111168 **N95-17583**

J52-17
31988
p-9

RE-ENGINEERING THE MULTIMISSION COMMAND SYSTEM AT THE JET PROPULSION LABORATORY

SCOTT ALEXANDER
JEFF BIESIADECKI
NAGIN COX
SUSAN MURPHY
TIM REEVE

Operation Engineering Lab
Jet Propulsion Laboratory
California Institute of Technology
MS 301-345
Pasadena, California 91109-8099
{salex, jeffb, nagin, sooz, timr@devvax.jpl.nasa.gov}

ABSTRACT

The Operations Engineering Lab (OEL) at JPL has developed the multimission command system as part of JPL's Advanced Multimission Operations System. The command system provides an advanced multimission environment for secure, concurrent commanding of multiple spacecraft. The command functions include real-time command generation, command translation and radiation, status reporting, some remote control of Deep Space Network antenna functions, and command file management. The mission-independent architecture has allowed easy adaptation to new flight projects and the system currently supports all JPL planetary missions (Voyager, Galileo, Magellan, Ulysses, Mars Pathfinder, and CASSINI).

This paper will discuss the design and implementation of the command software, especially trade-offs and lessons learned from practical operational use.

The lessons learned have resulted in a re-engineering of the command system, especially in its user interface and new automation capabilities. The redesign has allowed streamlining of command operations with significant improvements in productivity and ease of use. In addition, the new system has provided a command capability that works equally well for real-time operations and within a spacecraft testbed. This paper will also discuss new development work including a multimission command database toolkit, a universal command translator for sequencing and real-time commands, and incorporation of telecommand capabilities for new missions.

INTRODUCTION

The Jet Propulsion Laboratory has a long history of building multimission ground data systems that are designed to be easily adaptable to new projects. The mainframe-based systems of the 1970s have been replaced by distributed, workstation-based systems as part of JPL's advanced Multimission Ground Data System (MGDS). The new MGDS provides flexible, extensible components that are easily adapted for new missions, but more importantly, can also support multiple missions concurrently. However, as these ground systems have evolved, it has become apparent that providing advanced tools that help simplify and automate the old way of doing business is not enough to support the small, low-cost missions of the future. In particular, the uplink process has been very labor intensive for planetary missions and it must be re-engineered to provide the simple command capabilities that will be needed for missions with cheaper, more autonomous spacecraft and for operators wanting remote telescience capabilities.

The Operations Engineering Lab (OEL) has developed and refined the MGDS Command Subsystem to be an adaptable, low-cost, multimission component of the overall uplink process. As part of our development work, the OEL is working with the sequencing teams and developers at JPL to re-engineer the uplink process so it can provide seamless, easy-to-use capabilities for spacecraft commanding. The goal is to provide an off-the-shelf command package that can support large to small missions that need to command through the Deep Space Network (DSN).

MGDS COMMAND SYSTEM DESCRIPTION

The MGDS Command functions include real-time command generation, command translation and radiation, status reporting, remote control of DSN antenna functions, and command file management. A distributed, network-based, graphical interface is provided to give real-time command radiation status to users at remote sites. This interface was implemented in X/Motif. The Command System provides security functions including authentication for two user privilege levels, internal security checks, a central node for controlling all command radiation processing, a configuration control environment for command files, and a mode for non-interactive Command viewing.

The primary control function of the Command system is to permit real-time transmission of command files and memory loads from the ground to a spacecraft. The Command Control Graphical User Interface (GUI) (Figure 1) provides real-time, interactive control of the command transmission and radiation to the spacecraft. The connection between Command and the DSN is a secure process controlled by the Data System Operations Team at JPL. These operators allocate the connection resources to a project mission control team after ensuring a clean commanding interface.

Command files are first transmitted to the DSN and held at the receiving end until the completeness and integrity of the file transfer can be verified. Once there, the user is free to put the files in the queue of the Command Processor Assembly for radiation to the spacecraft either at that moment or some later specified command window. The user also uses the Command GUI to remotely control the configuration of the antenna in terms of when actual radiation of commands.

