DESIGN OF A GENERIC TRAINING DEVICE CONTROL CONSOLE USING ADA

V. Faconti
L. B. McDonald, Ph.D.
Harris Government Support Systems Division
Winter Park, Florida

ABSTRACT

Several factors set the stage for control console designers who wish to compete in today's training environment. Chief among these are various DoD initiatives to reduce the costs and increase effectiveness of training systems. The DoD mandate to use Ada* is a good example. This paper documents a program of research aimed at developing a design approach to realize the DoD cost-effectiveness goals in the training device control console area. This approach features increased use of modular generic sofware solutions which can be applied over a wide range of situations. At the same time, the approach allows for modification to accommodate specific requirements as needed. A functional baseline was developed based upon reported console design studies and then expanded through developmental testing and user surveys. User reactions and Ada lessons learned are also discussed.

INTRODUCTION

Over the past few years, the cost of simulation hardware has plummeted while the cost of software has skyrocketed. While much of this hardware cost reduction is due to advances in miniaturization, the dominant effect has been the result of modularization of hardware. Instead of designing and building dedicated hardware for specific applications, engineers select modules that will perform the required functions. In order to fulfill the varying functional requirements, these hardware modules must have a built-in flexibility such as programmable functions or expansion capabilities that can be implemented by the end user without hardware modification. Hardware modules with this flexibility are used in a number of applications, thus spreading the development cost over a large number of units and lowering unit cost.

To achieve lower costs in software, the training systems industry must develop flexible modular software packages that can be used repeatedly in a number of training system projects, thereby, distributing development costs over a number of applications. This emphasis on reusable training system software modules is in keeping with the larger DoD Ada Initiative for weapon systems in general.

In order for hardware manufacturers to develop modules that will be used repeatedly, they had to first determine what functions must be performed repeatedly. This same function analysis is required by software developers in order to determine what software functions will be used repeatedly in training systems. The U.S. Air Force has performed an analysis of flight simulator functions and has developed the following list of simulation (functional) modules for a weapon systems trainer (as defined in the SOW for Modular Simulation Design.):

Aerodynamics	Visual
Flight Controls	Navigation
Flight Station	Support
Instructional System	Electronic Combat
Motion	Radar
Propulsion	Weapons

In order to be flexible enough to be used repeatedly on separate flight simulators, these simulation software modules must consist of a large number of packages that will carry out the various functions likely to be needed in the diverse applications of the future.

In this paper we will concentrate on the module labeled by the Air Force as Instructional System. To achieve the needed flexibility for the module, we must first determine the functions the module must perform in the future. Since the primary purpose of the instructional system is to support the simulator instructor/operator, these functions can be derived by doing an analysis of instructor/operator functions.

BACKGROUND

The imposition of Ada fully supports a generic instructor console concept, and further, even simplifies its implementation. Figure 1 illustrates the advantages realized by reusability of Ada software. Since Ada software is designed to compile and execute on any Ada-compatible processor, reusable instructional software source code need only be developed once. Each subsequent implementation requires only compilation on the target processor and the writing of driver packages to meet the specific needs of the instructor and interface hardware.

*Ada is a registered trademark of the U.S. Government, Ada Joint Program Office.

