GALILEO Project

MONITOR WORKSTATION SYSTEM
Architecture Design Document
Tecnical report n. 1

A. Balestra, P. Marcucci, M. Pucillo, C. Vuerli

October 1990

Pubbl. Osservatorio Astronomico di Trieste n. 1341

Overview

This is a preliminary version of the Architectural Design Document (ADD) for the monitor software of the TNG Control System (TCS) to be built inside the Monitor Workstation (MWS).

The software described in the following refers to a first, simplified, system needed to qualify the primary mirror subsystem. It will be taken as a prototype for the further development of the complete system.

There shall be two environments on the MWS, one off-line dedicated to the housekeeping of Characteristics files, and one on-line wich shall communicate with the user and the TCS subsections, monitoring all the telescope activities.

Files and Tables

The structure of the MWS file system shall be as follows :

/tng/data/config	contains system configuration files
/tng/data/def	contains editable characteristics files PCF, CCF, DCF, ... in the format required by off-line editor
/tng/data/compiled	contains the characteristics files in compiled format, as required by programs in the VMEs and in the MWS. File names will be in the format xxxnnn.cmp, where xxx is the editable file name and yyy is the number of the node to which the file will be downloaded.
/tng/data/offline	contains all the files needed by off-line utilities, e.g. masks, definitions, ...
/tng/exec	contains executable files
/tng/sources	contains source files
/tng/lib	contains libraries
/tng/lib/sources	contains libraries sources
/tng/help	contains on-line help files

Files structure

Configuration Files

A description of the structure and internal organization of configuration files follows.

```
/* === DCF (Display characteristic file) ==== */
struct dcftype
     {
     int       dcfnum;            /* unique identifier of the window */
     int       nodenum;           /* (see /tng/data/config/ncf) */
     int       wind;              /* window # (node relative) */
     int       xoffs,yoffs;       /* window offsets */
     int       xdim,ydim;         /* window dimensions */
     int       color;             /* window color */
     char      title[81];         /* window title */
     };
typedef struct dcftype dcfrec;

/* === NCF (Nodes characteristic file) ==== */
struct ncftype
     {
     int       nodenum;           /* unique identifier of the node */
     char      nodename[81];      /* node description */
     char      arpa_node[16];     /* ex: 125.000.000.002 */
     int       byte_sex;          /* YES=swap bytes, NO=as is */
     int       send_data;         /* YES=send Tm+Data, NO=send only Tm */
     };
typedef struct ncftype ncfrec;
```

Characteristic Files

A description of the structure and internal organization of characteristic files follows.

```
/* ==== PCF (Parameter characteristic file) ==== */
struct pcftype
     {
     int       key;               /* unique identifier of VME */
     char      operator_id[31];   /* name of the operator */
     char      creation_date[7];  /* aammgg */
     char      update_date[7];    /* aammgg */
     char      title[66];         /* title of the file */
     char      comments[2][66];   /* two lines of comments */
     char      ccf[21];           /* related CCF file */
     };
typedef struct pcftype pcfrec;

/* ==== DCM (Descriptor characteristic file) ==== */
struct dcmtype
     {
     int       key;               /* DCM identifier (master descr.) */
     int       validity;          /* VALID / UNVALID record */
     char      acronym[21];       /* descriptor acronym */
     int       upcom_vme;         /* VME id (see /tng/data/config/ncf)*/
     int       size;              /* data size */
     int       ring_buffer;       /* ring buffer to be used */
     int       convert;           /* YES/NO convert to physical units */
     int       intr_low_limit;    /* lower input limit in eng. units */
     int       intr_high_limit;   /* higher input limit in eng. units */
     int       check_limits;      /* YES/NO check input limits */
     double    low_alarm_thr;     /* low threshold for ALARM */
     double    high_alarm_thr;    /* high threshold for ALARM */
     double    low_attn_thr;      /* low threshold for ATTENTION */
     double    high_attn_thr;     /* high threshold for ATTENTION */
```

```
    int       wind_id;          /* output window id (see /tng/data/config/dcf)*/
    int       x_wind_pos; /* internal x position inside window */
    int       y_wind_pos; /* internal y position inside window */
    char      phy_unit[11];     /* acronym for phys. units to display */
    char      dispform[2];      /* display format (s,f,n...see xmask) */
    double    coeff[MAXCOEFF]   /* IPM polynomial coefficents */
    };
typedef struct dcmtype dcmrec;

/* ==== CCF (Command characteristic file) ==== */
struct ccftype
    {
    int       key;              /* unique identifier of VME */
    char      operator_id[31];  /* name of the operator */
    char      creation_date[7];/* aammgg */
    char      update_date[7];   /* aammgg */
    char      title[66];        /* title of the file */
    char      comments[2][66];  /* two lines of comments */
    char      pcf[21];          /* related PCF file */
    };
typedef struct ccftype ccfrec;

/* ==== MCM (Microcommands) ==== */
struct optype
    {
    int       convert;          /* YES/NO convert in eng. units */
    int       min_value;        /* minimum value allowed */
    int       max_value;        /* maximum value allowed */
    double    coeff[MAXCOEFF];  /* IPM matrix */
    };
typedef struct optype oprec;

struct mcmtype
    {
    int       key;              /* unique command identifier */
    char      acronym[21];      /* command acronym */
    int       queue;            /* DELAYED/NODELAYED */
    int       counter;          /* max number of operands */
    int       min_exec_time;    /* min. time extimated for execution */
    int       max_exec_time;    /* max. time extimated for execution */
    int       tm;               /* Tm par. to verify after maxexectime*/
    int       tolerance;        /* error/1000 allowed in Tm parameter */
    int       node;             /* node (see /tng/data/config/ncf) */
    oprec     oper[MAXOPER];    /* operands (see optype) */
    };
typedef struct mcmtype mcmrec;
```

