AIAA Meeting Papers on Disc, 1996, pp. 342-352
A9635039, AIAA Paper 96-3513

Multiprocessor computation in the research flight simulation environment

Matthew W. Blake
NASA, Ames Research Center, Moffett Field, CA

Joseph R. King
NSI Technology Services Corp., Sunnyvale, CA

Craig Piers
NSI Technology Services Corp., Sunnyvale, CA

AIAA Flight Simulation Technologies Conference, San Diego, CA, July 29-31, 1996, Technical Papers (A96-35001 09-01), Reston, VA, American Institute of Aeronautics and Astronautics, 1996

We describe a new multiprocessor implementation of a very complex research flight simulation using essentially 'commercial off-the-shelf' hardware and system software. A description of the simulator and its primary components is provided, followed by a description of the real-time system as implemented. The real-time system spans three synchronized processors and can be expanded to 11 processors on the current hardware platform. The implementation has provided a common environment between the multiprocessor real-time computer, integration and test station, and desktop development systems. The system has provided significant performance improvements and increased efficiency of operations and support. Several potential improvements are also described. (Author)

96-3513
A96-35039
AIAA-96-3513-CP

MULTIPROCESSOR COMPUTATION IN THE RESEARCH FLIGHT SIMULATION ENVIRONMENT

Matthew W. Blake, NASA Ames Research Center
Joseph R. King, NSI Technology Services Corporation
Craig Pires, NSI Technology Services Corporation

ABSTRACT

This paper describes a new multiprocessor implementation of a very complex research flight simulation using essentially Commercial Off The Shelf (COTS) hardware and system software. A description of the simulator and it's primary components is provided followed by a description of the real-time system as implemented. The real-time system spans three synchronized processors and can be expanded to 11 processors on the current hardware platform. The implementation has provided a common environment between the multi-processor real-time computer, integration and test station, and desktop development systems. The system has provided significant performance improvements and increased efficiency of operations and support. Several potential improvements are also described.

INTRODUCTION

In the aircraft training simulator market, simulations often run on multi-processor computers to gain the cost efficiency of fewer computer systems utilizing less expensive processors. The simulation developer incurs the time and expense of partitioning the software and verifying the complex interaction and timing between processors only once, then numerous identical copies of the training simulator are sold. In the research environment the software is constantly being modified to perform a new function. Under this environment, it has traditionally been easier to develop and test the simulation by having the code run sequentially on one processor. This avoided the difficult verification task of ensuring each process was independently functioning correctly and that each process was communicating in the proper sequence with the other processes. Unlike the training market, in the research environment there is no economy of scale for any extra expense in developing and testing the simulation across multiple processors since additional copies of the simulation are not created and sold.

When used for research purposes, flight simulation has generally focused on one flight regime and only that part of the vehicle and it's systems are simulated. In the case of full-mission simulation of a commercial transport, the entire environment that the crew experiences must be provided. If all systems are simulated (no actual avionics boxes), the size of the simulation software gets extremely large. The Advanced Concepts Flight Simulator (ACFS) is one such simulation, representing a complete full-mission simulation of a commercial transport aircraft and it's related systems and environment. The ACFS software includes approximately 500,000 lines of code, split approximately evenly between the programming languages FORTRAN and C. The ACFS is one of two full-mission flight simulators at the Crew-Vehicle Systems Research Facility (CVSRF) at NASA Ames Research Center. The ACFS is used to study aspects of human factors in aviation safety as well as methods to improve aviation operational efficiency.

The ACFS simulates a generic advanced twin engine mid-range transport (similar to a Boeing 757). The ACFS consists of a reconfigurable cab on a 6 degree-of-freedom motion platform with a color night/dusk Out-The-Window (OTW) visual system. The cockpit includes control sticks and rudder pedals, conventional center console with throttle levers and control panels, an overhead panel for control of other aircraft subsystems, and 8 video display screens across the main panel for flight, guidance, navigation, and status displays. There is a Mode Control Panel (MCP) on the glare-shield for autoflight control and two Control Display Units (CDU) for interfacing with the Flight Management Computer (FMC) simulation. This configuration provides full-mission aircraft functionality including complete auto-flight, auto-throttle and Flight Management System (FMS) capability, yet the hardware and software configuration can be changed as needed to address specific research requirements. The ACFS can be configured to operate with the other CVSRF simulator, a Federal Aviation Administration (FAA) certified Boeing 747-400 simulator as well as an elaborate Air Traffic Control (ATC) simulator. Unlike most training simulators and the 747-400 simulator at the CVSRF, the ACFS does not include any actual aircraft avionics boxes so the entire simulation is fully programmable to address any research requirements. Figure 1 shows a system diagram of the main components of the ACFS.

