GENERIC ENVIRONMENT FOR INTERACTIVE EXPERIMENTS

Robert G. Fainter
Sue R. Guy
Joseph F. Maynard
Timothy E. Lindquist

Technical Report No. CS830009

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

August 11, 1983

ACKNOWLEDGEMENTS

This research was supported in part by the Office of Naval Research under ONR Contract Number N00014-81-K-0143, and Work Unit Number NRSRO-101. The effort was supported by the Engineering Psychology Programs, Office of Naval Research, under the technical direction of Dr. John J. O'Hare.

Many people other than the authors have made significant contributions to GENIE. The spouses of the authors, Lily K. Fainter, Lyndon Guy and Vicki Maynard, all provided encouragement and support during long hours spent programming. We offer our special thanks to them.

Navy Petty Officers First Class Hanson, Spengler and Spinelli gave us our introduction to Carrier Air Traffic Control. They gave us an entire day of their time to explain this complex task. Dr. Mel Moy, of the Naval Personnel Research and Development Center (NPRDC), San Diego, California, provided his assistance for two days.

Dr. Roger Ehrich and Mr. Shuhab Ahmed designed and wrote DMS, without which GENIE could not exist in its present form.

Mr. Richard Critz, the VAX systems manager for the Department of Computer Science, provided systems programming support for GENIE and went out of his way to give us access to the VAX system.

Mr. Robert Hansen, formerly of BDM Corporation, provided us with the documentation to a program called BLUEMAX, which crystallized our thinking on the movements of vehicles in three-space.

Other program development support has been provided by Dr. H. Rex Hartson, Ms. Deborah Johnson, Mr. Tamer Yunten, Ms. Christine Rieger, Mr. Markku Hakkinen, Dr. Robert C. Williges, Mrs. Beverly Williges, and Dr. Joel Greenstein.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The Generic ENvironment for Interactive Experiments is a complex of five programs designed to provide a test-bed for Human-Computer interface dialogue experiments. It is designed to approximate the movement of points in three-space and is therefore applicable to a number of situations encountered in everyday life. The intention of GENIE is to provide a program wherein the dialogue with the user can be structured so that the program appears to be different for different dialogues.

This document is structured into two general parts: one dealing with the use of GENIE on the DEC VAX 11/780 system and the other with the modification of GENIE for some specific study. It is important to note that the DMS often referred to in this document is just part of a software system designed for user-friendly software modification. This document is not designed to usurp the DMS facilities for software modification but to provide specific information about GENIE so that the use of DMS as a software modification tool will be eased.

For the purpose of GENIE to be fulfilled, an appropriate task had to be found. A brief introduction to Naval Aircraft Carrier Air Traffic Control (CATC) provided the idea for this task. The job of the Air Traffic Controller in the Navy is to guide aircraft returning from a mission to a safe landing on the deck of an Aircraft Carrier. There are many well-defined, specialized procedures for accomplishing this task. In fact, CATC is simply a special case for a much more general task: the control of vehicles or groups of vehicles in some area.

There are any number of real-world applications of the general task. An inexhaustive list consists of

1. air traffic control,
2. control of combat strikes on a ground target,
3. interception of unfriendly aircraft,
4. docking of satellites or other spacecraft with an orbiting space station or
5. in-flight refueling of aircraft.

If altitude is ignored, the problem reverts from three-space to two-space and an entirely new variety of applications presents itself. For example,

1. control of refueling operations of ships at sea,

2. control of combat troops in a battlefield operation,

3. control of movement of trains in a railyard,

4. control of trucks arriving at a warehouse,

5. control of distribution of goods in a warehouse or

6. control of automobiles on a city's streets.

GENIE can be made to appear to be any of these tasks simply by changing the interface of the computer with the human operator. In each of the instances cited above, GENIE would play the part of the vehicle driver, while the subject using GENIE would be the agency controlling the vehicles. By changing the display that GENIE presents to the user and the language and Input/Output (I/O) device(s) that the subject uses to communicate with GENIE, the experimenter can reproduce any of the scenarios mentioned above, or any other scenario whose basic premise is the control of some points moving in space.

Therefore, GENIE allows for experimentation into the realm of human-computer dialogues. Similar tasks can be presented to subjects using different dialogues. Differences in the subjects' performance can be measured, giving some quantitative basis for the rating of various schemes of human-computer dialogues.

1.1 TASK-SOFTWARE INTERFACE

When the general task of vehicular control is performed, there are four "actions" happening at the same time:

1. the vehicles are moving,

2. the controller is observing the vehicles' relative positions,

3. the controller is communicating with the vehicles and

4. time is passing.

Using the facility of Process Concurrency provided by the Dialogue Management System (DMS) [EHRI82], GENIE can do these four things at the same time. In addition, GENIE also allows the experimenter to control the sequence of events in time, therefore providing a fifth concurrent process.

When these five processes are running, GENIE provides an accurate movement of vehicles in time, a facility for the

controller to give instructions to the vehicles and a facility for the experimenter to control what events happen. GENIE also provides the display of the vehicles.

In its initial inception, the display that GENIE presents to the world looks something like that in Figure 1.

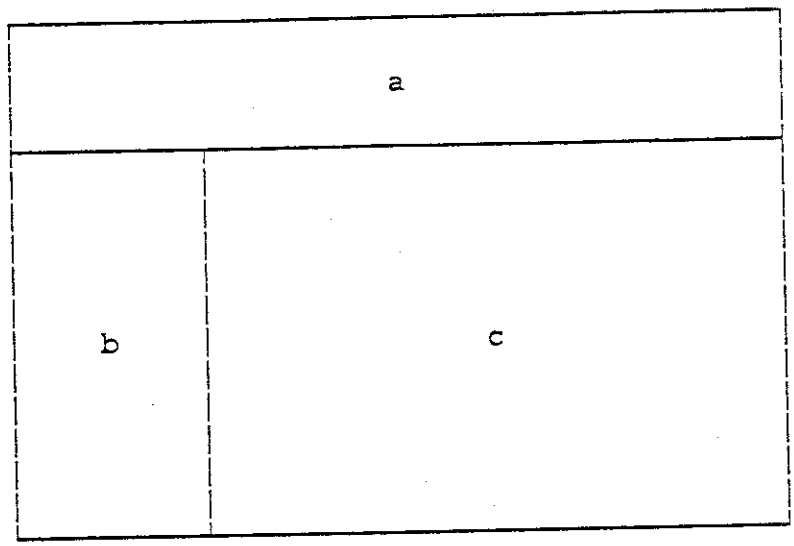

Figure 1: GENIE's Display

In Figure 1, the area represented by (a) provides information about the destination of the vehicles, (b) provides information about one particular vehicle selected by the controller and (c) provides a plan view of the relative positions of all vehicles under consideration. Of course, since the experimenter would be using GENIE to experiment with various forms of dialogue, he or she would probably want to define the screen to be used with a given experiment. This would, of course, require the partial re-programming of GENIE. DMS will, eventually, provide tools to aid in this re-programming. At the present time, appropriate changes must be made to the process controlling the display.

Subject commands are entered and echoed on a separate terminal, called the subject's console. Response to subject queries are typed on that console.

The language currently used by the subject and/or the experimenter looks very much like Air Traffic Control jargon.

This is because GENIE's genesis was in Air Traffic Control. It is also anticipated that the experimenter will want to manipulate the language. It would, therefore, be necessary to partially re-program the controller's language processor.

2. DESCRIPTION OF INPUT LANGUAGES

Any program input stream which is interpreted as a language must undergo three analysis phases:

1. Lexical analysis
2. syntactic analysis and
3. semantic analysis.

The lexical analysis of the input languages in GENIE is handled by the function LLSCAN in each processor, as explained later. The syntactic analysis is handled by the DEC facility LLPARS. The semantic analysis is handled in the GNEDBAS process.

2.1 LLPARS

LLPARS is a DEC/TR-90 software facility which produces syntax directed parsers. The facility consists of a compiler-compiler and a driver routine. The compiler-compiler accepts as input an LL(1) grammar specification. This grammar describes the syntax of the commands and supported command options and also names routines to be called as various commands and command fragments are recognized. These routines are called semantic action routines.

The compiler-compiler produces a set of tables based on the grammar specifications. When linked with these tables, the driver routine, called LLDRV, becomes a parser for the specified command language. The input stream, containing the command to be parsed, is accessed by the parser through a call to a user provided scan function, called LLSCAN. This function is called by LLDRV each time the parser needs a new token.

