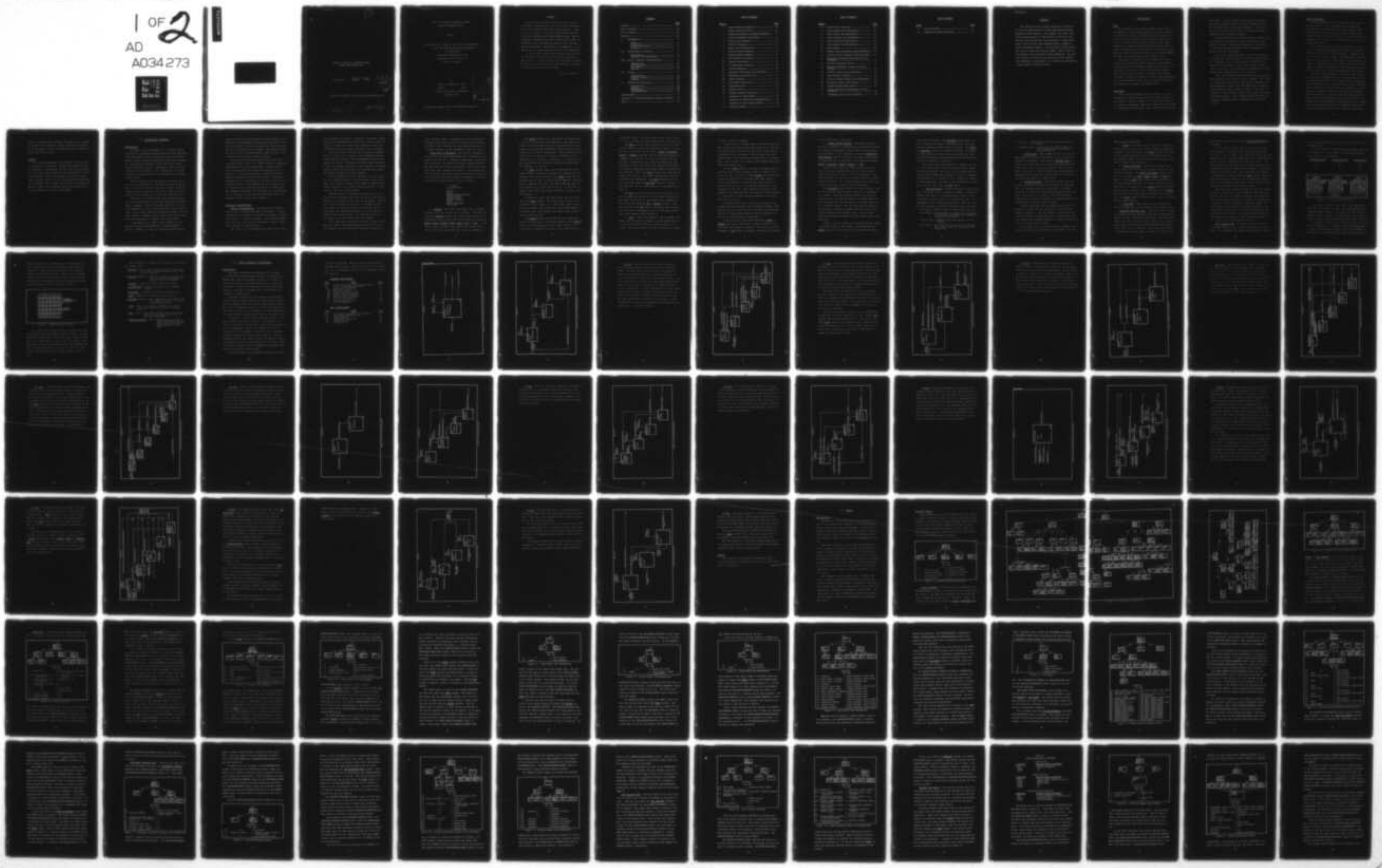
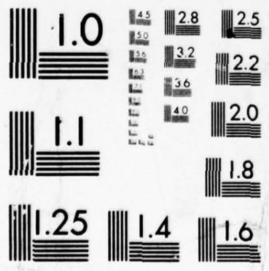
AD-A034 273 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
SOFTWARE DESIGN FOR MULTIMODE MATRIX DISPLAY PERCEPTION TESTS.(U)
DEC 76 R A LIEBACH
UNCLASSIFIED GE/EE/76-30 NL

