Consiglio Nazionale delle Ricerche

APLCMS Extended to support Graphic Facilities

V. Casarosa - E. D'Ettorre - E. Klein - R. Raffaelli - S. Trumpy

83

CNUCE

Divisione Servizio Elaborazione Dati

A cura di : V. Casarosa
E. D'Ettorre
E. Klein
R. Raffaelli
S. Trumpy

Vittore Casarosa *

Enrica D'Ettorre +

Ellen Klein +

Remo Raffaelli +

Stefano Trumpy +

APLCMS EXTENDED TO SUPPORT GRAPHIC FACILITIES

* IBM Scientific Center - PISA

+ CNUCE - Institute of CNR - PISA

Index

1 - Introduction

The use of APL in connection with some kind of graphic output device is becoming wider and wider. The combination of the interactivity of APL and the immediateness of graphic displays turns out to be very useful, especially in those applications where users have no familiarity with the computer.
In this paper we describe how an integrated system has been implemented by using APL under CMS and an IBM 2250 display unit.

The capabilities of interaction of the 2250 cannot be exploited under APL, as the APL system does not support that device. In order to display an image built in APL, it is necessary to write a description of it onto disk and then to enter an environment where a graphic package can be used. To further elaborate, under APL, the information acquired by interacting with the image, it is necessary to repeat the same procedure in the opposite way.
Although such procedure can be done without any intervention of the user, in any case it gives rise to a long response time mainly due to I/O operations on disk and to the swapping in memory between APL and a graphic environment.

In order to make more practical the use of the 2250, and in order to improve the performance, an interface for APLCMS has been written, so that a graphic package can be used directly from within the APL system and the graphic data can

be accessed directly in memory.
With this interface an APL user has at his disposal a set of basic APL defined functions that perform all the tasks involved in driving the 2250. These functions perform their tasks by invoking the routines of some existing graphic package. In the present implementetion the graphic package that we have used is the Fortran Graphic Support (FGS) Package (ref. 1). FGS is a set of Assembler written routines that can be called by Fortran, and can be thought of as a much simplified version of the Graphic Subroutines Package (GSP), the standard IBM package to drive the 2250.

Another way of making the 2250 directly available under APL could be that of modifying the APL supervisor so as to support the 2250 and of modifying the interpreter by adding commands or operators to use the new facility. The main disadvantage of this approach is the necessity of deep changes and additions to the supervisor, mostly in those parts devoted to interrupts handling.
By using the facilities already present in APLCMS, the approach that has been choosen leads to the same results without any change to the APL system, apart that of integrating the graphic package with the APL system into a single executable module.

2 - Functional description of the interface

In the design of the interface, one of the major concerns has been that of avoiding modifications to APLCMS and to FGS. The kind of solution to the problems that have been encountered is a consequence of this approach.

2.1 - Passing parameters between APL and FGS

APL and FGS use different conventions as far as data representation in memory is concerned.
APL data are represented in memory by entities called "M-entries". Each M-entry is composed by a set of fields providing a complete description of the data itself, that is its type (integer, boolean, etc), its size, and so on, followed by its value. FGS data representation, or more exactly FORTRAN one, is constituted only by the value of the data itself.
Internally, APL and FORTRAN use the same conventions for passing parameters from a calling routine to the called one: a list (PLIST) of the addresses of the parameters is created and R1 register points to that list. In APL such addresses are relative to the M-entries of the parameters, but in FORTRAN to their values.
The interface must build a new list of addresses as FORTRAN conventions require, by using the information provided in

the PLIST and in the M-entries of the parameters.

In addition, all the variables containing characters must be converted from the APL internal representation (Z-symbols) to the EBCDIC code used by PGS.
The different internal representation of boolean values and the different organization in storage of arrays have not constituted a problem because such kinds of variables are not allowed to be exchanged between APL and PGS.

Upon return from an PGS subroutine, only the conversion of characters from EBCDIC to Z-symbols needs to be done, as an PGS subroutine can modify only the value but not the type of the parameters.
If an PGS function has been called, the result of its execution is left by PGS in R0 register, as usual. Instead an M-entry is needed for every intermediate and final result of APL computations. The pointers to those M-entries are stored in a stack, called the "R13-stack", which is dinamically built while the execution of a statement is in progress. Upon return from an PGS function, therefore it is necessary to build an M-entry for the result and to properly update the R13-stack.

2.2 - Updating of relocated variables addresses

In order to display an image with PGS, it is necessary to build a program of orders and data into an area of main memory (called a graphic area) and subsequently this area must be moved into the 2250 buffer to display the image on the screen. Before being used, a graphic area (that is a previously dimensioned vector) must be initialized by calling the proper PGS routine.
The initialization routine associates to the graphic area a control block which is acquired from CMS free storage, and which contains the address of the graphic area, among other information. Every PGS routine acting upon a graphic area, identifies it only by the address of its control block.