Once a command has been parsed, a semantic action routine is called to assemble the pertinent command details into a parameter list. This list is passed to a routine in the GNEDBAS process where the command is implemented through appropriate modifications to the database.

Thus, with the employment of the LLPARS facility, only the following components need be provided in order to implement each of the parsing processes.

1. A BNF-like specification of the command language,

2. A main routine to initiate the parsing operation by calling LLDRV,

3. An input scan function named LLSCAN and

4. The semantic action routines specified by the command grammar.

Each of these components is discussed in the following sections. More detailed information on the LLPARS facility may be found in [MORS79].

2.2 COMMAND LANGUAGE SPECIFICATION

2.2.1 Introduction

The GNESUBJ language and the GNEEXPR language are described by separate BNF-like grammar specifications. These specifications (which define LL(1) type grammars) are reproduced in the GENIE source code. A more thorough treatment of LL(1) grammars and the use of BNF (Backus-Naur Form) for the description of languages is provided in [MCKE70]

As noted earlier, the LLPARS compiler-compiler receives these grammars as input and produces a set of tables which is used to drive the parsing operation. Therefore, each BNF-like specification may be viewed as describing not only the command language syntax, but the entire operation of the GNESUBJ or GNEEXPR process. Each language specification consists of three major sections which are described below.

2.2.2 Terminal Definition Section

The first major section of a language specification is called the Terminal Definition Section. This section as its name indicates, defines the terminal tokens used in the grammar. Both the GNESUBJ language and the GNEEXPR language define error, end-of-line (EOL), and numeric digit tokens. The GNEEXPR language also defines a character string token. These terminals have no semantic values, but are the names used to represent specific token values returned by the scanner, LLSCAN.

For example, in both language specifications, the digit token is defined as having the value three. Thus, when an LLSCAN function encounters a numeric digit in the command input stream, the function returns the value three. Similarly, if an undefined character is encountered by LLSCAN, a value of zero is returned, thereby indicating that an error exists in the command.

2.2.3 Keyword Definition Section

The second section of a language specification is the Keyword Definition Section. Here, all keywords for a control language are specified. For example, the GNESUBJ language includes the words "ALTITUDE", "CLIMB", "DESCENT", "SPEED", and "TURN", for directing aircraft through the marshalling operation. Similarly, the GNEEXPR language employs the words, "BEGIN", "CANCEL", "CREATE", and "EMERGENCY", for establishing the air traffic control scenario for a given experiment. All keywords used in GNESUBJ and GNEEXPR languages are contained in the Keyword Definition Sections of the language specifications reproduced in the GENIE source code.

Like a terminal token, each keyword is associated with an integer value. This value is returned by LLSCAN when the associated keyword is encountered int he input stream. For example, in parsing the GNESUBJ command:

100 SAY ALTITUDE

LLSCAN first returns the value 3 which is the value assigned to a digit. It will return this value until the whole call sign has been recognized. LLSCAN then returns the values 946 and 900 on the two subsequent calls since these are the values assigned to the keywords "SAY" and "ALTITUDE" respectively. On the next call, LLSCAN returns the value two, indicating the end of the command by specification of the associated end-of-line termnal token.

2.2.4 Production Section

The third section of a language specification is the Production Section. This section is the heart of the grammar specification as it contains the list of productions (or rules) which define the syntax of the command language being implemented. These productions also specify the semantic action routines to be called as various command fragments are recognized. The productions appear in a BNF-like form with each production being defined as an order-dependent collection of keywords, terminals and non-terminals. Each non-

terminal then is defined in another production which also may contain non-terminals. This production hierarchy continues until the non-terminals are defined in terms of keywords and terminals only.

For example, the following production is the first (highest level) production for the GNESUBJ language.

```
AIR-CONTROLLER    = EOL {PROMPT} COMMAND AIR-CONTROLLER
                  |  ERROR {ERRMSG} EOL AIR- CONTROLLER
```

This rule dictates that on its first call from LLDRV, LLSCAN should return the end-of-line token (EOL). In other words, initially there is no command to be parsed. This condition triggers a call to the semantic action routine, PROMPT, to prompt the subject for a command. The word "COMMAND" in this production is a non-terminal defined in subsequent productions and describes the various GNESUBJ commands. Thus, the command received by the PROMPT routine from the subject should be one of the commands defined by the non-terminal "COMMAND".

The last component on this line of the production is the non-terminal "AIR-CONTROLLER". Since this is also the non-terminal being defined, the operation is recursive. In fact, since it is also the highest level non-terminal for the grammar, the entire parsing operation is recursive. When one command has been parsed, the production is repeated: the EOL token is returned by LLSCAN, PROMPT is called to get another command from the user, and the new command is parsed. Thus, once initiated, the parser operation continues indefinitely, and may be stopped only by some external means. In the control modules, parsing is stopped by execution of special HALT instruction.

The second line of the AIR-CONTROLLER production is called an error recovery production. This production is applied whenever an error is discovered in the command being parsed. When a token value returned by LLSCAN does not match the token value type expected, as defined by the production being applied, an error has occurred. At this point, the current production is abandoned and the error recovery production is applied. ERRMSG is called to print a message to the subject and the rest of the command is ignored. Normal processing resumes with the return of the EOL token by LLSCAN and the recursive application of the ASR-CONTROLLER production. A more detailed description of the LLPARS error recover process is provided in [Morse 1979].

The highest level production of the GNEEXPR grammar appears as follows:

```
EXPERIMENTER_CMD    = EOL {READIN} COMMAND EXPERIMENTER_CMD
                    | ERROR {ERRMSG(0)} EOL EXPERIMENTER_CMD
```

From examination of this production and the remaining productions of the grammar, it should be clear that operation of the GNEEXPR process is analogous to that of the GNESUBJ. Process. The semantics of the languages differ, but the parsing operations performed on the commands essentially are identical. Hence, the description of other aspects of the processes will be made with respect to the GNESUBJ Process only. However, implementation differences in the GNEEXPR Process will be highlighted.

3. EXECUTION LOGIC OF GENIE

3.1 DATA ORGANIZATION

With the exception of local variables within procedures, all of the data manipulated by GENIE is contained in a binary tree ordered on the identification number of each vehicle or in arrays transmitted between processes by DMS using VAX mailboxes. For a complete description of DMS, the user is referred to the work of Roger Ehrich [EHRI82]. In the current implementation, this ID number is an aircraft call sign. An in-order traversal of this tree would list the call signs in ascending numerical order. This organization allows fast access on any vehicle, since the vehicle call sign is specified in any command dealing with a particular vehicle. The algorithms which manipulate this tree are found in the module "gnetree" located in the library [fainter.genie2]genielib.tlb and are taken from Knuth [KNUT73].

The VAX mailboxes and DMS are designed to transmit byte strings. However, it would be most inconvenient for the user's software to have to convert program data to byte strings. Therefore, great care was taken to make sure that the variable declarations (in PASCAL and FORTRAN) agreed EXACTLY in the transmitting and receiving processes. In this way, the receiving process can "reassemble" the data into the same form in which it was sent based upon information available at compile time. GENIE modifiers are warned, cautioned and otherwise injoined to maintain the same care in data organization. This is so critical because the processes do not share an execution environment so data passed incorrectly cannot in any way be spotted by the VAX system. The receiving process can only assume that the data is correct. If it is not, very unpredictable results follow.

Some of the most intransigent errors that occurred during the development of GENIE were caused by small oversights in data transmission. In short, be careful.

Each node of the tree has three fields: left and right pointers and vehicle information. The vehicle information field contains such things as the vehicle's polar coordinates with respect to the destination, the vehicle's x and y position on the screen, the vehicle's speed and acceleration in each dimension and various other data. See the source code type declarations for a complete description.

The tree is created when the GNEDBAS process is created. The tree's root node and the root node of the left subtree (which is the only node of the left subtree) are created by GNEDBAS. Creation of the tree causes the pointer variable "p_tree" to contain the address of the head node of the tree.

The call sign associated with this node is -1. This signals two things to the display process, GNEDISP. Firstly, the information contained in this node can be used to control the format of the display and secondly, the nodes soon to be transmitted are to be interpreted as vehicular nodes and handled accordingly. The information contained in this node is important and, due to the use to which this information is put, the names of the fields of the tree are not always mnemonic to the actual data.