Help Files

A description of the structure and internal organization of help files follows. (TBW)

Processes

Six main tasks can be foreseen for the MWS, as can be seen from fig. 1. This structure, which takes into account the characteristics of the UNIX operating system, devotes a single process to each task, providing also a fast dynamic interprocess communication mechanisms.

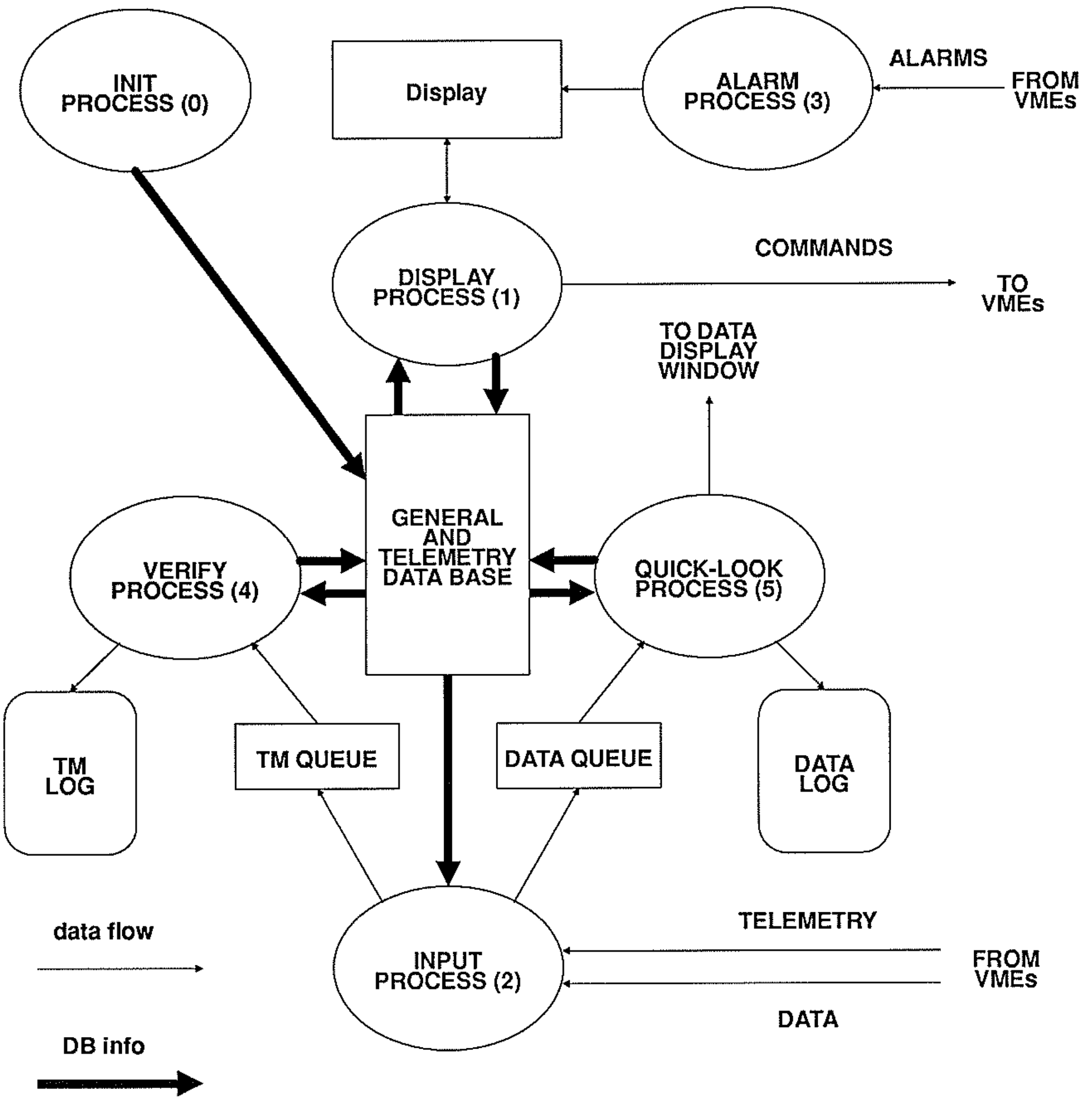


Fig. 1 : MWS task structure

Processes description

INIT (0)

The INIT process starts at the very beginning of MWS operations. Its task is to load the internal Data Base with the contents of definitions files, and to create the process structure as depicted in fig. 1. After all processes are created and inited, INIT puts itself in a sleep state, from which it will be waked up either by the unexspected death of a child process (due to a non recoverable error), or by a signal from DISPLAY, passing on a request for an ordinate shutdown of the system, issued by the operator.

Process creation procedure is implemented using fork and exec system calls. The former duplicates the process while the latter starts a new image in the same procees. The original process is then called "parent process", the other is the "child process". If necessary, parameters can be passed from the parent to the child at the moment of the exec call. The parent process knows the process id of all the children and this feature will be used for the ordinate shutdown of the system. The children can also recover the parent process id through a simple sistem call; this may be useful for sending alarm signals to the sleeping parent on special situations.

System shutdown - TBD

DISPLAY (1)

The task of supervisor of the interactive activities of TCS is taken on by the DISPLAY process. It is actually divided into three sections, each carrying on a well defined and independent activity:

i) monitoring of user input

ii) validating and sending commands to TCS components

iii) displaying of telemetry data