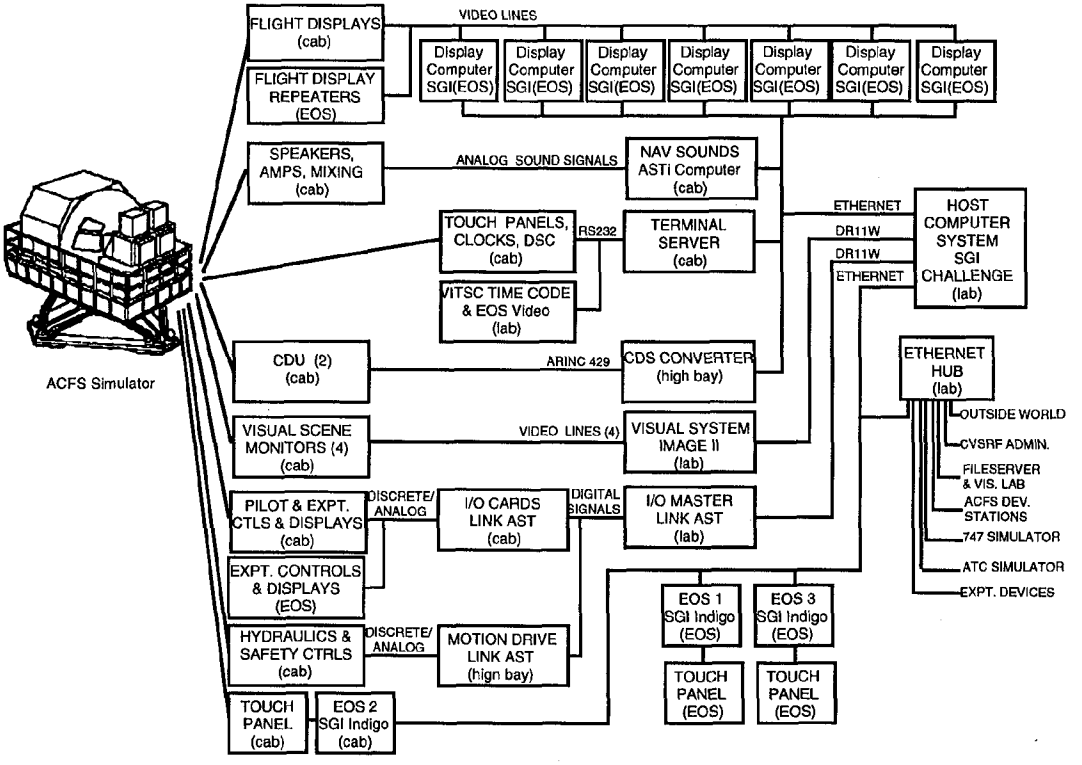

Figure 1: ACFS System Diagram

The ACFS was originally built in the mid-1980s and has gone through many upgrades and enhancements since. This paper addresses the recent replacement of a late 1980s Digital Equipment Corp. (DEC) VAX computer. In replacing this computer system, the goal was to increase reliability, provide significant spare processor and memory capacity, and greatly increase the efficiency in developing, testing, and operating simulation experiments without creating extensive custom real-time control software. The goals of increased reliability and direct performance (processor speed and memory) were easy to achieve as many computers provide improvements in these areas over the mid-1980s VAX that was being replaced. To achieve the goal of improved productivity using COTS systems in a research flight simulation environment led to the purchase of a Silicon Graphics Inc. (SGI) Challenge host computer and a set of SGI workstations.

HOST COMPUTER ARCHITECTURE

The real-time host computer is an SGI Challenge L multi-processor computer capable of housing 12 processors. The current implementation described here utilizes only four of the available processor slots. Each of the processors is a model R4400 with a 200 MHz. clock speed. The supporting single processor SGI workstations include Indy, Indigo, and Indigo2 models with various processors.

MEMORY MANAGEMENT

The SGI Challenge host computer is configured with 128 Megabytes of main memory, expandable to 2 Gigabytes. All critical program variables and constants are assembled in one section of shared memory called Global Common. The current Global Common utilizes approximately 634K bytes of memory. The Global Common memory is partitioned into sections along aircraft systems boundaries such as navigation systems, flight dynamics, etc. A common Global Common file is processed into both C and FORTRAN INCLUDE structure files so that the C and FORTRAN routines will map to the same variables. Other processes, executing in other processors, that access the shared memory include the data collection write to disk process, the Experimenter/Operator Station (EOS) processes, and the FMC process.

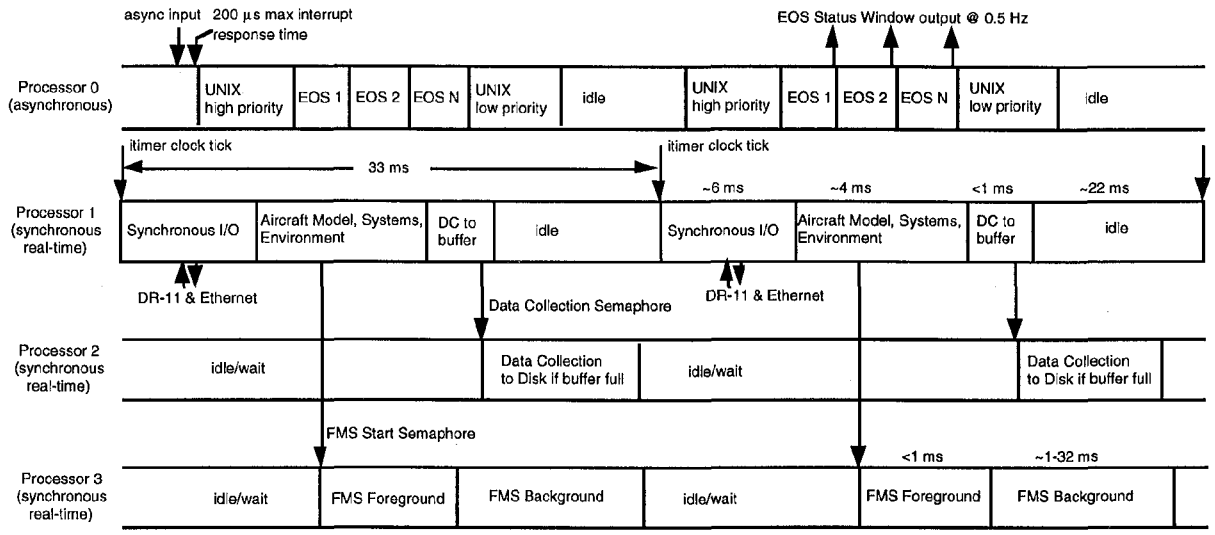

Figure 2: Sample processor activity time-line

PROCESS CONTROL AND SEQUENCING

SGI's IRIX (Version 5.3) Operating System (OS) offers a number of services and routines to control the execution of processes running on their multi-processor computers. These services include:

- Locking processes into memory
- Isolating processors from any external activity and allowing only the selected processes to run on those processors
- Setting process priorities that will run at levels higher than any other "User" process and not be adjusted
- Synchronization of processes on different processors