1 OF 2
AD
A034273



1 OF 2

AD A034273



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

GE/EE/76-30

1

SOFTWARE DESIGN FOR MULTIMODE MATRIX DISPLAY PERCEPTION TESTS

GE/EE/76-30

Robert A. Liebach
Captain USAF

DDC
RECEIVED
JAN 10 1977
A

Approved for public release; distribution unlimited

(See form 1473)

GE/EE/76-30

SOFTWARE DESIGN FOR MULTIMODE MATRIX DISPLAY PERCEPTION TESTS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

Robert A. Liebach, B.S.E.E.
Captain USAF
Graduate Electrical Engineering

December 1976

ACCESSION for
NTIS White Section ✓
DDC Buff Section ☐
UNANNOUNCED ☐
JUSTIFICATION ....
BY ....
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

Approved for public release; distribution unlimited

Preface

This report summarizes the progress made towards developing a computer-based system to perform perception tests using light-emitting diode (LED) displays. Since it was not possible to test the design, the report discusses in depth the system development process. The requirements definition and initial design phase which are discussed can be successfully applied to any software development project. This report is written for a reader who possesses a basic knowledge of software development.

I wish to thank Captain Larry Goble of the Air Force Flight Dynamics Laboratory for his support throughout this project. A special thanks goes to my thesis advisor, Captain J. B. Peterson, for his encouragement, assistance and guidance through the past months.

Robert A. Liebach

Contents

List of Figures

List of Figures

List of Tables

Abstract

The Multi-Mode Matrix Display Program is attempting to test the acceptability of using light-emitting diode displays in USAF aircraft. The informal and formal functional specifications for the software necessary to perform the desired tests have been developed. Formal functional specifications were developed using a technique called structured analysis. The functional specifications were used to design the software system. This design was accomplished using structured design techniques. The software design is presented using structure charts together with functional descriptions of all modules and definitions of the interfaces between modules.

I. INTRODUCTION

Scope

The Multi-Mode Matrix Display (MMM) project office is attempting to develop light-emitting diode (LED) displays to replace CRT displays currently being used in USAF aircraft. Part of the effort being conducted by the MMM office is to perform human factors research on dot matrix symbology to determine the acceptability of using LED displays. Specifically, the MMM office is interested in testing the effects on human perception caused by the rotation, vibration and degradation of an LED display image.

This thesis is concerned with designing the software necessary to accomplish the MMM project test objectives, without making any assumptions concerning the hardware architecture which will be used to implement the system. The software design consists of designing the necessary subsystems, programs or modules, and their interconnections. It is not the intent of this thesis to devlop design specifications or to produce any program code.

Objectives

When developing computer software there is a great tendency to start producing code at the very beginning of the project. This has proven to be a very poor approach to software development, since these projects have frequently experienced cost overruns, missed schedules and disgruntled users. Even in projects where top-down structured programming techniques were used, the resulting software has frequently been plagued with problems. A prime example of this latter case is IBM's New York Times system which has many maintenance problems.

In virtually all of the above cases, the problems were the result of poor software design (or the lack of any software design). The primary objective of this thesis is to avoid the problems mentioned above by designing the system software architecture.

To meet the primary objective, it is first necessary to develop the top-level system performance specifications on a functional basis. This consists of precisely describing the user inputs to the system and the outputs desired by the user. The algorithms for information storage, transfer and manipulation are also described.

After the functional specifications have been defined, the structure of the software can be designed. The design is accomplished through the application of structured design techniques, which reduce the complexity of programs by dividing them into functional modules. This makes it possible to create complex systems from simple, relatively independent modules. The resulting design facilitates the debugging, modification and maintenance of the software system.

Design Constraints

Design constraints are the conditions which specify how the system is to be implemented. The design constraints do not necessarily define the particular items which will be used to realize the system. Rather, they identify limits which may later be used in the selection or creation of the hardware or software which is used to implement the system.

Good design practices advise that details of the design, and binding implementation decisions, be postponed to later phases of the system life-cycle. Typically this is not the case however, since a designer may be given premature budget or hardware constraints. Frequently the designer is constrained to use a particular piece of hardware which has already been purchased, even though that hardware may not result in the best implementation of the system.

