An Automation Language for Managing Operations (ALMO) in the Deep Space Network

P. F. Santos[1] and P. Pechkam[1]

Configuring a set of devices for pre- and post-track activities in NASA's Deep Space Network (DSN) involves hundreds of keyboard entries, manual operations, and parameter extractions and confirmations, making it tedious and error prone. This article presents a language called Automation Language for Managing Operations (ALMO), which automates operations of communications links in the DSN. ALMO was developed in response to a number of deficiencies that were identified with the previous languages and techniques used to manage DSN link operations. These included a need to (1) provide visibility to the information that resides in the different link devices in order to recognize an anomaly and alert the operator when it occurs, (2) provide an intuitive and simple language capable of representing the full spectrum of operations procedures, (3) mitigate the variations in operating procedures experienced between different tracking complexes and supports, and (4) automate overall operation, reducing cost by minimizing work hours required to configure devices and perform activities. With ALMO, for the first time in DSN operations, operators are able to capture sequences of activities into simple instructions that can be easily interpreted by both human and machine. Additionally, the device information, which used to be viewable only via screen displays, is now accessible for operator use in automating their tasks, thus reducing the time it takes to perform such tasks while minimizing the chance of error. ALMO currently is being used operationally at the Deep Space Communications Complex in Canberra, Australia. Link operators at the Madrid, Spain, and Goldstone, California, communications complexes also have received training in the use of ALMO.

I. Introduction

The Deep Space Network (DSN) is a worldwide network of spacecraft tracking and communications complexes located in Madrid, Spain; Canberra, Australia; and Goldstone, California. Each complex is capable of performing multiple missions simultaneously, all of which involve operating communications links. A DSN communications link is a collection of devices used to track and communicate with a spacecraft [2]. Examples of devices include antennas of various sizes, from diameters of 26 meters to 70 meters; transmitters; and receivers. Track supports must share a complex's capabilities, devices, and time allotments/constraints.

---

[1] Applications Development Section.

Before the Automation Language for Managing Operations (ALMO) was developed, DSN link operators used a macro language that provided a certain amount of link-configuration automation but had limited flow control, imposed a limit on the size of a script, and provided only a few constructs for performing link activities. It also did not provide ways of accessing device monitor data and provided only limited mechanisms for automatic parameterization. These limitations forced operators to create different versions of a script for use with different tracking supports, making maintenance of these scripts difficult.

Another language called BasicScript also was used to automate link activities. However, it lacked the constructs that automation needs to easily perform DSN operations, imposed size restrictions on scripts, limited usage of variables and subroutines, and required that operators understand every detail and all the code used to monitor and control various devices. In addition, it also employed a one-layered architecture, making the operator responsible for writing all the code for monitoring link and device status. This made it tedious and difficult for operators to understand and fully modify a script, especially should last minute changes be required.

II. Goals

ALMO was developed in response to the growing need to reduce operations cost by simplifying and minimizing the amount of time required to operate the DSN and the need to reduce operator workload, the number of errors, and the recovery time oftentimes associated with complex manual operations and resource limitations.

ALMO developers conducted a series of meetings with the cognizant device system engineers and programmers and with representative DSN link operators in an effort to fully understand the various ways devices and links are operated at the different communications complexes for different tracking supports. ALMO developers also spent a considerable amount of time with operations personnel at Deep Space Station 14 (DSS 14) during real-time configuration and tracking supports to fully comprehend the process and environment in which DSN operations are done. Knowledge gained from years of interacting with DSN link operators also was incorporated into the design and syntax of ALMO. The result was a language that is

Extensible: Capable of representing the full spectrum of operations procedures

Flexible: Allows for the variations in operations procedures between different operations complexes and tracking supports

Robust: Provides the constructs necessary to identify problems, accesses device monitor data and diagnostic information, and tailors general procedures to specific circumstances

Maintainable: Easy to update and maintain

User natural: Readable and usable by both computer and human operator [2]

III. ALMO

A. Definition

The Automation Language for Managing Operations, ALMO, was implemented using the C language and an embeddable scripting language called Tcl/Tk (Tool Command Language).

