LBL--27588

DE91 004287

The W.M. Keck Telescope Segmented Primary Mirror Active Control System Software

R.W. Cohen*, S. Andreae, A.K. Biocca,
R.C. Jared, J. Llacer, J.D. Meng,
R.H. Minor, and M. Orayani
Engineering Division
Lawrence Berkeley Laboratory
University of California
Berkeley, CA 94720

*California Association for Research in Astronomy
65-1120 Mamalahoa Highway
Kamuela, HI 96743

Primary funding was provided by the California Association for Research in Astronomy. This work was supported in part by the Director, Office of Energy Research, Office of Health and Environmental Research, Physical and Technological Division of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

LBL-27588

The W.M. Keck telescope segmented primary mirror active control system software

R.W. Cohen*, S. Andreae, A.K. Biocca, R.C. Jared, J. Llacer,
J.D. Meng, R.H. Minor and M. Orayani
Engineering Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, CA 94720
and
*California Association for Research in Astronomy
65-1120 Mamalahoa Highway
Kamuela, HI 96743

ABSTRACT

The active control system (ACS) uses both parallel and distributed processing techniques to measure and control the positions of the 36 segments of the Keck Observatory Telescope primary mirror. The main function of the software is to maintain the mirror figure; to accomplish this goal the software uses a predictive, "feed-forward" mechanism which effectively increases the system bandwidth for the most important sources of perturbation.

The software executes on a set of twelve 68000-family processors under the supervision of a VAX workstation. An array of nine parallel I/O processors collect and process data from 168 displacement sensors and transmit motion commands to 108 actuators. Three additional processors simultaneously compute actuator commands, monitor system performance, compute sensor control parameters and communicate with other observatory computers. The software is highly optimized for speed.

1. INTRODUCTION

The positions of each of the 36 hexagonal segments of the W.M. Keck Observatory Telescope primary mirror are constrained by the passive support system (PSS), the mirror cell mechanical structure, and the active control system, a computerized servo control system. The rigidity of the mirror cell adequately controls three of the six degrees of freedom of each segment: x and y motion in the plane of the segment and rotation about the normal to the segment. The remaining three degrees of freedom, tilt about two axes in the segment plane and piston along the normal to the segment, are controlled dynamically by the ACS.

The ACS is designed to limit mirror surface fluctuations to below 40 nm RMS over the 10 meter diameter primary in the face of perturbations induced by gravitational deflection, thermal instability of the PSS and external low-frequency stimulation. In addition, the ACS must control the overall orientation of the primary in the mirror cell, respond to mirror segment motion commands during alignment and calibration operations, monitor performance parameters of the mirror during astronomical observations and support the processes of building-up, testing and servicing the primary mirror.

The active mirror control system consists of three main components: 168 displacement sensors that measure relative segment positions, 108 actuators that move each segment in three dimensions and a servo control algorithm implemented as a software program which executes on a set of 12 processors. Control of the mirror is accomplished by measuring the relative positions of adjacent mirror segments using the 168 differential capacitance displacement sensors. A 171-element sensor-space error vector is constructed from the displacement sensor data and from three attitude variables which define the overall piston and tilts of the primary mirror in the mirror cell. The error vector is multiplied by a 171 x 108 control matrix that transforms the error vector from sensor-space to actuator-space. A feed-forward vector to compensate for the limited bandwidth of the control system is added to the product of the matrix multiplication; the result is a 108-element vector of changes in actuator lengths required to restore the optimal mirror figure.

This paper describes the software of the ACS. We emphasize the software that implements the servo control algorithm, the "ACS Control Loop." We also discuss the other programs which support the control loop and we describe the hardware and software environment in which these application programs run. Reference 1 provides an overview of the Active Control System. For a detailed analysis of the control loop from the point of view of control theory, see Ref. 2.

2. ENVIRONMENT

Figure 1 shows the hardware architecture of the ACS. The VME crate contains 12 processors, a bus controller, an Ethernet interface, a shared memory module, an ADC board for monitoring node box power supplies and a programmable master clock module which triggers downlink packet transmission. All processors share equal status on the VME bus; any processor can operate in bus master or slave mode.

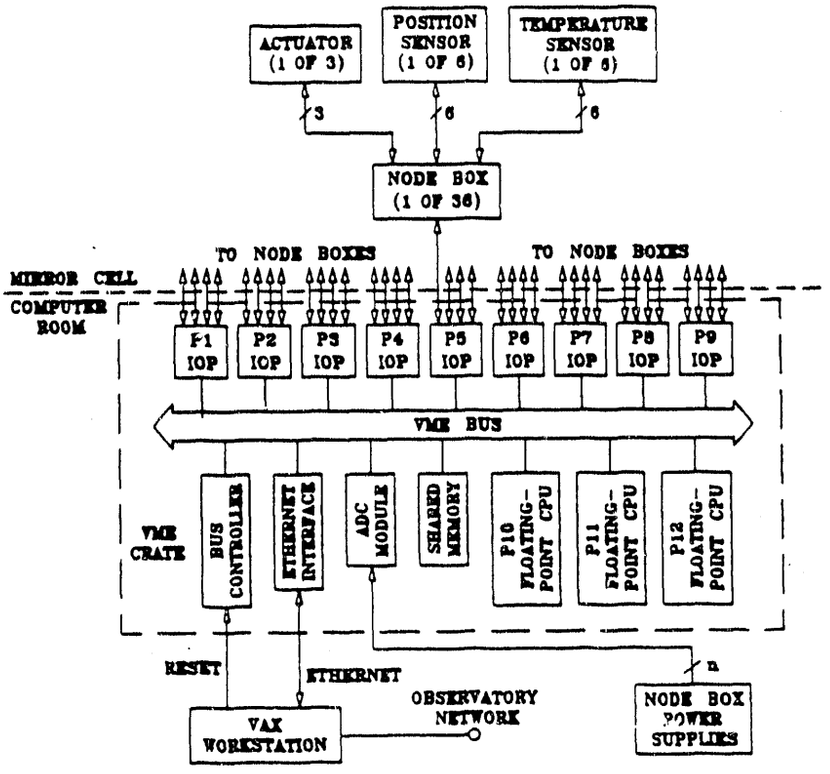

Fig. 1: Overview of ACS hardware

VxWorks, the real-time operating system used on the VME processors, provides a multi-tasking runtime environment with pre-emptive priority scheduling, inter-task synchronization and communication mechanisms and Transmission Control Protocol/Internet Protocol (TCP/IP) network support over Ethernet and on the VME backplane. VxWorks includes a symbolic debugger and a command shell which allows interactive execution of most C language expressions including invocation of any loaded routine and symbolic references to any loaded variables. On the VAX workstation we run the VMS operating system supplemented with TCP/IP support software. We use the Digital Equipment Corporation (DEC) Configuration Management System (CMS) for configuration control and the DEC Module Management System (MMS) to build and maintain all ACS code.