When this investigation was started, no firm decisions had been made concerning the hardware which would be used to implement the system. It was assumed that a minicomputer in the PDP-11 class would be used, but this assumption had no impact on the design which was developed. In addition, no assumptions were made concerning the operating system which would be purchased with the system hardware, nor were any cost constraints imposed on the design. As a result, the design developed in this thesis made no assumptions concerning the computer or peripheral devices (I/O and storage) which would be used. Therefore, the design which is presented makes reference to "input from a console" or "output to mass storage" without giving any description of the particular device (or possible operating system routines) which will be used.

Outline

The informal functional specifications for the system are discussed in Chapter II. Chapter III presents the formal functional specifications for the system using activity and data models which result from the application of structured analysis. Chapter IV presents the software design using structure charts together with functional descriptions of the modules in the design. Conclusions are presented in Chapter V along with recommendations for the effort necessary to complete the software development.

II. REQUIREMENTS DEFINITION

Introduction

The normal system life-cycle for a computer software (or hardware) development project consists of the following seven phases (Ref 1:5-8): conceptual, requirements definition, design, coding and checkout, testing and operational. Requirements definition is the second phase in the system life-cycle, and it is recognized as one of the most critical of all the phases. It is during this phase that a complete and consistent set of requirements is developed for the system.

Numerous problems typically occur in projects where a good set of requirements is not developed early in the project. Some of the problems which have resulted are missed schedules, cost overruns and dissatisfied users, since the system does not perform the way the user wants (Ref 3:2).

During the last several years, the computer industry has become increasingly aware of the importance of requirements definition. Several different methods or aids have been developed to assist in developing good system requirements. One of these methods is structured analysis (Ref 3), which provides a graphic language which can be used to specify the requirements of a system (see Chapter III).

A computerized aid which has been developed is the Problem Statement Language/Problem Statement Analyzer (Ref 6). PSL/PSA is a tool for describing both the hardware and software of information processing systems and recording the descriptions in machine-processable form. The data base which is created is analyzed for logical correctness and consistency by the computer. The computer system can also be used to produce various reports on all or any selected part of the data in the data base.

Requirements definition typically deals with three different subjects (Ref 3:4): context analysis, functional specification and design constraints. Context analysis deals primarily with the reasons why the system is to be created. As discussed in Chapter 1, context analysis is considered beyond the scope of this thesis. Functional specification is a description of what the system is to accomplish in terms of the functions it is to perform, and is discussed in the following section and in Chapter III. The conditions specifying how the system is to be implemented is the subject of the design constraints, which were discussed briefly at the end of Chapter I.

Functional Specifications

General Specifications. The system to be designed will be used to perform perception experiments. A typical perception test consists of displaying a sequence of symbols and recording a subject's responses. The software system will have three basic modes of operation: Set-Up, Experiment Execution and Data Analysis.

The Set-Up mode of operation allows the user to enter test parameters to perform a particular experiment. During this mode of operation, the system will output a series of messages to the user to choose the options and enter the parameters to be used during execution of the experiment.

When the system is put into the Experiment Execution mode, a message will be output to the experiment subject, requesting the subject to enter identification information (name, date, etc.). The system will then begin displaying a series of symbols (according to the parameters and options entered during the Set-Up mode) and recording the symbols and subject responses for later analysis.

The Data Analysis mode of operation is used to reduce the data gathered during an experiment into a more useful form. For example, commands can be given to generate confusion matrices or to execute collapsing routines, as discussed at the end of this chapter.

When the system is initialized (turned on), it will operate in an executive mode which will request the user to choose one of the three basic modes of operation. When the user wishes to change from one mode to another, he will first enter a command to terminate the current mode of operation. The termination command causes the system to return to the executive mode, from which the user can again choose any of the basic modes of operation.

The system must be designed so that it is easy to use. Thus, the system will output requests to the user to input commands, and the parameters associated with the command. However, since a frequent user of the system will probably not require the system to request the parameters associated with a command, the system will allow the user to input a command and its parameters at the same time.

Set-Up Mode of Operation. This mode of operation allows the user to choose the options and to input the parameters to be used for a particular experiment. When put in the Set-Up mode, the system will output a message telling the user that it is in the Set-Up mode. The user may then enter any of the commands from Table I. If any other command is entered, the system will respond with an error message and request that the command be reentered. A description of each of the commands of Table I follows.

