Semantic Definitions of Spacecraft Command and Control Languages Using Hierarchical Graphs

M. Elwakkad Zaghloul*
The George Washington University, Washington, D.C.
and
W. Truszkowski†
NASA Goddard Space Flight Center, Greenbelt, Maryland

In this paper a method is described by which the semantic definitions of spacecraft command and control languages can be specified. The semantic modeling facility used is an extension of the hierarchical graph technique, which has a major benefit of supporting a variety of data structures and a variety of control structures. It is particularly suited for the semantic descriptions of such types of languages where the detailed separation between the underlying operating system and the command language system is system dependent. These definitions are used in modeling the systems test and operation language which is a command language that provides means for the user to communicate with payloads, application programs, and other ground system elements.

Introduction

IN this paper a method is described by which the semantics of command languages can be specified. The semantic modeling facility used to describe the command language is an extension of the hierarchical graph technique[1-4] which has the major benefit of supporting a variety of data and control structures. This means that the hierarchical model allows hierarchical specifications depending upon the different aspects of the modeled system. This is particularly suited for the semantic descriptions of such type of languages where the detailed separation between the underlying operating system and the command language system is system dependent and detailed modeling depends on the particular implementation.

A command language is defined as a set of commands which are used to communicate the user to an underlying system. These commands are of interpretive type, i.e., each time the user specifies a command to the system, it is immediately executed. Each command within the language is usually built out of a small set of syntactic primitives and contains arguments which refer to the conceptual entities within the command. Usually the command language system is part of a system with a set of tasks that ultimately the user wants to accomplish. The semantic of each command within the language means the definitions of the set of operations and procedures associated with each command to accomplish the user's task or subtask. The semantic definitions of the space flight control centers languages are important for the implementers because they clearly provide the meaning of each command in the language and model the language processor clearly and simply. This provides the implementers with a basic set of structures in which to base an implementation. As for the users, those definitions provide the user with a clear picture of the program execution and the underlying operation associated with each command.

The semantic model chosen is the hierarchical graph. A hierarchical graph or H-graph is a finite set of directed graphs over a common set of nodes, organized into a hierarchy. It provides a sufficiently general set of primitives which makes it possible to construct a model that will reflect the command language as closely as possible. The model must not preclude the possibility of being extended for new language features and for several implementations, but at the same time it must not be so general that it loses usefulness as a guide to the specific language implementation. The H-graph models described in the following sections provide these means.

The Hierarchical Graph Technique

Basically, the H-graph structure is composed of a finite set of edges connecting pairs of nodes in a one-way manner. Nodes are represented as boxes (rectangle or oval) with a designated entry node and edges as arrows connecting nodes. The edges leaving each node may contain unique labels. There may not be two edges leaving a node with the same label, but two edges leaving a node with different labels may end at the same node. Thus, parallel edges are allowed. Hierarchy is introduced into the above structure as follows. A universe U of atomic units is assumed. A hierarchical graph or H-graph over U is a graph in which the nodes are considered to be containers, each of which contains either an atomic unit from U or an H-graph over U. Thus, in terms of the usual flow charts, the H-graph is a flow chart in which each box contains either an atom or another flow chart whose boxes may in turn contain atoms or flow charts, and so forth to any depth.

The attributed hierarchical graph is an H-graph in which each node may contain atomic value (defined through function set v), structure graph value (defined through function set h), and some attributes (defined through function set a) representing some compile-time properties. Figure represents an example of attributed H-graph model of three element integer array A whose values are 2, 3, and 4 respectively. Detailed formal definitions of these function are given in Refs. 3 and 4.

In the definitions of command language semantics using the H-graph, the language should be translated into the graph language using the graph grammar.[1,2] Each graph grammar rule specifies a nonterminal on the left and an H-graph on the right. The context-free graph grammar works the same as an ordinary context-free grammar to generate a graph in its language. In generating a graph using the grammar, nodes with nonterminal values are replaced by graph nodes using the appropriate grammar rules. Three graphs may contain other nodes with nonterminal values. When all such nodes have been replaced by terminal nodes, the resulting graph is the

Received October 23, 1981; revision received July 8, 1982.

*Assistant Professor, Department of Electrical Engineering and Computer Science.
†Member, Data Systems Technology Office.