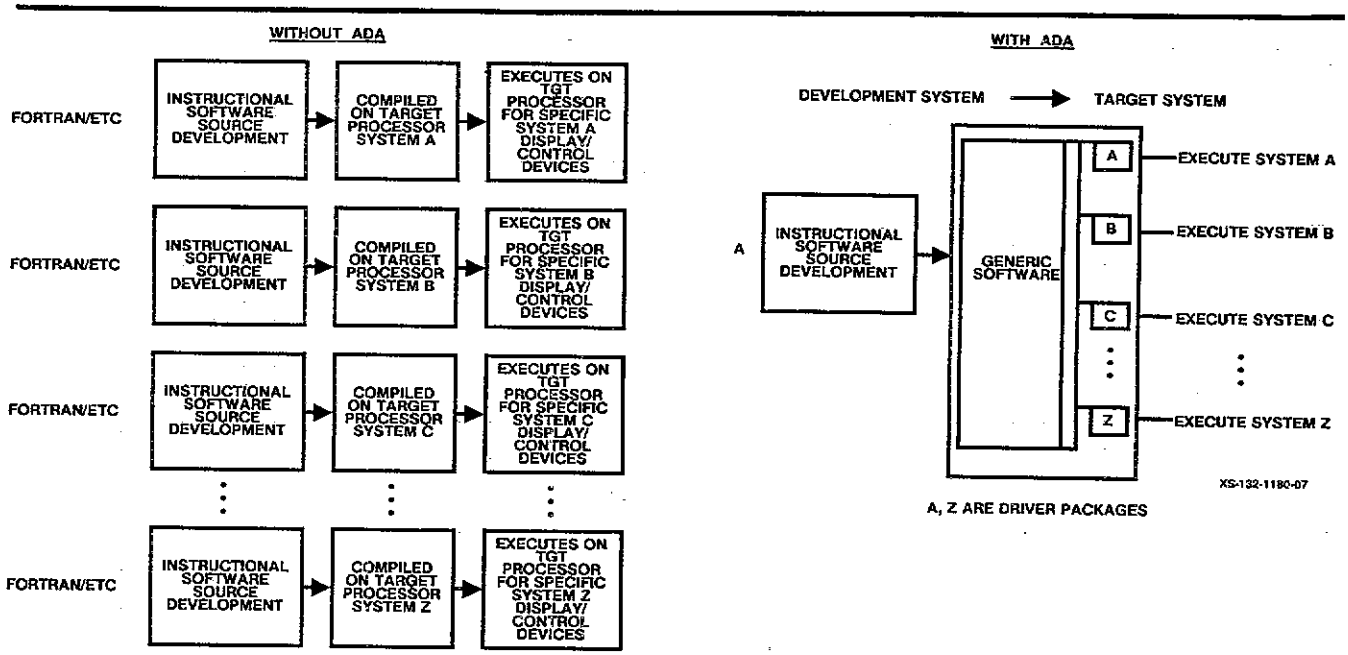


Figure 1. Advantage of Ada

Accordingly, we are now presented the tools needed to explore the concept of a Generic Instructor/Operator System (IOS). The combination of microprocessor power/cost, advanced raster or flat panel display technology and Ada Offers this opportunity.

As a more pragmatic understanding of training technology has evolved over the years, so has an understanding of the instructor's role in a simulation training environment, and with that, a clearer understanding of the functions that are concomitant with this role. Much of the material discussed below provides an understanding of the functions performed by current and future instructors. The premise of this work is based on an understanding that the role of the instructor station is to provide a mechanism by which the instructor can view and alter the training problem.

Many desirable features of control consoles have been well documented in the literature, several are listed in Table 1. Figure 2 is a sequential flow of instructor/operator functions and how they interact with the other modules and functions of a flight simulator. Generic IOS software must perform the functions labeled as IOS functions in order to be reusable on a large number of flight simulators as other classes of simulation training devices.

The design approach followed here was driven by a desire to capitalize on technology, and at the same time meet the needs of a great number of training situations. Therefore, modularity became a central performance requirement. With the performance aims understood, the next step was to lay out the functional basis for building the system. The functions of a generic control console should be common to a great many control situations.

Table 1.
Summary of Desirable Control Console Features

INSTRUCTIONAL FEATURES	LISTED BY (SEE REFERENCES)
RECORD/PLAYBACK	1, 4, 7
REMOTE REPEATER DISPLAY	1
HARDCOPY	1, 4
MANUAL FREEZE	1,4
AUTOMATIC FREEZE	1,4
PARAMETER FREEZE	1, 4
DEMONSTRATION	1, 4
DEMONSTRATION PREP	1
AUTOMATIC MALFUNCTION FAULT INSERTION	1, 4, 7
AUTOMATIC MALFUNCTION INSERTION EXERCISE PREP	1, 4
INITIALIZE FUNCTION	2, 3, 5, 7
PERFORMANCE EVALUATION FUNCTION	2, 3, 7
DEBRIEF STUDENT FUNCTION	2, 3, 7
DATA MANAGEMENT FUNCTION	2, 3, 5, 7