Using these OS features provides the ability to force real-time behavior on all processors other than the one running non-real-time UNIX processes. The processor load is currently split as follows:

- Processor 0: UNIX OS, EOS, Asynchronous Input/Output (I/O)
- Processor 1: Synchronous I/O, Main simulation calculations
- Processor 2: Data Collection
- Processor 3: FMC process

Processors 1 - 3 are run in an isolated mode with only the specified processes locked in place and at the highest priority. Since some important simulator functions are still performed on processor 0, a "Single User" mode was developed. In this mode, all non-essential processes and services are disconnected from the simulator. Outside network activity is minimized by isolating the real-time networks with filtering bridges and routers within the Ethernet communication hub. At simulation start-up, all essential files are copied to local disks so that simulator operations can be performed independent of the shared file-server system by unmounting the file-server disks from the real-time host. This isolation mode allows full use of the file-server for development while minimizing impact to the real-time simulation.

Frame timing is currently provided by the ITIMER utility providing a resolution approaching 1 microsecond.

Communication between the various real-time processes running on different processors is via shared memory Global Common (GC) with synchronization using SGI Arena/Semaphore signals. The Arena/Semaphore signals have a guaranteed latency of less than 200 microseconds and in practice have provided approximately 10 microseconds latency. Figure 2 shows a high level diagram of the time line of two real-time frames with the process semaphore signals and main processor activities.

MAIN MODEL & SYNCHRONOUS I/O PROCESS

The main aircraft model and synchronous I/O functions are performed by processor 1. The model is partitioned into approximately 30 subsystems such as acrodynamics, engines, FMC communication, I/O to instruments, etc. Each subsystem may contain hundreds of subroutines. Execution of these subsystems is controlled by the model scheduler (MODSCH). MODSCH controls in what order and how often each subsystem is executed.

MODSCH determines this schedule based on an ASCII database called the Model Attribute Table (MAT). By modifying this MAT database, operators can quickly change the characteristics of the simulator by enabling/disabling modules or increasing/decreasing the execution rate. The 30 subsystems are partitioned into approximately 120 MAT table entries to increase scheduling flexibility. The highest frequency loop is currently 30 Hz (33 ms per frame). Figure 3 shows a high level diagram describing one 33 ms frame of MODSCH. Figure 4 shows the sequencing attributes of one subsystem as described in the MAT table as well as the definitions of MAT table options.

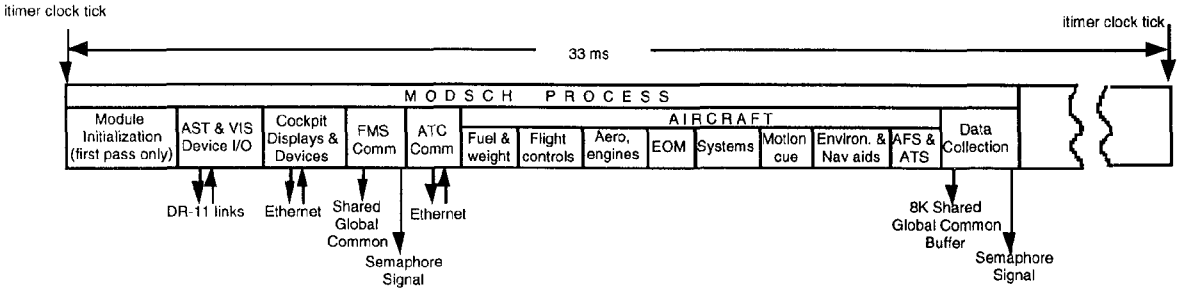

Figure 3: MODSCH timeline for one frame

MAT.DAT TABLE EXTRACT:

```
MODEL_NAME:                 FLAPSLAT
ENABLED:                    ENABLED
FRAMES:                     MASK 010000010000010000010000010000
GROUP:                      FCSURF
TIMES_CALLED_INIT:          2
DELAY_BEFORE_RUN:           NO
RUN_DURING_FREEZE:          NO
RUN_DURING_RESET:           YES
RUN_AFTER_EXCEPTION:        NO
RUN_DURING_STOP:            NO

MODEL_NAME:                 SPOILERS
ENABLED:                    ENABLED
FRAMES:                     MASK 000010000010000010000010000010
GROUP:                      FCSURF
TIMES_CALLED_INIT:          2
DELAY_BEFORE_RUN:           NO
RUN_DURING_FREEZE:          NO
RUN_DURING_RESET:           YES
RUN_AFTER_EXCEPTION:        NO
RUN_DURING_STOP:            NO

MODEL_NAME:                 NEW_AEROCS
ENABLED:                    ENABLED
FRAMES:                     ALL
GROUP:                      AERO
TIMES_CALLED_INIT:          10
DELAY_BEFORE_RUN:           NO
RUN_DURING_FREEZE:          NO
RUN_DURING_RESET:           YES
RUN_AFTER_EXCEPTION:        NO
RUN_DURING_STOP:            NO
```

MAT.DAT TABLE DEFINITIONS:

MODEL_NAME - Function name as listed in MODEL.STC (max. 15 bytes)

ENABLED - Model is active this run: ENABLED - the model will be called as requested.
DISABLED - in the MAT but disabled (i.e. not called).

FRAMES - Which frames model is called; held in a 30 element mask.
ALL - all frames (111111 ...)
EVEN - all even frames (i.e. 101010 ...)
ODD - all odd frames (i.e. 010101 ...)
NONE - only called during initialization mode.
MASK - a binary mask (e.g. MASK 001001001001)
SEQ - a number sequence (e.g. SEQ 1,5,8)

GROUP - the name of a model group defined in file acfs_dat:
MODEL_GROUP_FILE.DAT. Each model is associated with one group. The default is NONE. Models should be assigned to groups of related models.

TIMES_CALLED_INIT - How many times model should be called during initialization mode. (Range: 0 through 200)

DELAY_BEFORE_RUN - a model can be temporarily kept from running immediately after initialization or after each reset for a preset number of cycles. Each process can be set to delay a different number of cycles.