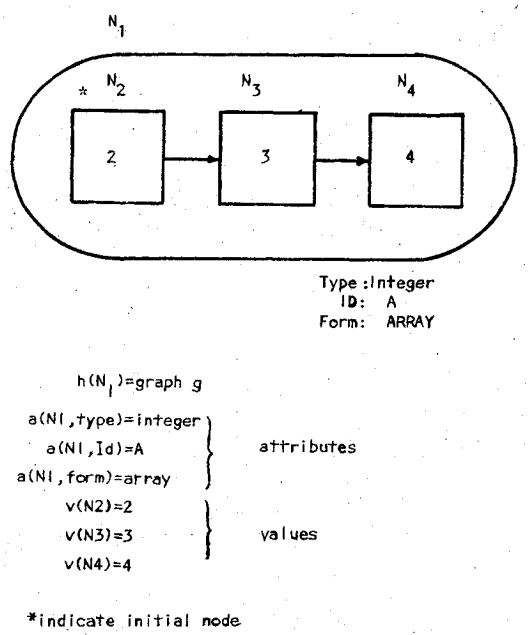

Fig. 1 H-graph model of three-dimensional array.

graph language defined by the graph grammar. An example to illustrate the graph grammar is shown in Fig. 2 where the conditional IF statement is illustrated. The ⟨Boolean expression⟩ value (true or false) which is stored at temporary node TEMP is assisgned to node BRANCH. The node BRANCH contains the edge label of edge to be followed during execution. The symbol # denotes the null value node. The ASSIGN instruction assigns the value of node TEMP as the value of node BRANCH. The nodes TEMP and BRANCH are common nodes used to communicate data between instructions. In an H-graph, if the entry node contains the name of an operation (for example, ASSIGN), the graphs they contain are termed instruction graphs.[1-4] Semantic functions are used to define instruction graphs in this model, where each instruction when applied to an H-graph in a certain state results in changing the state to the sequent state. The data structure used in this model is the set structure which permits a higher level of modeling of the command language. This type of data structure imposes a set of construction and accessing primitives on the sets of nodes of the graph which coincide with the standard set theoretic operations.

Thus, the semantic model for each command in the language is represented by a graph structure based on the hierarchical graph structure and an accessing primitive defined by the data structure used in the model. The syntax of the resulting graph defines a control structure which shows the state transition from an initial state of abstract machine into a final state or subsequent states. A set of semantic functions is used in mapping an H-graph into an H-graph to specify the meaning of the language components relative to one another. The semantic functions can be expressed in terms of other semantic functions in hierarchical order, where the conditions of application of such functions are included in their representation. It should be noted that the application of an instruction scheme as defined by the semantic function is strictly local; that is, the applicability of an instruction is determined entirely by the values of nodes that are either modifiable or pure inputs. The inaccessible nodes can neither affect the applicability of an instruction nor be modified by execution of an instruction. Such nodes are simply carried along into the next state without change. The use of locally defined instructions is crucial to most analysis of this type because instructions that are not locally defined may have various kinds of side effects or other undesirable effects that make their analysis difficult.

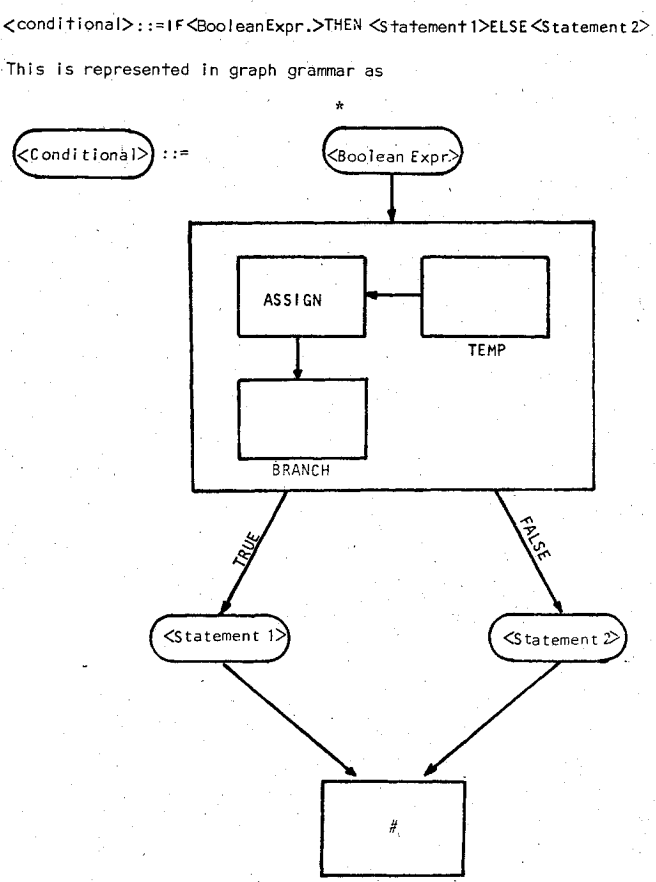

Fig. 2 H-graph model of conditional statement.