Any time the user is connected to a DSN station, the station returns monitor data which is displayed in the Command GUI for inspection by the operator. Monitor data contains information about the current antenna configuration, acknowledgments of command file radiation, and constant status information including alarms, files at the CPA, and receipt of command blocks.

Command files are generated prior to transmission using the Command system or the Sequence software system. Both processes are similar. A spacecraft command sequence is formulated and constraint-checked and then the actual commands are entered as command mnemonics, encoded abbreviations (with parameters) that tell the spacecraft what commands to perform. The command mnemonics are translated into a spacecraft-ready file that contains binary translations of the mnemonics, spacecraft identification information, start and acquisition codes, and file integrity and error-detection information. Once the command files are prepared, they are stored and made available to the Command system through a secure database that checks command formats and user permissions. Before transmission, the command files are reformatted for recognition and radiation by the DSN (Figure 2).

LESSONS LEARNED

When the MGDS Command System was completed, existing projects were required to transition from the mainframe MCCC Command System. Voyager was chosen as the first project to transition since it had entered its interstellar cruise phase. Their experience provided multiple lessons learned about simplifying the user interface and reducing the number of steps in the uplink tasks.

When the Mars Observer (MO) Project came on line as a new project, they had no prior system for comparison. Their experience was different since they had a much higher command rate than the Voyager mission. They had also decided not to implement the real-time command translation capability in the MGDS Command System as a cost-cutting measure. This meant that all of their command files, even those with only a single non-interactive command, had to be prepared off-line using the more complex Sequence software. As a result, the project was having difficulty keeping up with its command rate, even in the early cruise phase. When the spacecraft went into emergency mode, commanding became a 24-hour activity with many engineers required in the process.

There were two lessons learned from the MO use of the command system. First, eliminating the real-time translator during the mission planning phase resulted in increased costs in the mission operations phase. Second, the number of steps needed to prepare commands had to be reduced. In particular, the use of the security-controlled Command GUI had to be re-evaluated. The GUI was required to perform even simple file reformatting functions, with no options for a command-line interface or batch-mode. This reliance on a graphical interface prevented automating some steps with simple scripts because a user had to be sitting at the computer, pushing each button in turn. It became apparent that we had to provide an off-line command generation capability that was based on non-graphical, less restricted, command-line interfaces. The secure GUI was still essential for transmission and radiation of commands. With this re-design, DSN resources are only required for the final transmission and radiation of the Command files to the spacecraft. The impact of this off-line capability on required network resources is significant.

Thus, the Command system interface was redesigned to allow users to generate command files in an off-line environment without requiring a connection to the command control GUI. First, the translation and reformat functions were developed into separate, stand-alone programs. The translation program translates text mnemonic commands into an intermediate SpaceCraft Message Format (SCMF) file containing binary commands expected by the spacecraft. The reformat program packages the binary commands into the form expected by the DSN. These programs can be started up by the user on the UNIX command line or script, as well as by the central command system. The off-line capabilities have also allowed script automation to reduce the number of manual, interactive steps involved in the generation of command files. A graphical interface shell was built using the JPL-developed PERL scripting language and OELSHELL interface building tool.

This off-line translation toolkit also found extensive use in spacecraft flight testbed facilities where no connection to the DSN was allowed. Testbeds provide an environment for testing and validating commands on a mock spacecraft. The testbed command system sends commands directly to the ground support equipment.

Another lesson learned was the need to streamline and simplify the end-to-end uplink process. The uplink process involves multiple operations and development teams. This creates a system with multiple tools and interfaces, forcing the user to learn how to operate across several different boundaries. From a project perspective, there should only be a single interface to the uplink process that would allow a single user to perform all functions including spacecraft sequence generation and translation, ground sequence of events schedule generation, real-time command preparation, mnemonic translation, and command transmission and verification. The OEL has worked closely with the Mars Surveyor Project to implement an integrated, graphical interface tool that allows a single user to seamlessly perform end-to-end functions in the uplink process.