APL interpreter routines perform a dynamic storage management making a garbage collection after the execution of every statement: the amount of storage occupied by temporary E-entries is returned to workspace free storage and permanent E-entries are relocated for optimizing storage occupation.
When using PGS from APL, a graphic area must be a global variable so that the related E-entry is never released to workspace free memory. Moreover, because of the relocation of E-entries, it is necessary to update the contents of the control block with the new address of the graphic area. This is done in the interface by means of a table containing an entry for each graphic area that has been initialized. The

entry contains the printname of the graphic area and the address of its control block.
Every time a routine using a control block address is called, the interface looks up the table to find the printname of the involved graphic area. The printname is then used to look up the workspace SYMBOL TABLE to find the actual address of the variable. The old address is replaced by the new one in the control block and eventually the FGS routine is called.

2.3 - Handling of errors and program interrupts

The interface performs as many controls as possible to avoid erroneous usage of FGS. For example, a check is made that a graphic area is initialized before being used; the number and the type of parameters are checked before calling an FGS routine.
The detected errors are properly coded for an easy debugging and an error message is typed by the interface. Control is then given to the APL interpreter routine which handles the errors. This prevents from further executing a function containing a line with an erroneous call to an FGS routine.

In addition to that, when entering in the interface the Program Interrupt Handler of CMS is set to point to a location of the interface.
This avoids the occurrence of an APL System Error and subsequent clearing of the workspace if a program check arises when control is in the FGS routines.
In case of program check, the interface resets the FGS environment and goes back to the APL error handling routine.
In any case, before leaving the interface, the handling of Program Interrupts is restored to its previous state.

3 - Implementation under APLCMS

The interface described in the previous section has been implemented in Assembler/360 language so as to make use of the possibility existing in APLCMS of jumping to an Assembler written routine during the execution of an APL statement. The names of the Assembler routines must have been previously "declared" to the system by inserting them in a special table called QTAB. This table contains the names of the entry-point of the routines and their addresses at execution time. The addresses are filled in by the Loader during the link-editing of the QTAB, the routines, and the APL system at system generation time. An APLCMS user can jump to a routine contained in QTAB by using an extension of

the dyadic I-beam operator. The right I-beam operand must be a literal vector constituted by the name of the called routine and by those of its arguments. The left operand must be "101" or "102" depending whether an explicit result is expected or not.

In order to make the interface program known to API, the interface name has been added to those in QTAB. The FGS routines are known to the interface by means of a table containing their names and their addresses.
The APL-FGS system is obtained with a system generation by link-editing into a single CMS module the APLCMS supervisor and interpreter, the interface program, and the FGS routines.

Once the system has been generated, an FGS function or an FGS subroutine can be called with the I-beam operator having 102 or 101 as left operand, respectively. The right operand is constituted by the name of the interface followed by the name of the graphic routine and by those of its parameters.

4 - Using APIFGS

As previously stated, in the APL-FGS system, all graphic routines of the FGS package are called by the APL I-beam operator. In order to simplify the procedure of calling the graphic routines, a set of APL functions exists which incorporates the use of the I-beam operator. For every FGS function and subroutine, there exists a corresponding APL-FGS function which has the same name and serves the same purpose. Clearly the sequence of the calls to the graphic routines, referred to as the 'display program', and its logic, is the same whether using APL-FGS or the FGS package. A list of all APL-FGS functions has been included with their calling arguments, explicit results, and error reporting. For complete information on the details of these functions, refer to 'Fortran Graphic Support User Manual', IBM Technical Report CSP014-513-3513. Before offering the list of the API graphic functions, some remarks will be made on the 'display program'.

4.1 - The display program

The fundamental structure on which the APL-FGS system operates is the display program. It consists of a sequence

of calls to the graphic functions. These graphic functions can be separated into three general areas:

- driving the display unit
- managing data
- handling communication between the 2250 and the user's program

Driving the Display Unit

The display program is built up in a reserved area of main memory called the 'graphic area'. The graphic area is initialized by a routine which at the same time sets up a corresponding control area where all system information on the graphic area will be stored. Most important, this system information includes the starting address of the graphic area and the current displacement for the next available position in the graphic area.

Since more than one graphic area, with its corresponding control area, may exist concurrently in main memory, the user must use the address of the control area in order to access its graphic area. Therefore the address of the control area is used as an argument in the functions operating on graphic areas. The use of a graphic area is terminated by using the proper function.