B. Benefits

Operators using ALMO are given the ability to specify, retrieve, and store device or subsystem (S/S) information that previously was viewable only from screen displays. It is crucial that operators be able to access this information, since it can be used to monitor and control the subsystems; determine the logical

flow of a script that they want to write to automate a task; and detect anomalies during an activity, enabling them to develop more intelligent and robust scripts.

For example, when an operator sends a directive to a subsystem, the subsystem returns a directive response, which is simply an acknowledgment that the directive was received. It does not indicate the successful or unsuccessful execution of a directive [4]. Additionally, the ability to associate a directive response with its specific directive is crucial in determining whether or not a directive was successfully sent. ALMO's operator directive (OD) command provides mechanisms that identify directive responses with their associated directives. Furthermore, it is important to determine whether a directive had the intended effect. With ALMO's monitem, verifySts, and eventSubscribe commands, the operator can use monitor data and/or event-notice messages to verify successful or unsuccessful execution of directives. Monitor data are data provided by a subsystem that reflect its state and parameter values and are stored in a predefined format. Event-notice messages are textual messages that relay information about the state of a subsystem or a value of a subsystem parameter. However, when monitor data and event-notice messages are generated, they are not explicitly tied to any directive. The operator must rely on his/her experience to determine which directive was most likely to have caused the subsystem to send the event-notice message or to generate the monitor data change. The monitem, verifySts, and eventSubscribe commands allow an operator to verify directive execution by specifying which monitor data items and event-notice messages to check. Moreover, each ALMO command has a built-in error-detection and -notification capability such that the operator is alerted when an anomaly occurs. The operator also is given a description of the anomaly and presented with several courses of actions to take. Of course, the operator is allowed to ignore an anomaly and continue with the next command in a script.

ALMO also provides mechanisms for automatic parameterization. For the first time, operators are able to retrieve parameter values from electronic sources, i.e., the subsystem's monitor data and sequence of events (SOE) using ALMO's param and monitem commands. Operators also may specify default values if they wish. This is a useful feature because operators usually are required to refer to hard copies of data, i.e., the SOE and the network operations procedures (NOP), for parameter values that need to be entered manually. With ALMO, the chances of error and the time spent in parameter validation and verification is greatly reduced.

C. ALMO Architecture

ALMO commands are parsed, interpreted, and executed via an engine called the ALMO engine, which is a component of the network monitor and control (NMC) software.

ALMO uses the NMC software as its means of communicating with the various DSN link devices or subsystems. It provides real-time monitor and control of the DSN. The NMC uses temporal dependency networks (TDNs) to automate pre- and post-track calibration of subsystems. A TDN is a directed graph of interconnected nodes that represents an end-to-end sequence of operations. It also specifies sequential and parallel operations. Each node in a TDN is called a block [2]. A block contains pre- and post-track conditions and directives that are sent to the subsystems sequentially. A directive is a control message sent to an individual subsystem instructing it to perform a specific task. A block may contain one or more directives to more than one subsystem and is written in ALMO.

In the NMC environment, there are two engines that communicate with the ALMO engine: a connection engine, which is part of the NMC workstation software, and a TDN engine, which is part of the NMC automation software. The connection engine is used to route and log directives to subsystems as well as to collect event information for its operations logs. It is a method by which the operator can collect directives from simultaneously running blocks and confirm their sources and destinations for security reasons. Otherwise, the ALMO engine is fully capable of routing the directives directly to the subsystems themselves. On the other hand, the TDN engine is used for routing user information collected from the operator, information tables, or monitor data from other outside sources. Figure 1 illustrates the ALMO

architecture, which can be broken down into four key areas: the command interpreter, the event handler, the intercommunication protocols, and the graphical user interface (GUI).