The ACS computer system communicates with each of the 36 node boxes on the mirror cell via 36 individual bidirectional serial communication links which transfers data at 250 k bits/s. Node boxes contain control electronics for the displacement sensors, temperature sensors and actuators, and also provide the interface to these devices. Bit serial data transmission between the computers and node boxes is used to reduce the number of high-power drivers and receivers required in the node boxes, thereby minimizing power dissipation on the back of the mirror.

Nine I/O Processors (IOP's, P1-P9) run identical code in parallel. Each IOP is connected to four node boxes via eight serial ports operated in half-duplex mode. Received data is moved into dual-port memory by direct memory access (DMA) controllers while the four ports used to transmit uplink data are operated in polling mode. Based on a 68010 microprocessor running at 10 MHz, the IOP's also perform front-end processing of downlink packet data including lowpass digital filtering. These boards are optimized for high-speed serial data transfer. Processor loading on the IOP's is ~ 80%.

Three floating-point processors (P10, P11 and P12) perform the control function calculations, communications, system monitoring and error logging and mirror emulation functions. These are based on 68020 microprocessors with 68881 floating-point coprocessors. Processor loading on the floating-point processor which performs the control matrix multiply is ~ 60%; the remaining two floating point processors are ~ 50% loaded. Peak VME bus loading is ~ 50%.

Programs running on all 13 processors communicate with each other using TCP/IP. The VAX is connected to the VME crate via Ethernet and the VME processors communicate over a backplane network with packet buffers in shared memory. In addition to TCP/IP, programs running on the VME processors can assert VME bus interrupts to alert other processors of time-critical events. A facility exists to manipulate semaphores using indivisible VME bus transactions. This allows controlled access by multiple processors to shared resources such as message buffers. Tasks and interrupt service routines running on the same processor can communicate and synchronize with each other using mailboxes and queues. VME processes also share data stored as interprocessor global variables in dual-port or shared memory.

Application software is developed and cross-compiled on the VAX workstation and downloaded to the VME processors as object modules. The loading and initializing of the application code on the VME processors is controlled by initialization programs running on the VAX which invoke subroutines on VME processors using a TCP socket-based remote procedure call (RPC) mechanism. The Internet File Transfer Protocol (FTP) is used to download data files and object modules from the workstation to VME memory.

3. CONCEPTUAL DESIGN OF THE MIRROR CONTROL SYSTEM

Figure 2 is a context diagram of the mirror control system which shows the interactions between the control system and its environment. Mirror control software is represented by the circle at the center of the diagram. Rectangles represent external entities with which the control system interacts. Solid lines represent data flow; the direction of the arrow indicates flow direction. Dashed lines represent event flows. An event is an occurrence which takes place external to the system at a specific point in time; events elicit responses from the system.

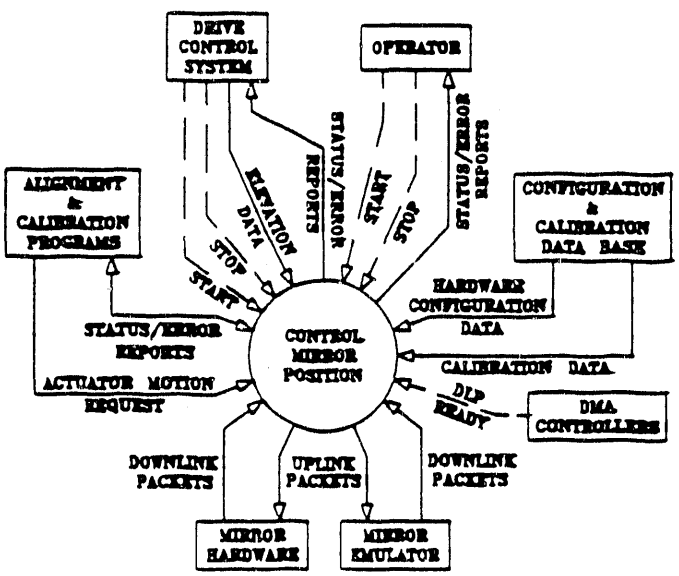

Fig. 2: Mirror control system context diagram

The main function of the mirror control system is to compute the actuator motion commands needed to maintain the primary mirror in its optimal figure from the position and temperature sensor data in the downlink packets (DLP's) transmitted by the mirror hardware and the elevation data from the telescope drive and control system (DCS). Actuator motion commands are sent to the mirror hardware in uplink packets (ULP's). As shown in the context diagram, a software mirror emulator can take the place of the real mirror hardware for the purpose of testing the mirror control software.

Commands to start and stop mirror control are issued by an operator or DCS. Telescope zenith angle and zenith angle rate-of-change data provided by DCS are used to compute sensor and actuator corrections and to predict the effects of gravitational deformation during slewing. Alignment and calibration programs can directly change actuator lengths and sensor range. Sensor and actuator calibration data and hardware configuration data are provided by database management programs.