XS-132-1181-07

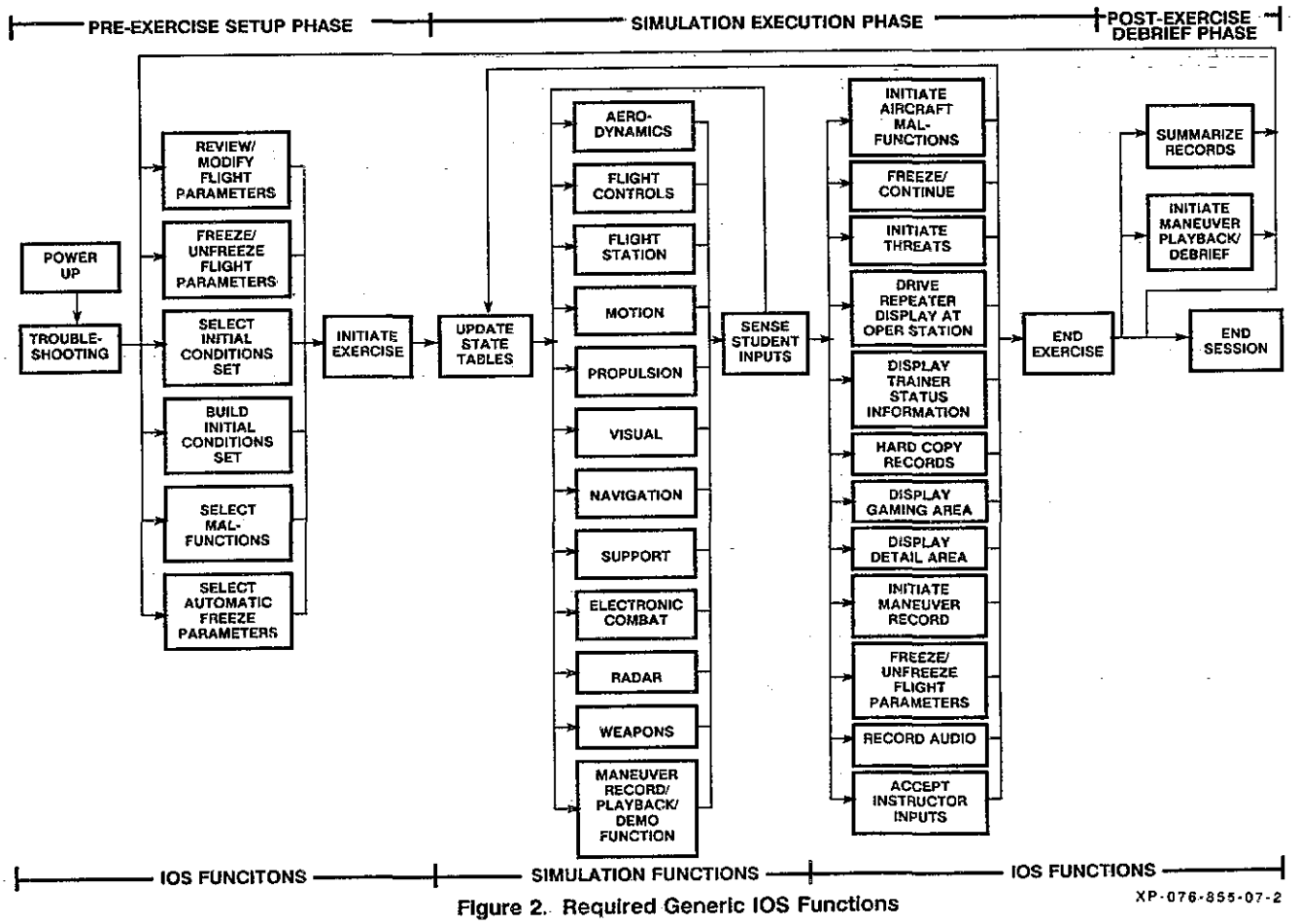


Figure 2. Required Generic IOS Functions

DESIGN MODEL FOR A GENERIC IOS

While the functions in Figure 2 are all different, many of the underlying software tasks are the same. Consequently, the list of functions can be distilled down to the following elements.

1. Display data to the instructor in textual format. This function can be used to not only display data resident in the simulation data pool but also to display data contained in mass memory for elements such as initial conditions, mission scenarios, environmental data sets, navigation data sets, tactical data sets, etc.

2. Display data to the instructor in symbolic format. There are several variations of this kind of a display, e.g., navigation maps, tactical maps, GCA, formation flying, weapons loading, etc. However, this task is essentially reduced to mapping state data pertinent to the student's position into a symbolic view of the problem.

3. Accept data singularly from the instructor to alter the instantaneous state of the simulation. This function can be used to execute initial conditions, insertion of malfunctions, reset training scenarios, alter specific symbol values and other basic functions in the system.

4. Accept data in blocks to redefine the problem. This function can be thought of to serve several different features listed in Table 1. Initial condition data, environmental data sets, reset, record/playback and demonstrations are a few examples.

5. Store data in blocks for later retrieval. Again, this function can serve several of the features listed in the reference Table. In particular, scenario generation functions can be performed by this type of a generic element. In addition, storing data for record/playback, demonstration and initialization is also performed by this element.

6. Perform mathematical functions on simulated data. This capability would be used to display massaged data to an instructor either on a CRT format or via some hard copy mechanism.

Table 2 lists the required IOS functions and indicates which software shell element satisfies that requirement.

Table 2.
Feature Comparison

INSTRUCTIONAL FEATURES	SHELL ELEMENTS					
	1	2	3	4	5	6
RECORD/PLAYBACK				X		
REMOTE REPEATER DISPLAY	X	X				
HARDCOPY	X	X				
MANUAL FREEZE			X			
AUTOMATIC FREEZE			X			
PARAMETER FREEZE			X			
DEMONSTRATION				X		
DEMONSTRATION PREP					X	
AUTOMATIC MALFUNCTION FAULT INSERTION			X			
AUTOMATIC MALFUNCTION INSERTION EXERCISE PREP			X			
INITIALIZE FUNCTION				X		
PERFORMANCE EVALUATION FUNCTION						X
DEBRIEF STUDENT FUNCTION	X	X				X
DATA MANAGEMENT FUNCTION	X		X			