1. Command Interpreter. The command interpreter consists of the ALMO command library and the interpreter that translates the ALMO command into machine language and executes it before going to the next ALMO command. The command interpreter incorporates Tcl, an embeddable and interpreted scripting language that provides generic programming facilities as variable substitution and control flow. An extension to Tcl called Tk is included to provide graphical routines. Lastly, commands that were designed and developed mainly for interacting with subsystems are added to the mix. The whole of these parts forms ALMO. Thus, the ALMO library is comprised of the Tcl interpreter, Tcl commands, Tk commands, and ALMO commands. The ALMO library is shown in Fig. 2.

2. Event Handler. The event handler is responsible for collecting and processing "events" or notices of activity from subsystems, the GUI, and/or the external and internal infrastructure. More importantly,

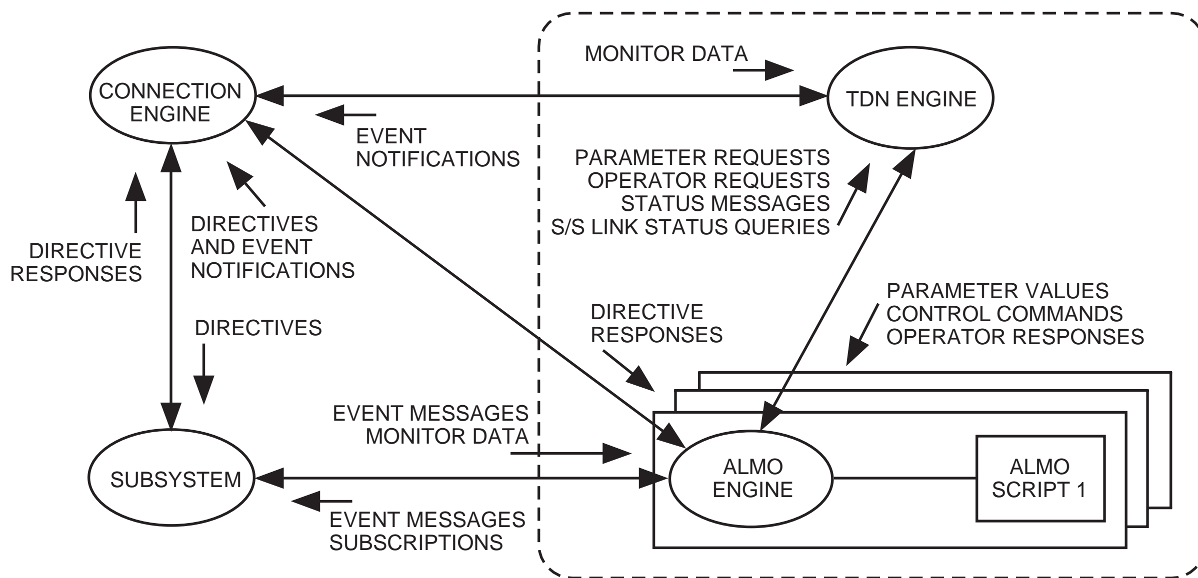

Fig. 1. ALMO architecture.

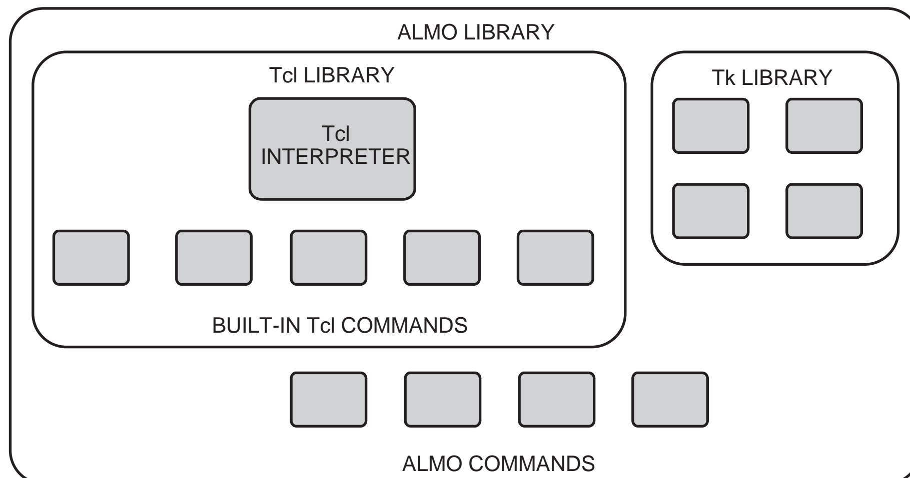

Fig. 2. ALMO library.