Another node of the tree is also created at initialization time. This is the so-called "tail" node.

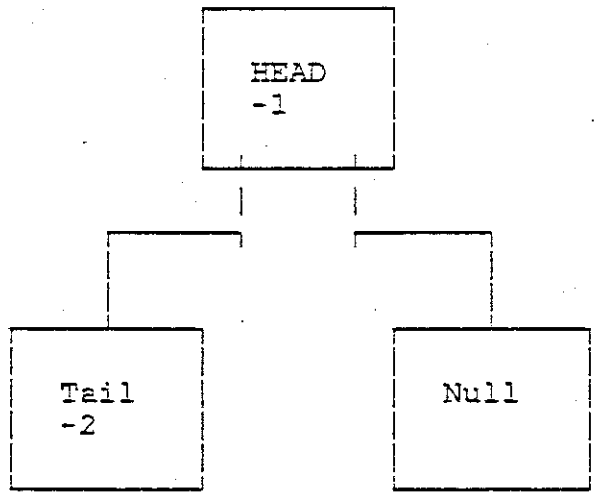

Figure 2: The Initial Tree Structure

It has a call sign of -2 and when it is received by the display, the display knows that no further vehicular information will be sent for the moment. At the present time, no other information in this node is used. In order to maintain the "search tree" structure of this tree, the tail node is placed in the left subtree of the head node. In fact, it is the only node that will ever be in the left subtree. Figure 2 is a diagram of the initial state of the vehicle tree.

GENIE has the capability of allowing five different types of vehicles to be used at any one time. Initial information on each type of vehicle is stored in the VAX file [fainter.genie2]gnebook.dat which is assigned the logical name "bookin". This file is read by the procedure "initialize" in the module GNESEMA. If the GENIE user desires to change the characteristics of the vehicles, this is the file to change. The first five lines of the file contain the so-called "book" information on each vehicle. The next six lines contain probabilities for the various emergencies. This information could be used if the experimenter wanted to assign emergencies to vehicles on a pseudo-random basis as the vehicle is created. (Since the seed of the random number generator is not changed, the same "random" pattern of emergencies would occur each time GENIE were run.) The last line contains information used as default for parameters that can be changed with the experimenter's DEFINE command.

3.2 ORGANIZATION OF THE PROGRAM COMPLEX

As mentioned above, GENIE is organized into five processes. They are named as follows:

1. GNESUBJ
2. GNEEXPR
3. GNETIME
4. GNEDBAS
5. GNEDISP

GNESUBJ is a DEC VAX 11/780 PASCAL program which, using DMS, creates GNEEXPR and then invokes a FORTRAN subroutine called GNELAN2, which recognizes the subject's command language. GNEEXPR is a PASCAL program whose sole purpose is to invoke the FORTRAN subroutine GNELAN1, which recognizes the experimenter's command language. GNELAN1 and GNELAN2 use the VAX system service LLPARS to analyze the commands input by the experimenter or the subject. Execution of the first

"define" command in the experiment profile (see below) will cause GNEDBAS to be created. Execution of the "begin" command causes GNETIME to be created. If there are no "define" commands in a profile, then GNETIME will create GNEDBAS when the "begin" is executed. Once GNETIME is created, it will, once every three seconds thereafter, request that GNEDBAS produce new locations for each vehicle currently extant. The new location produced by GNEDBAS will be based on the 3-second elapsed time and on the speed of the vehicles. As each new location is determined, GNEDBAS requests that GNEDISP display a blip in the calculated position. (GNEDISP as a process is created by GNEDBAS when the first request is made.)

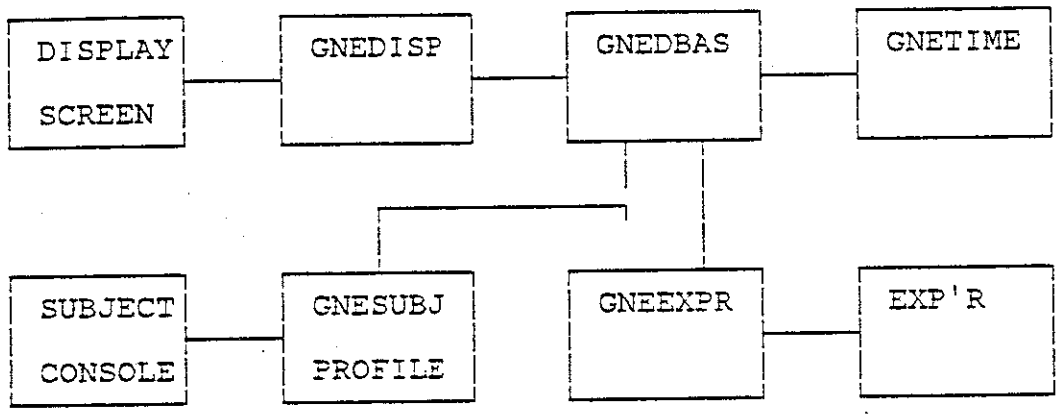

Figure 3: Steady State Macro Structure of GENIE

Figure 3 depicts the steady state (after all process creation has occurred) of GENIE and the interprocess communications links.

GNELAN1 and GNELAN2, after recognizing and analyzing each experimenter's or subject's command, respectively, requests that GNEDBAS make the appropriate change in the database of vehicles. All five of these processes will run until the experimenter issues the HALT command; at that time, GNESUBJ will terminate. DMS will then insure that all other processes will also be terminated. Since the processes GNEDISP and GNESUBJ are the processes that directly interface with the controller, it is suggested that these are the processes most likely to undergo changes by the experimenter.

3.3 ORGANIZATION OF EACH PROCESS

3.3.1 The Language Processors

Even though the GNESUBJ and GNEEXPR processes are separate, they will be considered together here because they differ only in the language that they recognize. A few prefatory remarks will deal with the differences, and the rest of the language processor discussion will apply to both processes.

3.3.2 GNESUBJ

This is the process which initiates GENIE. It creates the experimenter process and then calls the FORTRAN subroutine GNELAN2. GNELAN2 recognizes commands from the subject and, after verifying that the command is correct, sends a 10-element integer array to GNEDBAS indicating the parameters of the command. GNELAN2 also sends a 2-character string to GNEDBAS, indicating which semantic procedure in GNEDBAS is to process the 10-element array.

3.3.3 GNEEXPR

This is the process that would be changed if the experimenter wanted to add new experimenter commands. For information on making changes to this process, see the report written by Sue R. Guy [GUY82]. GNELAN1 is a FORTRAN program which recognizes the experimenter's commands. It is straightforward and well documented except for the call to the sub- routine LLDRV. LLDRV is a system service for using LLPARS. LLDRV will call the appropriate subroutine provided in the remainder of GNELAN1. For a complete description of LLPARS and LLDRV, the reader is referred to the documentation for GNELAN1 and the system documentation for LLPARS.

After GNELAN1 recognizes a command, it sends a 2-character string to GNEDBAS indicating the meaning of the command and it also sends a 10-element array containing the parameters of the command. GNEDBAS then effects the appropriate change in the database.

The structure of a GENIE "program" is given in the Experiment Profile section. If the GENIE "program" contains "define" commands, execution of the first of these will cause the creation of the GNEDBAS process. In fact, the process will be created and will start execution before the action specified in the "define" command takes place. Part of the initial execution of GNEDBAS is the reading of the "bookin" file, part of which contains defaults for those variables

which the "define" command can change. After "bookin" is read, the first "define" takes effect. If there are no "define" commands, the "begin" command causes the creation of the GNEDBAS and, since "define" commands are not permitted after the "begin", the defaults read from "bookin" are never changed.

3.3.4 GNESUBJ Process and GNEEXPR Process MAIN Routines

The function of the GNESUBJ and GNEEXPR processes MAIN routines is to initiate the parsing operation by calling the parsing driver, LLDRV. Each MAIN routine also serves to define and initialize common variables used in the parsing operation by the scanner, LLSCAN, and the semantic action routines.

The MAIN routine defines the command input stream as an 80-character string. This string is set to blanks, indicating to LLSCAN that initially, there is no command to be parsed.