YES - delay this model before running
NO - run this model according to schedule

RUN_DURING_FREEZE - model continues to be scheduled during freeze mode.

YES - call this routine during freeze mode.
NO - Don't call this routine during freeze mode.

RUN_DURING_RESET - model is called with initialization mask during RESET mode.

YES - call this routine during RESET mode.
NO - Don't call this routine during RESET mode.

RUN_AFTER_EXCEPTION - model continues to be scheduled even after an exception has been caught (e.g. zero divide).
YES - continue calling this routine despite error.
NO - stop scheduling this model after an exception.

RUN_DURING_STOP - model is called once while the simulation is stopping.
YES - call this routine during STOP mode.
NO - Don't call this routine during STOP mode.

Figure 4: Sample MAT table and MAT table definitions

FMC PROCESS

The FMC simulation is computed on processor 3. The FMC simulation is run as two processes; a Foreground process and a Background process. The Foreground process handles the I/O to the CDUs, I/O to the aircraft model, and computation of guidance information (comparison of actual to planned path and needed corrections). The Foreground process is run synchronously with the aircraft model running in processor 1. This provides real-time response to crew actions (keystrokes) and for continuous guidance to the aircraft autoflight system. Communication of data is through Global Common shared memory and initiation of the FMC Foreground process is through semaphore signals. The Background process handles flight planning which includes establishing way point and path information, aircraft performance computations, navigation data processing, and processing the CDU page formatting. The Background process runs in an asynchronous mode, and runs whenever calculations are required and when the Foreground process is not running. This design is consistent with the software operation of most commercial transport aircraft FMC avionics systems.

DATA COLLECTION PROCESS

To conduct human factors research, large amounts of information regarding complete status of the simulated aircraft systems and all crew activities must be recorded for post simulation analysis. To accomplish this requirement the ACFS Challenge host computer system was configured with sufficient hard disk capacity to save the collected experiment data during the real-time simulator operation. The current configuration provides 2 Gigabytes of space for data collection. Software tools have been developed to post-process the data into other formats to meet researcher needs or provide a "quick-look" at the data for verification.

Implementation of the data collection software was accomplished using three run-time modules and an off-line data definition compiler. The host initialization module (DRG_INIT) and the real-time module (DRH_REC) runs in processor 1 within the main real-time load. A slave process that performs disk I/O (DRG_SLAVE) runs on processor 2.

Setup of data collection begins with a specification of the desired simulation variables and collection rate (between 1 Hz and 30 Hz). This list is edited into an ASCII data definition file with the suffix of .rec. This file is compiled with the off-line data definition compiler (DRC) program to create the runtime files with the .dra, .drs, and .log suffixes.

During real-time operation an 8K byte buffer residing in the Global Common shared memory area is packed with the predefined experiment data at the end of each 30 Hz frame by the DRH_REC module running in processor 1. The DRH_REC module then sends a SGI Arena/Semaphore signal to the DRG_SLAVE module running on processor 2. The DRG_SLAVE module then starts up and moves the 8K byte buffer into one of two 121K byte disk I/O buffers and checks to see if the 121K buffer is full. If it is full, DRG_SLAVE identifies the alternate I/O buffer as the current collection buffer for use at the end of the next simulation frame. DRG_SLAVE then initiates a disk write operation of the full buffer. This utilization of two buffers is called double buffering.

Run-time control of the data collection system is through the EOS. The data collection file to be written to disk is comprised of three main sections. The first section contains header information which is entered from the EOS and includes the simulator crew names and/or numbers, experiment number, run number, date, etc. The second section contains definitions of the data to be collected which is taken directly from the pre-processed list. The third and largest section of the data file is the actual experiment run data collected in real-time.

EXPERIMENTER/OPERATOR STATION PROCESSES

The Experimenter/Operator Station (EOS) currently runs on processor 0. The process runs at a high priority non-real-time status. The EOS is similar to the Instructor/Operator Station (IOS) on a conventional training flight simulator but features have been added to the ACFS EOS to support the research requirements. The ACFS EOS configuration was designed to allow complete simulator control from the on-board EOS station, at the rear area of the cockpit enclosure, or from a remote control room area located within the facility. Any number of EOS processes can run concurrently so several users can monitor different simulator activity simultaneously.

SGI Indy workstations were selected for the EOS to provide the simulator user the desired Graphical User Interface (GUI). The EOS process actually runs in the SGI Challenge host computer via remote login. Communication between the EOS Indy systems and the host uses standard Terminal Control Protocol/Internet Protocol (TCP/IP) protocols imbedded within the SGI computer OS. The host resident EOS processes are executed at a high priority non-real-time mode. Touch sensitive screens were installed on each of the 19" EOS displays and the display format was designed to use the touch screen or trackball/mouse input to activate functions and enter data.