the event handler is key in responding to events, allowing an operator to write event-driven blocks—i.e., instead of processing instructions sequentially, the event handler waits for events to come in and then acts upon them.

There are various forms of events. Internally, there are events produced by the GUI or messages from another process with information or a request. In addition, the engine should be able to respond to control commands from the user. Such commands include pausing and running. The script also may need to make parameter or display requests, for example, to another engine. Likewise, the engine should be able to receive and process the responses from other engines. Moreover, asynchronous communication, i.e., non-blocking communication, is established so that any messages will be stored and kept in an input/output (I/O) buffer until read. During script execution, the event loop becomes active via calls to event-handling routines; otherwise events do not get processed until the command finishes. All of the above help in providing an event-driven engine that can process ALMO commands in a script while at the same time react to any GUI requests or events from subsystems or other engines.

3. Intercommunication Protocols. The ALMO engine interacts with a number of different processes and subsystems. In order for it to exchange information with these, however, all participants must be able to understand each other. This is accomplished using intercommunication protocols that define a common language.

UNIX pipes are used to establish connection between the ALMO engine and the TDN engine, which also is a UNIX process, and text messages are used as the communication protocol. Since TDN is part of NMC automation and the NMC workstation employs the DSN common services for communicating with subsystems, the ALMO engine inherits this additional foundation, which includes the monitor and control services (MCS). MCS is based on an implementation of the distributed computing environment (DCE) and distributed file system (DFS) and uses remote procedure calls (RPCs) for interprocess communication. Figure 3 shows the intercommunication protocols used by ALMO.

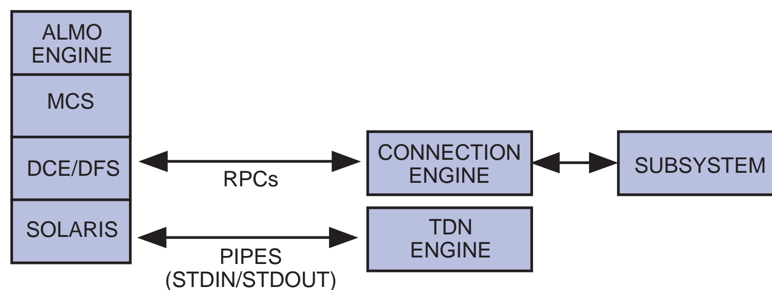

Fig. 3. Intercommunication protocols by ALMO.

4. Graphical User Interface. A graphical user interface, called the block viewer, is provided to display ALMO blocks, which are scripts that are written in ALMO. A block viewer has the ability to show an active script during execution. An arrow pointing to a line in an ALMO block indicates which command currently is being processed. Perhaps most importantly, the GUI allows the user to control execution of the block, e.g., pause, resume, and reduce or increase execution speed. The graphical user interface is developed entirely in Tk, the graphical extension to ALMO's Tcl code. A snapshot of the block viewer is provided in Fig. 4.