In designing the abstract model of a command language, it is necessary to represent the interaction between the command language system and the underlying system. This is represented in this model by the task command semantic function. This type of function is used by the model to represent the calling of the underlying operating systems to initiate the particular task named within the function and to start execution with the specific input set of nodes. This is convenient in defining the semantic of a command language where all of the tasks are assumed to be defined to the underlying system, and the job of the command language system is the appropriate initiation of a particular task.

Spacecraft Command and Control Languages

The above techniques are used to describe the semantic descriptions of the spacecraft command and control languages specifically, the systems test and operation language (STOL).[5-7] STOL is a well-defined model of the command and control languages currently in use by payload operation control centers (POCC). A major purpose of languages modeled by STOL is to support in an efficient and effective manner man/system interaction. To more fully appreciate the complexity and sophistication of the current command control languages one has merely to casually observe the evolution of spacecraft and their corresponding ground support functions from the rather simplistic approaches of the early 1960s to the complexity and sophistication of the 1980s. As Table 1 shows there has been over the last two decades[8] an increase in the complexity and sophistication of the class of unmanned spacecrafts. A major aspect of ground command and control systems which clearly indicates the complexity of growth is the operational concept of command management. A command management system is that portion of the ground command and control complex which forms the kernel of the man/system interface.

Table 1 Evolution of spacecraft and ground command and control, 1960-1980

1960s	1980s
Spacecraft evolution	
Simplistic spacecraft design	Accurate attitude control mechanisms
No onboard processors	Data recording capabilities
Low command/telemetry rates	Sophisticated command and telemetry handling systems
Single experiment packages	Programmable on-board processors
	High command/telemetry rates
	Multiple experiment packages
	Varying degrees of autonomous activity
Ground command and control evolution	
Slow teletype communication among ground tracking stations	Central facilities for real-time and batch commanding
Simple commanding capability	Multimission operational control centers
No central control	Intensive man/system interfaces
24 h/day manual monitoring	Highly coupled resources
	Advanced data distribution
	Advanced tracking and data relay concepts

The major functional and operations support requirements of a command management system are as follows:

1) To generate command sequences for maneuvering the spacecraft and controlling the experiment.

2) To provide procedures and computer programs for avoiding and recovering from certain potentially dangerous situations.

3) To manage onboard stored command processors (loading, execution, verification, continuity, and coordination).

4) Prepare onboard computer (OBC) loads of software modification and data updates.

5) To provide mission operations with a preplanned intergrated chronological event/command activity report on a continuous basis.

6) To provide mission computing services essential for effective ground control of the inflight payload.

7) To conduct operational feasibility studies.

8) To determine spacecraft equipment misalignments and provide for corrections.

9) To support ground systems simulations, integrations, and full ground systems readiness demonstrations.

Figure 3 shows the overall commanding process and the central role played by the command generation function. The major functions of a command management system[9] are displayed in Fig. 4.

A prime mechanism for providing the interactive user support in Fig. 4 is through a control center command language. Command languages, similar to programming languages, have evolved from primitive assembly language-like concepts to the high-level functional capabilities associated with today's programming languages (e.g., ADA). STOL, the example command and control language discussed in this paper, is a high-level functional language capable of supporting all the major activities associated with ground-based command and control activities including the highly man/machine interactions associated with command management. This language was designed by spacecraft

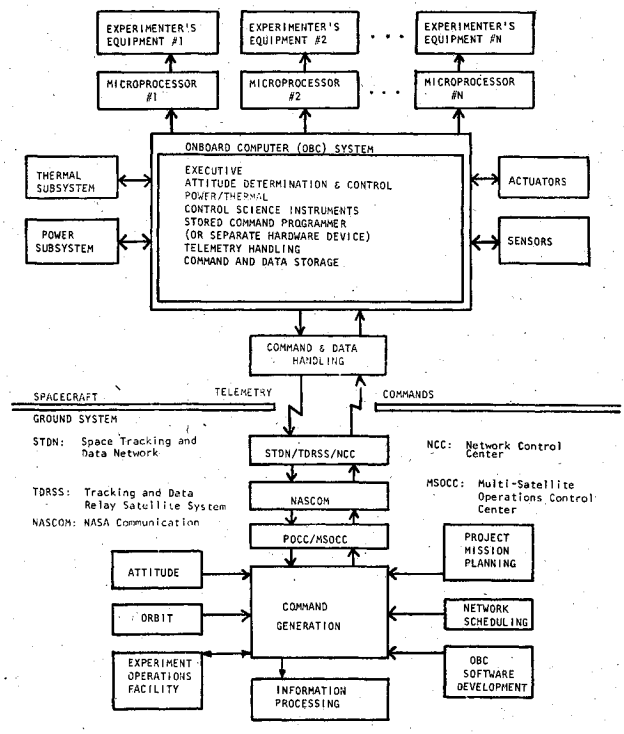

Fig. 3 Overall commands process.

