DEVELOPMENT OF A GRAPHICAL DISPLAY ON THE DMS TEST BED

FINAL REPORT

NASA/ASEE Summer Faculty Fellowship Program -- 1987

Johnson Space Center

Prepared by:	Robert A. Donnelly, Ph. D
Academic Rank:	Associate Professor
University & Department:	Auburn University Department of Chemistry Auburn University, AL 36849
NASA/JSC	
Directorate:	Engineering and Development
Division:	Avionics Systems
Branch:	Flight Data Systems
JSC Colleague:	Michael M. Thomas
Date:	August 26, 1987
Contract Number:	NGT 44-001-800

ABSTRACT

The DMS test bed is a model of a data network aboard Space Station. Users of the network share data relevant to the functional status of various systems (Guidance, Navigation and Control, Environmental Control and Life Support, etc.) aboard the Station. Users may inquire the status of myriad sensors, obtaining readings of Station subsystem status in real-time via the Data Acquisition and Distribution Service.

This project is aimed at development of a graphical display of the status of a simulation of the Environmental Control and Life Support System. Two broad issues were addressed: (1) Flexible, extensible software design; (2) The impact of utilizing standard processors, languages, and graphics packages implementing the software design concept.

Our experience gained with DEC hardware, the DEC implementation of the GKS graphics standard, and with Ada can be summarized as follows. DEC hardware seems adequate to the display tasks which form one part of the present research project. Highly graphics-intensive applications run slowly on the MicroVax GPX when it is programmed using the various DEC implementations of the GKS standard. Use of the GKS standard for graphics can provide code portability provided the FORTRAN/77 to GKS binding is used. Since no binding to Ada is currently available it makes sense to provide a procedural interface between Vax Ada and the FORTRAN/77 binding. This was accomplished, and can be used by other generations of graphics programmers who must program in the development environment provided by DEC.

I. INTRODUCTION

The DMS test bed is a model of a distributed communications network which is proposed for the Space Station. Users of the network have access to data generated by several subsystems likewise attached to the network. Users may inquire the status of these subsystems (and their components) through DADS (Data Acquisition and Distribution Service). Each element of requestable data (a subsystem or sensor status, or an engineering measurement) is uniquely identified by a measurement identifier (MSID).

One purpose of the present project is development of an interactive graphical display of the status of a simulation of the Environmental Control and Life Support subsystem (ECLSS). Sets of MSID's defined in the graphics software are transported over the network to the ECLSS simulation software using DADS. DADS then returns the simulated status of sensors aboard ECLSS, which are then displayed on a high-resolution computer graphics terminal. The software package is interactive, allowing the user to display the status of various components of the ECLSS subsystem.

A second, equally important, aspect of this research project is concerned with the impact of "standard" hardware and software on implementation of the software package. Developers of the DMS test bed have declared the MicroVax GPX as the standard display device. GKS is a well-known standard for rendering two-dimensional graphical images, which is available on DEC hardware. Finally, Ada has been adopted as the standard programming language aboard the Space Station. Accordingly, the software package was developed using a mixture of FORTRAN and Ada (for reasons to be discussed below). It is anticipated that lessons learned in this research can be helpful to those who design the final software aboard Space Station.

II. SOFTWARE DESIGN

Two features of the software design were judged especially important. First, the software should be designed in such a way that additional display capabilities could be added with a minimum of alteration to the original package. Second, the software package should be transportable across host processors equipped with GKS software.

The first consideration was accomplished by coding a control structure with "stubs" for added simulations as they become available. A control section written in Ada is essentially a menu-driven display processor, the highest level of which allows the user to select a subsystem for display. The display requirements for each subsystems can be made completely invisible to the main routine. Alternately, the display requirements can be met using a graphics parser written in FORTRAN and included as a package accessible by the main Ada supervisory routine. The graphics parser is general enough to provide all graphics services. Its capabilities can easily be extended as needed by extension of its dictionary of recognized graphics commands. Network access is provided by a separate Ada package which can be modified independently of the supervisory routines.

A second feature of the program design is its minimal reliance on "hard-coded" display items. We accomplished this by utilizing ASCII input files containing graphics commands directed to the graphics parser. Dynamic components of each display window are maintained as retained graphical segments which are defined once and for all at startup time and inserted into each display window as required. This strategy results in a flexible display manager: Static components of each display window can be changed by merely editing the corresponding

file with a text editor without recompiling the display program itself. Design of retained segments representing standard components of a schematic display, for example, make it easier for system-wide users to interpret graphical displays of system status.

The second consideration, software transportability, has been addressed by mixing languages in the demonstration program. The reasons for this are discussed in the next Section, where we consider the impact of the development environment on program design.

III THE IMPACT OF STANDARDS ON SOFTWARE DESIGN

GKS is itself merely a method for the device-independent management of two-dimensional graphics displays. As such it includes procedures for storage and manipulation of graphical segments, for implementation of software windows and view ports, and for interaction with graphical input devices such as the keyboard and mouse. Device independence allows the applications programmer to concentrate on the display task at hand, without the need to consider the specific characteristics of a particular hardware device. The functional con trol of graphics displays afforded by the GKS routines is judged adequate for the graphical manipulations required in the current display task.

The DEC/Vax implementation of the GKS standard has a significant impact on programmers writing in FORTRAN/77 or Ada. The GKS package used in this research is revision 3.0 of the DEC Vax software produced by Digital Equipment Corporation. The issues we consider in this Section are: (1) Existence of standard interfaces to GKS through FORTRAN and Ada; (2) Documentation and ease of software use by graphics novices and experts; (3) "Low-level" support provided for control of graphics input devices and display hardware; and (4) The availability of expert consulting services from the software vendor.