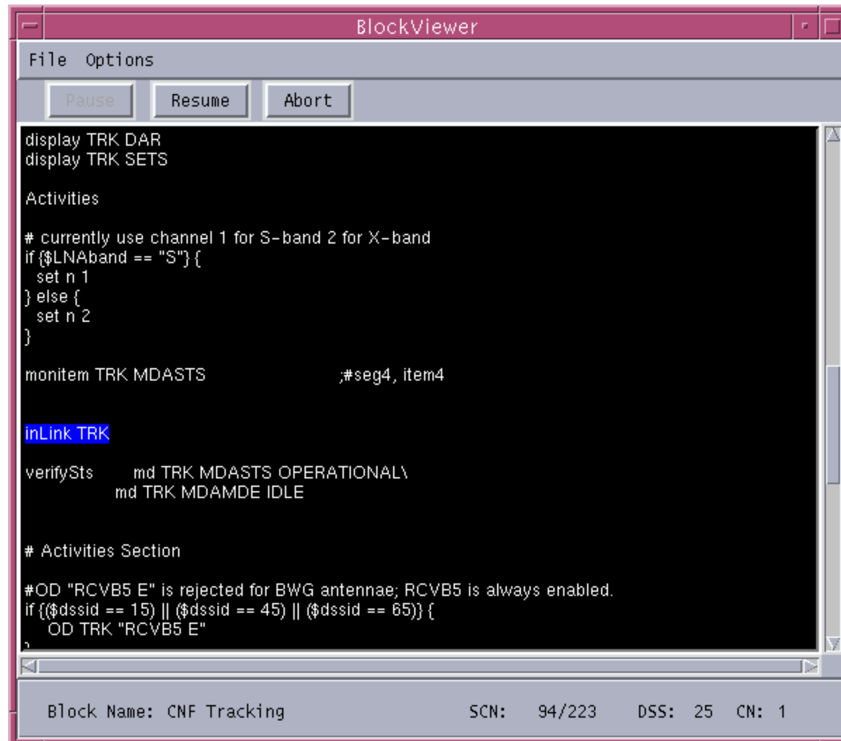

Fig. 4. The block viewer.

D. Language Description

1. Language Features. ALMO provides many constructs for performing DSN link operations. Some of the features are

(1) Specifying a block's associated display information

(2) Sending messages to the display log

(3) Declaring, polling, subscribing, and fetching monitor data

(4) Obtaining parameter inputs from either the operator or the electronic source

(5) Querying for a subsystem's link status, i.e., is the subsystem in the link or not?

(6) Suspending and resuming block execution

(7) Querying for a subsystem's short and long directive destination code (DDC) name

(8) Verifying directive status via monitor data, event message, and/or operator confirmation

(9) Prompting the operator to make a selection among several options

(10) Prompting the operator to enter a parameter value

(11) Obtaining the current time

(12) Comparing, adding, and subtracting time

(13) Subscribing and unsubscribing to event-notice messages

(14) Calling one or more blocks within a block

(15) Delaying block execution

Table 1 provides a complete list of ALMO commands.

Table 1. ALMO commands.

ALMO command	Description
startBlock	A required command that defines the start of a block.
display	Defines and declares a display associated with a block.
monitem	Declares, subscribes, polls, and fetches monitor data to be used in the block.
param	Sends a parameter request.
Activities	A required command that indicates the start of activities in a block.
inLink	Queries the connection for subsystem link status.
getShortDDC	Returns the short DDC name if the subsystem is in the link.
getLongDDC	Returns the long DDC name if the subsystem is in the link.
selectOp	Invokes a dialogue box and prompts the operator to make a selection.
inputOp	Invokes a dialogue box and prompts the operator to enter a string input.
getUTC	Returns the current time in universal time coordinated (UTC).
cmpUTC	Compares the current time with a given time.
addTime	Returns the new time after adding <period of time> to <time>.
subTime	Returns the new time after subtracting <period of time> from <time>.
compare	Performs either a string or numerical comparison.
OD	Sends a directive.
opsLog	Sends a message to the log.
verifySts	Verifies the status of a directive or a subsystem using one or more monitor data, event-notice message, and operator confirmation.
call	Runs another ALMO block (child block) while the caller block (parent block) waits for the child block to finish. When the child block is finished, the parent resumes execution.
release	Runs another ALMO block (child block) while the caller block (parent block) continues execution after it has "released" the child block. The parent block does not wait for the child block to finish and runs in parallel with the released block.
eventSubscribe	Subscribes to events from the event-message server.
eventUnsubscribe	Unsubscribes to an event.
delay	Sets delay in block execution.
wait	Suspends block execution for a given period of time.
endBlock	A required command that defines the end of a block.