The first two tasks are strictly dependent on the user actions at the MWS console. They execute in an asynchronous way, with the first one accepting and parsing user input, while the second validates and transforms it into a meaningful set of TCS commands.

The third task is a synchronous one and is activated by a signal from a periodic clock (e.g. every second). It takes its input from the general Data Base, and writes the selected data to the active windows on the MWS screen. The selection of the data to be written and of the windows where they will go is made following the contents of PCF and DCF tables.

User Interface - TBD

Command Validation and sending - TBD

The display of incoming telemetry data is guided by two tables: DCM and DCF. The first, using the MSD of the TM data as input key, get the window identifier that is used as input key to the DCF table. Here are described in detail the identifier of the output window (nodenum+wind) and the positions of the TM data inside the output window. The windows used for TM data display can be moved by the user, and the type of TM data displayed can be modified via the DCM tables.

INPUT (2)

Telemetry data incoming from the TCS components are collected by the INPUT process. The mechanism adopted to communicate with the TCS is based on the socket structure, as defined in the BSD interprocess communication system.

The sequence of operations performed by INPUT is as follows:

1) a virtual circuit is opened, to allow TCS components to request a connection;

2) after a request of connection, a permanent circuit is established with the requester over which data exchange can take place (precedence is given to telemetry connection);

3) all incoming data are collected in a circular buffer, placed in a shared memory segment, where VERIFY process can read them.

Step 2 is repeated for each TCS node, so that a set of communications entry points is built up, each with an associated circular buffer. The whole input operation is structured in such a way that only two data moves will take place: one from the socket to the circular buffer, and the other from the buffer to the Data Base, thus minimizing the total overhead.