MAIN also defines a keyword array used as temporary storage for the keywords recognized during the parsing of a command. Once a comand has been parsed, the semantic action routine for the command may examine the keyword array contents to determine various command options specified by the subject. For example, the GNESUBJ ALTITUDE_CHANGE command consists of a call sign followed by the keywords "CLIMB AND MAINTAIN" or "DESCEND AND MAINTAIN" followed by an altitude. After command parsing, the first entry in the keyword array contains the keyword "CLIMB" or the keyword "DESCEND". Thus, by examining this entry, the semantic action routine can determine which command option was entered by the subject.

Similarly, the MAIN routine defines an integer parameter array for temporary storage of integer values recognized during command parsing. Using the example of the GNESUBJ ALTITUDE_CHANGE command, the integer parameter array contains, after command parsing, the call sign as its first entry, and the altitude as its second entry. The MAIN routine also defines an array for temporary storage of integer parameter digits. This array is called the symbols array. During parsing, individual digits are stored in the symbols array until it is determined by the grammar that the entire integer parameter has been parsed. At this point, a semantic action routine, NUMERIC, is called to remove the digits from this array and convert them to a single integer value. This value then is stored in the integer parameter array.

For each array described above, a separate pointer is needed to mark the last array position filled, or to indicate the number of entries made during parsing of a command. These pointers also are defined in the MAIN routine and are set to zero.

In addition to the arrays and pointers described above, the MAIN routine for the GNEEXPR Process defines a weather flag and a 30-character string for temporary storage of weather information. The weather flag, set by the semantic action routine, SETFLAG, indicates to LLSCAN that the next string of characters constitutes weather information and should be stored in the weather string. This facility may also be used for speech production. The experimenter specifies weather conditions for an experiment in the GNEEXPR START_SIMULATION command. Weather information does not follow any fixed format and hence, cannot be described in a grammar production. Therefore, special treatment is necessary for recognition and storage of weather information.

Specific variable names and declarations for the processes' common variables can be found in the MAIN routines' code.

3.3.5 The Scanner Function, LLSCAN

LLSCAN is the FORTRAN scan function called by the LLPARS driver, LLDRV. The function analyzes the first non-blank character in the command input stream and returns an integer code, indicating the type of symbol found. The symbol is removed from the command line and placed in one of the temporary storage arrays. The operation of LLSCAN is described in greater detail in the following paragraphs.

First, LLSCAN determines if the command line is blank. If this is the case, LLSCAN returns the value 2, the value associated with the end-of-line terminal token (EOL). If the command is not blank, LLSCAN finds the first non-blank character by removing any leading blanks from the command line. LLSCAN then determines if the character is a numeric digit (a character in the range '0' through '9'). If the character is a digit, LLSCAN removes it from the command line and stores it in the first available space in the symbols array. LLSCAN returns the value 3 which is the value associated with the numeric digit terminal token. If the symbol is not a digit, LLSCAN checks whether the symbol is the asterisk ('*'). If it is, LLSCAN stores the asterisk in the symbols array and returns the value 42. (The asterisk was assigned this value by the compiler-compiler.) If the character is neither a digit nor an asterisk, it is assumed to be the first letter of a keyword. LLSCAN removes this

letter, and any following letters, from the command stream and builds a keyword. LLSCAN passes this keyword to a support function, KEYWRD, and returns the value returned by this function. If the keyword is one supported by the language, KEYWRD stores the keyword in the keyword array and returns the unique code associated with it. This code is the integer assigned to the keyword in the Keyword Definition Section of the language specification. If the keyword is not supported by the language, KEYWRD simply returns the value zero which is the value associated with the error terminal token.

The LLSCAN function for the GNEEXPR Process performs essentially the same operations as the LLSCAN function just described. The GNEEXPR LLSCAN function does not test for the occurrence of an asterisk since the asterisk is not used in any GNEEXPR command. However, the GNEEXPR language does include tokens not defined in the GNESUBJ language that must be handled by the GNEEXPR LLSCAN. Treatment of these additional tokens is described below.

After removing any leading blanks from the command line, the GNEEXPR LLSCAN function checks the weather flag. If it is set (value of true), the next string of characters constitutes weather information. As weather information is followed by the keyword "DECK" in the START_SIMULATION command, LLSCAN removes all characters preceeding the keyword "DECK" and stores the characters in the weather string described in Section 3.3. LLSCAN then resets the weather flag (value set to false). After performing the check for a digit, the GNEEXPR LLSCAN function determines whether the character is a comma (","). If it is, the comma is removed from the command line, and the value 44 is returned. (This value was assigned to the comma symbol by the compiler-compiler.) Similarly, the GNEEXPR LLSCAN function checks if the symbol is a hyphen ("-"). If the symbol is a hyphen, LLSCAN removes it from the command line and returns the associated value, 45.

3.3.6 Semantic Action Routines

3.3.6.1 Overview

Semantic action routines are the routines named in the productions of the language specifications. These routines are called by LLDRV as various commands or command fragments are parsed. Based on the types of operations performed, these semantic action routines form two distinct groups of routines: the general group of semantic action routines, and the command-specific group. These routine groups are described in greater detail below.

3.3.6.2 General Semantic Action Routines

The general group of semantic action routines are used in both the GNESUBJ and the GNEEXPR processes and aid in the general parsing of commands. The routines ERRMSG, NUMERIC, PROMPT, and HALT are the routines which form this group Each of these routines has been mentioned previously in this paper but is described further in the following paragraphs.

ERRMSG is called by LLDRV during application of the error recovery production. The routine also is called by several of the command-specific semantic action routines upon detection of certain semantic errors. ERRMSG displays an error message to the subject (or the experimenter) and sets the command line to blanks.

The NUMERIC routine is called when all digits of an integer value have been parsed. NUMERIC removes the digits, stored in character form in the symbols array, and converts them to a single integer value. NUMERIC then stores this value in the integer parameter array. During command parsing, if a non-digit symbol is encountered where a digit is expected, the production is abandoned and the error recovery production is applied. Hence, NUMERIC is assured of the correctness of the values stored in the symbols array at the time of routine invocation. As a result, NUMERIC does not perform type checking on the symbols array contents.

The PROMPT routine (named READIN in the GNEEXPR Process) reads in the next command to be parsed. The routine also prepares the system for analysis of the command by resetting the keyword and integer parameter array pointers to zero. The PROMPT routine is called when parsing of a command has been completed or abandoned due to the presence of errors in the command line.

The GNEEXPR HALT semantic action routine is called when an GNEEXPR HALT command has been parsed. The GNEEXPR HALT routine terminates execution of the GNEEXPR Process by executing a FORTRAN STOP command. However, prior to executing the FORTRAN STOP, the GNEEXPR HALT signals the end of the interactive session by sending a message to the Data Base Position Update Process. This Process then may terminate after notifying the Screen Control and Timing Process that the session has ended. As the GNESUBJ Process receives input only from the subject's terminal, this process must be stopped independently by issuing the GNESUBJ HALT command. Upon completing the parsing of the HALT command, the GNESUBJ process invokes the GNESUBJ HALT semantic action routine which terminates the GNESUBJ Process by executing a FORTRAN STOP.

3.3.6.3 Command-Specific Semantic Action Routines

The command-specific routines are the semantic action routines called when a command has been parsed. Each command type is associated with a unique command-specific routine. The routines gather and code information stored in the keyword and integer parameter arrays. This information then is stored in an integer array that is passed to a corresponding routine in the Data Base Position Update Process. The corresponding routine to be invoked is indicated in the message by 2-character strings preceeding the parameter array. The character strings and the associated Data Base Position Update routines are summarized in Table 1. The procedures are located in the GNESEMA module.

Many of these command-specific routines perform limited semantic checks prior to passing the information to the corresponding routine in the Data Base Position Update Process. Routines corresponding to commands which specify call signs invoke a routine, CHCKSIN, to validate that the call sign is in the accepted range of values. Maximum values also exist for integers describing concepts such as distances and altitudes. Therefore, semantic action routines corresponding to commands involving these concepts also perform value range checks. If a value lies outside the accepted range, the semantic action routine calls ERRMSG to print an error message. At this point, processing of the command is terminated.

A summary of the functions of the command-specific semantic action routines is presented in the tables below. More detailed information on the routines is included in the routines' internal documentation .