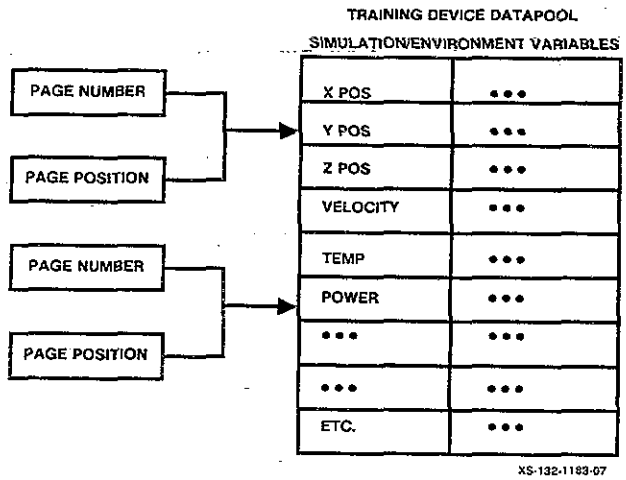
XS-132-1182-07

Implementation Considerations

One of the key issues briefly mentioned earlier which allows this approach to be fully achievable is the use of a data driven system. This means that all interaction from the instructor to the simulation problem can occur through data pool variables (in the traditional sense). Thus, a textual page is simply a collection of ASCII characters with pointers to variables in the simulation datapool. Alteration of these variables occurs through some mechanism (be it keyboard, touchscreen, mouse, voice, etc.) mapped to that page. A data driven approach allows the instructor to identify data he wishes to modify and to actually modify that data. This concept is illustrated schematically in Figure 3. A similar analogy can be drawn for activation of mission segments, for example. By using a data driven methodology to index into mission data sets, a textual page can be used in the same manner as mentioned above to identify the data set to be recalled from mass storage as illustrated in Figure 3.

Organizing the instructional function in a training device in this manner clearly supports the object oriented definition required for design using the Ada programming language. Each generic element in this system can function as an object, with other elements of hardware also serving as objects in the design of the overall object model.

The following discussion addresses a generic model and how it is defined in order to implement the instructional



XS-132-1183-07

Figure 3. Display and Modification of Datapool Variables

features needed in a training device. A typical model is illustrated in Figure 4. All elements except the simulation state block are part of the generic process. This presentation shows a single input device which takes some action on the simulator state by, for example, changing the value of the altitude. Similarly, it could affect the instructional state itself by activating a new display image on either of the two displays shown or activating the store/retrieve data blocks shown. To further illustrate, consider the block marked "Store data blocks". This would handle storing data for record/playback, storing data for CRT page usage, storing data for missions, etc.

As shown in Figure 5, the block from the previous illustration can be broken down further into sub-blocks, each performing a generic function. For example, block #1 is responsible for storing sequential data at a prescribed frequency rate onto some mass storage device. For record/playback and demonstration the frequency rate would be a number greater than 1. However, when used for initialization and reset, the frequency would be set to 1 and one block of data would be stored. Block 2 is a CRT page index for textual pages, while block 3 is the block for symbolic or graphic pages. The last block shown is the training scenario system where scenario design is structured in hierarchical sets. In a typical case, these sets would consist of a set of initial conditions, a set of environmental conditions, a set of navigation aids required to support the problem, and a set of automated features needed such as malfunction insertion, procedures monitoring, etc. The next level in the hierarchy then would be a definition of those sets to be used. This flexibility in design allows the users to customize the software package to fit their needs and makes the package reusable in a large number of applications.

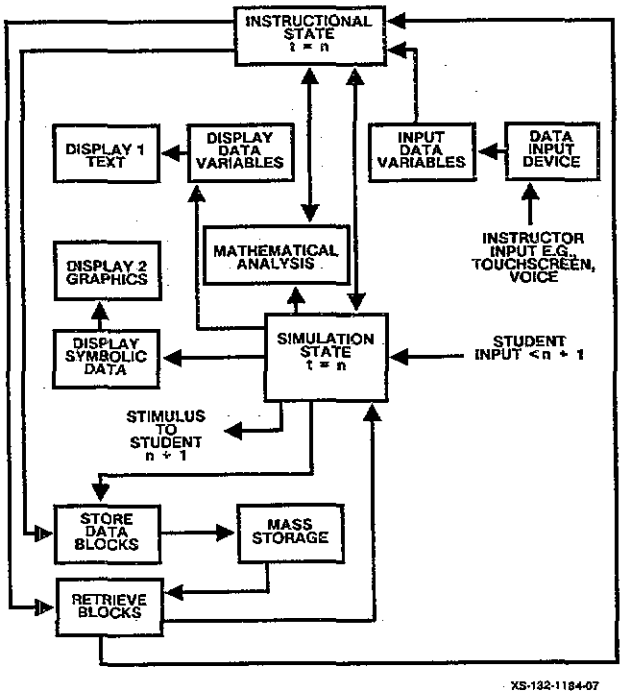


Figure 4. High Level Simulation Model

Feature Methodology

The implementation methodologies required by this approach are very flexible in that, by adhering to a strict data driven approach on many of the functions, variations of those functions are simple to implement. A few examples will be presented to illustrate this point. Consider first, textual CRT pages, pages with descriptive text identifying variables within the simulation problem as well as field location on a display screen. A data-driven approach requires that textual pages are subdivided into two pieces similar to Singer-Link's ASPT approach. First would be a fixed portion ASCII string which would contain all of the non-changing information on the screen such as variable identification, units and other similar items.