Table I
Set-Up Commands

LIBRARY
MASKS
SYMBOL DISPLAY DYNAMICS
TIME PARAMETERS
STIMULUS LIST
DISPLAY SEQUENCE
PRINT
SET-UP PARAMETERS
END SET-UP

1. LIBRARY. The system must provide the capability to input, store and recall a set of symbols. When the user enters the LIBRARY command, the system will respond by requesting the user to choose one of the following functions: CREATE, STORE, DISPLAY, PURGE, ALTER, LIST, or END. If any other command is entered, the system will respond with an error message and request that the command be reentered.

The CREATE command causes the system to respond with a request for the symbol characteristics. The system must provide the capability to enter the characteristics of any symbol, by defining the individual elements of the display which are to be lit or extinguished. The parameters can be entered one at a time, or a series of parameters can be entered (e.g., parameters_1, parameter_2, ...). The user can also specify that the display be cleared, in preparation for defining a new symbol.

The STORE command causes the characteristics of the symbol which is currently displayed to be entered into the library. The system responds to the STORE command by requesting that an index for the symbol be entered. When the index is entered, the system will check that the index is not identical to an index already in use. If the index is invalid, the system will request that the user enter a different index.

To delete a symbol from the library, the user would enter the PURGE command. The system would respond by requesting an index number. If a valid index number is entered, the system would delete the corresponding symbol; if an invalid index were inputted, the system would respond with an error message.

The DISPLAY command allows the user to display any symbol in the library. The system responds to the DISPLAY command by requesting the user to enter an index. If a valid index is inputted, the system will display the corresponding symbol. The system will respond with an error message if an index is entered which is not in the library.

The ALTER command allows the user to test the symbol display dynamics. After this command is entered, the system will request the user to input a ROTATE, ADD/DELETE, JITTER or VIBRATE function, and the associated command parameters (see 3 below). The user may specify that only one, or a combination (e.g., rotate and vibrate) of the functions be used. The system will respond by performing the selected functions on the symbol which is currently being displayed. Thus, the ALTER command allows the user to preview the effects of the symbol modifications prior to performing the experiment. The ALTER functions will remain in effect until the END ALTER command is inputted, after which the user can again enter any of the commands in Table I.

The LIST command allows the user to request the system to output the indexes of all the symbols currently in the symbol library or to output the number of symbols in the library (by using the ALL or NUMBER options). When the user enters the END command, the system will return to the Set-Up mode of operation, where it will again accept any of the commands shown in Table I.

2. MASKS. During performance of the experiment, the system may be required to display a random symbol (called a mask) before and/or after the stimulus symbol is displayed. The system must be capable of producing two kinds

of masks: static and dynamic.

A static mask is a symbol which has random characteristics which do not change during the display time. The system must be able to produce and store these masks prior to performance of the experiment. Dynamic masks are random symbols which are created during performance of the experiment. Unlike static masks, dynamic masks change appearance during the display time.

When the MASKS command is entered, the system will request the user to choose whether static or dynamic masks are to be used. If the user enters STATIC, the system will respond by asking the user to input the total number of masks to be generated and stored, the maximum number of elements to be lit for a mask and any constraints on the randomness of the mask. The randomness constraints will consist of requiring that the mask consist of line segments, or a constraint on which adjacent elements of a lit element may also be lit.

For dynamic masks, the system will request the user to input the time interval between lighting elements. The system will also request the user to input the persistance time for an element (the length of time an element is to be lit) if an LED display is being used.

If the user does not respond with either STATIC or DYNAMIC, the system will output an error message and request that the command be repeated. After the mask characteristics or END option is inputted, the system will return to the Set-Up mode of operation.

3. SYMBOL DISPLAY DYNAMICS. The system must provide the capability to perform certain modifications (such as rotation or vibration) to the symbol displayed during the performance of the experiment. After the SYMBOL DISPLAY DYNAMICS command is entered, the system will respond by asking the user to enter one of the following options: ROTATE, ADD/DELETE, JITTER, VIBRATE, or END.

When the user enters the ROTATE option, the system will respond by requesting the user to choose between a constant or random rotation. If the user chooses a constant rotation, the system will request that the rotation angle be inputted. For a random rotation, the system will request the maximum allowable angle of rotation.