TABLE 1

Data Base Position Update Process Subroutine Codes

Subroutine Name	Code	Activating Process
alt_change	gg	GNESUBJ
hand_off	hh	GNESUBJ
info_req	ii	GNESUBJ
marshal_req	jj	GNESUBJ
blip_tag	kk	GNESUBJ
range	ll	GNESUBJ
ship_pos	mm	GNESUBJ
spd_chg	ss	GNESUBJ
stat_area	nn	GNESUBJ
vctr	oo	GNESUBJ
wave_off	qq	GNESUBJ
ship_head	aa	GNEEXPR
color_p	tt	GNEEXPR
create_flight	bb	GNEEXPR
definition	rr	GNEEXPR
emerg	cc	GNEEXPR
halter	dd	GNEEXPR
spd_chg	ss	GNEEXPR
commence	ff	GNEEXPR

TABLE 2

GNESUBJ Process Command-Specific Semantic Action Routine Summary

Routine Name	Command Name	Parameters	R	Database Update Routine
ALTCHIN	ALTITUDE_CHANGE	call sign, vertical direction, altitude	C E	alt_change
HANDIN	HAND_OFF	call sign, controller code, button	C E	hand_off
INFOIN	INFORMATION_REQUEST	call sign, info request code	C E	info_req
MRSHLIN	MARSHAL_INSTRUCTION	call sign, fix type, radial, distance, altitude, eat	C E	marshal_req
RARDCMD	RADAR_DISPLAY	display code, plane code		blip_tag
RANGECMD	RANGE_COMMAND	range		range
SHIPCMD	SHIP_COMMAND	ship direction code	C	ship_pos
STATCMD	STAT_COMMAND	call sign	C	stat_area
VCTRIN	VECTOR	call sign, direction, radial	C E	vctr
WAVEIN	WAVEOFF	call sign	C E	wave_off

where R = support routines called,
C = CHCKSIN, and
E = ERRMSG

TABLE 3

GNEEXPR Process Command-Specific Semantic Action Routine Summary

Routine Name	Command Name	Parameters	R	Database Update Routine
CHANGEH	CHANGE_HEADING	direction, radial,	E	ship_head
COLR	color	call sign, color	E	color_p
CREATEP	CREATE_PLANES	plane type, number of planes, radial, distance, altitude	C E	create_flight
DFN	definition	various	E	definition
EMER	Emergencies	emergency action, call sign, emergency type	C E	emerg
SPEED_C	speed	call sign, speed	E	spd_chg
STRTSIM	START_SIMULATION	deck condition, radial, weather	E	commence

where R = support routines called,
C = CHCKSIN, and
E = ERRMSG

3.3.7 GNETIME

The purpose of GNETIME is to "kick" GNEDBAS every so many seconds (specified by the experimenter) so that GNEDBAS will produce a new configuration of the vehicles. It does this by calling the DMS service "long_delay" with the number of seconds between traversals. Upon returning from "long_delay", GNETIME will request that GNEDBAS perform the update operation. The 'pp' sent to GNEDBAS is the cue for this action. GNETIME then waits for a message from GNEDBAS indicating that the update has occurred before returning to hibernation. This process is repeated as long as GNETIME is active.

3.3.8 GNEDBAS

This is actually the heart of GENIE. It creates and maintains the binary tree structure in which information about each vehicle is stored. It performs a traversal of this tree upon prodding by GNETIME. During the traversal, it produces the new position of each vehicle based upon the laws of particle motion in Newtonian physics. As each new position is computed, that vehicle is displayed by GNEDISP.

Additions to the tree are effected by the procedures "create_flight", "ctcecrtp" and "tree_insert" on the behest of the GNEEXPR process. Since valid call signs are non-negative integers of three digits, each digit being between 0 and 7 inclusive, a search tree order is maintained if vehicle nodes are inserted into the right subtree. The "tree_insert" procedure does this insertion. The procedure "create_flight" allows the creation of up to four vehicles in a group at a time. The procedure "ctcecrtp" acquires storage for the new node and assigns values to each of its fields. The procedure "tree_insert" places this newly created node in its correct place in the tree.

Nodes are automatically deleted when the vehicle that the represent gets within some specified distance of the destination and is on a final approach to that destination. The final approach is a radial line from the destination along which the vehicles travel in order to arrive at the destination. The implication is that vehicles must arrive from one compass direction. A vehicle is considered to be "on final" if its radial from the destination is within ten degrees of the final approach radial. Currently, that distance is one-half of a nautical mile. The procedure "tree_delete" does the actual deletion.

The interactions of deciding to delete a node are complex and warrant some discussion here. In the "change" procedure

of the GNEMOVE module, each vehicle's distance from the destination is checked each time a new position is computed. If the distance is less than or is equal to one_fourth mile and the vehicle is within 10 degrees of the final approach course, a boolean variable called "has_landed" is assigned the value "true". For the moment, that is all that happens. This node is returned to "traverse" in GNEDBAS, which immediately sends it to GNEDISP. GNEDISP examines the boolean variable and, if it is true, GNEDISP erases the node from the screen (if the boolean were false, it would still be erased, but it would also be redrawn) and deletes the call sign from the active vehicle list. Upon return from GNEDISP to "traverse", the boolean is examined again and, if it is set, "traverse" invokes "tree_delete" to get rid of it. This complication is necessary because one procedure (change) makes a decision which affects the operation of two entire processes and these processes need information and time to adjust to the deletion of a node. (Careful inspection of the source code will reveal that the boolean variable "new_plane", which is declared at the same time as "has_landed", serves a similar purpose. GNEDISP must know if a node in the tree is new, so that the call sign can be inserted in the active aircraft list. So, "ctcecrtp" initializes "new_plane" to true and, after the node has been sent to GNEDISP for the first time, "traverse" sets "new_plane" to false.)

The above discussion covers the creation and deletion of vehicle nodes. During the life of a node, it undergoes updating once during each traversal of the tree except for the head node, which never is updated. One traversal occurs each time the timer process, GNETIME, signals the data base process. During the traversal, the node is sent to the "change" procedure. (See the module GNEMOVE.) This procedure calls several other procedures which actually modify the node.

Procedure "new_pos" computes the new (x,y) position on the screen and the new radial position with respect to the destination. Close inspection of the code which performs the calculations for this procedure could raise questions as to its validity.

Each aircraft's position is actually computed on two sets of coordinates: the origin of the (x,y) plane is the lower left corner of the display portion of the screen, while the origin of the polar plane is the destination. See Figure 4. If this seems confusing, consider the following: the standard mathematical polar coordinate system has its initial side as the x axis and angles increase into the first quadrant, effectively to the left. The polar coordinate system used in GENIE (Figure 5) has its origin at the destination and the initial side is parallel to the y axis with angles

increasing TO THE RIGHT. The origin, O, of the polar system is always placed at the (x,y) location of the destination in the Cartesian system. New coordinates for each aircraft must be computed in each plane.

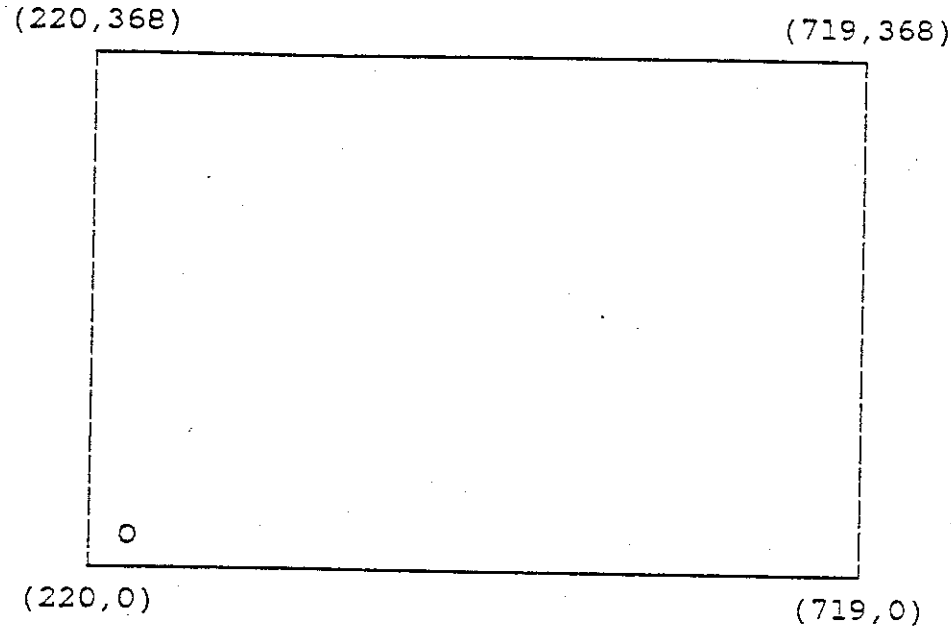

Figure 4: The Cartesian Plane Used by GENIE

Since the polar axes have been rotated, a transformation had to be found to map the polar system into the Cartesian system. The standard x=r cos - and y=r sin - would not work. Since GENIE's x and y axes are simply reversed from that of the standard polar plane, the transformation was simply x=r sin - and y=r cos -. Therefore, "new_pos" uses this translation to compute the new coordinates of each aircraft.