The second part is an update table which is maintained whenever a particular page is activated, and this data is output to the screen surface on some cyclic basis, for example, twice/second. This feature can be easily implemented by constructing the desired page layouts in an off-line mode. The pages are constructed by using the actual fixed textual information required in the proper screen locations and by utilizing a method to identify the data fields where simulation parameters are to be displayed through the use of either the defined symbol dictionary name or some superset of that name tailored for the user population. In addition, by defining action codes located on the screen, it is possible to implement a coding mechanism by defining variables in

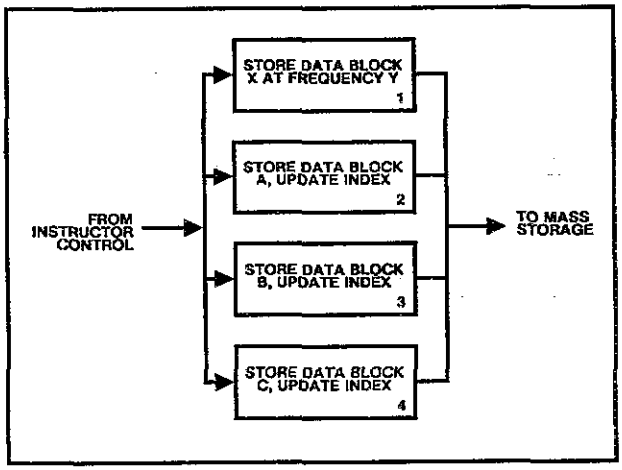


Figure 5. Detail of Store Data Block

the datapool which activate many functions. For example, in one case a symbol dictionary location might have an action code which directs, when selected, that a numeric keyboard input will provide data to be inserted into the simulation problem in a certain location in memory. Alternatively, on a different screen or on the same screen for that matter, a location could have the action code, that when activated, adds 1 to the page number being displayed. The result of this would be the selection of a new CRT page.

This approach can be extended to scenario generation by first defining a series of subsets consisting of, as mentioned, initialization sets, environmental conditions sets, etc. The structure of the scenario can be created by collecting these sets into segments, usually numerically coded. Therefore, the same operators that are used for creation of CRT pages may also be used for creation of scenarios with the exception of their storage structure.

Examples of Model Application

The discussion to this point has, of course, been theoretical and abstract. However, this concept has been implemented recently as part of an R&D project conducted by the Harris Corporation in conjunction with the Naval Training Systems Center. The system model was based on the experimenter/operator system (EOS) currently used in conjunction with the Visual Technology Research Simulator (VTRS) located at Naval Training Systems Center. The VTRS itself served as a test bed for evaluation of not only this abstract concept of developing an instructor control system, but also the implementation of a microprocessor-based modular instructor console programmed using Ada.

The system recently developed in Ada is shown in Figure 6 and consists of a series of Ada tasks and packages that are invoked by a series of events. There are four events that control the execution of the software programs, namely, a timer which activates tasks on an iterative basis, the touchscreen, change data from the simulation, and a keyboard. Once execution occurs, the processing tasks being cued by the event tasks would then output information to one of three output elements. One task, of course, is to modify data for CRT pages themselves, either alphanumeric or graphic. Another task would be to provide hard copy of CRT page data and the third task is to transmit data to the simulator to alter its particular state.

Most of what is shown in Figure 6 is generic in the sense that it can be used on any training simulator which follows certain precepts. That is, a simulation data pool or similar repository for current data must be available and a CRT system is used for control. It must be pointed out that, of course, the graphics are unique to the hardware used except in the structure of the model. Only one task would have to be replaced, and that is the task that deals directly with the input/output to the graphics processors. In fact, the entire model is based on this approach and there are, of course, certain graphic depictions which are unique to this training device. However, the methodology for implementation allows insertion and removal of different tasks fairly simply. This technique was also demonstrated during the current research project.

Figure 7 is an example of the application in Ada of the concept of a generic task. As mentioned previously, the function of the CRT page and related control hardware is to provide information to the instructor and further to allow the instructor to insert information into the simulation problem by creating an online page editor. The page created simply defines data associated with a particular function and the generic display task can instantiate any type of page. As can be seen from the figure, all actions associated with the page are defined in the generic task. Thus, by creating a specific and unique data base for each function, a variety of display pages can be created.