The ADD/DELETE option allows the user to modify a symbol by having the system randomly remove or add portions of a symbol. When this option is chosen, the system requests the user to input the number of elements which are to be randomly added and the number of elements which are to be randomly removed. The user will have the option of entering one add/delete parameter which will be used for all the symbols in the stimulus list, or he can enter a different add/delete parameter for each symbol in the stimulus list.

Jitter causes the symbol to be randomly located within a specified region of the display. After entering the JITTER option, the user will be asked to input the allowable range of movement. The VIBRATION option allows the user to specify the range of movement and the frequency that the displayed symbol should be vibrated. The JITTER and VIBRATION options are mutually exclusive. If neither of these options is chosen during the Set-Up mode, then the symbols displayed during performance of the experiment will not be jittered or vibrated.

After any of the above commands are entered, the system will respond with a request to choose another symbol modification option. Thus, the user can specify that symbols displayed during the experiment are to have multiple modifications (i.e., symbols are to be rotated and vibrated simultaneously). To return to the Set-Up mode of operation, the user would enter the END option.

4. TIME PARAMETERS. During performance of the experiment, the system will be required to display a sequence of symbols consisting of an acknowledgement symbol, a mask, the stimulus symbol followed by a second mask, and a redisplaying of the acknowledgement symbol. There are several time intervals associated with this display sequence:

Mask Time_1 -	length of time to display the first mask
Mask Delay_1 -	time interval between the end of displaying the first mask until the beginning of displaying the stimulus
Stimulus Time -	length of time the stimulus is to be displayed
Mask Delay_2 -	the delay between the end of displaying the stimulus until the beginning of the second mask

Mask Time$_2$ - the length of time the second mask is to be displayed

Prompting Delay - maximum time the system should wait for a response before displaying a prompting message

5. STIMLUS LIST. The stimulus list is a set of symbols which are taken from the library to be used during a particular experiment. After entering STIMULUS LIST, the system will request the user to enter the indexes for the symbols to be used as the stimulus list. The user also will be requested to specify whether the symbols should be displayed in order from the stimulus list, or if the system should take them in a random order.

6. DISPLAY SEQUENCE. This command allows the user to enter parameters to control the sequence in which symbols are displayed during the experiment. One option which can be specified is the acknowledgement symbol to be used. The user may also choose a reinforcement option which causes the system to display a correct/incorrect acknowledgement after each subject response.

Another option the user can specify concerns the removal of a symbol from the stimulus list based upon the subjects responses. When this option is chosen, the user will be requested to input the number of consecutive times the subject must correctly respond to a symbol for the symbol to be removed from the list.

In addition, the user can also choose an option which causes the system to redisplay the same symbol if the subject responds incorrectly.

7. PRINT. The system must be capable of producing a print-out of the symbols and subject responses during the execution of the experiment. The PRINT command allows the user to specify that the system is to print the index of the displayed symbol, the symbol display dynamics, the subject's response and response time.

8. SET-UP PARAMETERS. The system must allow the user to store all of the set-up parameters used for a particular experiment. The SET-UP PARAMETERS command allows the user to enter the STORE, RELOAD, PRINT and ERASE options. With the STORE option, the user can specify that the current set-up parameters are to be retained in permanent storage for later use. The RELOAD option allows the user to recall a previous set of parameters and the PRINT option causes the current set-up parameters to be printed out. The ERASE option is used to delete a set of parameters from permanent storage.

9. END SET-UP. This command causes the system to exit the Set-Up mode and to return to the executive mode.

Experiment Execution Mode. After the set-up parameters and options have been specified, the user can command the system to begin executing the experiment. When the system enters this mode, it will output a message to the experiment subject, requesting that identification information (name, date, etc.) be inputted. The system will continue to display this request until a response is received from the subject or the user enters the <u>END EXPERIMENT MODE</u> instruction.

If the subject responds with an ID, the system will then display the acknowledgement symbol indicating that the test is about to begin. The system will then begin displaying the mask, stimulus symbol, mask and acknowledgement symbol sequence according to the set-up parameters. For each response from the subject, the system will record the displayed symbol, the symbol display dynamics, the subject's response and response time. The experiment will continue to run until the subject enters the <u>STOP</u> command, which will cause the system to output an "end of experiment" message. The system will then redisplay the request for ID information in preparation to test another subject.