After the new position of the vehicle has been computed, its angular motion is computed by the procedure "turning". Aircraft turn at a rate of three degrees per second until they are within 10 degrees of the assigned heading. In this procedure, the difference between the current vehicle heading and the assigned heading is computed. If this difference is greater than the amount that the vehicle will turn in the NEXT traversal, then compute the turn and assign the vehicle the new heading. Then, if the vehicle is within 10 degrees of the assigned heading, reduce the rate of turn by

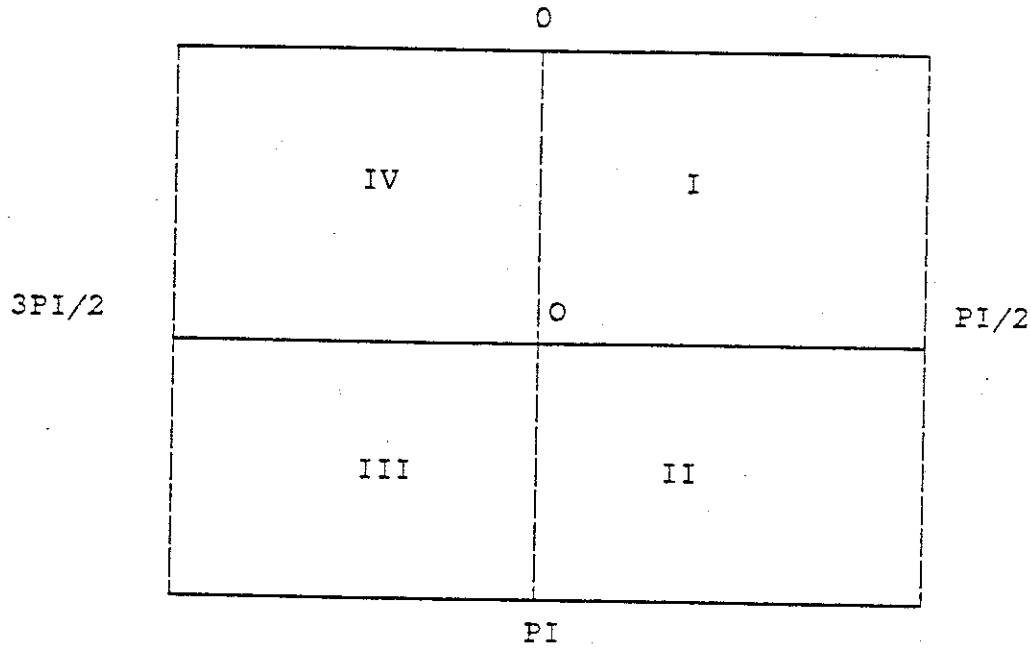

Figure 5: GENIE's Polar Coordinate System

one-half. If, however, the vehicle will overshoot its assigned heading on the next traversal, then simply make the current heading the same as the assigned heading and set the rate of turn to zero.

Next, "change" decides if the aircraft is in a position from which a landing may be made. (The aircraft must be "on final" and within 10 miles of the destination.) If this is the case, the "prepare_to_land" procedure lowers the landing gear, begins slowing the aircraft to approach speed and begins the aircraft's descent to the destination. This is called putting the aircraft in "landing configuration." If the aircraft is already in landing configuration, then the algorithm described above decides if the aircraft is ready to land. If, at any time after the aircraft begins its final approach to the destination, it departs from the final approach course, the "go_around" procedure is executed. This causes the aircraft to climb back to 5000 feet altitude and accelerate to cruise speed (after the climb, which is at the aircraft's climb speed.) Finally the new airspeed and altitude are computed based upon acceleration and vertical speed. Aircraft speed and altitude change at rates that approximate those of high performance jet aircraft. Climbs and descents are made at a rate of 4,000 feet per minute except if the aircraft is below 5,000 feet, a descent is made

at 2,000 feet per minute. Aircraft acceleration is 20 knots per second while deceleration is 30 knots per second.

After all of the preceding has been done to each node, the information in the node is sent to the display process and the traversal of the tree continues first down the right subtree and then down the left subtree.

At the end of each traversal, GNEDBAS is ready for another command. If the command is from one of the language processors (GNELAN1 or GNELAN2) the appropriate semantic procedure is called, which then makes a change in the binary tree. GNEDBAS loops eternally on the boolean variable "forever", waiting at the "accept" statement for a command.

The program GNEDBAS actually comprises a main program and two separately compiled modules. This organization precludes the necessity of recompiling all of GNEDBAS when a change is made in only one part. Any change, however, does necessitate re-linking all of GNEDBAS. The modules which must be linked to GNEDBAS are GNEMOVE and GNESEMA. GNEMOVE has the code to perform the update operations on the nodes of the tree. The procedure "change", which is called from the traversal procedure, is the last procedure in the module GNEMOVE. The module GNESEMA contains the semantic procedures for the experiment profile language and the subject's command language. These procedures determine the actual meaning of the various GENIE commands. The reader is referred to the GENIE source code for the documentation of these procedures. It is conceivable that this module might be a candidate for modification.

3.3.9 GNEDISP

This process draws the screen image and places blips representing each vehicle on the screen. It makes use of a slightly modified version of the CORE graphics system. Two modifications were made to CORE. The first allows for leading or trailing blanks in a text string. The second adds a parameter to the TEXT subroutine that allows the user to specify the length of the text string to be written. These modifications will be described later in this discussion. This process has been developed to run on the Digital Equipment Corporation's GIGI system. In its present form, it will not run on any other device.

At the present time, there are four discrete sections which comprise the the screen image. The display in Figure 1 has remained much the same since its original design. The difference is that the area defined by (a) has been divided into two areas. The leftmost area retains its original pur-

pose, providing information about the vehicles' destination. This rightmost area provides a list of the active vehicles. In this implementation, the call signs of the aircraft are displayed.

GNEDISP is a single PASCAL program which is comprised of six routines along with the body of the main. The names of the routines are

1. ACTIVE
2. CARRIER
3. PLANE
4. REFRESH
5. SCREEN
6. SHIPSTAT

ACTIVE maintains the list of the active vehicles. The structure of this list is a tree with each entry having several attributes as well as pointer to the left and right children of that node. The call sign of the vehicle, any emergency information and a color are kept. The color, at present, is used to indicate to the controller that the plane is a new entry onto the screen or that a plane has an emergency. If the plane is a new entry, the call sign is written in yellow. If the plane has an emergency situation, the call sign is drawn in red. Provisions have been made to color the call signs for those vehicles which are under control of the "computer" during task allocation experiments.

CARRIER is the routine which draws the carrier where the experimenter wants it to be placed. It may also draw a marshalling vector to which the planes must be guided by the controller so that they may land.

PLANE draws and/or erases the planes from the screen. Its function is determined by an action parameter passed to it. It may also write information about the plane next to it. This information may be any combination of the following: call sign, altitude, speed or squawk. The combination is chosen by the experimenter. At present, there are two shapes which the plane may take. The first is a filled box. This shape indicates that the plane is controlled by the subject controller. The plane may also appear as a filled circle. This represents that the plane is under control of the computer. At the present time, the software has not been completed to handle this allocation.

REFRESH is called when the controller requests information a particular vehicle. It will display this information in the section of Figure 1 represented by (b). This information includes the call sign, heading, fuel status, landing gear position, altitude, distance to the destination, bearing to the destination, estimated time of arrival at the destination, the communication frequency currently being used and remarks about the plane. The remarks, at present, are limited to emergency information and are written in red.

SCREEN draws the boxes which make up the screen. It also writes the heading information into all of the boxes.

SHIPSTAT updates the information on the destination status area. At present, the only information which changes is the time.