Once the display program has been constructed in a graphic area, the graphic area must be transferred from the main memory to the buffer of the 2250 in order to obtain a display. This buffer must be initialized when the session

with the 2250 begins. With initialization the buffer becomes known as the 'buffer area', and a 'control buffer area' is set up in main memory. The control buffer area contains system information which is updated whenever operations are made on the buffer area. The use of the buffer area is terminated by using the proper routine.

The 2250 display unit reads, decodes, and executes the instructions of the display program. Since the image on the screen fades rapidly, it must be continuously redrawn in order to create a steady image. This regeneration of the image is accomplished by repeating the execution of the display program via a special 'branch' function called at the end of the display program. However, a short display program results in a too fast regeneration of the image and can result in damage to the display screen. Therefore a call to a proper function is included in the display program to prevent the regeneration cycle from being repeated more than once in 25 ms.

Managing Data

The 2250 display unit is able to display two types of information, points and segments, and characters. The points and segments may be entered only when the 2250 is in 'graphic mode', while the characters must be entered when the 2250 is in 'character mode'. A set of functions exists for setting the display unit to the desired mode. When in either mode, calls to appropiate functions are needed to

define the data to be displayed. A point is defined by its coordinates, a segment by the coordinates of its end points, and a character simply by a character.

Handling Communication Between the 2250 and the User's Program

The display program can include calls to functions to communicate with the 2250. Three features are offered: the light pen, the alphanumeric keyboard, the programmed function keyboard. In many cases the capabilities of these features overlap and the selection of one feature over another is dependent on user preference.

The light pen is a fiber-optic device which is applied directly to the screen for individualizing a specific point for further manipulation.

The alphanumeric keyboard is a typewriter-like keyboard which is used for inputing data onto the screen for use by the display program. The area on the screen where the data will appear is indicated by a special character called the 'cursor'.

The programmed function keyboard contains 32 keys and 32 light indicators which are generally used as switches for controlling options which have been programmed by the user in the display program.

4.2 - The API graphic functions

The API graphic functions can be divided into the following groups according to the operations they perform:

1) Initialization Functions
   - INEA
   - INGA

2) Termination Functions
   - TMGA
   - TMEA

3) Order Generation Functions
   Graphic order generation functions
   - GEVM
   - GEVMI
   - GEEM
   - GEEMI

   Character order generation functions
   - GECEE
   - GECEF
   - GECLE
   - GECLF

   Control order generation functions
   - GSRT
   - GTRU
   - GECS

4) Data Generation Functions
   - GTEXT
   - GXY
   - GXYF

5) Transmission Functions
   - WREA
   - RDEA

6) Control Functions
   - GEXEC
   - GSTCP
   - INCSE
   - RMCSE
   - SPFK

7) Input Data Functions
   - RDTC
   - RDXYE

8) Maintenance Functions
   SAIF
   SDSF
   KDSF

9) Attention Handling Functions
   KATBT
   KATNW

10) Error Management Functions
   KEBB
   EBDSF

1) Initialization Functions

INEA - initialize the buffer area

Argument: none

Explicit result: none

INGA - initialize a graphic area

Argument: a scalar which specifies the length of the graphic area

Explicit result: address of the graphic control block

2) Termination Functions

TMGA - terminate a graphic area

Argument: a scalar which is the address of the graphic control block corresponding to the graphic area to be terminated

Explicit result: none

TMEA - terminate the use of the buffer area

Argument: none

Explicit result: none

3) Order Generation Functions

- Graphic order generation functions
The following functions have the same argument and explicit result.

GEVM - enter vector mode absolute
GEVMI - enter vector mode incremental
GEPM - enter point mode absolute
GEPMI - enter point mode incremental

Argument: the address of the graphic control block

Explicit result: none

- Character order generation functions
The following functions have the same argument and explicit result:

GECBP - enter charcter mode basic protected
GECBF - enter character mode basic unprotected
GECLP - enter character mode large protected
GECLF - enter character mode large unprotected

Argument: the address of the graphic control block

Explicit result: none

- Control order generation functions
The following functions have the same argument and explicit result:

GSRT - Start regeneration timer
GEOS - end of order sequence

Argument : the address of the graphic control block

Explicit result: none

GTBU - unconditional branch

Arguments:
left - address of the graphic control block
right - a scalar specifing an address in the buffer area

Explicit result: none

4) Data Generation Functions

GTEXT - used to display characters

Arguments:
left - the address of the graphic control block
right - a character vector

Explicit result: none

The following functions have the same arguments and explicit result.