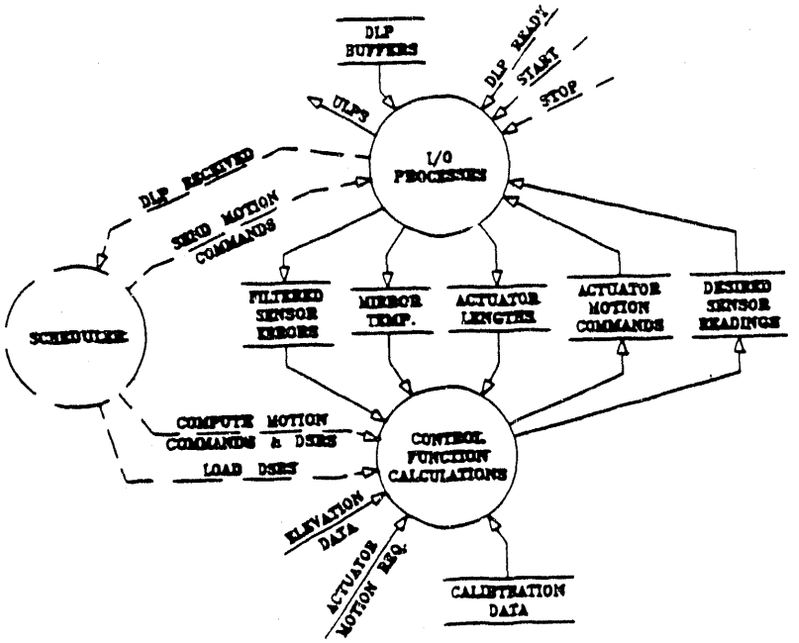

Fig. 3: Mirror Control System Top-Level Transformation Diagram

A top-level transformation diagram of the mirror control system is shown in Fig. 3. In this diagram horizontal parallel lines represent data stores, solid circles represent data transformations and the dashed circle represents a control transformation. A data transformation uses input data to derive output data and may be triggered by one or more event flows. A control transformation takes only event flows as input and produces only event flows as output. This diagram is a simplified explosion of the mirror control system shown in the context diagram.

From the broad perspective of Fig. 3, the mirror control system is divided into two data transformation processes which communicate with each other via data stores. The I/O process receives DLP's, extracts sensor and actuator data, computes and filters sensor error data and transmits actuator commands to the mirror hardware. The control function calculation (CFC) process takes sensor and actuator data provided by the I/O process and computes the next set of actuator commands which will ultimately be transmitted to the mirror hardware by the I/O process. The scheduler is a control transformation which determines the relative timing of all critical control loop events.

The I/O process is triggered by the DLP ready signal issued by the DMA controllers after a complete set of DLP's have been received and stored in memory buffers. DLP's arrive every 10 ms. (The rate of DLP construction and transmission is programmable but is nominally 100 Hz.) The DLP arrival rate is the fundamental clock frequency of the mirror control system; all control loop events are timed relative to the arrival of DLP's. When the DLP set arrives, the I/O process triggers the scheduler which causes a transition from one control loop state to the next. Each cycle of the control loop is divided into a number of discrete states by the scheduler; there are nominally 50 control loop states; thus the control loop cycle period is 500 ms.

At a particular control loop state the scheduler triggers computation of desired sensor readings and actuator motion commands by the CFC process. Desired sensor readings are the displacement sensor values which would be expected in the absence of segment position errors. The segment position error at each sensor is the difference between the desired sensor reading and actual sensor reading for that sensor. At another control loop state the scheduler triggers the CFC process to update the desired sensor readings actually used by the I/O process. At yet another control loop state the scheduler triggers the I/O process to transmit the actuator motion commands previously computed by the CFC process.

CFC process functions are performed only once during each 500 ms control loop cycle whereas I/O process functions repeat at every control loop state or 50 times per control loop cycle. After triggering the control loop state transition and concurrent with CFC processing, the I/O process extracts displacement sensor, temperature sensor and actuator length data from the DLP buffers. The I/O process computes segment position errors from the raw displacement sensor data and desired sensor readings provided by the CFC process; these position errors are then passed through a lowpass digital filter. Filtered sensor errors, temperature data and actuator length data are updated and made available to the CFC process at each control loop state although the CFC process samples this data only once per control loop cycle.

Control loop timing is illustrated in Fig. 4. Five plots are shown with a common abscissa which represents time in units of control loop states. Three complete control loop cycles are shown. The top plot illustrates displacement error at a particular sensor; the difference between minimum and maximum displacement error depends primarily on the rate of change of telescope elevation but is no greater than about 28 nm. The second plot shows CFC process activity which starts at control loop state 0 where filtered sensor error data is sampled and completes at about control loop state 35 after which actuator motion commands and new desired sensor readings are available but have not yet been acted on.

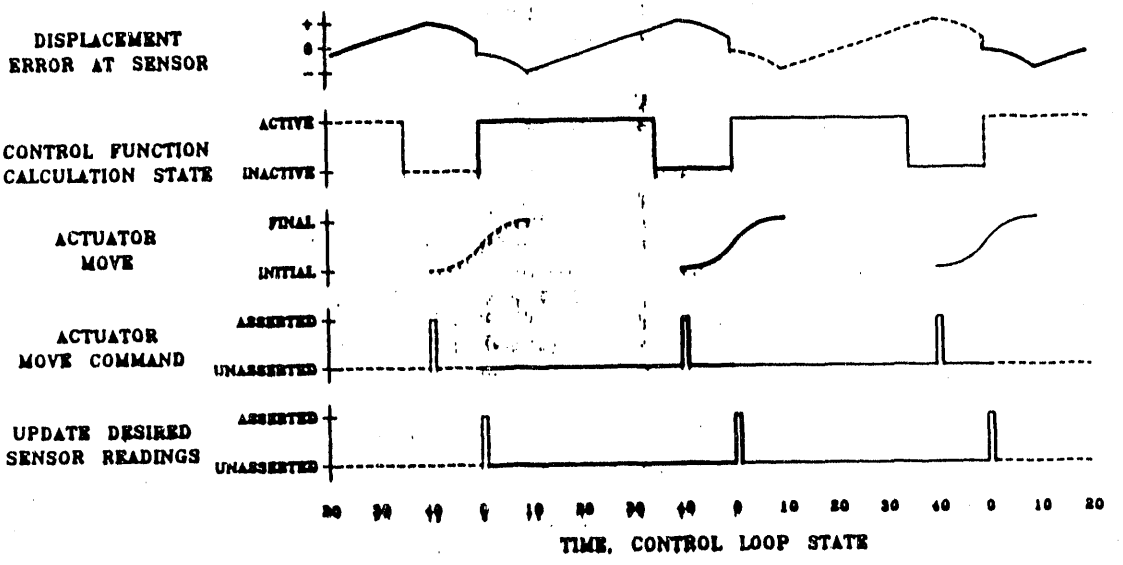

Fig. 4: Control loop timing

Actuator move profiles are shown in the third plot. Actuator motion commands generally take about 200 ms to complete, depending on the move size, and are profiled to minimize mechanical energy injection to the mirror cell. Actuator move commands are issued shortly after completion of CFC, as shown in the fourth plot. Desired sensor readings are updated about halfway through actuator moves. Since displacement errors are computed by taking the difference between desired sensor readings and actual sensor readings, there is a discontinuity in sensor error value at the time desired sensor readings are updated. The exact time at which desired sensor readings are updated is chosen to minimize the integrated error stored in the lowpass filters.