GNEDISP is the controller. It initializes several variables, draws the screen and then waits at an ACCEPT for a REQUEST from GNEDBAS. Once the request has been made,. a RECEIVE is encountered. As GNEDBAS traverses the tree, it sends information to GNEDISP. The information which is passed is a node from the tree. The first packet i of information that is received pertains to the destination, in this case, an aircraft. This information is used to draw the carrier at one of the five predetermined locations and also to fill in the destination status area. This is accomplished by calls to CARRIER and SHIPSTAT. The following packet(s) either vehicle information or is the node which signals the end of a traversal of the tree.

If the information refers to a vehicle, a check is made of its existence status. If the vehicle, plane, has landed, then the plane is erased from the screen by an invocation of the PLANE with the action of erase and its call sign is deleted from the list of active vehicles by a call to ACTIVE with the action of delete. If it is a new plane, the call sign is inserted into the active vehicle list and if the position of the plane is within the range of the screen, it is drawn. If the existence status shows that it has neither landed nor that it is a new plane, the plane is erase from its last location and then redrawn at its new position, provided that it is within the range of the screen. These actions are preformed by calls to PLANE and ACTIVE with the appropriate actions. If the controller has requested information about this plane, a call is made to REFRESH to update the AIRCRAFT STATUS area.

If the packet signals the end of a traversal, the program returns to the ACCEPT to await the next traversal. This looping is never stopped by any actions of GNEDISP. The program is forced to exit after the experimenter's profile executes a HALT. Due to this forced exit, the GIGI is left

in a state where neither of its cursors are turned on. An exit handler is on the way to take care of this problem. For the time being, the experimenter can manually reset the GIGI by entering set-up mode and changing the VC parameter to VC3.

4. USING GENIE

GENIE is designed to run under the control of DMS. There are two reasons for this:

1. GENIE is the testbed for DMS as a software development tool.

2. DMS is the only available software that provides for concurrent execution of several program parts on the DEC VAX 11/780.

Running GENIE is a three-step process.

1. Create the experiment profile.

2. Make logical name table assignments.

3. Run the image.

Each of these steps is outlined below.

4.1 STEPS IN RUNNING GENIE

4.1.1 The Experiment Profile

The experimenter controls the actions that take place during the experiment via the experimenter's command language. In effect, the experimenter writes a GENIE "program", which specifies what actions are to take place and the time for the execution of these actions. NOTE: In the following description of commands, the command keywords are in upper case letters. This is for emphasis only. These commands should be in lower case letters in the experiment profile. At this time, GENIE's language processors do not recognize upper case letters. A future version of GENIE will be more friendly in this regard, but time constraints have prevented that from happening yet.

The experimenter has seven commands at his/her disposal: DEFINE, BEGIN, CREATE, EMERGENCY, SPEED, COLOR and HALT. These will be dealt with one at a time.

The DEFINE command is used to establish the initial configuration of GENIE for a given trial. Each DEFINE command may have one of five parameters. Several DEFINE commands may be used. The operands are

1. SHIP LOCATION

2. REFRESH RATE

3. SCREEN DIMENSION

4. TAG

5. APPROACH HEADING

The SHIP LOCATION defines where in the radar display area the ship symbol will be located. The words SHIP LOCATION must be followed by one of the following: C, for the center of the display area; UL, for the center of the Upper Left quadrant; LL, for the center of the Lower Left quadrant; UR, for the center of the Upper Right; and LR, for the center of the Lower Right. These are the only possible locations at the present time.

The words REFRESH RATE should be followed by a digit between 1 and 9. This will be the number of seconds that elapse between each traversal of the tree and subsequent drawing of the screen. Rates of less than 3 may produce unpredictable results when there is a large number of vehicles to be displayed because traversals may occur before the results of the previous traversal has been transmitted over the lines.

A two or three digit number follows the words SCREEN DIMENSION. This number defines the distance from the left edge of the radar display to the right edge in nautical miles.

The TAG option of the DEFINE command defines what information is to be displayed in the text tag beside each blip on the screen. After the word TAG comes one or more of the words CALL SIGN, AIRSPEED, ALTITUDE. These words may be specified in any order. If more than one is used, they must be separated by commas.

The APPROACH HEADING option defines the final approach bearing that the aircraft must fly to land. The words are followed by a 3-digit integer in the range of 0 to 360, specifying the direction.

The BEGIN command has no operands. It specifies the point at which GENIE is to draw the screen and begin processing the action commands. The BEGIN must follow ALL of

the definition commands and must precede ALL of the action commands. (The action commands are CREATE, EMERGENCY, SPEED, COLOR and HALT.)

The CREATE command is an instruction to produce a new aircraft for display. The experimenter specifies, in the following order, after the word CREATE, the type of aircraft (XTF-1, XTF-2, XTF-3, XTB-1, RCU-1), the number of aircraft in the flight (1-4), the call signs of the aircraft (if more than one in the flight, they must be separated by commas), the location relative to the destination (a radial FROM the destination in the range of 0-360 and a distance FROM the destination, in the range 0-300), and the altitude, specified by the keyword ANGELS followed by the altitude in thousands of feet. See Appendix A for an example of a CREATE command.

The EMERGENCY command specifies that a given aircraft has some trouble that may require special handling. The keyword EMERGENCY is followed by the call sign of the affected aircraft and, if the emergency is being created, the name of the emergency. The valid emergencies are HYDRAULIC, FIRE, COMM TRANSMITTER, COMM RECEIVER, TACAN, ADF, IFFSIF, BATTLE DAMAGE, GYRO, SLATS, AILERONS, RUDDER, ELEVATOR or LANDING GEAR. Only one of these may occur on an emergency command, but several emergency commands may refer to a single aircraft. The occurrence of the emergency implies that the system in inoperative. If the keyword CANCEL follows the call sign, then the specified emergency no longer applies.

The SPEED command specifies that the chosen aircraft is to accelerate or decelerate to the given speed. The acceleration used for this speed change is 20 knots per second. The keyword SPEED is followed by the call sign and the new speed.

The COLOR command is not currently inplemented because of hardware constraints. (GIGI does not handle small areas of color well.) When it is implemented, the experimenter can change the color of an aircraft by following the word COLOR with the call sign and color. The valid colors will be BLACK, WHITE, RED, GREEN, BLUE, MAGENTA, CYAN and YELLOW.

The HALT command terminates the trial. It has no operands.

The experimenter's language interpreter reads commands from a disk file which has been created in advance by the user of the system. (You may create this file using the editor of your choice.) Each command is made up of two parts:

1. the elapsed time since the beginning of the trial that this command is to be executed, and

2. the command itself, described in the preceding several paragraphs.

The time is specified as follows: The first two columns of each record in the file must contain the number of minutes, the third and fourth columns must contain the number of seconds. After that, the experimenter command is entered in the file. For example, if the record

0730create xtf-1 1 122 180 20 angels 120

appeared in the profile, seven minutes, thirty seconds into the trial the create command will be executed.

There is a certain structure that must be observed in the experiment profile. All "define" commands must be placed together at the beginning of the file. These must be followed by one "begin" command (if the begin command is omitted, the "display" process will never be created and you'll never see the screen.) After the "begin" command comes all the rest of the commands for the experiment. Note that the "define" and "begin" commands should have time delays of 0000. Appendix A has a sample profile.

The default file type for profiles is .PRO but this is not required. If you do give your profiles this file type, you need only specify the filename when you run the com file described in the section of this document on logical name table entries. Note that the profile must be almost totally unprotected in order for the ONR account (see below) to properly access the profile. Therefore, after the profile is created, the protection must be set to group: read, write, execute and delete. Use the DCL command

set protection=g:rwed {profile name}

to do this.

4.1.2 Logical Name Table Entries

GENIE makes extensive use of the VAX's logical name facility. In general, however, the user of GENIE need not concern him/herself about that. If GENIE is being run in the Human Factors lab, execution of the com file [fainter]hfgenie will do all the required logical name assignments. This com file will accept as a parameter or prompt you for the name of the file containing your experiment profile.

