEUVM. It returns the CVMID of the corresponding calling UVM. Both entry points have the same calling sequence. The accept as the first parameter, a pointer to an 8-byte area into which the resulting VMID will be placed on return.

Examples:

```
  del vmid char (8);
                                          when used by a CVM
  del ixeuvm entry external;

  Call ixeuvm (vmid);
```

<u>or</u>:

```
  del vmid char (8);
                                          when used by a UVM
  del ixecvm entry external;

  call ixecvm (vmid);
```

VI. GMIS 2 SPECIFIC VIRTUAL MACHINE MODULE DESCRIPTION

A. The Manager Virtual Machine

There are two main functions that the MGR must perform: autolog on user's CVM, and force off a CVM when the UVM which requested it is no longer using it. These are both provided in the PL/I routine, VMMGR, which is autologged when the CP system is ipled.

In order to know the CVM-UVM association and their account passwords, VMMGR when first logged on, reads from a file called 'UVMCVMID MAINT A1' which contians 24-bytes records, each of which is a UVMID -CVMID -CVM account password triplet. The file is encrypted and the manager reads it (decoded) from a second file and then destroys the second file. The only decoded version kept is in core. The end of the file is indicated by the terminating triplet; logon-logon-logon. Currently VMMGR can accommodate 100 triplets.
After reading the content of the 'UVMCVMID MAINT A1' file into its storage, VMMGR will then wait for any UVM to send him a message. The 'initialz' character string in the message text and 'M' in message type indicates to VMMGR to autolog a CVM.

When woken up by a timer interrupt, VMMGR will perform the second task (force off an idle CVM) by querying the list of CVMs and forcing off any CVM without an attached active UVM.

VMMGR should be on the logon list of VM/370 and is never logged off unless the system is brought down.

B. USER VIRTUAL MACHINE

The user virtual machine software consists of an initialization module - GMISINT and the various language dependent user virtual machine interface moduels, e.g. APL, TRANSACT, TROLL, etc.

GMISINT

This module provides a standardized user entry point to GMIS. It prints the GMIS message of the day and communicates with VMMGR to autolog its CVM before communications with the CVM begins. Basically, it performs the following functions in sequence.

1. read 'GMIS MSG C1' and prints it to terminal
2. obtain the MGR VM ID and the CVM interface module name from the parameters provided by its EXEC profile.
3. call IXECVM to obtain the CVMid
4. log on the CVM by requesting the MGR via VMCF to autolog it.
5. sleep 5 seconds to allow for the CVM to be autologged.
6. check via CP to see if CVM is logged.
7. spool and tag a punch file to CVM from UVM (in case the communication mechanism is via punch files).
8. punch a card to CVM <u>and</u> send a message to CVM via VMCF to ensure that the CVM gets the message to switch to the desired interface module. (Both are required since the CVM may already be in either mode.)

APL User Interface

The APL user interface is contained in an APL work space called SEQUEL. It performs 3 major functions: 1. initializes the communication environment, and 2. sends a query and 3. receives the result of the query and translates it into APL.

Initialization is performed by the APL function SEQOPEN. It basically does the following:

1. call getid to obtain the UVMid
2. read 'file CVMid C1; to obtain CVMid
3. read 'DBVM id B1' to obtain dbvmid
4 spool and tag punch file to CVM

Query is sent by using the APL function QUERY which actually consists of SEQPUT (sends a query) and SEQGET (obtains the results of the query). The b disk is used for communication with the CVM. Query results are stored on files by the CVM on the b disk in APL variable formats.

C. Communication Virtual Machine

The communication virtual machine software consists of a standardized initialization routine and the interface modules. For each user machine environment and data base machine combination, a unique interface module is coded, for example, APL - SEQUEL, PL1 - SEQUEL etc. The purpose of an interface module is to handle the data transfer and conversion functions of a user machine-data base machine communication.

CVMINIT

This standardized initialization routine is given control when the CVM is first logged on. It will wait for a message from its corresponding UVM which will indicate a switch in environments (an 'X' in the first byte of the message). The code in the message header is used to set a Pl/l return code indicating the specific interface module to be loaded in place of CVMINIT. The EXEC that contains the call to CVMINIT will interpret this return code from CVMINIT and load the appropriate interface modeule.

APLSEQ

This routine handles the data formating and data transfer between the APL user machine and the SEQUEL data base machine. It is loaded into the CVM as a result of the CVMINIT module. APLSEQ communicates with SEQUEL via the VMC facility but it communicates with APL via punch files and shared disks. The first category of communication is achieved using the IXEVMC module and its entry points. The latter communication category is achieved using the RDRINIT, XWTAPL, RDRTERM modules. The dbvmid is obtained by reading the

'DBVM id bl' file and the UVMid is obtained, as usual, by calling IXEUVM. APLSEQ has to interpret the type of message from UVM. A UVM can request that another CVM module be loaded so that it can access a different data base. Thus APLSEQ, like other CVM modules, has to be able to detect the signal for switching modules. In the case of a VMC message, the message header will contain this important information. In the case of using punch cards as means of communication, the one character returned by XWTAPL is used as the equivalent of the message type character in the message header of a VMC message. This character can be discriminately changed by the UVM simply by issuing a CP command.
APLSEQ receives blocks of messages from the data base VM and manipulate these blocks into APL variables which are written onto the disk shared by the CVM and UVM. Currently APLSEQ decodes each block as they arrive but a simple modification can be made so that decoding will be performed after all the blocks are received. This is not crucial to the operation of the CVM, besides, a 1 block message can usually accommodate all the results of a normal query.

PL/I Interface

The PL/I multi-user interface consists of a duplicate set of sequel entry points and a cum module, PLISEQ. These control all the functions of initialization, switching and query processing. All communication paths utilize VMCF and it's entry points as previously described. (A prior knowledge of the PL/I programming interface is assumed - see GMIS user's guide appendex F).

In the User's Virtual Machine, the duplicate set of SEQUEL entry points (TRANSIN, SEQUEL, GETROWS, AND WINAUP) is contained in the procedure TRIFUFI (similar to the GMIS I version). Operationally then, the user must first invoke GMISINT with the "PL1SEQ" parameter, to set up a CVM. Then using a Pl/I program (such as TRANSFER) a call is made to TRANSIN which performs some simple initalization (in the UVM only). Next, when the query is ready for processing and placed in the external variable QSTRING, a call should be made to SEQUEl. Now, instead of a full copy of SEQUEL, XRAM, RAM etc., the SEQUEL entry in TRIFUFI is called which transmits the query to the CVM via VMCF, and then goes into a VMCWAIT for an interrupt from the CVM. The CVM receives the query and extracts the database VM id from it and then sends the query to that VM, and waits for an interrupt from the database VM. The database performs the query and returns its results via VMCF. The only processing the CVM does is to write any table data on the user's 340 desk and then return the rest of the result (column names, cordinality, format descriptors etc) to the UVM via VMCF. This serves as a wake-up to the UVM, and completes the processing done by the CVM, which goes back to an initial wait state. The UVM stores the information in the appropriate external variables and returns to the caller.

To retreve the rows of a table after a successful SEQUEL call (Code = 2) calls should be made to the entry point GETROWS, which re-accesses the 340 disk and then calculates the necessary blocks # and then performs the direct access read. It extracts the row indicated by DISPROW and fills in QSTRING appropriately.

To terminate the communications, a call to WINDUP should be made, to release the CVM and stop VMCF communcations.