In an effort to make the software as flexible and reusable as possible, the display task was designed such that the user can create and modify screens of data (called frames) without re-compiling the code. To create a display frame, the user will select the CREATE FRAME option from a menu, respond to the prompt and name the new frame. The user will then see the Edit Field menu in Figure 8. From this menu the user can insert, delete, copy and move fields on the frame as well as create and delete other frames. If the user selects INSERT FIELD, the Insert Field menu in Figure 9 will appear. To monitor a certain value in the simulator datapool,

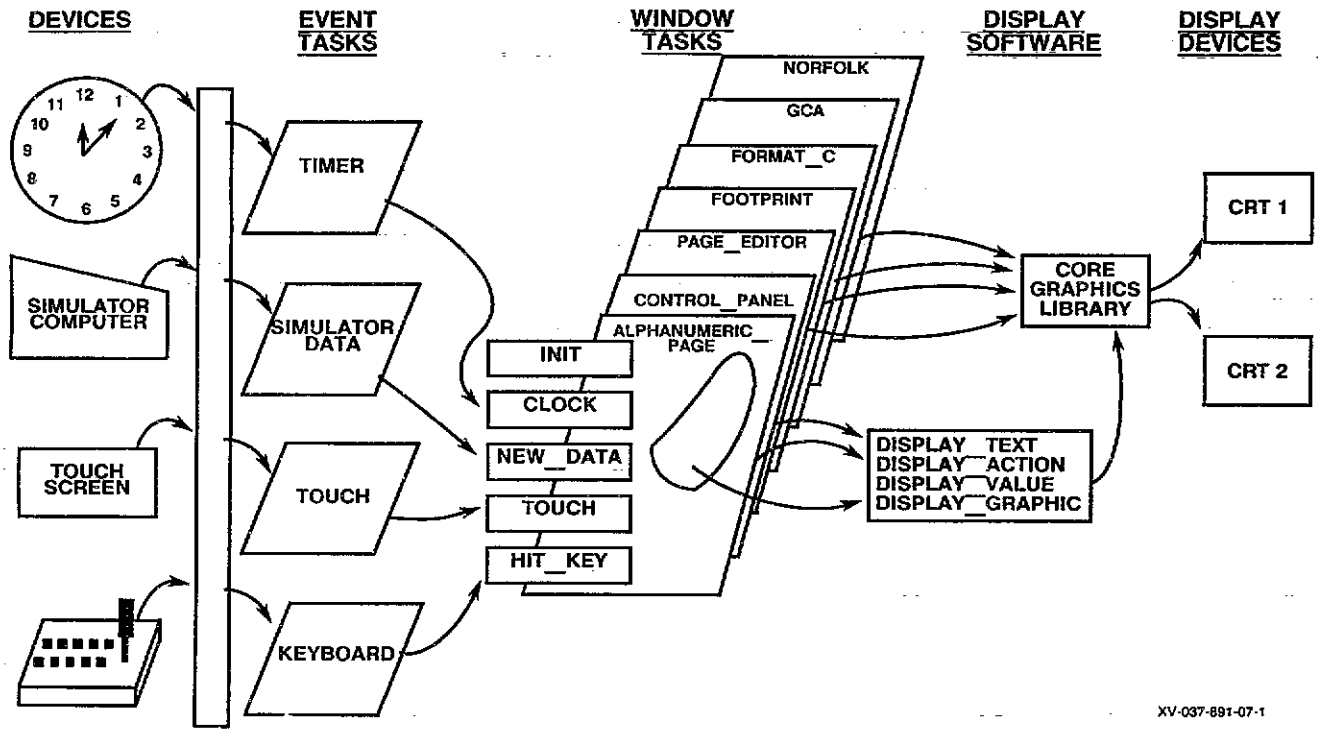


Figure 6. Current MODIOS Software

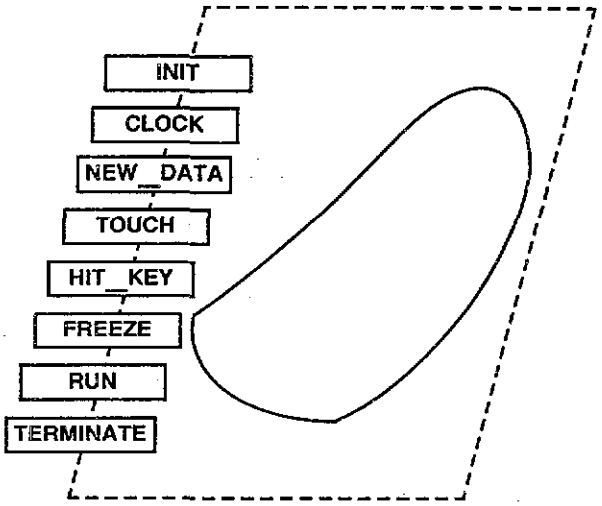


Figure 7. A Generic Display Task

such as altitude, the user will touch the appropriate square and see a prompt to enter the datapool variable name. A prompt will then appear to touch the desired location. The user will touch the desired location on the screen and the value for the desired parameter will be displayed at that location whenever that frame is displayed. The user will then touch the INSERT TEXT option, insert via keyboard the desired text and touch the desired location. To insert a control function, such as NEXT PAGE, the user will touch the INSERT TOUCH SCREEN option, designate the location on the frame, select the desired action (GO TO FRAME X) from a menu and name the desired frame. With this level of flexibility in frame design the reusability of the software is assured.