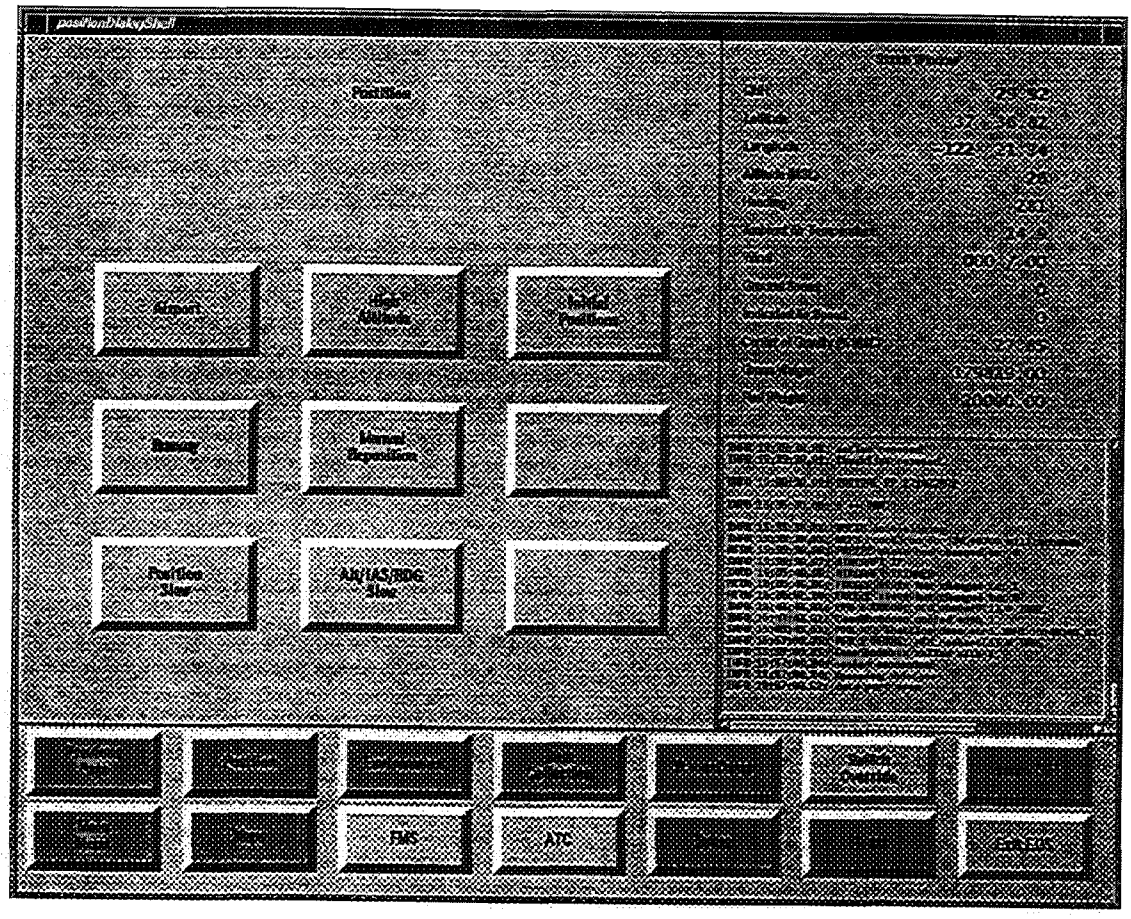

Figure 5: The Position Page screen of the EOS

The EOS graphical displays (or pages) were developed with the Builder Xcessory (BX) GUI editor developed by Integrated Computer Solutions (ICS) Inc. The main emphasis of this approach was to provide fast prototyping of new EOS pages. This was achieved by customizing the BX editor to become an EOS development editor (called EOS_bx). The EOS_bx is linked with the ACFS shared memory (or its own local version for isolated development). A set of customized Motif widgets were integrated with BX. These widgets have predefined properties and a small set of fields targeted for the ACFS-EOS environment. The ACFS EOS widgets are as follows:

- EOS Simple Button
- EOS Arrow button
- EOS Form
- EOS Set Value
- EOS Temporary Key Pad
- EOS Scale
- EOS Page

The EOS process is a conventional Xwindow process so it can be "pushed" and "popped" with other graphical windows as well as scaled to different sizes.

The EOS screen layout consists of four main areas; the Page Area (center and upper left), the Footer (along the bottom), the Status Window (upper right), and the Message Logger (below the Status Window). An example EOS screen is shown in Figure 5.

The Page Area is the main window of the EOS. It contains buttons and displays that allow the operator to call up different pages, input values, and monitor values. There are many pages in the EOS that can be selected and appear in the Page Area. The Footer, also referred to as the Hot Button Area, displays buttons which are always accessible, regardless of what page is displayed in the Page Area. Buttons here indicate if they cause an action, such as starting the FMS, or call up a page, such as Environment. The Status Window is a dynamic display of aircraft and environment conditions. What is

displayed in the Status Window can easily be changed for each experiment. The Message Logger window provides important system or simulation messages to the operator. The EOS currently has pages supporting the following functions:

- Manual Reposition: Allows the operator to position the aircraft by entering latitude, longitude, heading, altitude, gross weight, and fuel weight.
- Initial Positions: Several "canned" positions, including aircraft configuration.
- Environment: Allows control of weather conditions, turbulence and visibility.
- Visual Control: Allows control of the OTW visual system.
- Preset Weather: Allows control of visual scene ranging from clear to zero/zero conditions.
- Aircraft Setup: Controls aircraft configuration and ground facilities.

Additional pages are created for each experiment's specific requirements.

Buttons on the EOS are color coded to indicate their type of action. Blue colored buttons call up new pages, pop-up windows, or input value windows. Pink colored buttons cause a specific action to take place. Pink button actions that take place include Screen Dump, Freeze, starting and stopping the FMS and ATC link, and Reset. Some of the pink action buttons also can be a different color to indicate a different current status. These specialized buttons are as follows:

- FMS: The color of the FMS action button indicates the state of the FMS. When the FMS is active, the button is yellow. When the FMS is not active, the button is pink.
- ATC: The color of the ATC action button indicates the state of the ATC link. When the ATC link is active the button is yellow. When the ATC link is not active, the button is pink.
- Reset: This button sets the simulator back to the last entered Initial Position (IP), regardless of whether the IP was entered manually or preset. The button turns yellow during the reset and back to pink when the reset has finished.
- Freeze: This button toggles the Simulation in and out of the Freeze State. The state of the Simulation is displayed with text and color on this button. Freeze on is indicated by red and the words "Freeze On". Freeze off is indicated by pink and the words "Freeze Off".
- Exit: Activating this button exits the EOS utility program.

Buttons are activated when contact is released, not initiated. If a button is accidentally pressed utilizing this technique, the operator's finger can slide contact to the side and disarm the button without activating it. Also, the operator can prepare for activation by pushing the button then watch some other screen for information on when to activate. The operator then does not need to look at the EOS screen, he simply releases his finger from the screen to activate the function.