The system will allow the subject to input two special responses. The first is an indicator that the subject has hit the wrong key when making a response. This indicator can be given between the time the subject has responded to a symbol and before he responds to the next symbol. If the subject indicates that he has hit the wrong key, the system will flag the stored data for his response and the experiment will continue. The second special response is an "I don't know" indicator, which the subject can use if he is not sure what the displayed symbol was.

<u>Data Analysis Mode</u>. The system must be capable of performing analysis on the results of one or more experiments. To accomplish the analysis, the system is required

to generate confusion matrices and also to perform a collapsing routine.

There are three different confusion matrices which are to be generated. As an example, assume the following for the results of an experiment:

Symbol Presented	Subject Response	Reaction Time
A	A	.5
A	C	.6
E	F	.4
E	F	.5
D	D	.6

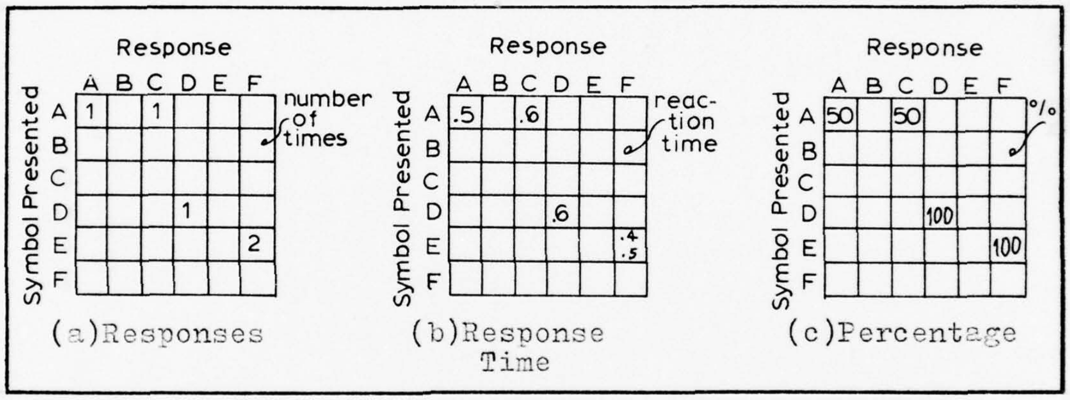
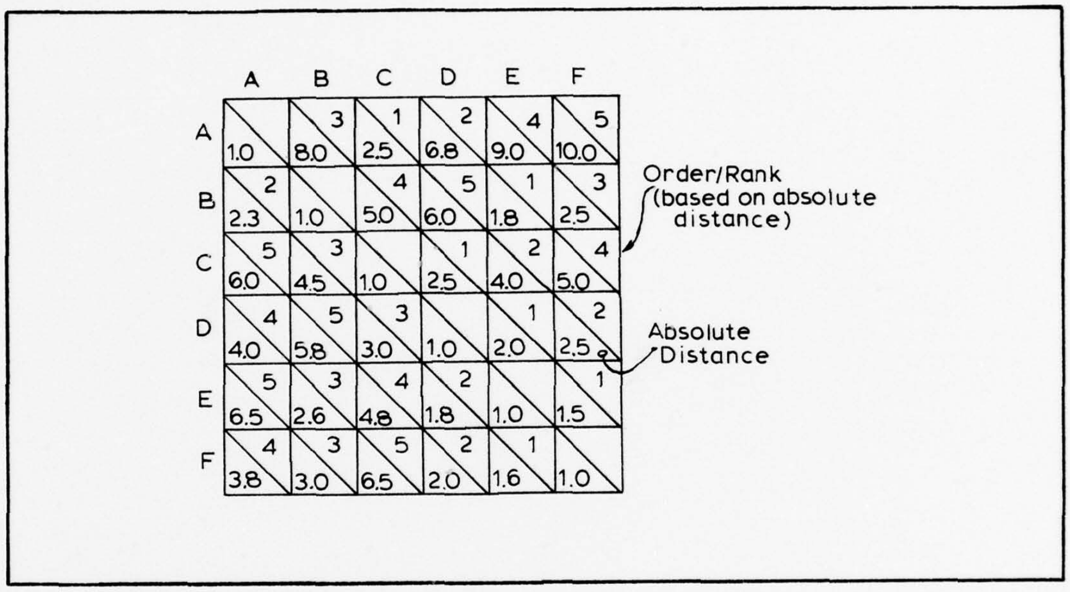


Figure 1. Sample Confusion Matrices

Figure 1(a) shows the first type of confusion matrix, which gives the number of times a given symbol-presented/subject-response occurred. In Figure 1(b), the subject's response time is plotted. Figure 1(c) shows the percentage of times a given symbol-presented/subject-response occurred.