2. Sample Block. A block written in ALMO is concise, readable, and easy for operators to understand, while ALMO's simple, high-level commands perform the tedious and critical functions. ALMO is able to greatly reduce the amount of code an operator has to write and understand. In fact, a nine-page block written in BasicScript was reduced to one page when converted to ALMO. The block shown in Fig. 5, written in ALMO, configures the DSN's metric data assembly (MDA) subsystem and currently is being tested at the Deep Space Communications Complex in Canberra, Australia.

An ALMO block consists of two sections:

(1) Declarations—variable- and parameter-declarations section using one or more param, monitem, and display commands.

(2) Activities—contains one or more of the following:

(a) Directives to subsystems using the OD command

(b) Directive verification using the verifySts command, which verifies successful execution of a directive via one or more directive responses, monitor data, and event-notice messages. If a directive cannot be verified with any one of the three methods, the operator may be prompted for confirmation using the confirmOp command.

(c) Link-status checking using the inLink command

(d) Time manipulations and comparisons

(e) Status and diagnostic messages to be sent to the operations log

(f) Invocation of other blocks

(g) Event-message subscriptions

The example in Fig. 5 defines parameters, sends directives to a subsystem identified as TRK (the generic subsystem name for the MDA), and checks the subsystem after each directive is sent. The first command executed will be startBlock, which takes one argument—the name of the block—that it uses for logging purposes. In addition, the command handles initialization of the engine, including establishing the communication route from the engine to the subsystem. On the other hand, endBlock (line 82) is used to indicate the end of the block and handles shutdown of the engine as well as logging the end of block execution. If the engine were to quit before endBlock was reached, the engine would exit the same way with the exception that it would be considered an "abnormal exit."

The group of commands between startBlock (line 1) and Activities (line 27) is considered the declarations section, where parameter requests are collected and information is passed between the TDN engine and the ALMO engine. These param commands are compiled and processed when Activities (line 27) is reached. After the user has filled in values for the parameters, block execution then proceeds to the if statement on line 30 in what is considered the activities section. The if statement simply determines the value of n depending on what the variable LNAband is set to, as determined by the value of monitor data item SOE_LNAband (line 16).

To check the health of a subsystem, a monitor data item must be subscribed to and the monitor data value checked. This can be accomplished with the monitem command on line 38. The following if statement (line 40) compares the value of MDASTS to the text OPERATIONAL. If they match, then execution proceeds to the next command. Note that the same logic can be accomplished using a simpler implementation, verifySts md TRK MDASTS OPERATIONAL.

If the subsystem is healthy, then the block will send a number of directives via the OD command. The directives configure the MDA for spacecraft tracking, but more importantly, after most of the directives is a verifySts command that checks the subsystem to confirm that the directive actually performed the intended action.