developers, integration teams, and operation centers. It is a command language which includes several features adaptable to requirements of the NASA communities. STOL is an interpretive lanuage intended to control the payload integration, test, and operation. Commands submitted for interpretation may come from several sources. They may be entered from the console attached to the system. In this mode denoted as "console mode," the commands are executed immediately. One of the commands permitted in this mode causes a procedure to be initiated, placing the system in the "procedure mode." A procedure is a named collection of commands that reside in the system procedure library. When the system enters a procedure mode, the procedures commands are executed automatically and sequentially, unless directed by certain commands to alter the sequential statement execution. The procedure control commands cause the procedure to terminate and to return to the operational mode in effect during the initiation of the procedure. The user may abort or halt the procedure at any time by entering the appropriate command from the console. On halting a procedure, the execution of the procedure is stopped temporarily and the system is "suspended procedure mode"; the user may examine the progress of the procedure or input all commands that are valid in the console mode. In this mode the user may restart the suspended procedure with any command within the procedure body, thus causing a return to procedure mode, or abort activity at the console and enter the console mode directly.

There exist commands within the language that allow assignment of arithmetic and logic operations beside other commands to provide the initiation or termination of specific task within the underlying system. Also, commands exist that allow the user to send output data to various types of displays. Two types of variables exist in the STOL language: global and local. Global variables are accessible to the STOL system as well as to the underlying system. The local variables are defined only to the STOL system and are not defined to the underlying system. Figure 5 shows the functional flow of the STOL subsystems. Figure 6 describes the classification breakdown of the various commands with the STOL language.

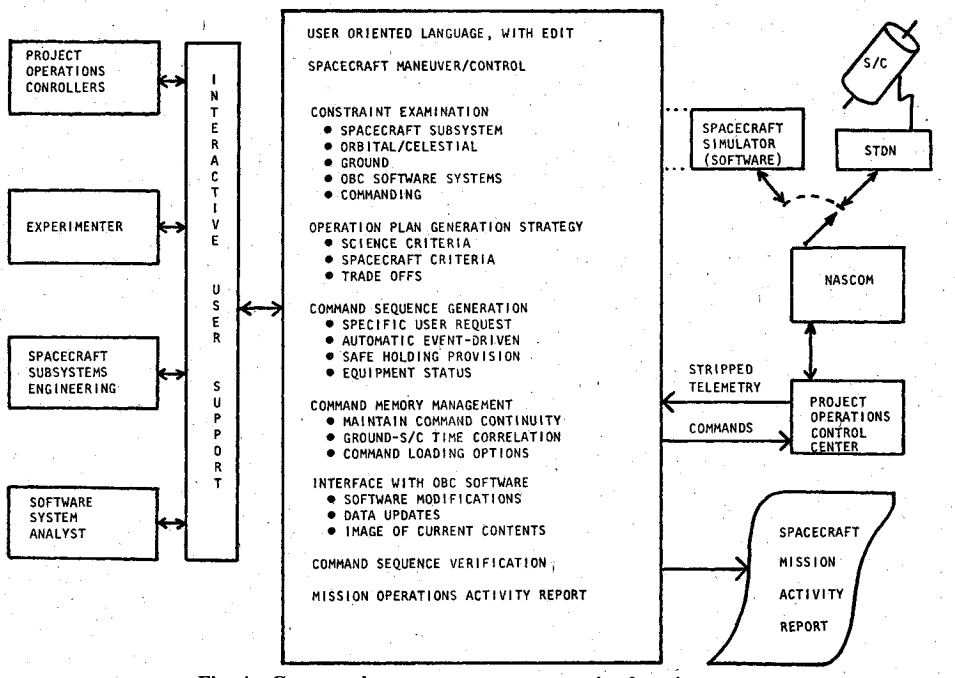

Fig. 4 Command management system: major functions.