A collapsing routine uses a predictor matrix to reduce the data contained in a confusion matrix into a more useful form. The collapsing routine consists of performing a comparison of a confusion matrix with a given predictor matrix (see Figure 2). The comparison is accomplished by first finding the cells in a confusion matrix which correspond to a given rank (order number) of the predictor matrix, or which correspond to a given range of absolute distance taken from the predictor matrix. The value contained in the corresponding cells of the confusion matrix are then summed to produce the desired result.


Order/Rank
(based on absolute distance)
Absolute Distance

	A	B	C	D	E	F
A	1.0	3 8.0	1 2.5	2 6.8	4 9.0	5 10.0
B	2 2.3	1.0	4 5.0	5 6.0	1 1.8	3 2.5
C	5 6.0	3 4.5	1.0	1 2.5	2 4.0	4 5.0
D	4 4.0	5 5.8	3 3.0	1.0	1 2.0	2 2.5
E	5 6.5	3 2.6	4 4.8	2 1.8	1.0	1 1.5
F	4 3.8	3 3.0	5 6.5	2 2.0	1 1.6	1.0

Figure 2. Sample Predictor Matrix

For example, the confusion matrix in Figure 1(b) could be collapsed using the predictor matrix of Figure 2 to give the Rank 1 reaction time. First, the rank 1 cells from Figure 2 are found (cells AC, BE, CD, DE, EF, and FE). Then the entries in the corresponding cells of the confusion matrix are found (the entries are 0.6, 0.4, and 0.5). The values of the entries are then summed to give a result of 1.5.

The following is a summary of the commands used during the analysis mode:

<u>LOAD DATA</u> - this command allows the user to enter the ID for each of the experiment results which are to be analyzed

<u>CONFUSION</u> (option) - after this option is entered, the user can specify which confusion matrices are to be generated

<u>RESPONSE</u> - generates response confusion matrix

<u>REACTION TIME</u> - generates reaction time confusion matrix

<u>PERCENTAGE</u> - generates percentage confusion matrix

<u>PRINT</u> - outputs the generated confusion matrix

<u>PREDICTOR</u> (option) - this option allows the user to enter a predictor matrix and perform a collapsing routine

<u>ENTER</u> - after this command is entered, the system will request the user to input a predictor matrix

<u>RANK</u>, number - this command causes the system to collapse the confusion matrix based on the given rank number

<u>ABSOLUTE DISTANCE</u>, range - this command causes the system to collapse the confusion matrix based on the given absolute distance range

III. FORMAL FUNCTIONAL SPECIFICATIONS

Introduction

The formal functional specifications of the system were developed using structured analysis, as mentioned in Chapter II. Structured analysis consists of creating module diagrams using a blueprint like language. These diagrams show a "top-down" decomposition of the system being analyzed, from both an activity and a data-oriented perspective as defined below.

Structured analysis uses the concept of modularity to analyze complex system structures by successively breaking down the subject matter into smaller, well defined modules. The goal of structured analysis is to break the system down into modules which are small enough to be easily understood and whose interfaces with other modules can be clearly seen.

To describe a complex system completely, there are two basic aspects which must be covered: the functions performed by the system and the things upon which the functions act. In structured analysis, the functions are called "activities" and the things are called "data". Structured analysis decomposes the system twice: once based on its activities (called the activity model) and again in a separate model based on its data (called the data model). Thus, the system is examined twice, resulting in a thorough functional specification of the system.

The next section of this chapter presents the activity model for the system. The last section of the chapter presents the data model. The reader is referred to Appendix A for a presentation of how to read structured analysis diagrams.