GXY - used to display a point or a vector
GXYP - used to position the electron beam without displaying

Arguments:
left - address of the graphic control block
right - a 2-element vector specifing the (x,y) coordinates on the screen

Explicit result: none

5) Transmission Functions

The following functions have the same argument and explicit result:

WBEA - move the specified graphic area to the buffer
RBEA - move the buffer area into the specified graphic area

Argument: the address of the graphic control block

Explicit result: none

6) Control Functions

The following functions have the same argument and explicit result:

GEXEC - start buffer regeneration
GSTOP - stop buffer regeneration

Argument: none

Explicit result: none

INCSB - insert cursor

Arguments:
left - an address in an unprotected graphic area where the cursor is to be inserted
right - a scalar which represents the position where the cursor is to appear

Explicit result: none

RMCSR - remove cursor function

Argument: none

Explicit result: none

SPFK - set the light keys of the programmed function keyboard

Argument: an integer vector or a scalar between 0-32 representing the number(s) of the light key(s) - the scalar '0' will turn off all lightened keys

Explicit result: none

7) Input Data Functions

RDTC - read to cursor

Arguments:
left - an address in an unprotected graphic area where the cursor is to be inserted
right - the maximum number of characters to be read

Explicit result: the character vector which has been read

RDXYP - gives the coordinates (x,y) of the last light pen attention

Argument: none

Explicit result: a vector of two elements containing the (x,y) coordinates

8) Maintenance Functions

The following functions have the save arguments and explicit result:

SAIR - set address in a graphic area or in the buffer area
SDSP - set displacement

Arguments:
left - the address of a graphic control block if the starting address is set in a graphic area - the

argument is equal to 0 if the starting address is set in the buffer area
right - a scalar representing the address

Explicit result: none

KDSP - keep displacement in the graphic area or in the buffer area

Argument: the address of a graphic control block if the starting address is set in a graphic area - the argument is equal to 0 if the starting address is set in the buffer area

Explicit result: the scalar representing the displacement

9) Attention Handling Functions

The following functions have the same argument and explicit result.

KATNT - keep an attention
KATNW - keep an attention or wait

Argument: none

Explicit result:
K = 0 - no attention
K = 1:32 - the programmed function keyboard key 1,32
K = 33 - an 'End of order sequence' order has been executed during buffer regeneration
K = 34 - the end key of the alphanumeric keyboard has been depressed
K = -bufadd - a light pen switch detection has been done upon a displayed image

10) Error Management Functions

KERR - keep error code in the specified graphic area or in the buffer area

Argument: the address of a graphic control block if the starting address is set in a graphic area - the argument is equal to 0 if the starting address is set in the buffer area

Explicit result: none

ERDSP - error display and/or print

Argument: an interger variable or constant whose content can be 0,1,2, or 3
code = 0 upon the occurrence of any error no message is sent to the user and the APL function that caused the error is resumed
code = 1 the buffer regeneration is stopped and the error message is written on the screen - the user may choose to continue his program or not
code = 2 upon the occurrence of any error the related error code and message will be printed
code = 3 upon the occurrence of any error, both the operations described above for CODE = 1 and CODE = 2 occur contemporaneously

Explicit result: none

5 - Conclusions

The implementation of the APL-PGS system required a reduced effort, thanks to the chosen approach of not modifying the APL supervisor, and of using an existing graphic package. The simple architecture of the PGS package has overridden the disadvantages due to the fact that it was designed to be called from a FORTRAN environment.

An interesting result of this work is the possibility of generalizing the use of this interface for dinamically calling any precompiled routine. The routine is loaded into the workspace free storage and then it is executed. All linkage-editing and loading operations are performed by the use of the CMS loader features.

This system has demonstrated in a practical and effective way the impact that can have on the applications the use of a really interactive graphic system. Expert users have at their disposal a tool constituted by a set of basic functions with which they can easily build programs utilizing the graphic and interactive capabilities of the system. The debugging and tuning of these programs is greatly siplified by the facilities provided by APL to this purpose.

The same set of basic functions is a powerful tool also for sytem programmers, for building programming packages to be used by non professional and casual users (ref. 2,3). The

use of displayed menus to be selected by means of the light pen, and the possibility of issuing 'help' requests at any moment, with an immediate and detailed response, are two features very useful in this type of applications which can be very easily exploited in the APL-FGS system.

References

1. I. Casazza, C. Paoli, I. Bartolo, G. Preziosc:
Fortran Graphic Support User Manual
IBM Pisa Scientific Center
Technical Report CSP014, August 1972.

2. V. Casarosa, S. Trumpy:
Interactive interpolation and display of surfaces
Proceedings of API Congress 73
North Holland Publishing Company, Amsterdam 1973.

3. V. Casarosa, S. Trumpy:
Conversational building and display of solid objects
Proceedings of the VI API user Conference
Coast Community College District, California, May 1974