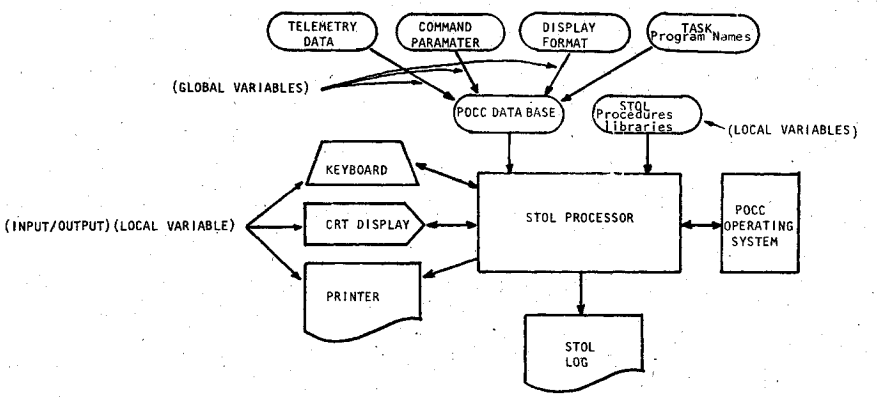

Fig. 5 STOL functional block diagram.

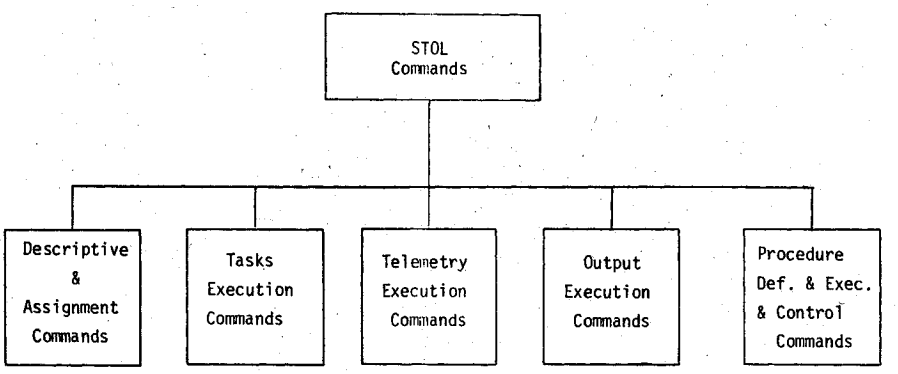

Fig. 6 STOL commands classifications.

In the semantic model described in this work, no attempt is made to describe all of the resources available with the underlying operating system or any deadlock prevention mechanism within the underlying systems. The semantic of the language will be described to the point of requesting the processing of a particular task from the underlying system. All other details are internal to the underlying operating system and generally depend on the STOL implementation.

In the following design, the various STOL system mechanisms will be separated to provide for many concurrent processes and functions in the system. The general organization of the STOL model will be as shown in Fig. 7. Three distinct processors within the system require three abstract interpreters: operation, procedure handler, and input interface.

The operation processor is responsible for processing all of the different classes of the STOL commands. It starts execution whenever it is called by the input interface processor. The different states of the abstract interpreter representing the operation processor are described by the graphs of all of the different classes of STOL command. The operation processor initiates messages to activate the underlying operating systems for each requested task within the underlying system and receives replies accordingly. Activation

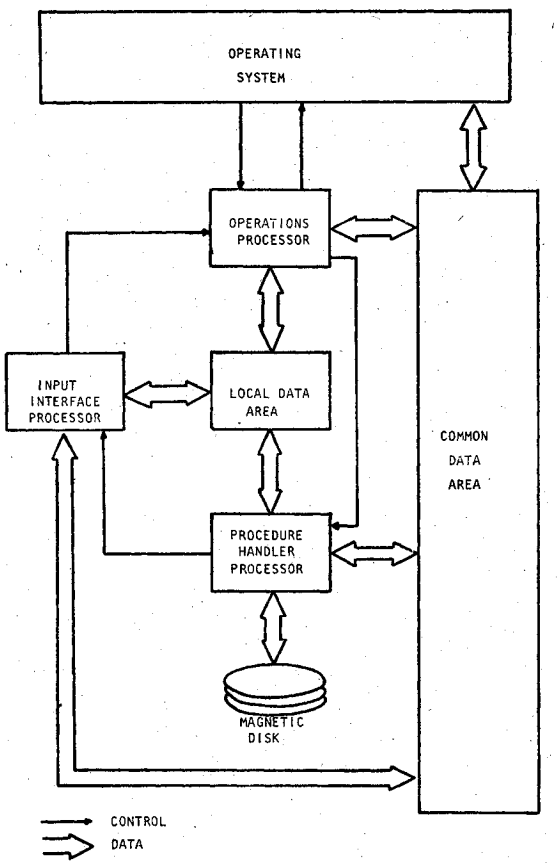

Fig. 7 General STOL processor elements and control flow.

of the procedure handler processor is performed by the operation processor.