The successful experience of the early projects using the MGDS Command system eased the transition of the remaining projects. All of the JPL planetary missions have now transitioned successfully to the MGDS Command System and the mainframe-based

Command system was decommissioned a year earlier than originally planned.

RE-ENGINEERING COMMAND TRANSLATION

Since both the Sequence and Command software provided capabilities for a user to generate command files, there were common translation capabilities duplicated in both systems. The OEL has worked closely with Sequence developers to re-engineer the translation process and develop a universal command translator that can be used by both subsystems. The redesigned system includes the use of advanced graphics and object-oriented techniques.

The translation functions in the Sequence system were based on manually building mnemonic-to-bit translation information in each project's unique command macro language. These project-specific adaptations were time-consuming and error-prone. The command translation process in the Command software was based on a multimission Command Definition Language (CDL) that can be used to specify command mnemonic-to-bit definitions and constraints. The CDL file is compiled into a project's Command Database. A command database is built for each project, but the language compiler, database interpreter, and translator software is multimission. In the re-designed uplink process, the command database interpreter and translator software was re-built as generic, universal libraries that could be called by both Command and Sequence software. This multimission, common approach will significantly reduce uplink costs.

An illustrative example of CDL code follows:

```
! define a memory load message
MESSAGE: memload-msg(buf: 200)
   FIELDS
      data: 160    ! 160 bit local variable
   END FIELDS

   ! declare the kinds of arguments that will
   ! be entered by the user
   LOOKUP ARGUMENT: name
      ! lookup value below in hex
      CONVERSION: HEX
      LENGTH: 8
      'MEMLOAD' = 'A9'
   END LOOKUP ARGUMENT
   NUMERIC ARGUMENT: address
      ! user to enter number in hex
      CONVERSION: HEX
      LENGTH: 16
      ! acceptable range
      '00FF' TO 'FFFF'
   END NUMERIC ARGUMENT
   NUMERIC ARGUMENT: aword
      ! user to enter number in hex
      CONVERSION: HEX
      LENGTH: 16
   END NUMERIC ARGUMENT

   ! read mnemonics from user input
   READ ARGUMENT name
   READ ARGUMENT address
   REPEAT 1 TO 10 TIMES
              (COUNTING WITH nwords)
      READ ARGUMENT aword
      data := data // aword
   END REPEAT

   ! combine converted input into a message
   ! counters like "nwords" are 16 bits
   buf := name // address // nwords // data
END MESSAGE
```

It defines a memory load message that can load up to ten words, sixteen bits each, into a certain area of memory. If the user's mnemonic input was, for example,

MEMLOAD; 0A48; 1; 22; 333

the resulting hex output would be:

A9 0A 48 00 03 00 01 00 22 03 33

where the first byte is an op code that signifies a memory load instruction, the next two bytes are the address to load the data into, the next two bytes are the number of words in the data, and the remaining six bytes are the data itself.

Since CDL files can become very complex, a command generation toolkit is being developed to facilitate their creation and browsing. The CDL toolkit will include a graphical CDL editor, a CDL parser and compiler, and various report generators. In the future, some text based on-line reference tools and a smart editor to help a user create mnemonics are planned.

The first step taken in the development of the toolkit was to determine the data structures for holding the information contained in a CDL file. These structures are accessed through a library that is used by all tools in the toolkit. Here, an object-oriented approach was used. For example, CDL has several types of processing routines. So, one of the classes was for that of a general processing routine. A subclass of the general routine is a message routine. Arguments are also objects, with lookup arguments and numeric arguments derived from a common, more general, argument class. For the CDL code above, there is one instance of a message routine, **memload-msg**. There are instances of both kinds of argument objects: **name**, **address**, and **aword**.