4.1.3 Running GENIE

If your application does not require user modification of GENIE, you may run it without copying it to your directory. In order to do this, you must log onto to the ONR account. See Roger Ehrich, Tim Lindquist or Bob Fainter for the password and the procedures for this account. Once you are on this account, set the default directory to [fainter.genie2]. This will give you read and execute access to GENIE. (The procedure for obtaining your own copy to mess with at will will be explained later.) Then, with the RUN command, execute the program ENTRY. This invokes DMS. DMS will ask you for the image name. Respond with the name GNESUBJ. This is all you need to do. GENIE is now running and any further communication must be by the experiment profile or subject commands. To terminate the run of GENIE, enter the command HALT on the subject's console. If for some reason GENIE becomes hung up and will not respond, you may break out of DMS with CNTL Y. This will return you to VAX DCL level.

4.2 THE SUBJECT'S INPUT LANGUAGE

The subject's input language is currently small, comprising only five commands. These commands are, however, powerful and allow the subject to do a great deal.

The SHOW command is a request to the system to place detailed information about one vehicle in the "Aircraft Status Area." The syntax is SHOW call sign.

All other subject commands are instructions to the vehicles and are prefaced with the call sign of the appropriate vehicle.

The TURN command causes the vehicles to turn. After the keyword TURN, the direction RIGHT or LEFT is specified, followed by the keyword HEADING, followed by a 3-digit number specifying the ground track relative to the vehicle's current location that it is to fly.

The altitude change command causes the vehicle to climb or descend to a given altitude. Following the call sign will be the key phrase CLIMB AND MAINTAIN or DESCEND AND MAINTAIN, followed by an altitude specification of the form ANGELS <alt>, where <alt> is the assigned altitude in thousands of feet, in the range 0-200.

The SPEED command assigns a new speed to a vehicle. Following the call sign and the word SPEED is the speed in nautical miles per hour (knots) that the aircraft is to fly.

The SAY command allows the subject to obtain certain selected information on any extant vehicle without disturbing the display in the "Aircraft Status Area." The format is call sign SAY SPEED or ALTITUDE or HEADING of FUEL STATUS. The information is returned to the subject's terminal, NOT to the GENIE display.

4.3 MODIFYING GENIE

The directory [fainter.genie2] contains language files for all programs and modules in the GENIE complex. This source code is in the library GENIELIB.TLB. A given module can be removed from this library by using the com file [fainter]extract. This com file prompts for the name of the module that you want to remove.

4.3.1 Source Listing

For the time being, contact Bob Fainter, X5853, VAX username FAINTER, for source listings.

4.3.2 Modifying GENIE

If you want to make modifications to GENIE for your own experiment, you must copy, into a directory to which you have write access, the library [fainter.genie2]genielib.tlb. Extract all the files, using the source listing as a guide. The files with type PAS are PASCAL programs; those with type FOR are FORTRAN programs. Make the desired editing changes and compile those modules that were changed. If you have just copied GENIE, you must compile all of the files. Any files that you change must be compiled and linked before running. Compile PAS files with the PASCAL command; FOR files with the FORTRAN command. There are COM files in [fainter.utility] to do the linking. LINKGENIE will relink all 5 processes; LINKEXPR will link the experiment profile process; LINKSUBJ will link the subject language process; LINKDBAS will link the data base handler; LINKTIME will link the timer process; and LINKDISP will link the display process. You have read and execute access to [fainter.utility]. Notice: the Computer Science department will not be responsible for the execution of user-modified versions of GENIE.

4.3.3 Language Modifications

The modular design of the parsers developed using the LLPARS facility enhances system flexibility as UAC and EP languages may be modified with only minor changes to the corresponding process software. Some changes, such as replacement of one keyword in a command with another existing keyword, requires only the command's production in the language specification be changed to reflect the replacement. Deleting a command can be accomplished by simply removing all references to the command from the language specification productions. (However, the deleted command's command-specific semantic action routine and the corresponding Data Base Position Update routines also should be deleted since they will no longer be executed.) Even the most complicated language modification, namely the addition of a new command, requires only minor software changes, as the following example illustrates.

Suppose researchers want to add a new EP command that will allow them to change the carrier's deck condition during an experiment run. This command, to be called a CHANGE_DECK command, is to have the following syntax:

CHANGE DECK TO OPEN
or
CHANGE DECK TO CLOSED

The software modifications needed to incorporate this command are summarized below.

1. In the language specification, add the CHANGE_DECK command as an option to the non-terminal COMMAND. Specify a command-specific semantic action routine to be called when a command of this type has been parsed. For example, the COMMAND non-terminal may include the line:

   | CHANGE_DECK {CHANGE_D}

   where CHANGE_D is the name of the command's command-specific semantic action routine.

2. Add a production to the language specification describing the command's syntax. Specifically, add the following production:

   CHANGE_DECK = CHANGE DECK TO DECK_COND

   where DECK_COND is the non-terminal described in the language.

3. Check the Keyword Definition Section of the language specification. Since the keywords "CHANGE" and "TO" currently are not defined as part of the EP language, add these keywords to the Keyword Definition Section and assign them unique key codes. So that these commands will be recognized as part of the EP language, add these keywords and their associated codes to the keyword lists in the LLSCAN support routine, KEYWRD.

4. Code the command-specific semantic action routine, CHANGE_D, and the corresponding Data Base Position Update routine so as to implement the command once it has been parsed.

As the preceding example illustrates, even a language modification involving the addition of a command with new keywords is easily performed. In fact, most language modifications can be accomplished without affecting the parsing operation for existing commands.

4.3.4 Running the Modified GENIE

In order to run a modified GENIE, you must have the EXE files for the procedures that you modified (they come automatically from the linking command) and the EXE files for the unmodified procedures. These are located in [fainter.genie2]. Copy the unmodified EXE files into your directory. Also copy the file gnebook.dat into your directory. Now, proceed as outlined in RUNNING GENIE, above, except after giving your username twice, DO NOT set the default directory to [fainter.genie2]. This way, DMS will execute the GNEEXPR that is in your directory.

REFERENCES

EHRI82

Ehrich, R. W. "The DMS Multiprocess Execution Environment". ONR Technical Report. (To appear).

GUY82

Guy, Sue R. "Design and Implementation of the Generic Interactive Environment's Expreiment Profile Process and User Aircraft Process." Project report, Master of Information Systems, Virginia Polytechnic Institute and State University, Department of Computer Science, 1982. Unpublished.

KNUT73

Knuth, D. E. The Art of Computer Programming: Vol 3 / Sorting and Searching. Addison-Wesley Publishing Company, Reading, Massachusetts; Menlo Park, California; London; Amsterdam; Don Mills, Ontario; Sydney 1973.

MCKE70

McKeeman, W. M., Horning, J. J. and Wortman, D. B. A Compiler Generator. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970.

MORS79

Mores, John A. DEC/TR-90 LLPARS Users Manual, Digital Equipment Corporation, 1979.

Appendix A

TRIAL SESSION

```
0000define refresh rate 3
0000define screen dimension 50
0000define tag call sign
0000define ship location ul
0000define approach heading 315
0000begin
0015create xtf-1 1 110 180 20 angels 100
0245create xtf-2 1 211 045 30 angels 140
0315create xtf-1 1 111 180 25 angels 100
0320emergency 211 hydraulic
0330emergency 211 landing gear
0500create xtf-3 1 344 090 50 angels 100
1000emergency 211 cancel landing gear
2000halt
```

The above session runs for 20 minutes while several aircraft are created and two emergencies are processed. Users are invited to try this session.

Appendix B

VAX FILES FOR GENIE

The following is the list of files in [fainter.genie2]genielib.tlb. The "extract" com file should be given these names and types to properly extract the information from the library.

ACTIVE.PAS
GNETYPE.PAS
ACTIVESTAT.PAS
GNEVARI.PAS
AIR.GRM
IENCODE.FOR
CARRIER.PAS
OLDSCREEN.PAS
COREPROCS.PAS
PLANE.PAS
DESTSTAT.PAS
REFRESH.PAS
DISPLAY.PAS
RENCODE.FOR
EXPER.GRM
SCREEN.PAS
GEN1CONST.PAS
SHIPSTAT.PAS
GEN1TYPE.PAS
VECHSTAT.PAS
GEN1VAR.PAS
GNEBOOK.DAT
GNECONS.PAS
GNEDBAS.PAS
GNEEXPR.PAS
GNELAN1.FOR
GNELAN2.FOR
GNEMATH.PAS
GNEMOVE.PAS
GNERAND.PAS
GNESEMA.PAS
GNESUBJ.PAS
GNETIME.PAS
GNETREE.PAS