The procedure handler processor is responsible for reading each procedure command from the particular procedure disk and performing the procedure control functions. The procedure handler processor activates the input interface processor which in turn serves in routing the data and control to the operation processor. In this case, input data are accepted either from the console or from the procedure (through the procedure handler processor). The console input always has priority over the procedure input.

In addition, there is a COMMON data area which is shared between the STOL system and the underlying operating system. In this COMMON data area, lie the COMMON tables used both by the STOL and the operating system; for example, the global variable list, the snap names list, the format names list, the chart names list, and the COMMON system nodes. A local data area is defined within the model, which is accessible to the STOL subsystems but not to the underlying system.

In this organization, each processor is represented by an abstract interpreter. Each interpreter is a sequentially working abstract machine. The states of each abstract interpreter are defined in terms of the hierarchical graphs. Messages are to be sent between these abstract interpreters. The messages are used to activate each interpreter to start from an initial state. When the final state is reached, each interpreter waits for another call in a no-operation state. All of the well-formed combinations of computations of all the abstract interpreters of the subsystems form the semantic definitions of the STOL processor system.

Language Examples

To illustrate the techniques used to define the semantics of the various commands within the spacecraft control centers command language, the following command examples were chosen. Those commands are processed at the operation

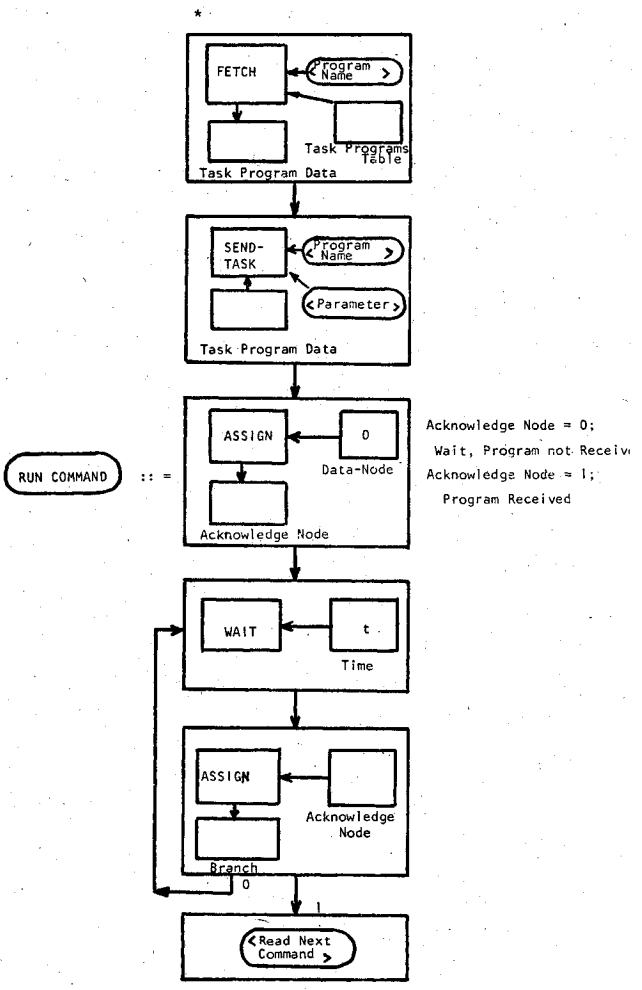

Fig. 8 H-graph model of RUN command.

processor of the abstract model of the STOL processc illustrated in Fig. 7.

The RUN Command

The RUN command initiates the execution of an ap plication program within a set of tasks defined to the systen The task programs table of references is defined by names an is known to the command language processor. The synta description of the RUN command is represented by the BN as:

RUN ⟨program name⟩ $\{,\langle\text{parameters}\rangle\}_0^n$

Examples are

RUN	READTLM	Start running task READTL
RUN	RDCAMC, 1000	Start running task RDCAMC with parameter 1000

Figure 8 illustrates the graph structure used to represent tl semantic of the RUN command. In this graph nontermin nodes are drawn as ovals and terminal nodes are drawn rectangles. Nodes may contain data items whose values a given or the name of a certain operation which is expressed terms of semantic functions. The graph structure shown Fig. 8 defines the sequence of states of the abstract machin The task program table is a node which contains the set of a task programs available to the system and each is defined i its name. The FETCH instruction which is described in terr of semantic functions denotes the search for the specific ta within the set of all task programs. If the task program is n

found, an error results. If it is found, its name is sent to the underlying system, signaling the start of execution of such task. If any parameters are present, they are sent also. The system waits for t seconds until the operating system acknowledges the receiving and executing of the requested task. The acknowledge node which is accessible to the underlying system is assigned by the underlying system to mark this condition. The branch node contains the arc label of the arc to be followed during execution at the next branch point.