When an object is created, a parent object is specified. Whenever an object is destroyed, all of its children are automatically destroyed as well. For the example above, **name**, **address**, and **aword** are children of **memload-msg**. So if the user of the graphical editor chooses to delete the **memload-msg** routine, the code for the editor is simply one call to destroy the appropriate parent object and all of the child objects (which are not useful by themselves) are automatically cleaned up.

CDL objects can refer to each other. For an easy example, the **READ ARGUMENT address** statement is itself an object (in this case, of class input processing statement and child of **memload-msg**). It contains a reference to the object corresponding to the argument to be read. Thus, if the CDL editor user changed the name of the **address** argument, when the CDL code was saved the **READ ARGUMENT address** statement would automatically be written with the new name. Note that for this example, the editor will not allow the **address** argument to be destroyed until the reference to it in the **READ ARGUMENT address** statement is changed or the statement removed altogether. It is easy to get a list of references to any object. There are many constructs in CDL not shown in the example that lead to a single object being referred to in several places.

The CDL language was designed years ago as part of the old mainframe-based command system. It is missing some important functionality such as arithmetic and comparison operators. CDL was also written before the Telecommand standard, so some of its constructs are outdated and intended for tasks such as embedding error polynomials into the binary commands. In the new Command system, any functionality not present in the CDL language must be added as hard-coded 'user hooks' to the command translation software, creating additional expense for development and testing. Thus, as part of our re-engineering efforts, we are incorporating important enhancements and simplifications to the CDL language. For some of these enhancements, we are investigating the use of other process control languages such as Spacecraft Control Language (SCL) in the uplink process. With the object-oriented approach taken and the goal of reducing class-specific code, we expect it to be easier to make changes to the language.

We are also investigating extending CDL to include information that would typically be found in a command dictionary such as telemetry verification points and flight rule constraints. The graphical CDL toolkit is also being enhanced to provide a complete command definition and dictionary toolkit with hypertext references to other mission documentation.

Other recent development work includes porting our code to multiple UNIX hardware platforms, ANSI-C, and XPG-4 open standards. In addition, we are incorporating the 1987 Consultative Committee for Space Data Systems (CCSDS) "Telecommand" standards into the MGDS Command System. All future JPL missions will comply with this standard.

TELECOMMAND IMPLEMENTATION

The Telecommand service model is a layered model which more or less parallels the ISO Open Systems Interconnect model. The highest two layers of this model, the Application Process layer and the System Management layer, have not yet been specified in detail. It is still up to the individual project to define procedures and data structures in these layers. The layers below this, however, *have* been specified in detail. Our response to the standard addresses the Packetization, Segmentation, Transfer, Coding, and Physical Layers

In the Command subsystem, the Telecommand (TC) standard is being implemented as a generic, batch-mode "wrapping service." Clients of the service supply the data to be wrapped in ASCII formatted files called Command Packet (CMD_PKT) files. The service takes multiple, one or more, CMD_PKT files as input, wrapping the data from each file record and time-order merging the results into an SCMF file.

CMD PKT file format

The format of the CMD_PKT file follows the CCSDS Standard Formatted Data Unit (SFDU) standard. The I-data (user) section of the file is organized as a header section followed by a series of data sections. The section boundaries are defined with special markers, and the information within these sections is organized in a "keyword = value" format. An example of a header section follows:

```
$$MPF   COMMAND PACKET FILE
*CMDPKT       SEQTRAN.CMDPKT/JOB001
*OPERATOR     Frank Zappa
*PROGRAM      SEQTRAN - MARS
         PATHFINDER  V19.0  APR 29, 1994
*CREATION     JPL 94-131/09:58:59
*BEGIN         *****  NO DATA  *****
*CUTOFF        *****  NO DATA  *****
*TITLE         *****  NO DATA  *****
*ZERO          *****  NO DATA  *****
*CMDFIL        *****  NO DATA  *****
*FILSIZ        6
*SISVER        04/27/94
*FRMVER        1
*CDUACQLEN  22
*CDUACQ        55
*CLTUSSQLEN 2
*CLTUSSQ       EB 90
*CLTUTSQLEN 8
*CLTUTSQ       55
*CLTUDLY       ENDSTART/BITS/0
*FRMPERCLTU 1 $$EOS
```