LESSONS LEARNED

The work discussed in this paper was conducted as an IR&D project by the Harris Corporation and utilized the facilities of the Visual Technology Research Simulator (VTRS) at NTSC facilities in Orlando, Florida.

In the pursuit of completing the overall design and the resulting implementation, many lessons were learned. These included a much deeper appreciation for the power of Ada and more insight into the design of a generic instructor station console. Throughout the design and development, emphasis was placed on the use of Ada and Ada design methodologies. Results were gathered throughout the project and more specifically during a user evaluation. The following paragraphs summarize these findings.

Data were gathered during three phases of the project.

a. Software Development;

b. Hardware/Software Integration;

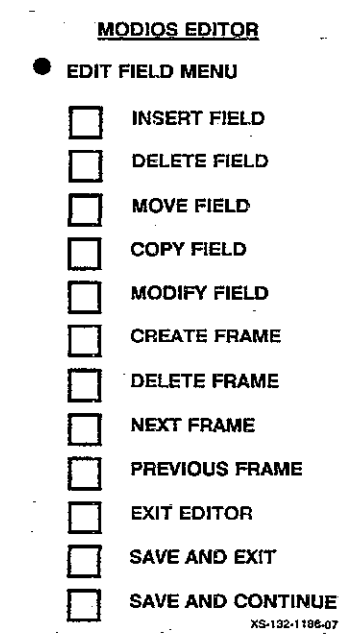


Figure 8. Screen Editor Options Menu

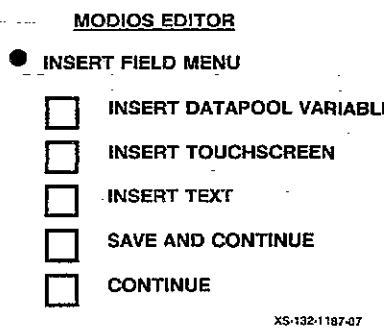


Figure 9. Insert Field Options Menu

c. User reaction to the ability of the design to effectively train and operate the Visual Technology Research Simulator (VTRS).

The purpose of the evaluation was to develop data to validate the instructor console station concept, and provide feedback for system improvement. The scope of the evaluation was established by a list of twenty-two evaluation questions which covered both general and specific areas of investigation. Participants in the evaluation, during the software development and hardware/software integration phases, included both Harris and Naval Training Systems Center technical team members.

Software Development and Ada

Findings with respect to Ada as an implementation tool are consistent with those being reported with other Ada projects. Ada is a robust language that offers many capabilities that did not

previously exist in many high level languages. Since an object oriented design approach was utilized on this project, project personnel had to learn Ada as well as how to design with object definitions in mind. Ada provided a direct application of the design in which objects were mapped easily into a programming solution. A very strong front end definition was obtained. The major finding with respect to Ada is that the training of Ada and Ada design methodologies is less painful than originally expected. Additionally, the software integration took approximately two weeks to complete. This quick integration process can be directly attributed to the front end definition and the level of abstraction that was attainable with the use of the Ada language. Debug time was minimal and strongly aided with an effective symbolic debugger. The use of generics expedited the addition of program units. A new task that consisted of a color driven graphics display was designed, coded and tested in less than one week. This was achievable due to the highly structured code, as well as reusable software packages designed into the framework of the instructor console software.

Hardware/Software Integration

The software system was developed on a SUN 3/160C computer system utilizing the Verdix Ada compiler. The system offered an excellent tool in which to do software development. It has the power of the Unix operating system, and the processing power to comfortably support four to five software engineers. This system served as both the development platform and the target run-time system. The conclusion reached is that the Sun workstation offered an excellent development environment but could not provide the processing power required to achieve the iteration rate required for real-time operations in the configuration used. Table 3 summarizes the results of the execution speeds. The slow updates are attributed to the Unix operating system and how input/output is now implemented by the operating system. Bypassing Unix I/O drivers and directly addressing the graphics processor should speed up the through-put rate to an acceptable level. From all indications, given a different I/O interface, Ada is suitable for real-time applications.