There are several interfaces to GKS procedures provided by DEC. Each programming language (Ada, Basic, C, FORTRAN, Pascal, etc.) uses a different interface, so that no single set of procedure calls exists. That is, the names of the routines are not the same in different languages, and the number and type of arguments passed to procedures differ from language to language. Here we focus on three such interfaces, Vax FORTRAN/77 to VAX GKS, ANSI FORTRAN/77 to GKS using the standard FORTRAN binding, and Vax Ada to Vax GKS. As of this writing there is no standard binding between Ada and GKS. Two such bindings have been proposed independently by ANS and ISO, but neither is available on DEC machines at present. (Parenthetically, we remark that an alternative interface to graphics routines is provided by UIS, the native graphics interface on the MicroVax. This is probably the highest performance, most powerful interface, but could not be used in this research. First, it is specific to the MicroVax GPX, and therefore requires non-portable code. Second, its use requires compilers and linkers which are unavailable on the DMS MicroVax. Third, DEC currently supplies no interface to UIS routines from Ada.)

The net result of the profusion of interfaces to GKS and their varying levels of documentation and functionality is that we chose to use a mixed-language program: The control structure was written in Ada, while direct access to graphics was provided by the FORTRAN/77 binding to GKS. We accomplished this by writing an Ada package specification containing pragma interfaces which allowed importing the FORTRAN/77 binding to GKS. Several procedures found to be only accessible through FORTRAN are accessed only through that language. The net result of this strategy is the generation of a single, standard interface between either FORTRAN/77 or Ada and the portable GKS binding. Transportability of the graphics software is thus possible

between processors equipped with FORTRAN/77, Ada, and GKS. Only a single level of documentation on the interface to GKS need be provided to FORTRAN and Ada programmers using this system.

We found the documentation provided by DEC to be generally inadequate. The highest level of documentation is provided for Vax FORTRAN/77 to Vax GKS. There is only minimal documentation on the FORTRAN/77 binding to GKS, and no documentation on the Vax Ada to Vax GKS interface. Several features of the FORTRAN/77 binding to GKS are completely unspecified in the system documentation, even though this interface is claimed to meet the highest standard of portability between processors equipped with GKS software.

This lack of adequate documentation is particularly undesirable. The complexity and profusion of procedure calls required to perform simple tasks is certain to confuse programmers with little training in graphics. Code fragments produced to document each procedure call are discussed briefly, but examples of only the simplest type of procedure call are given. The environment in which a given procedure call can be made is not generally discussed, so that one is required to read and study the entire two-volume manual set in order to trace errors. DEC has a rather unorthodox model of segment storage which can be confusing to the novice programmer. Its method of segment definition requires careful study, and is largely undocumented. Finally, access to graphics primitives (a circle, for example) is provided through graphics drawing primitives (GDP's). This is a most unorthodox method of generating these primitives, is tedious to code, largely undocumented, and error-prone. We have had extensive experience with a wide variety of graphics software and hardware. This implementation of GKS is by far the most complex one we have encountered. In fact, though

Ada was unknown to us at the start of the research project, it was much easier to learn that language than it was to become familiar with GKS.

Tasks requiring precision control of graphics input devices are not easily accomplished in the Vax implementation of GKS. We have been unable. for example, to find a means of controlling the foreground and background colors of a "pop-up" menu from software. The font used in displaying the menu items is not under software control, unless one is prepared to write a special-purpose graphics handler routine (outside of GKS). Control of hardware windows appears possible only by use of the non-standard Vax FORTRAN/77 to Vax GKS interface. Importantly, the GKS software does not make use of the specialized hardware aboard the MicroVax GPX. Highly graphics-intensive applications run unacceptably slowly on the MicroVax for this reason.

Finally, we find that software support from DEC appears to be largely inadequate. Those to whom we have spoken are largely unknowledgable of grahics in general, and of procedural inter-relationships in DEC's GKS in particular. There seems to be little support given to the standard FORTRAN/77 binding to GKS.

IV. CONCLUSIONS

Our experience gained with DEC hardware, the DEC implementation of the GKS graphics standard, and with Ada can be summarized as follows. DEC hardware seems adequate to the display tasks which form one part of the present research project. Highly graphics-intensive applications run slowly on the MicroVax GPX when it is programmed using the various DEC implementations of the GKS standard. Use of the GKS standard for graphics can provide code portability provided the FORTRAN/77 to GKS binding is used.

The lack of adequate documentation on the GKS software makes it unlikely that novice graphics programmers will be able to use the system easily. It is therefore likely that modern display techniques, which make use of multiple windows and view ports, and of segment transformation techniques, will remain largely the province of experienced graphics designers. This is unfortunate, as the use of modern techniques makes for quickly-developed, expandable, transportable computer codes.

The Ada programming language is easy to learn, and possesses many desirable features. It is strongly favored in recursive applications, such as the menu-driven display processor developed here. Since no binding to Ada is currently available it makes sense to provide a procedural interface between Vax Ada and the FORTRAN/77 binding. This was accomplished, and can be used by other generations of graphics programmers who must program in the development environment provided by DEC.

Finally, it appears that there are in fact no "standards" yet available on DEC hardware. Though GKS is a graphics standard suitable for two-dimensional display management, the only interface presented to meet a portability standard is provided by the largely undocumented binding to FORTRAN/77. No similar binding of GKS to Ada exists. Indeed, no link between Ada and the UIS procedures native to MicroVax yet exist. The net result is a lowest-common-denominator software implementation which is not designed to make most efficient use of existing hardware.