The formal definitions of the semantic function FETCH in Fig. 8 are

FETCH (task program table, ⟨program name⟩; task program data)

$$= \text{set } h \text{ [task program data, } h(g)\text{]}$$

where the function set h sets the graph structure g to node task program data and g is the graph defined by the access instruction, i.e.,

$$g = \text{access (⟨program name⟩, task program table)}$$

$$= \begin{cases} \text{member } (\chi \mid \chi \epsilon h(\text{task program table}) \; \Lambda Id(x) \\ \qquad = \text{⟨program name⟩}) \\ \text{otherwise ERROR} \end{cases}$$

The ERROR node represents a special node that indicates that an error results. If g is the ERROR node, execution halts.

The SEND-TASK instruction is a task command semantic function which is used to signal the underlying system to start executing the specific program named with the specific parameter and data copied into the task program data. The formal definition of such an instruction is

SEND-TASK (⟨program name⟩, ⟨parameter⟩, task program data) =

EXECUTE: SEND-TASK (⟨program name⟩, ⟨parameter⟩, task program data)

The EXECUTE instruction is used to generate the required signal code to the underlying system. The signal code is defined through the instruction name SEND-TASK and the data in the respective nodes in its argument.

Telemetry Execution Commands

Telemetry commands are responsible for activating a telemetry task within the system. Each telemetry task is responsible for the acquistion of telemetry data, conversion of these data from raw telemetry counts into usable values, and all other telemetry activities. Example of telemetry commands is the ACQUIRE command. This command is used to send an underlying telemetry task handlers the data type and the associated parameter. The ACQUIRE command is represented by the BNF as

ACQUIRE ON, ⟨data type⟩, ⟨parameter⟩

where ⟨data type⟩ and ⟨parameter⟩ are system data and system parameters, respectively. The H-graph of this command is shown in Fig. 9 where the instruction ACQUIRE-ON initiates the acquisition task within the telemetry task handler in the underlying system. The data type and the parameter nodes are passed to the telemetry task handler. Therefore

ACQUIRE-ON (⟨data type⟩, ⟨parameter⟩) =

EXECUTE-TELEM: ACQUIRE-ON (⟨data type⟩, ⟨parameter⟩)

The EXECUTE-TELEM instruction generates the required signal code to the underlying telemetry task handler. The signal code is defined through the instruction name ACQUIRE-ON and the data in the respective nodes in its argument.

Output Execution Commands

An example of the output execution command is the PAGE command which initiates a predefined display page to be displayed on the CRT screen. The command fetches the page display data from the page name reference table and sends these data with page parameter to the display handler for execution. The page display data are assumed to be created beforehand and stored in a data base which is available to the STOL processor. The syntax of this command is

PAGE ⟨page name⟩ $\{$, ⟨page parameter⟩ $\}_0^n$

where ⟨page parameter⟩ specifies the device name. For example,

PAGE POWER, KG4 display POWER on device KG4

The H-graph for the semantic definition of this command is shown in Fig. 10 where the instruction PAGE initiates the display handler task to display the data in the node page display data with the specified parameter. The definition of this instruction is

PAGE (page display data, ⟨page parameter⟩) =

EXECUTE: PAGE (page display data, ⟨page parameter⟩)

In all of the above definitions the ⟨read next command⟩ instruction node appears at the end of each graph representation. This instruction is used to control the actions of the operation processor with respect to the modes of operation of the STOL processor. As discussed in previous sections each command processed within the operation processor in Fig.7 can be initiated either from a console or a procedure body, and the system operation mode may change depending on the execution of each particular command. The ⟨read next command⟩ instruction tests the mode of operation

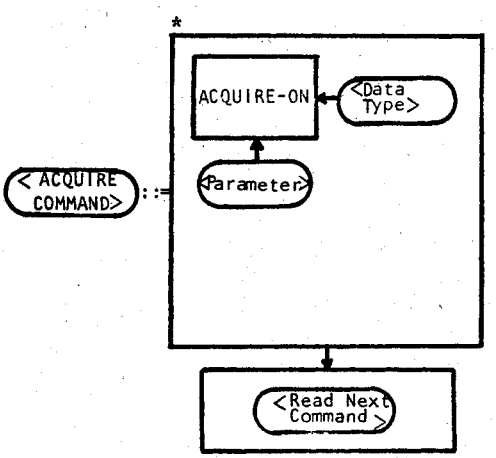

Fig. 9 H-graph model of ACQUIRE command.