Table 3. Execution Times

MEASUREMENT	TIME (MSECS)
SYSTEM TIME CALL	.383
GRAPHIC WINDOW TASKS:	
ALPHANUMERICS	
DISPLAY A PAGE	2208.000
DISPLAY MAIN MENU	2141.000
CONTROL PANEL	
CLOCK UPDATE (ENTIRE) (0:00:00)	83.000
FORMAT C DATA TABLE	
ENTIRE DISPLAY UPDATE	1588.000
GCA GRAPHIC	
ENTIRE DISPLAY UPDATE	449.000
GRAPHICS UPDATE ONLY	422.000
MAP GRAPHIC	
ENTIRE DISPLAY UDATE	199.000
GRAPHICS UPDATE ONLY	189.000

XS-132-1188-07

User Reaction

User reaction was outlined by utilizing ten Marine Corps helicopter and fixed wing pilots. A short training scenario was used to expose each instructor pilot to the generic console. After each pilot completed an exercise, detailed questions were answered. User reaction to the use of the generic instructor console was very positive. The operator console was used to control a training mission involving the SH60B helicopter simulator. A limited selection of features including: call up of initial conditions, in-flight store and recall, change of instructor controlled parameters, hierarchical menus, and the page editor were implemented successfully. The generic instructor console station did adapt to the control needs of the VTRS. Specifically, the instructor pilot's reaction can be summarized as follows:

a. The touchscreen approach with many control features selectable by touching the face of the display screen is an efficient and effective way to control the training problem. The instructor pilots particularly liked the way in which the in-flight store feature was implemented.

b. All subjects noted that the use of color for CRT displays added to the readability and presentation of the training mission parameters.

c. No subject experienced difficulty in console operations. Four of the subjects remarked that the console was easier to operate than those that they had used before. The menu driven approach allowed access to all information within a hierarchical structure.

d. All subjects were able to use the instructor console station within a 15 minute training and orientation period.

CONCLUSION

The results of this work lead the authors to two basic conclusions. First, the concept of a generic IOS is certainly feasible. Hardware technology has reached a level where common modules can be structured to implement all requirements in an instruction console, particularly if designers maintain open architecture such as VME, multi-BUS, PC, BUS, STD bus, etc. Further, by designing the software using a data driven methodology, major elements of the instructional software itself can be used in a variety of trainers.

The second conclusion deals with the use of Ada. Our experience has been that Ada allows you to reach an operation state much quicker than was previously experienced. The problem of servicing I/O must be overcome and from indications

among the commercial software developers this is happening. Adding new tasks to an Ada system is as advertised, i.e., simple and easy, thus lending more credence to the generic IOS concept.

BIBLIOGRAPHY

1. Caro, Paul W., Pohlmann, Lawrence D., and Isley, Robert N. (1979) Development of Simulator Instructional Feature Design Guides, Seville Technical Report TR 79-12, Seville: Pensacola, Florida 32505

2. Charles, John P. (1983) Device 2F119 (EA-6B) WST Instructor Console Review, Technical Report: NAVTRAEQUIPCEN 81-M-1083-1, Naval Training Systems Center, Orlando, Florida 32813

3. Charles John P. (1984) Design Guidelines for Trainer Instructor/Operator Stations, Technical Report: NAVTRASYSCEN 83-C-0087-1, Naval Training Systems Center, Orlando, Florida 32813

4. Faconti, V., Mortimer, C. P. L., Simpson, D. W. (1970) Automated Instruction and Performance Monitoring in Flight Simulator Training, AFHRL-TR-69-29, Air Force Human Resources Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio

5. Stark, E. A., Faconti, V., Mortimer, C. (1971) Study to Determine the Requirements for an Experimental Training Simulation System, NATRADEVCEN 69-C-0207-1, Naval Training Systems Center, Orlando, Florida 32813

6. Charles, John P., (1983) Device 2E6(ACMS) Air Combat Maneuvering Simulator Instructor Console Review, Technical Report: NAVTRAEQUIPCEN 82-M-0767-1, Naval Training Systems Center, Orlando, Florida 32813

7. Osborne, S. R., et al. (1983) Three Reviews of the Instructional Support System (ISS) Concept, Technical Report: NATRAEQUPCEN 81-C-0081-1, Naval Training Systems Center, Orlando, Florida 32817

ABOUT THE AUTHORS

MR. VICTOR FACONTI is the Manager of Advanced Engineering, Harris-GSSD/Orlando. He is responsible for system analysis and design for the simulation product line, and has technical and financial responsibility for operation IR&D program, direct support to all Business Development activities and system design for ongoing projects. Mr. Faconti has over twenty years experience in the design of flight simulation systems. He was technical champion in the area of instructor stations and features design. He holds a BS (1964) and MS (1968) from Adelphi University and and has several patents and publications in the area of flight simulation system design.

DR. BRUCE MCDONALD received his Ph.D. in Industrial Engineering from Texas A&M University. As a Program Engineer with Harris Corporation, he manages programs in the advanced Engineering Division. Dr. McDonald has extensive research and applications experience in training device concept formation, design and evaluation as well as user-computer interaction.