COMMUNICATION TO SUBSYSTEMS

Communication with most simulator subsystems is done with UNIX socket procedures over Ethernet lines. This includes the communication with eight flight display computers, the sound generation computer, the EOS computers, other simulators (the ATC simulator, Boeing 747 simulator, or other simulators not co-located), and several additional devices. Standard TCP/IP communication protocols have been employed in order to simplify support for communications with the wide variety of platforms that are used on the ACFS, and to provide an easily supported connection capability to research systems.

Different TCP/IP packet protocols are used for different devices. The TCP packet protocol is used for asynchronous stream devices such as the touch screens and alpha-numeric displays. The User Datagram Protocol (UDP) packet protocol is used for the large synchronous data packets that drive the flight displays. Multicast UDP protocols are used for the Primary Flight Displays (PFD) due to the large amount of shared data used for the Captain's and the First Officer's displays. This allows both processors to receive the same data and reduce the amount of real-time network traffic. UDP packet protocol has the advantage over TCP/IP packet protocol of being much lower overhead on the processor due to the connectionless nature of the communication. Although UDP does not have as low an overhead as "raw" protocols, it does have the advantage of being widely accepted. This eases development by insuring documentation and debugging tools are available and simplifies routing to other networks to meet specific experiment communication requirements. It has been determined that the overhead difference between UDP and raw protocols is minor for the ACFS application.

By utilizing standard UNIX socket communication, virtually the same code can support both real-time experiment operations on the Challenge computer and simulation development on individual workstations. The remote processes (such as the flight displays) can run on the local development station using the same socket for internal communication rather than over an Ethernet line. Although the processes run slower in the development environment, the performance is usually adequate for developing or debugging new systems.

Communication with the OTW visual system and the simulator I/O device driver (a Singer-Link AST device for reading and writing to analog systems) is performed utilizing VME boards emulating a DR-11W interface. This is required as these systems are old enough that they do not support Ethernet communications.

SIMULATION ENVIRONMENT

SIMULATION OPERATION

Loading, starting and stopping the simulation is accomplished through graphical menus that execute script files. The same executable can be used in a variety of configurations and "customized" by the setting of environment variables and run-time parameters.

The normal procedure is to log into the ACFS account from one of the SGI Indy EOS workstations. A Motif compliant ACFS Simulation Control window appears and provides a list of the currently available simulator bases and simulator tool options for subsystems such as the FMS and for development tools. Desired options are selected and the Execute button activated to initiate the start-up operation. It requires approximately 30 seconds for the simulator to automatically load the software to all subsystems from the file-server or local disk and start up the entire simulation.

Once the simulation software is loaded the simulator is always in one of the following modes:

- Reset
- Freeze
- Freeze Off (run)

When the simulation has been reset, the simulation is returned to an Initial Position (IP). The desired IP can be selected from preprogrammed options available on the EOS Position Page. The IP may be on the ground at many different locations or in flight at some desired location, altitude, airspeed, etc. The Freeze mode stops a majority of the modules (aircraft, environment, data collection) and leaves the simulation in what appears to be a state of suspended animation. During Freeze, many system and I/O modules continue to function to allow the operator to inspect and control the simulation. All normal simulator operation continues when the Freeze is de-selected.

All simulator control functions and unique experiment functions are controlled from one or more of the EOS pages. The termination of the simulation is accomplished by selecting the Exit EOS button at the bottom right of the EOS display and then selecting Stop on the ACFS Simulation Control screen.

SIMULATION DEVELOPMENT

Simulation development is performed on one of three platforms. For initial development of an experiment, the engineer runs a version of the ACFS software (called the mini-ACFS) on a desktop SGI workstation. The mini-ACFS and full simulator have nearly identical software. The main aircraft model, FMC, and EOS are essentially identical. The communication to the flight displays is the same, but the socket connects to an abbreviated version of the display software that is running on the local workstation, rather than over an Ethernet line to the display computer for the cockpit. A few additional mini-ACFS specific control and display tools are available to make up for the lack of cockpit hardware availability. All these processes run in the one processor in the user's desktop machine instead of being distributed in the Challenge computer and over Ethernet lines in other computers. The system does not provide the process isolation and clock timing necessary for real-time and the limited processor capability limits the system to running slower than real-time. However, this is generally adequate for initial experiment development.

As the development process progresses, the engineer moves to a more powerful integration and test station that has several video display screens (vs. the one on his/her desk) in order to verify the functionality and interaction of the new system with all the ACFS software subsystems. Again, the development station utilizes essentially the same code that is used on the full mission simulator, only differing in some control and scheduling routines. Due to the increased screen space, this configuration can run the exact flight display software. As with the desktop mini-ACFS, the test and integration station does not run real-time, however this is generally adequate for most debug and test procedures.

For final test and verification, the entire ACFS is utilized. This provides the exact environment that will be used for experiment operations. This is particularly useful for tuning scenario code and event timing for experiment-specific operations. Following this phase, experiment operations and data collection takes place. Figure 6 shows a diagram representing the development, integration and test, and real-time facility interface.