The header section contains global file information. For example, the value of the 'FILESIZ' keyword tells you the number of data sections which follow. 'CDUACQLEN' and 'CDUACQ' together form a specification of the acquisition sequence to be used for this file. 'CDUACQLEN' is the number of octets in the acquisition sequence and 'CDUACQ' is the smallest repeat pattern. Using the above record, the Telecommand wrapping service would generate 22 octets of 55 hex.

Each data section contains ASCII hexadecimal data to be wrapped, along with enough information to fill in the Telecommand headers. Here is an example:

```
$PKT          SCGNLD
PKTVER        1
SEQFLGS       FIRST
CHECKSUM      947D
VC            1
LENGTH        12
APPID         0
OPENWIN       82-080/11:40:00.000
CLOSEWIN      82-080/12:00:00.000
FRMSEQ        0
FEC           EACSUM55AA
CTRLCMD       NO
BYPASS        YES
PACKETIZE     N
FRAMING       YES
SEGMENTING    NO
DATA
        0A01 0000 0200 0001 0002 0003
        0004 0005 0006 0007 0008 0009
$EOP
```

Following the DATA keyword is a sequence of ASCII hexadecimal words. This represents the binary data to be wrapped. The format and structure of this data is known to the higher layers of the CCSDS Telecommand service model (system management and application layers). The values of other keywords enable the wrapping service to fill in the TC headers. For example, the value of the VC keyword tells the wrapping service what to put into the 6-bit 'virtual channel ID' field of the TC transfer frame header.

The creator of this file also has control over which layers of wrapping are applied to the data. The wrapping service concerns itself with the following layers:

- TC packetization layer (TC packets)
- TC segmentation layer (TC segments)
- TC transfer layer (TC transfer frames)
- TC coding layer (Command Link Transmission Units (CLTUs), consisting of TC codeblocks)

For example, consider the keywords PACKETIZE, SEGMENTING, and FRAMING. PACKETIZE and SEGMENTING are both set to NO, while FRAMING is set to YES. This means that the TC wrapping service will consider the data to be the contents of a TC frame, and will only prepend a TC frame header (and may also append a Frame Error Control word, if the FEC keyword is set to a value other than NONE), before creating a CLTU. If PACKETIZE were set to YES, the wrapping service would consider the data to be the contents of a TC packet, and would apply a TC packet header. Then, if SEGMENTING and FRAMING were both set to YES, the TC packet would be broken into TC segments, and then each TC segment would be wrapped as a TC frame, before creating a CLTU. Currently, all eight permutations of (PACKETIZE, SEGMENTING, FRAMING) are allowed by the wrapping service, though only three may be legal: (NO, NO, YES), (YES, NO, YES), and (YES, YES, YES). This flexibility makes the name 'CMD_PKT' something of a misnomer; perhaps 'CMD_TC' would have been a better choice.

Currently, each data section of this file will result in one or more CLTUs. Normally, only one CLTU will be created per data section; the only thing which can affect this is the setting of the FRMSPERCLTU keyword in the CMD_PKT header section. If this is set to a value N, where N > 0, then no CLTU may contain more than N TC frames. So, if the amount of data in the data section is large enough that when it is segmented, more than N TC frames are created, more than one CLTU will result.

Each data record contains a timestamp as well. This may be specified as either a window (OPENWIN, CLOSEWIN) or an execution time (EXECTIME). Times are expressed in GMT relative to the spacecraft (SpaceCraft Event Time, or SCET). For a given data record, this means that the data in that record will be at the spacecraft, ready to be processed, at the given (EXECTIME), or within the given window (OPENWIN, CLOSEWIN).