No data analysis is made by INPUT process. All checks and computation intensive tasks are left to the VERIFY process.

The same set of operations is performed on the scientific data, but for the reading process which is the QUICK-LOOK process.

The circular buffer is a structure formed by the following fields:

flag : Specifies whether new items were read on the buffer

buoy : Index of the first not written byte

wrindex : most recent byte written

rdindex : most recent not read byte

block : array of the read/written bytes

Before reading n bytes from the buffer, a reader process does the following tests:

if buoy = 0, read only if wrindex > rdindex

if buoy > 0, read only if buoy > rdindex; if buoy = rdindex then set buoy to 0.

Before writing n bytes into the buffer, a writer process does the following tests:

if buoy = 0, write only if wrindex + n <= dimension of buffer; if not, set buoy to wrindex

if buoy > 0, write only if wrindex + n < rdindex.

ALARM (3)

Alarm conditions detected by TCS components are sent directly to the ALARM process, using the same operations as in the case of TM data. The permanent connections with TCS components are set up during the init phase, after which the process forces itself in a wait state. The arrival of an alarm message wakes up the process and alarm processing starts with a direct interaction with the MWS screen.

User interactions with alarms - TBD

Alarms display - TBD

VERIFY (4)

The VERIFY process is devoted to TM verification and storage. It receives its input from the INPUT process through a set of circular buffers (one for each transmitting node). The TM data are organized as described in the Tables section, and are processed following the information contained in the PCF, CCF and DCF tables.

A general Data Base is maintained in the shared memory segment, where all the processed TM data are written, thus representing the actual status of the TCS system. All other processes can gather relevant data from there (see fig. 1), again following the information contained in the definitions tables.

The Data Base is contained in a single shared memory segment; that segment is organized as a main structure whose elements are arrays of structures. The Data Base segment is created by the INIT process and all its children processes will receive the identifier of the new segment. Starting from the segment identifier, the children will extract a pointer to the segment.

If we name 'db' the pointer to the DB segment, the accesses to the data base will have the following form:

```
db -> file_name [k] .field
```

Example: the instruction

```
val  = db -> dcm [235].pvalue
```

extracts the phisical value of the parameter which tag is 235.

QUICK-LOOK (5)

Scientific data are received through a communication mechanism identical to that of TM data, are put directly in a file contained in a RAM disk. The task of the QUICK-LOOK process is to organize the data in a format suitable for storage, and to allow a degree of interaction between the user and the data themselves. A dedicated window on the MWS screen will be maintained by the process in order to perform those tasks.

User interface - TBD

Communication Mechanisms

The communication mechanisms are based on the tools that UNIX flavours sysV and Berkeley place at user's disposal. Berkeley interprocess communication system is supported by the concept of "socket"; a socket defines an entry point in a process where a communication channel can be opened. SysV system is restricted to processes running on a single machine and uses the system calls inherents the "messages"; such a mechanism allows a message to be sent from one process to another in a rather easy way.

Internode Communications

The choice of a mechanism for internode communications is forced, by the impossibility to use sysV tools, towards socket system. At the moment the used sockets are of SOCK_STREAM type: this guarantees a sequenced, reliable, two-way connection based byte stream. Two communication domains are provided: UNIX and Internet. The latter has been choosen for its higher degree of standardization on different types of machines. The supported protocol for these sockets, in the Internet domain, is TCP. Possible use of SOCK_DGRAM type sockets with UDP protocol is foreseen to be tested in the near future.

The general structure of communications is implemented with a client-server operating mode. Once a server receives a connection request, it can react in two ways: forking and executing a new process dedicated to successive data exchanging or mantaining the control over transmissions, polling the set of opened sockets using a select system call. The latter solution avoids scheduling problems and allows more efficient communication management and is now working.

Interprocess Communications

The interprocess communications are designed in order to work in an asynchronous mode. This is accomplished using signals and sysV messages together. A process needing to send a message, writes it in the suitable queue and then signals, through a SIGIO signal, the event to the receiver processes. After the signal has been delivered to the process, the signal handler provides the reading of the just written queue. During the execution of the handler, successive arrivals of SIGIO are blocked until its termination: at this moment only, other readings will be possible.