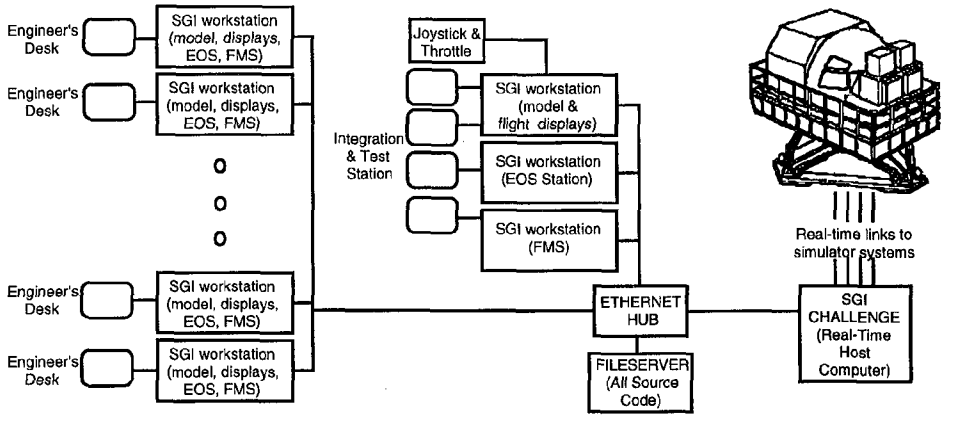

Figure 6: ACFS development, integration & test, and real-time interaction

DEVELOPMENT TOOLS

For any stage of experiment development or operations most of the same tools are available. For a complete suite of debugging tools, the user compiles in debug hooks (a compilation flag) and then invokes the SGI CaseVision tool set. This tool set includes a Graphical User Interface (GUI) front end to the debugger, a static code analyzer, complete interactive help and documentation, and much more. The primary tool used for ACFS development has been the CaseVision Debugger (CVD). CVD is a modern graphical-based debugger providing a high-level language-based source code display and complete features such as break-points, single step, display and modify. Although CVD extacts a small performance penalty, the impact when used in real-time for process inspection has been negligible. Use of the tool in this way does not cause frame over-runs.

If the user wants to simply inspect and deposit into application code variables, he invokes an in-house built tool called the Global Common Utility (GCU). This tool provides access to all Global Common variables (which includes most parameters users wish to inspect or change).

In addition, there are other tools available for plotting and analysis. The current plotting process uses a predefined set of Global Common variables edited into an ASCII file. A plotting data file is written to during the real-time experiment run. The public domain plot utility PLOTMTV is used to plot and print the collected data.

CONFIGURATION MANAGEMENT

Software modifications and upgrades to the ACFS are made frequently to support the ongoing human factors research. Some changes to the simulation software are retained to become part of the baseline, others are temporary for a given experiment and are removed and archived. At any one time there may be up to a dozen engineers working on various experiments or doing software maintenance for the ACFS. However, to ensure integrity of the simulation during actual experiment operations, all development activity is isolated from the real-time systems. A rigorous configuration management system is required in order to track these continuous software and data modifications efficiently. To meet these constraints and provide a productive development environment, all source files and data are maintained on a file-server. A SUN Sparc 10 computer system with dual CPUs, 96 Megabytes of memory, and 7 Gigabytes of disk storage provides the Network File System (NFS) support for the engineering workstations. Development activity on the ACFS host computer system uses the NFS disk system. Real-time simulation loads can be run directly from the file-server. Tested "experiment-ready" real-time executables and data files are copied to ACFS host local disks to ensure isolation from the development environment during experiment operation.

The simulation software is maintained in multiple directory trees representing a logical decomposition of the flight simulator systems. Related software modules including their support files and data structures are defined as a Computer Software Configuration Item

(CSCI). The UNIX Source Code Control System (SCCS) is used to maintain revision level changes at the source code and data level. In the current system, a simulation development project is accomplished by doing the initial development in a minimally controlled base. As work progresses, the software is migrated to a moderately controlled base for system integration. Finally, the software is validated in an experiment base which is then frozen for the duration of the experiment.

PERFORMANCE AND PRODUCTIVITY IMPROVEMENT

The improvement in processor performance over the older system was dramatic, approximately one order of magnitude for each processor (with potentially 12 processors available). Memory availability is much more than required and hardware reliability has been 100%.

Improved productivity was gained in several ways. First, through the use of COTS hardware and software the facility does not need to maintain as large a staff of software personnel devoted to supporting an in-house real-time operating system. This frees up more of the software staff to develop research-specific software. Additionally, SGI provides a state of the art software development environment. The key elements of this environment can be utilized with the real-time OS, so the user is not required to develop and test his software in a non-real-time environment and then port it to the real-time system for re-testing. Also, the user does not need to learn and stay proficient at two or more different environments, and the system support staff does not need to support a unique (often in-house developed) real-time system. The improved toolset and the use of the same toolset for development and real-time has increased productivity.

Productivity is also significantly improved through the use of many mini-ACFS systems. With minor modifications to the simulation startup and schedule, the entire simulation can run on small single processor SGI machines in non-real-time. This provides a system that NASA researchers can use on their desks for initial concept evaluation, as well as for ACFS engineers to implement changes, and do initial testing. This reduces the need for expensive duplicate simulators for development, or using dedicated time early in the development phase on the main simulator complex. Since the cockpit flight displays are driven by SGI computers, the same processes can run on the development computer concurrently with the model, again not in real-time, but fast enough for development of most new research systems.

FUTURE PLANS

The system, as developed, is successfully performing research experiments more reliably and with less development effort than the previous system, however, there are many improvements planned. Some of the improvements are listed here.

The EOS functions and asynchronous I/O are computed on the same processor as the UNIX OS. As new processor boards are added, these functions will be moved to isolated processors to ensure appropriate response time.

The EOS will have several enhancements. The primary improvement will be a map display to provide the operator with better situational awareness. A real-time graphical EOS data plotter will also be added. Different COTS systems will be periodically evaluated and may be considered as replacements.

Current message logging has a simple priority and message structure. A flag based priority system would provide the operator a cue to important error, warning, and informational situations as they arise. This kind of message system is similar to the Engine Indication and Crew Alerting System (EICAS) messages used in the aircraft itself.

SGI provides a "frame scheduler" capability to provide synchronization control of processes distributed across multiple processors. The frame scheduler also provides access to a much higher resolution clock (21 nanosecond resolution). As new advanced subsystems are added to the ACFS simulation or higher frequency calculations are needed, the SGI real-time frame scheduler will be implemented.