The CFC process samples filtered sensor errors. Without feed-forward corrections to the actuator motion commands there would be an uncorrected position error of magnitude proportional to the rate of change of telescope elevation and the rate of

change of mirror temperature and approximately inversely proportional to control loop gain. Computer models have shown that, without feed-forward, this lag would result in a settling time (time for system to reach sensor error less than system noise level) of about 30 s after completion of a fast move. This settling time could be decreased by increasing system gain, but higher gain results in longer transient settling times. Feed-forward based on the derivative of cell deformation due to gravity and temperature reduces maximum settling time to about 10 s and tracking lag to less than 3 nm, which is considered acceptable.

4. CONTROL LOOP IMPLEMENTATION

The ACS Control Loop is implemented as 15 tasks running on 12 processors; nine of these tasks are identical and run in parallel. Figure 5 shows the control loop tasks and the mirror emulator (Test Emulator Task) and the control and data flow between them. The notation used here is the same as that of Fig. 3. This diagram, however, represents the actual implementation rather than the conceptual design. The processor on which each task runs is also indicated on the figure. A description of each task follows.

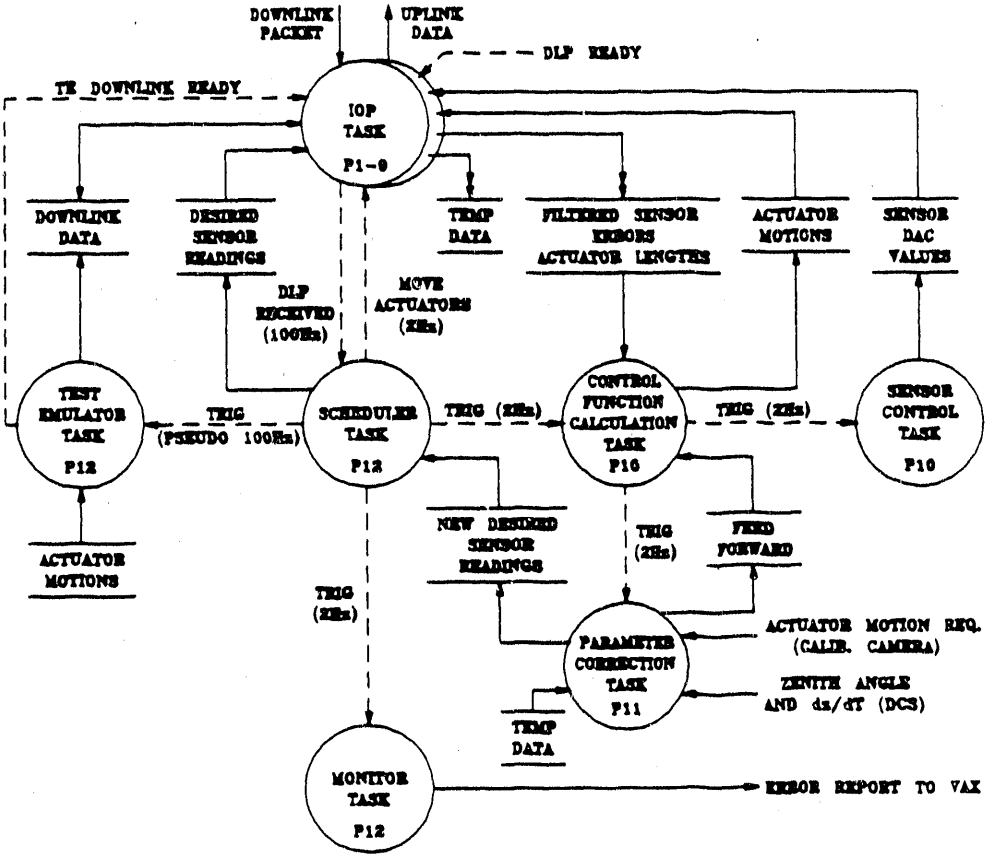

Fig. 5: Control Loop Tasks

4.1 I/O Processor Task

Each IOP runs exactly the same code. When a process must know which processor it's running on it can interrogate a memory location which contains a unique value determined by jumpers on the CPU board. Dual-port memory base addresses are also configured to complete the identification of the board.

The IOP task is posted by the DMA interrupt service routine (ISR) which services a local interrupt generated by one of the on-board DMA controllers when it has completed the transfer of an entire 40 byte DLP. The DMA ISR reprograms all four DMA controller channels in preparation for the next set of DLP's; only one channel is programmed to generate the

local interrupt which signals receipt of DLP's. The ISR then posts the IOP task and returns. Since DLP construction and transmission from all node boxes is triggered simultaneously and since all node box serial communication controllers are clocked by the same signal, DLP's from all node boxes arrive simultaneously; the interrupt from any one DMA controller channel signals the arrival of all four DLP's on the processor. Integrity of the serial links is verified by monitoring serial communication hardware status registers and by echoing some ULP data in DLP's.

One of the IOP tasks issues a VME bus interrupt which causes the scheduler task to increment the control loop state. Sensor, actuator and hardware status data are extracted from the DLP buffers and error counters are incremented as necessary. Displacement errors are computed by taking the difference between sensor displacement values received in the DLP's and desired sensor readings previously stored in IOP dual-port memory by the parameter corrections task. The sensor error values are then passed through a two-stage, lowpass, first-order, real-pole, recursive digital filter. Filter output along with actuator length and temperature sensor data are placed in dual-port memory where they are available to other processors on the VME bus. All numerical computations are performed on 32-bit integers and all data arrays are in hardware device space (see 6.2 - Map Generation).