<u>TC Wrapping Service</u>

This service is implemented as a single process which consumes one or more CMD_PKT files and produces a single SCMF (SpaceCraft Message Format) file. Each data record of the SCMF file contains a single 'spacecraft message', which in this case is a CLTU.

Each CLTU within a record may be preceded by an acquisition sequence, depending upon the PLOP (Physical Layer Operation Procedure) in use by the project. Currently two PLOPs are defined in the TC standard. In PLOP 1, CLTUs are individually radiated, meaning that the physical telecommand channel is deactivated after each transmitted CLTU. In this case every CLTU in the SCMF file must have an acquisition sequence prepended. In PLOP 2, the physical channel is not deactivated until the last CLTU in an 'upload' has been transmitted. For our purposes, this means that only the first CLTU of the SCMF file will be preceded by the acquisition sequence.

The TC wrapping service places the resultant CLTUs in ascending time order within the SCMF. Further, the timestamp in each record of the SCMF is the time of radiation of the first bit in the record. This means that in going from execution time in CMD_PKT file(s) to an SCMF, all times have to be backed off by the number of bits in the record (multiplied by the time of one bit at the current uplink rate), plus any inherent spacecraft delay time, plus the appropriate one-way light time. All of this is a fairly complex operation, since we are merging multiple CMD_PKT files, each of which can have a mixture of window and execution time records.

TC Wrapping Service Design

A modular approach was taken in the design of the wrapping service. It is decomposed into five primary modules, as follows:

1. CMD_PKT file I/O module.
2. SCMF file I/O module.
3. Light time module.
4. Telecommand module.
5. Main module.

The first four modules are implemented as libraries. The main module calls functions in these libraries. The CMD_PKT file module depends upon the Telecommand module as well, mainly for validation of TC header field values.

The CMD_PKT file I/O module isolates all of the knowledge of the format and structure of CMD_PKT files. Its set of exported functions allow record-oriented I/O (both reading *and* writing) of CMD_PKT files.

The SCMF file I/O module is directly analogous to the above, for SCMF files.

The Light time module contains functions which perform conversion between ground transmission times (TRM) and spacecraft event times (SCET). This module reads a LIGHTTIME files in order to perform its function.

The Telecommand module isolates all of the knowledge of the TC data structures. It contains a set of functions for validating all of the TC header fields values, as well as a set of functions for performing TC wrapping. This module also maintains a table of project-dependent Telecommanding data. Items such as default acquisition, start, and tail sequences, virtual circuit and application id mnemonics, TC codeblock size, and PLOP are included in this table. The main module is responsible for the overall control of the wrapping process, and deals directly with the time-ordering issue.