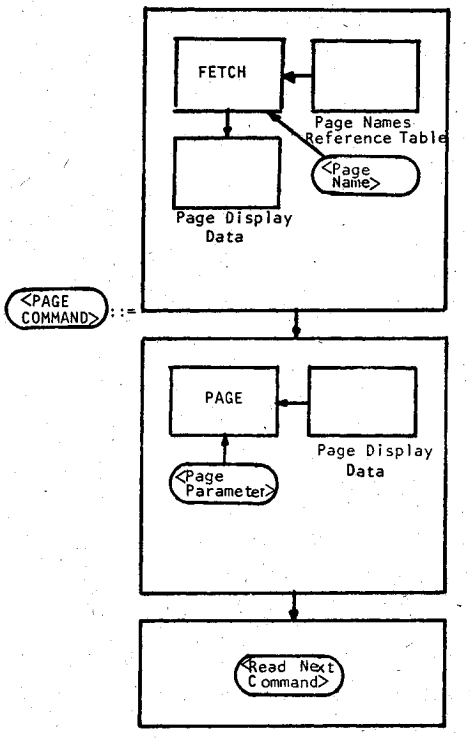

Fig. 10 H-graph model of PAGE command.

of the system. If a procedure mode is encountered, a signal to the procedure handler processor is sent to proceed on and send the next command in line. If a console mode encountered or suspended a procedure mode, no procedure call is issued and the operation processor waits for the next call.

Conclusions

A semantic model of the spacecraft command control languages was discussed in this Paper. This model is designed to model as closely as possible the complexity of such language, but at the same time it is extendable to accommodate new language features. The model provides definitions that are usable by language implementators as a base design for a reasonable implementation of the language. The semantic descriptions provided are based on the hierarchical (H) graph structure. It has the major benefits of supporting a variety of data structures and a variety of control structures. This provides a comprehensive model of the command language processor and serves as the basis for user understanding of the language. The systems test and operation language (STOL) was used to illustrate the complexity of such language where simple examples were chosen from the set of commands available in the language to show its properties. This leads to much deeper insight into the structure of such language and further extensions of new commands to the language.

References

[1] Pratt, T.W., "Application of Formal Grammars and Automata to Programming Language Definition," *Applied Computation Theory; Analysis, Design, Modeling,* edited by R.T. Yeh, Prentice-Hall, Englewood Cliffs, N.J., 1976, pp. 250-273.

[2] Pratt, T.W., "Pair Grammar, Graph Language and String to Graph Translations," *Journal of Computer and System Sciences,* Vol. 5, Dec. 1971, pp. 560-595.

[3] Basili, W.R. and Truner, A.J., "A Hierarchical Machine Model for the Semantics of Programming Languages," *ACM/IEEE Proceedings of Symposium on High Level Language, Computer Architecture,* Nov. 1973, pp. 152-161.

[4] Basili, W.R., "A Structured Approach to Language Design," *Journal of Computer Languages,* Vol. 1, 1975, pp. 255-273.

[5] DesJardins, R., "GSFC Systems Test and Operation Language (STOL) Functional Requirements and Language Descriptions," NASA Rept. X-408-77-100, June 1977.

[6] Zaghloul, M.E., "GSFC Systems Test and Operation Language, (STOL) Syntax Specifications," Computer Sciences Corp., Silver Spring, Md., Rept. CSC/TM-78/6309, Oct. 1978.

[7] Zaghloul, M.E., "GSFC Ssytems Test and Operation Language (STOL) Semantic Specifications," Computer Sciences Corp., Silver Spring, Md., Rept. CSC/SD-79/6143, Dec. 1979.

[8] Costa, S. Richard, "Mission Control," *AIAA/NASA Proceedings of the Symposium on Spacetracking and Data Systems, Assessment Theory,* Vol. 8, Sept. 15, 1981, pp. 117-129.

[9] Soscia, S., "Command Management System (CMS)," Informal NASA Goddard Space Flight Center memorandum, Code 514.