Uplink packets are transmitted to all node boxes during each execution of the IOP Task (every 10 ms). Each displacement sensor has one offset digital-to-analog converter (DAC) and one drive DAC. Since each node box can control up to six sensors, four sets of ULP's transmitted over four control loop states are required to update all sensor DAC's. Sensor DAC commands are sent even when DAC values have not changed so that the integrity of each serial link between IOP and node box can be verified upon receipt of each set of downlink packets. Most uplink packets contain commands to set sensor DAC's but once every control loop cycle (500 ms) the scheduler task sets a flag which causes the IOP tasks to transmit actuator motion commands which have been stored in IOP dual-port memory by the control function calculation task. Each ULP can hold three commands so motion commands can be transmitted to all actuators in one set of ULP's.

4.2 Control Function Calculation Task

At one particular control loop state the scheduler task initiates computation of desired sensor readings and actuator commands by posting the control function calculation task. This task samples filtered sensor error data from the IOP's and moves this data as well as actuator length data to P10 local memory. After this transfer of data is complete the control function calculation task posts the parameter corrections task and both tasks proceed to operate on the same data set.

The actuator motion command vector, **P**, is a 108-element actuator space vector given by

$$\mathbf{P} = g\,\mathbf{B}\,\mathbf{S} + \mathbf{ff}$$

S is the 171-element sensor space position error vector: 168 elements correspond to displacement sensors and three elements correspond to virtual attitude sensors which are computed from a linear combination of actuator lengths. **B** is a 171 x 108 linear transformation matrix which transforms the position error vector from sensor space to actuator space, g is the global control loop gain (a scaler) and **ff** is the feed-forward correction vector computed by the parameter corrections task.

The actuator motion command vector is converted from units of nanometers to units of actuator steps using actuator gain data provided by the calibration database; elements which are greater in magnitude than the maximum distance an actuator can move within one control loop cycle are clipped. The elements of this vector are then mapped-out to the IOP's where they will be converted to actual actuator motion commands by the IOP tasks when the actuator move command is issued by the scheduler task.

4.3 Scheduler Task

At specific control loop states, the scheduler posts the control function calculation task, initiates transmission of actuator motion commands and loads a new set of desired sensor readings to the IOP's. The states at which these events occur are programmable. The scheduler task is posted by one of the IOP tasks after each set of downlink packets is received.

4.4 Parameter Corrections Task

This task computes desired sensor readings and the feed-forward vector for actuator commands. There are three components to the desired sensor readings vector: the zenith angle and temperature components account for zenith angle and temperature-dependent sensor behavior, and an external component accounts for direct segment motion commands issued by external programs. The external component is required to prevent the system from attempting to nullify segment motions requested by calibration or alignment programs; it is computed by multiplying the accumulated segment motion request vector in actuator space by the A-matrix, the geometric matrix that contains the relationships between actuator motions and sensor displacements. Temperature and elevation corrections are computed from look-up table data provided by the calibration database manager. The three components of the desired sensor readings vector are summed and stored in an interprocessor global geometric sensor space array. At a particular control loop state the scheduler task takes this data, maps it into hardware sensor space and writes it to IOP dual-port memory where it is used to compute displacement sensor errors.

Feed-forward, based on the temperature rate of change of the primary mirror and the rate of change of the telescope in zenith angle, is especially important during slewing to limit settling time. Feed-forward corrections are essentially a prediction of position errors which will accrue during the forthcoming control loop cycle based on the current rate of change of mirror temperature and the velocity of the telescope in zenith angle. Calibration data for each actuator is taken at many mirror temperatures and telescope elevations. This data is fitted to an equation of the form:

$$g(z,T) = k_1 + k_2 \sin(z) + k_3 \cos(z) + k_4 z + k_5 z^2 + k_6 T + k_7 T^2$$

which represents the length of a particular actuator as a function of temperature (T) and the zenith angle of the telescope (z). Coefficients for each actuator are downloaded from a database on the VAX to data structures in VME memory when the control loop is initialized. Actuator-motion-request feed-forward corrections are computed by evaluating these equations in the differential form, so the feed-forward component of the actuator motion request is given by:

$$ff = [\delta g(z,T)/\delta z]_T \, \Delta z + [\delta g(z,T)/\delta T]_z \, \Delta T$$

where Δz is the change in zenith angle and ΔT is the change in mirror temperature expected during the next control loop cycle.

4.5 Monitor Task

This task performs two main functions: system performance parameters are computed and error counters are monitored.

Error counters record error events which may occur during normal operation of the control loop but which do not require program or operator notification or intervention on a per event basis. An error event is declared when, for example, a task does not complete within an allotted time interval, filter calculations do not complete in time or serial communication controller errors are detected. These events are recorded in error counters which are data structures consisting of a counter, fatal and non-fatal count limits and increments. The error counter mechanism is used to limit the processor time needed to track the event in order to preserve system timing relationships. When an error counter exceeds its non-fatal warning level an error message is logged and the non-fatal warning limit for that counter is incremented. When a counter reaches its fatal level, which is infinity for some counters, then, in addition to logging an error message, the control system is placed in idle mode.

In addition to monitoring error counters, this task monitors system performance parameters including sensor residuals which measure the performance of the control function calculations, length of the error vector which indicates overall system performance and mirror temperature gradients. The operator can view these parameters at any time using the ACS interface (ASCI, see Sect. 6.4 - ACS Interface). When any parameter exceeds a predetermined limit an error report is automatically logged and, if the parameter has exceeded a fatal limit, the control loop is put in idle mode.

4.6 Sensor Control Task

The sensor control task performs two functions:

1) Sensor drive and offset DAC input values are computed based on the selected sensor range and sensor calibration data;
2) Sensor equation coefficients are computed and stored for later use by the parameter corrections task. The sensor equation, which relates sensor output reading to sensor displacement, is discussed in detail in Refs. 3 and 4.

This task is posted each control loop cycle, but only performs its functions if the sensor data change flag is set. This flag is set via the ACSI to indicate that the global sensor range value has been changed or that new sensor calibration data has been downloaded.