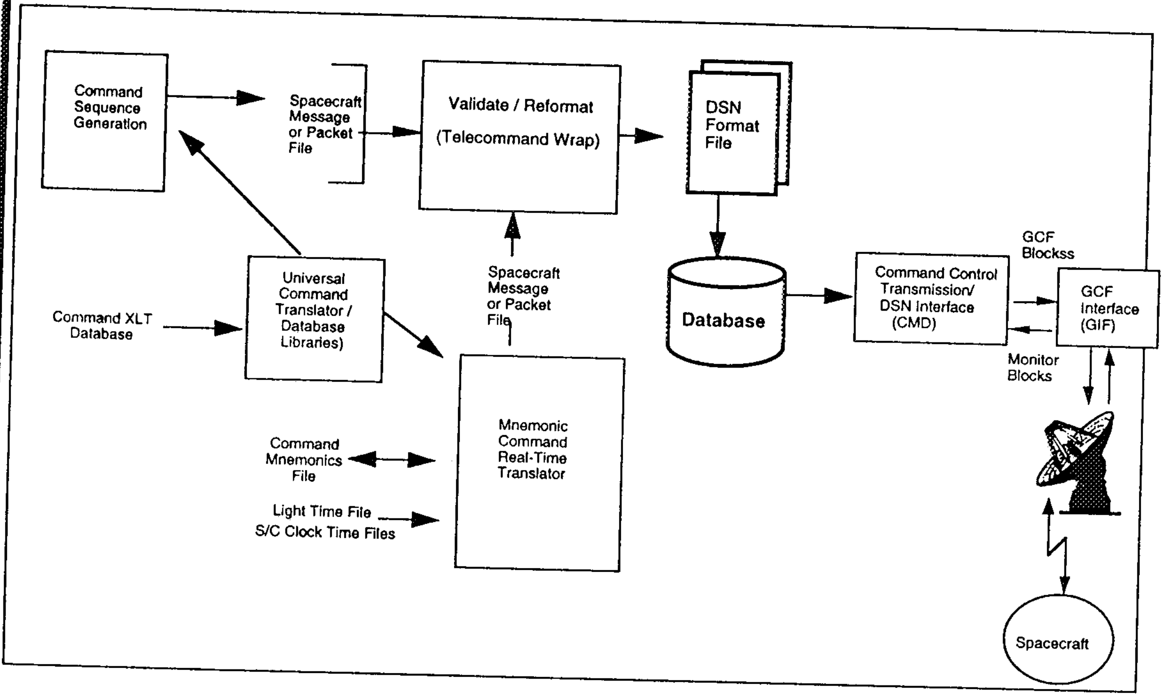
CONCLUSION

The Operations Engineering Lab has developed the JPL multimission command system to provide low-cost, adaptable, extensible uplink capabilities to new and existing flight projects. The goal in the ongoing re-engineering of the command subsystem is to create a set of independent tools to allow more flexibility for the user and to make any necessary customization faster and easier for future, low-cost missions.