The commands starting at line 65 determine what directive to send based on what time it is. If the current time is not within 5 minutes of its designated start time (stored in the variable bot, which stands for "beginning of track"), then it will time stack a directive (line 79). Otherwise, it sends a RUN directive immediately. Lastly, as described above, if the block has executed the way it was intended to execute, then endBlock (line 82) will be executed, and the block has finished running.

```
1  startBlock "CNF Tracking"
2
3  # Parameter Inputs
4
5  # For three way tracking mode, get uplink dssid and append it.
6  monitem NMC SOE_Way          ;# 1-, 2-, or 3-way tracking
7  if {$SOE_Way == 3} {
8     monitem NMC SOE_3WayDSS
9     set way ${SOE_Way}W${SOE_3WayDSS}
10  } else {
11     set way $SOE_Way
12  }
13
14  param dssid "DSS ID" conn SOE_Dss
15  param predSetID "Predict Set ID" conn FranzCode
16  param LNAband "Downlink Band" conn SOE_LNAband
17  param bot "Beginning of Track" conn SOE_Bot
18  param sam "Sample Rate" conn SOE_Doppsample
19  param way "Track Mode/Way" op $way
20
21
22  # Associated Displays
23  display TRK CNF
24  display TRK DAR
25  display TRK SETS
26
27  Activities
28
29  # Use channel 1 for S-band, 2 for X-band
30  if {$LNAband == "S"} {
31    set n 1
32  } else {
33    set n 2
34  }
35
36  inLink TRK       ;# verify that the TRK subsystem is in the link
37
38  monitem TRK  MDASTS             ;# seg4, item4
39
40  if ![compare OPERATIONAL $MDASTS] {
41    selectOp "The MDA is not operational" Continue Restart Abort
42  }
43
44  verifySts  md  TRK  MDAMDE  IDLE
45
46
47  # Activities Section
48
49
50  if {($dssid == 15) || ($dssid == 45) || ($dssid == 65)} {
51     OD TRK "RCVB5 E"
52  }
53
54  OD TRK "RANGE D"
55
56  OD TRK "SAM $sam" C:WAIT 7
57        verifySts md TRK [format "C%sSMP" $n] $sam
58
59  OD TRK "WAY $way" C:WAIT 7
60        verifySts md TRK [format "C%sWAY" $n] [string index $way 0]
61
62  OD TRK "PRED $predSetID" C:WAIT 7
63        verifySts md TRK DPRDID $predSetID
64
65  set time [subTime $bot 000:00:05:00]
66  # Same as variable "time" but with no colons and no DOY (per timed direct format)
67  set formattedTime [subTime $bot 000:00:05:00 -noDOY -nocolon]
68
69  if {[cmpUTC $time] == 1} {
```

Fig. 5. Sample block in ALMO.

```
70       OD TRK "RUN"
71       wait 7
72             verifySts md TRK MDAMDE RUN
73  } else {
74       OD TRK "RUN NORPT"
75       wait 7
76             verifySts md TRK MDAMDE RNHR
77
78       getLongDDC sd TRK
79       OD NMC "^$formattedTime $sd RUN"
80  }
81
82  endBlock "CNF Tracking"
```

Fig. 5 (contd)

IV. Summary

Several scripting languages have been developed to reduce the workload of operators and aid in the automation of operations in the DSN. However, these languages lacked certain attributes that a DSN automation language needs—a language that is simple and robust and that encompasses the full spectrum of DSN operations. ALMO was developed with these in mind and is continuously being improved to fully meet the needs of DSN operations. It has significantly reduced the amount of work required to configure and operate a communications link and has given operations personnel more control and flexibility during pre- and post-track operations. To date, ALMO is being used operationally at the Deep Space Communications Complex in Canberra, Australia. In addition, some research tasks have expressed interest in using ALMO to develop blocks for fault-detection and -recovery systems. All of this indicates that a ceiling on ALMO's flexibility and capability has not yet been reached.

References

[1] R. W. Hill, Jr., K. Fayyad, P. F. Santos, and K. Sturdevant, "Knowledge Acquisition and Reactive Planning for the Deep Space Network," *Working Notes of the 1994 Fall Symposium on Planning and Learning: On to Real Applications*, New Orleans, Louisiana: AAAI Press, 1994.

[2] K. E. Fayyad and L. P. Cooper, "Representing Operations Procedures Using Temporal Dependency Networks," SpaceOPS 92: Second International Symposium on Ground Data Systems for Space Mission Operations, Pasadena, California, 1992.

[3] R. W. Hill, Jr., S. Chien, K. Fayyad, P. F. Santos, and C. Smyth, "Planning for Deep Space Network Operations," *Proceedings of the 1995 AAAI Spring Symposium on Integral Planning Applications*, Palo Alto, California, pp. 51–56, March l995.

[4] B. B. Welch, *Practical Programming in Tcl and Tk*, New Jersey: Prentice Hall, Inc., p. 79, 1995.