5. DATA CAPTURE

Facilities are provided by the control loop for capturing data from the real-time system for remote or delayed analysis. There are two classes of data which can be captured: fast data and slow data. Fast data changes at each control loop state. Raw displacement sensor data, filtered sensor errors and actuator length data are collected by fast data capture. Slow data is computed once every 500 ms control loop cycle. Slow data capture saves the product of the error vector and the control matrix, actuator motion commands which include feed-forward corrections and desired sensor readings.

Data capture is controlled via the ACSI. Fast data capture can be captured at the fast data rate or an integer multiple thereof. There are two modes in which data capture can be run. In "continuous" mode data is captured continuously and the data capture buffers are treated as arrays of circular buffers; an index, accessible to VMS programs via the ACSI, is maintained to mark the position of the most recent data. This mode is useful for examining the recent behavior of the system after an event of interest has been detected. In "future" data capture mode the data capture buffers are treated as arrays of lists. Upon receipt of a control signal data capture begins and continues until the specified number of data samples have been collected or until the buffers are filled. This mode is useful for examining the response of the system to an applied perturbation.

Data capture provides the means to examine details of the running control loop; it is the most direct mechanism we have to measure and analyze the response of the system to real perturbations induced by rapid changes in elevation, temperature or external commands or emulated perturbations induced by the mirror emulator. With data capture and the software mirror emulator we can analyze and study in detail the behavior of the control loop before attempting to control the real mirror.

6. ANCILLARY PROGRAMS

Figure 6 shows all the cooperating programs which comprise the ACS. The ancillary programs are discussed in following sections.

6.1 Initialization

Initialization of the real-time system is accomplished in five stages:

1) All devices on the VME bus are reset via the bus controller. A bootstrap loader program running out of ROM on each VME processor automatically loads an executable image of the VxWorks operating system and system symbol table over the network from mass storage on the VAX.
2) External global variables are downloaded to their home processors and then distributed to all processors.
3) Application code is downloaded. Since external global variable addresses are not known until variable distribution completes, object code is not linked until loadtime.
4) Data structures are initialized from data files on the VAX.
5) Application tasks are spawned.

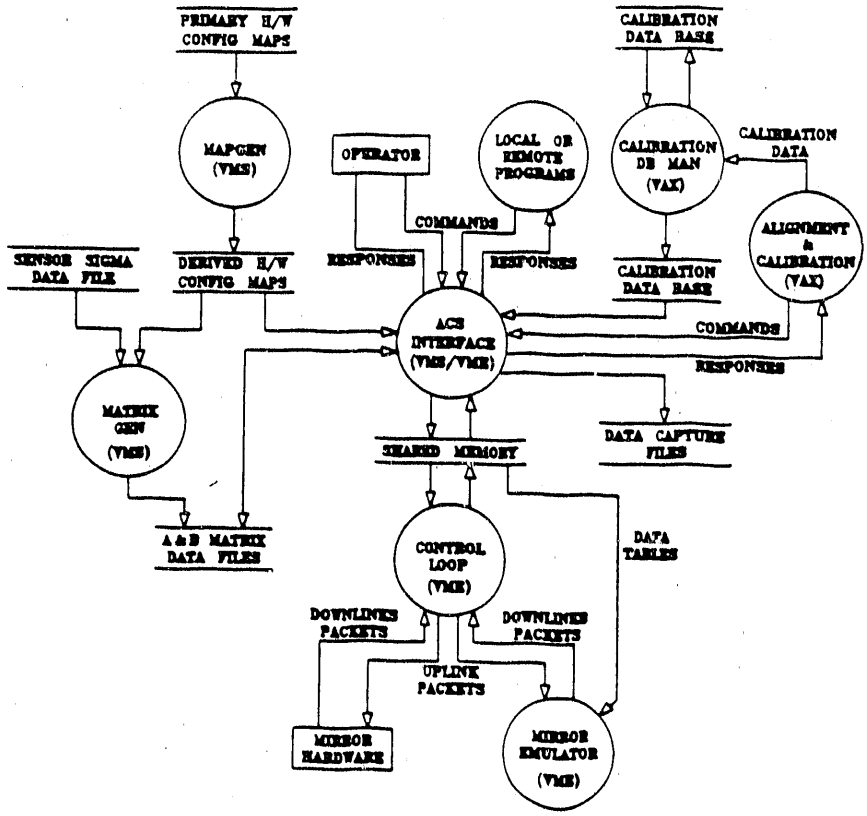

Fig. 6: ACS Software Overview

Upon successful completion of initialization, the control loop is running in idle mode and is prepared to receive commands via the ACS interface.

A set of low-level routines, automatically downloaded to the VME processors with the VxWorks operating system, are invoked by the initialization programs running on the VAX by the RPC mechanism. The remote procedures perform functions such as downloading data files and object modules from mass storage on the VAX to the remote processors using FTP, managing the VxWorks symbol tables and spawning tasks. At the program level, RPC's are virtually identical to local subroutine calls. We have designed our RPC package so that each remote procedure is invoked by a call to a single function to which arguments (including a target machine identifier) are passed in the normal manner; these are blocking functions which do not return until the remote procedure and the RPC network transaction have completed. Figure 7 is the structure chart for INIT_load_objs, the initialization program which downloads object modules specified in a data file called an Object Module List to the VME processors; it is only in the calls to the RPC's that the distributed nature of this program is revealed thereby vastly simplifying the program design.

A master configuration file, itself under configuration control, is used to specify the hardware configuration data files, calibration data files and application object modules, all of which are maintained under configuration control, from which the running system is to be constructed. Our configuration control system is used not only to control software development; it also provides a formal mechanism for organizing and controlling hardware configuration and calibration data files. Any new or previously defined combination of application code release, hardware configuration data set and calibration data set can be easily specified in the master configuration file and used to establish a running system.