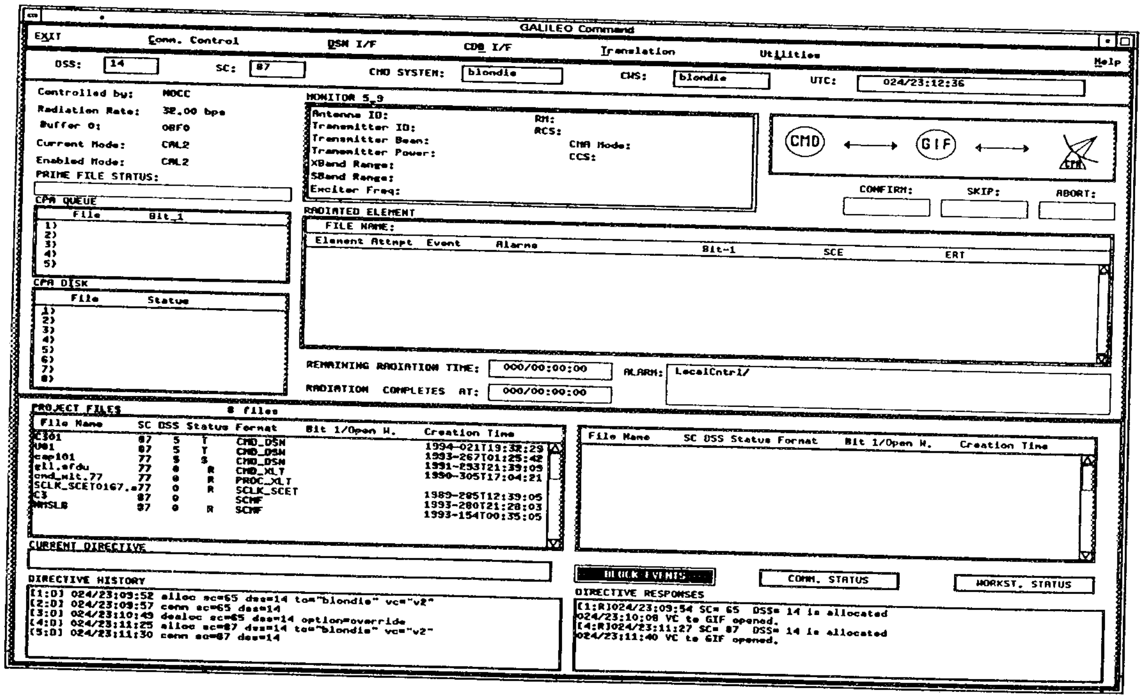
ACKNOWLEDGMENTS

This work was done at the Jet Propulsion Laboratory, California Institute of Technology, under a contract from the National Aeronautics and Space Administration. Thanks to the technical staff in the OEL, the early MGDS development teams, and the JPL Mission Operations Teams for their enthusiasm and support.

Data Flow for Commanding Process


Command Sequence Generation
Spacecraft Message or Packet File
Validate / Reformat (Telecommand Wrap)
DSN Format File
Universal Command Translator / Database Libraries)
Command XLT Database
Spacecraft Message or Packet File
Database
Command Control Transmission/ DSN Interface (CMD)
GCF Blockss
GCF Interface (GIF)
Monitor Blocks
Mnemonic Command Real-Time Translator
Command Mnemonics File
Light Time File
S/C Clock Time Files
Spacecraft

Command GUI



```
DSS: 14        SC: 87        CMD SYSTEM: blondie        CWS: blondie        UTC: 024/23:12:36

Controlled by:     MOCC
Radiation Rate:    32.00 bps
Buffer 0:          0BF0
Current Mode:      CAL2
Enabled Mode:      CAL2
PRIME FILE STATUS:

CPA QUEUE
     File         Bit_1
 1)
 2)
 3)
 4)
 5)

CPA DISK
     File         Status
 1)
 2)
 3)
 4)
 5)
 6)
 7)
 8)

MONITOR 5_9
Antenna ID:                              RM:
Transmitter ID:                          RCS:
Transmitter Beam:                            CMA Mode:
Transmitter Power:                           CCS:
XBand Range:
SBand Range:
Exciter Freq:

RADIATED ELEMENT
  FILE NAME:
 Element Attempt  Event     Alarms                 Bit-1        SCE        ERT

REMAINING RADIATION TIME:  000/00:00:00     ALARM: LocalCntrl/
RADIATION  COMPLETES  AT:  000/00:00:00

PROJECT FILES                 8 files
 File Name        SC DSS Status Format     Bit 1/Open W.    Creation Time
C301              87  5    T    CMD_DSN                     1994-021T19:32:29
U01               87  5    T    CMD_DSN                     1993-267T01:25:42
cap101            77  5    S    CMD_DSN                     1991-253T21:39:09
ell.efdu          77  0     R   CMD_XLT                     1990-305T17:04:21
cmd_xlt.77        77  0     R   PROC_XLT
SCLK_SCET0167.e77 77  0     R   SCLK_SCET                   1989-285T12:39:05
C3                87  0         SCMF                        1993-280T21:28:03
MMSLB             87  0     R   SCMF                        1993-154T00:35:05

 File Name     SC DSS Status Format     Bit 1/Open W.    Creation Time

CURRENT DIRECTIVE

DIRECTIVE HISTORY
[1:D] 024/23:09:52 alloc sc=65 dss=14 to="blondie" vc="v2"
[2:D] 024/23:09:57 conn sc=65 dss=14
[3:D] 024/23:10:49 dealoc sc=65 dss=14 option=override
[4:D] 024/23:11:25 alloc sc=87 dss=14 to="blondie" vc="v2"
[5:D] 024/23:11:30 conn sc=87 dss=14

DIRECTIVE RESPONSES
[1:R]024/23:09:54 SC= 65  DSS= 14 is allocated
024/23:10:08 VC to GIF opened.
[4:R]024/23:11:27 SC= 87  DSS= 14 is allocated
024/23:11:40 VC to GIF opened.
```

ORIGINAL PAGE IS
OF POOR QUALITY