General Data Base

TBW

Display Organization

The TCS presents TM data and scientific data through a multi-window display system. This is a costraint due to the various type of information to which the operator wants to access. In fact, there will be a window to manage the command input, one or more windows to present TM data, a window to display scientific data, a window to display alarms and alerts, and some other windows related to peculiar processes (such as the display of the telescope actual position in a small pictorial window). Antoher field of developement will be the implementation of a multi-screen workstation in order to display, for example, TM and alarms on a 800x600x8bits screen and scientific data on a 1280x1024x24bits screen.

Screen Layout

The layout of the TCS screen is not strictly defined, as the operator can drag and resize most of the windows across the screen. Keeping this in mind, becomes clear the fact that the windows have different priorities. An alarm window, for example, cannot be closed or resized by the operator; the only operations allowed on it will be the moving of the window in another position of the screen and the "iconization" of the window, that will keep, even as an icon, the color related to the last active alarm.

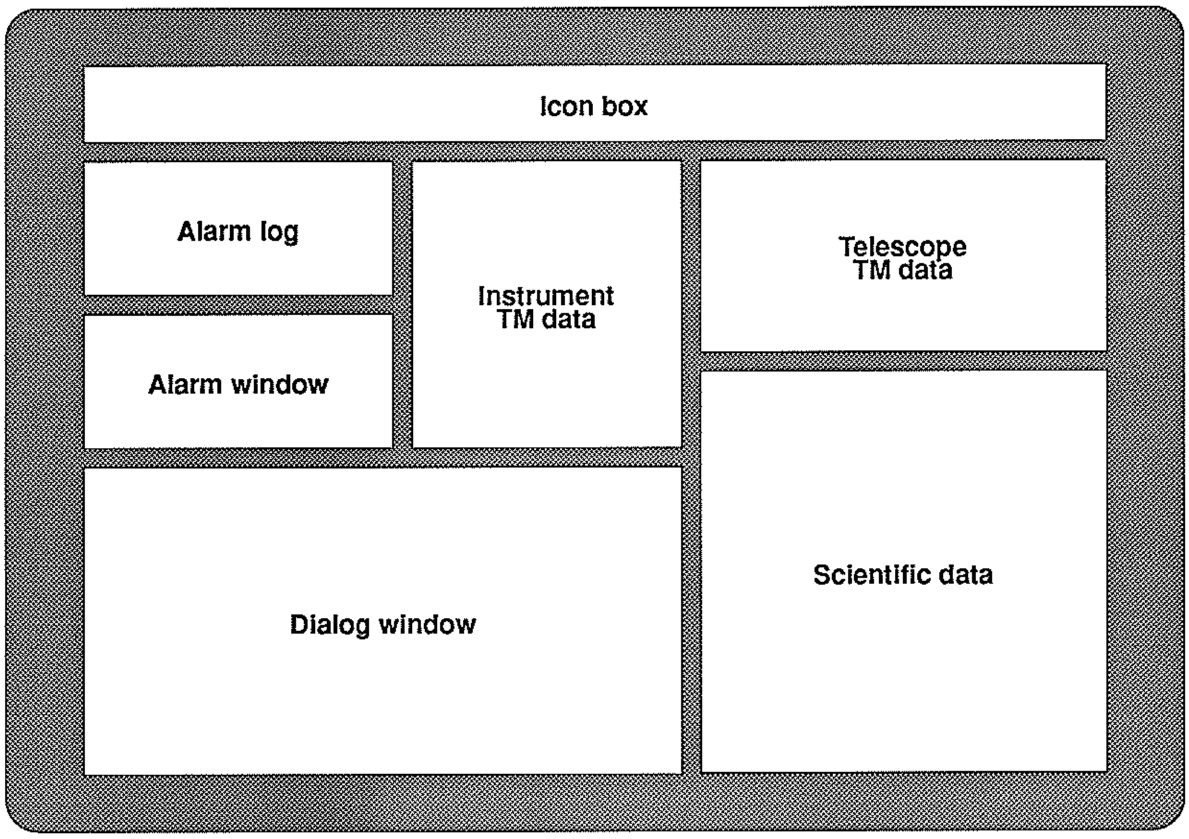


Fig. 2 : Example of an MWS screen layout

Windows Definition

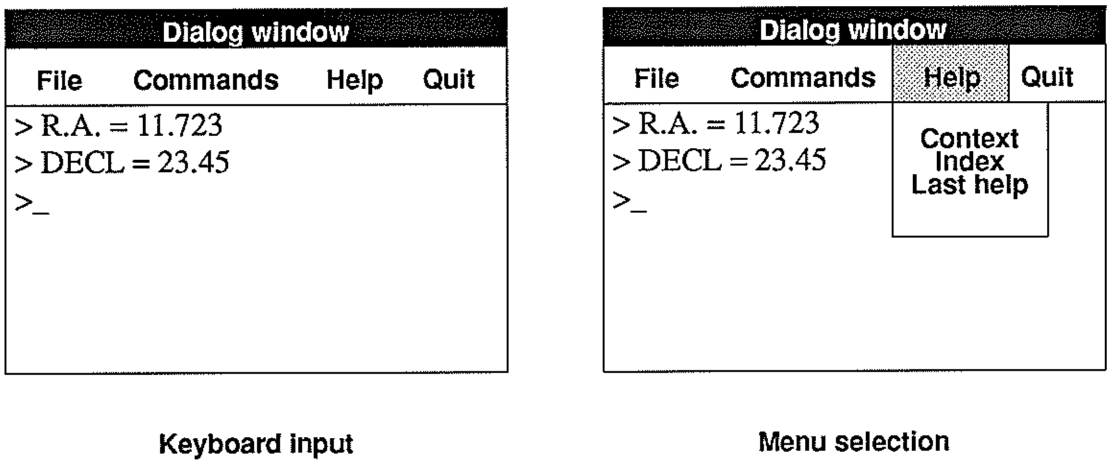
As seen in the "files structure" section, all the data that have to be displayed in a window provide an handle to a window structure, described in detail by the DCF data structure. In this data structure can be defined the node address of the display used to