6.2 Map Generation

The ACS map generation program addresses the problems of how to associate physical device locations with hardware connection paths and device addresses and how to inform the software of which devices are present and which are absent (physically or logically). In normal operation the ACS controls 108 actuators, 168 displacement sensors and as many as

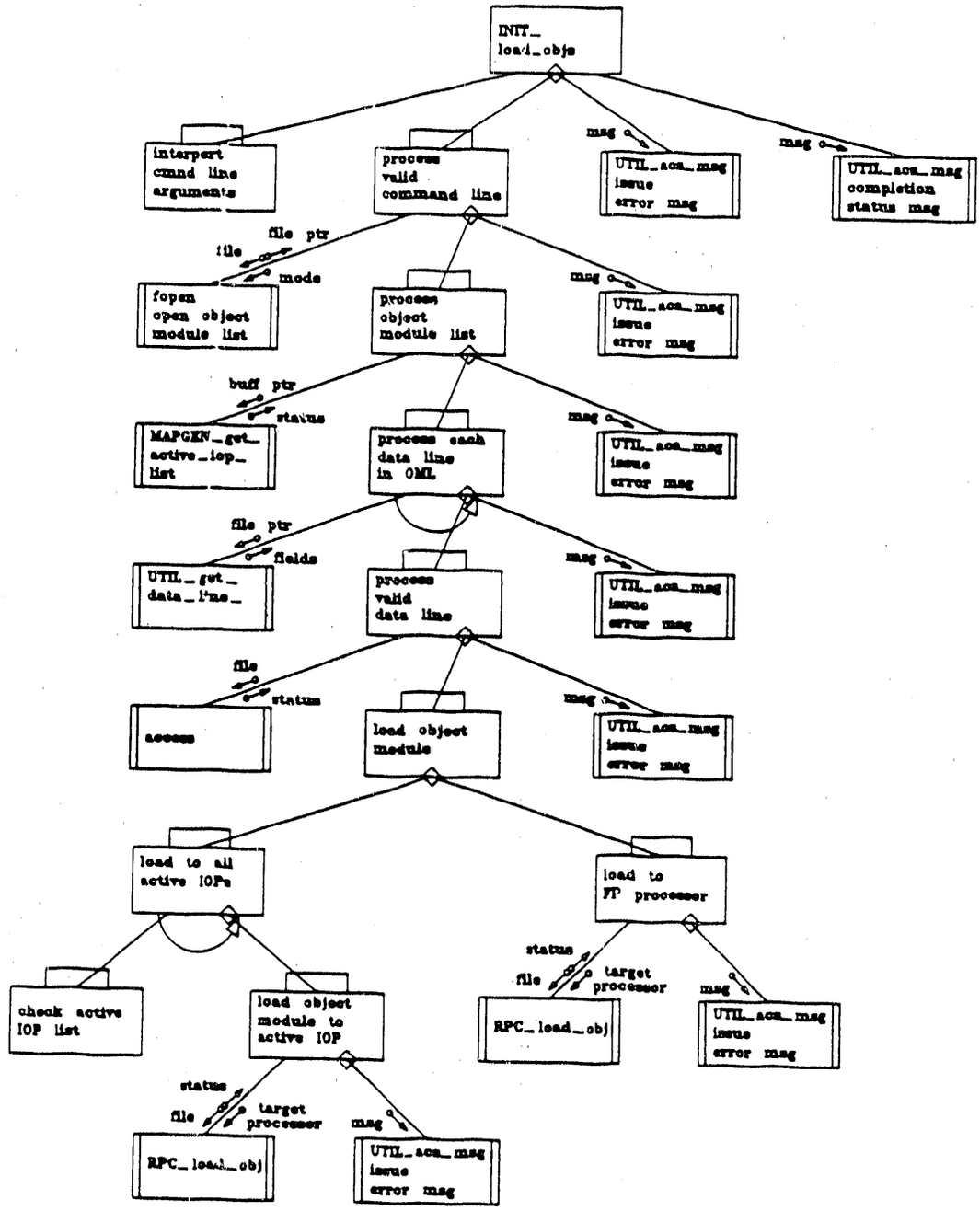

Fig. 7: INIT_load_objs structure chart

252 temperature sensors. Mounted on each of 36 mirror subcells is a node box which is connected to three actuators, as many as six temperature sensors and, depending on the segment type (inner ring, middle ring or outer ring), four, five or six displacement sensors. A given displacement sensor, for example, can be connected to any one of six ports on the node box and the node box can be connected to any one of four channels on any one of nine IOP's.

The mapping problem is compounded by the fact that the system may be required to operate (at diminished performance levels) with less than the full complement of segments during mirror build-up, for example, or a device may be declared non-operational under circumstances where it is undesirable to immediately replace that device and suffer the down-time required for installation and recalibration.

Central to the ACS mapping scheme are the definitions of geometric device number (geometric sensor number, geometric actuator number, geometric temperature sensor number) and hardware device number (hardware sensor number, hardware actuator number, hardware temperature sensor number). Geometric device numbers identify the location of a device in the global coordinate system of the primary. Mast[5] has defined the primary mirror global coordinate system and has identified sensor and actuator positions and numbers; we use these numbers for geometric device numbers.

Hardware device numbers are defined to contains all information necessary to address a device. Neither geometric device numbers nor hardware device numbers are associated with a specific device, instead; geometric sensor numbers, for example, identify one of the 168 displacement sensor positions on the primary and hardware sensor numbers identify one of the 216 possible connection paths between computer and sensor (of which 168 or fewer will be in use at any time).

The map generation program takes as input seven primary maps. Some of the primary maps, such as the actuator position map which contains the global coordinates of each actuator, are static. Other primary maps must be changed when the mirror is being built-up, a device is declared non-operational or when connection paths change. The actuator connection map, for example, which specifies the connection paths between actuators and node boxes and identifies those actuators which are to be considered inactive, must be updated when an actuator is connected to a different node box channel or is taken out of service.

The map generation program creates a set of ten derived maps from the primary maps. For example, the sensor map, which maps geometric sensor number to hardware sensor number, is derived from the active IOP list, the sensor connection map and the mirror map, all of which are primary maps. The derived maps provide the ACS programs with sufficient information to associate device locations with connection paths, to define the coordinates of the devices and to identify inactive devices.