An improved GCU is planned to allow more flexibility for inspection of variables in the main executive, MODSCH. The current GCU only allows predefined data files to be used. The new GCU will allow real-time editing of the variable list being used, as well as changing the display format, and browsing through each of the Global Common sections. A further enhancement should allow the changing of local variables as well as Global Common variables.

Planned improvements to the Software Configuration Management System include the ability to modify development configurations from the engineer's desk top with complete isolation from other simulation user activities by utilizing specialized UNIX scripts and MAKE files.

The integration and test computer is currently a single processor SGI workstation with two graphics displays. An SGI Onyx would provide the ability to closely model the multi-processor architecture that the host platform

has while also supporting more displays. These displays could be used for a simulated OTW display, more flight instruments, as well as operator/developer interaction. This should greatly improve integration and test efficiency.

In addition to the computer system enhancements listed above, there are three major hardware improvements planned. A new high quality OTW Visual System and a new I/O system will be installed in the ACFS during the next year. In addition, a commercial Heads-Up Display (HUD) system, similar to current systems gaining popularity with several of the airlines, is being considered for installation in the ACFS.

SUMMARY

A multiprocessor implementation of a very complex full-mission research flight simulation using essentially Commercial Off The Shelf (COTS) hardware and system software is described. The real-time system has provided a significant performance and reliability improvement. The development and real-time implementation has provided common environments between the multi-processor real-time computer, integration and test station, and desktop development systems. This common development, test, and operations environment saves resources in supporting many different platforms, physically moving files around, and user concern over data control. The use of common control and debug tools on all systems means there is no lost productivity by users being forced to learn new systems or move between different systems. There are several improvements planned to the simulator hardware and software to increase efficiency of development and operations even further.

References

Builder Xcessory 3.5 User's Guide
Integrated Computer Solutions Incorporated
Copyright 1995

Builder Xcessory 3.5 Reference Manual
Integrated Computer Solutions Incorporated
Copyright 1995

Advanced Concepts Flight Simulator
Programmer's Manual and ACFS Procedures
PM-020 Last Revision May 16, 1996

SGI IRIX On-Line Document Set
IRIX 5.3

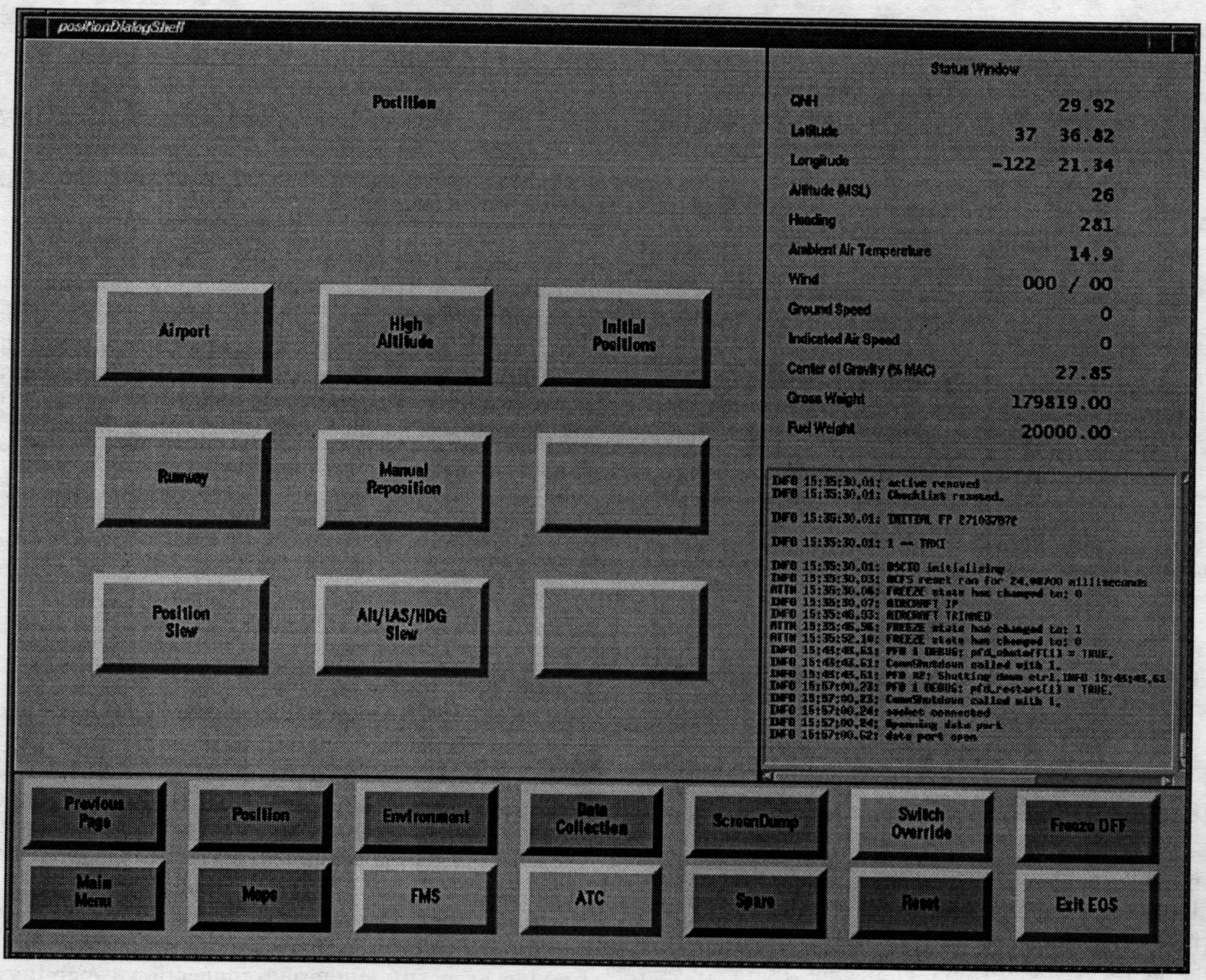

Figure 5: The Position Page screen of the EOS