contain the window (the same for all windows in a mono-screen organization), the positions and dimensions of the window itself, its color and the related title. By using an handle to this structure and the position informations found in DCM, the DISPLAY process knows exactly where data have to be displayed.

User Interaction

The interaction of the user with the telescope and instruments can be accomplished in, at least, two ways:

- interaction via the dialog window (see fig. 3);
- interaction via "control panels" (see fig. 4).

The first kind provides input either as a command entered at the keyboard or as a menu selection, while the second uses a "control panel" metaphora in order to give the user the sensation of a direct control of the telescope or of an instrument.



```
> R.A. = 11.723
> DECL = 23.45
>_
```

Fig. 3 : Keyboard commands vs. menu selection

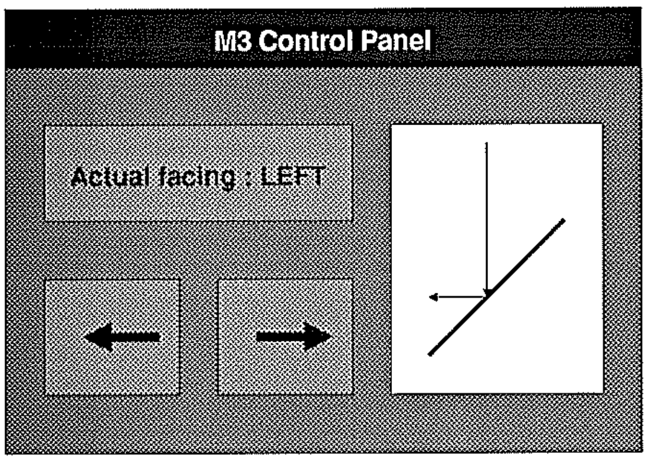


Fig. 4 : Control panel

The TM data display is directly related to the interaction flavor choosed. If the operator uses the "commands+menu" interaction, the TM data are displayed in a separate window (as seen in fig. 2) and the contents of this window can be user defined. In the other case, "control panel" interaction, the TM data are displayed in the same window where the operator uses slides, pushes buttons, etc.. An example of that is shown in fig. 4: here we see the buttons that control M3 position and the actual position (i.e. a TM data) of the mirror, either in a graphical manner (on the right) or in a textual way (on the left).

Error Management

TBD

Help Management

TBD

Information Logging

TBD

Data Base log

TBD

Telemetry log

TBD

Data log

TBD