6.3 Matrix Generation

This program is used to generate data files containing the **A** and **B** matrixes. An actuator space vector multiplied by the 171 x 108 **A**-matrix is transformed into sensor space. The **A**-matrix is derived entirely from the geometry of the primary mirror and the positions of the sensors and actuators. The displacement reading of a given sensor depends only on the orientations of the two segments which it bridges so each sensor depends on six actuator lengths. Thus, only six elements of each row of the **A**-matrix are non-zero.

The **B**-matrix, which transforms sensor space vectors to actuator space, is the psuedo inverse (neither matrix is square) of the **A**-matrix and is derived from the **A**-matrix using eigenvectors. Unlike the **A**-matrix, the **B**-matrix is not sparse.

When generating the matrix data files, account is taken of those segments which are not installed or are not operational and a facility is provided to define a standard error for each sensor. Quality factors of the control matrix are computed to provide the operator with an indication of the capability of a particular combination of segments, sensors and actuators to maintain good mirror figure.

6.4 ACS Interface

The ACS interface (ACSI) provides a program and user interface to the VME portion of the ACS. Read and write access to all interprocessor global data structures and variables is supported. Commands to start and stop the control loop, turn the test emulator on and off, control fast and slow data capture, move segments and load and retrieve data are issued via the ACSI. The ACSI is a distributed program; the VME portion of the program is implemented as a set of RPC's which are invoked by the portion of the program which runs on the VAX.

6.5 Calibration

There are seven complimentary calibration and alignment operations: star stacking, segment phasing, sensor calibration, actuator calibration, adjustment of secondary despace, adjustment of secondary tilt and adjustment of segment/mirror figure.

The calibration database manager (CalDB) provides facilities for archiving, retrieving and manipulating calibration and alignment data provided by several different types of calibration operations. CalDB is an interactive system with tools provided to select, fit and format data for use by the ACS.

For more information in this area see Ref. 6.

6.6 Maintenance

There are two maintenance programs: on-line maintenance and off-line maintenance. On-line maintenance is a hardware-fault-location program which runs on the ACS computer system independent of the control loop. It is intended to be run prior to an observing run to verify the operational status of ACS hardware or to isolate the source of errors detected by the control loop performance monitor task. Off-line maintenance is designed to be used in a test-bench environment, off-line of the main ACS computer and mirror hardware, to exercise, parameterize and certify node boxes, sensors and actuators. A single control program and menu-driven user interface supports both maintenance programs.

On-line maintenance divides the ACS into three subsystems: these are mirror hardware subsystem, VME subsystem and VAX subsystem. Each subsystem is divided into a number of field-replaceable units. A field-replaceable unit is a module, such as a processor board or a node box, which can be readily replaced with a spare. In some cases where sufficient data are available, on-line maintenance may isolate a failure to a more detailed level such as dual-port memory on a processor board. The off-line maintenance test algorithms are based on the assumption that, at any given time, there is no more than one failed unit in the system.

The VME subsystem consists of IOP's, floating point processors, Ethernet interface, shared memory, bus controller, master clock module and node box power supplies. The mirror hardware subsystem consists of node boxes, displacement sensors, temperature sensors and actuators. The VAX subsystem consists of Q-bus interface modules. The operator can choose to test the entire ACS, in which case all units of all subsystems will be tested, or she can test a single subsystem or a single unit or range of units within a subsystem.

Associated with each unit is a binary tree which represents the sequence of tests to be performed for that unit. A test tree consists of terminal nodes and non-terminal nodes. Non-terminal nodes represent tests; the result of a test, pass or fail, determines the next node to be visited in the traversal of the tree. Terminal nodes represent conclusions; a final test report is issued when a terminal node is reached.

Off-line maintenance includes tests to exercise and quantify the operational parameters of sensors and actuators. These tests are used for diagnosing problems with sensors and actuators and for certifying them prior to installation on the mirror cell. A summary of results is presented to the operator upon completion of a test. The operator can also choose to have summary and detailed test results saved in an archive file for later analysis or for maintaining a history of the device under test.

6.7 Test Emulator

The mirror emulator replaces primary mirror hardware with software for the purpose of testing the ACS software and for future concept development. The emulator intercepts ULP's containing actuator motion commands and sensor control commands and transmits DLP's containing position sensor, temperature sensor and actuator data back to the control loop. Actuator move profiles are simulated, actuator, sensor and wiffletree response characteristics can be programmed and various noise patterns and spectra can be introduced. The paper "Emulator for the W.M. Keck 10 m Telescope" describes the mirror emulator in detail.

7. CONCLUSIONS

Design, implementation and parameter and assembly testing of the ACS software are complete. Integration testing has been completed to the extent possible without the real telescope. Hardware and software have been exercised under conditions very similar to those expected during normal operation. We are now in the process of characterizing the dynamic behavior of the control loop using the mirror emulator. Prior to integration and testing with the telescope we will measure and understand control loop step and frequency response and phase lag. During the next year we expect to demonstrate that the ACS is indeed capable of controlling the figure of the Keck Observatory Telescope primary mirror.

8. ACKNOWLEDGMENTS

Primary funding was provided by the California Association for Research in Astronomy. This work was supported in part by the Director's Office of Energy Research, Office of Health and Environmental Research, U.S. Department of Energy under Contract No. DE-AC03-76SF00098. Reference to a company or product name does not imply approval or recommendation of the product by the University of California, the U.S. Department of Energy or the California Association for Research in Astronomy to the exclusion of others that may be suitable.

9. REFERENCES

1. R. Jared et al., "The W.M. Keck Telescope Segmented Primary Mirror Active Control System," in this volume.

2. J. Llacer et al., "Analysis of the W.M. Keck Telescope Segmented Mirror Control Loop," in this issue.

3. T.S. Mast, "Displacement Sensors: Control and Software," Keck Observatory Technical Note No. 240 (1988).

4. C. Witebsky, "Displacement Sensors: Hardware Parameters and Their Measurement," Keck Observatory Technical Note No. 247 (1988).

5. T.S. Mast, "The Geometry of the W.M. Keck Telescope Segmented Primary Mirror," Keck Observatory Technical Note No. 141 (1985).

6. C. Witebsky et al., "Alignment and Calibration of the W.M. Keck Observatory Telescope Segmented Primary Mirror," in this volume.

END

DATE FILMED

12/14/90