ACTIVITY DECOMPOSITION

Node	Title	Page
A-0	Perform Perception Experiment (Context)	22
A0	Perform Perception Experiment	23
A1	Process User Commands	25
A11	Get Valid Command	27
A12	Handle Set-Up Parameters	29
A13	Perform Library Function	31
A14	Perform Display Function	33
A15	Give Terminal Responses	35
A2	Execute Experiment	36
A22	Do Experiment Sequence	38
A3	Perform Analysis	40

DATA DECOMPOSITION

Node	Title	Page
D-0	Perception Experiment Data (Context)	42
D0	Perception Experiment Data	43
D1	Symbol Library	45
D2	Experiment Set-Up Data	47
D3	Experiment Data	50
D4	Analysis Data	52

Activity Model

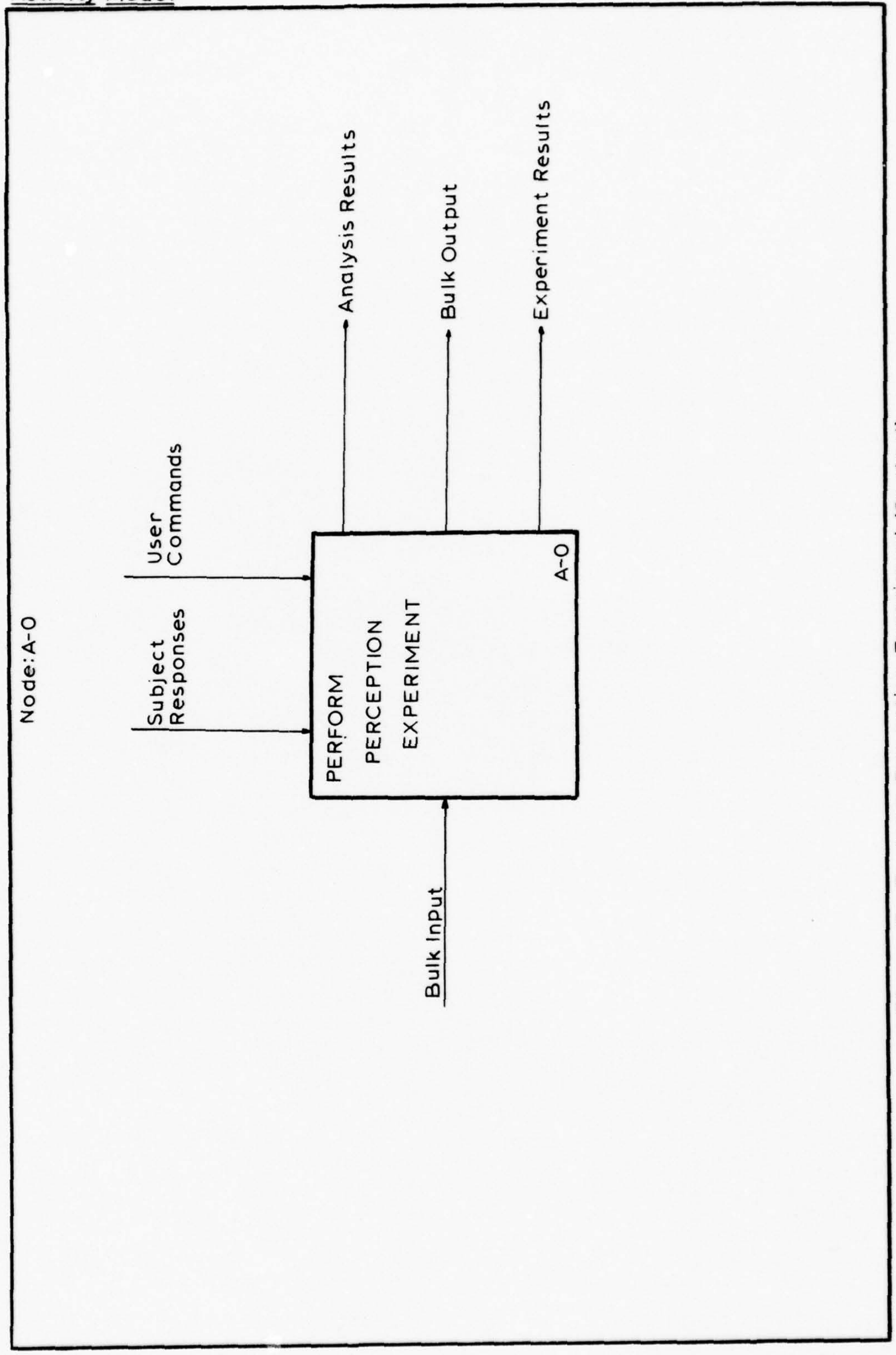


Figure 3. Perform Perception Experiment (Context)

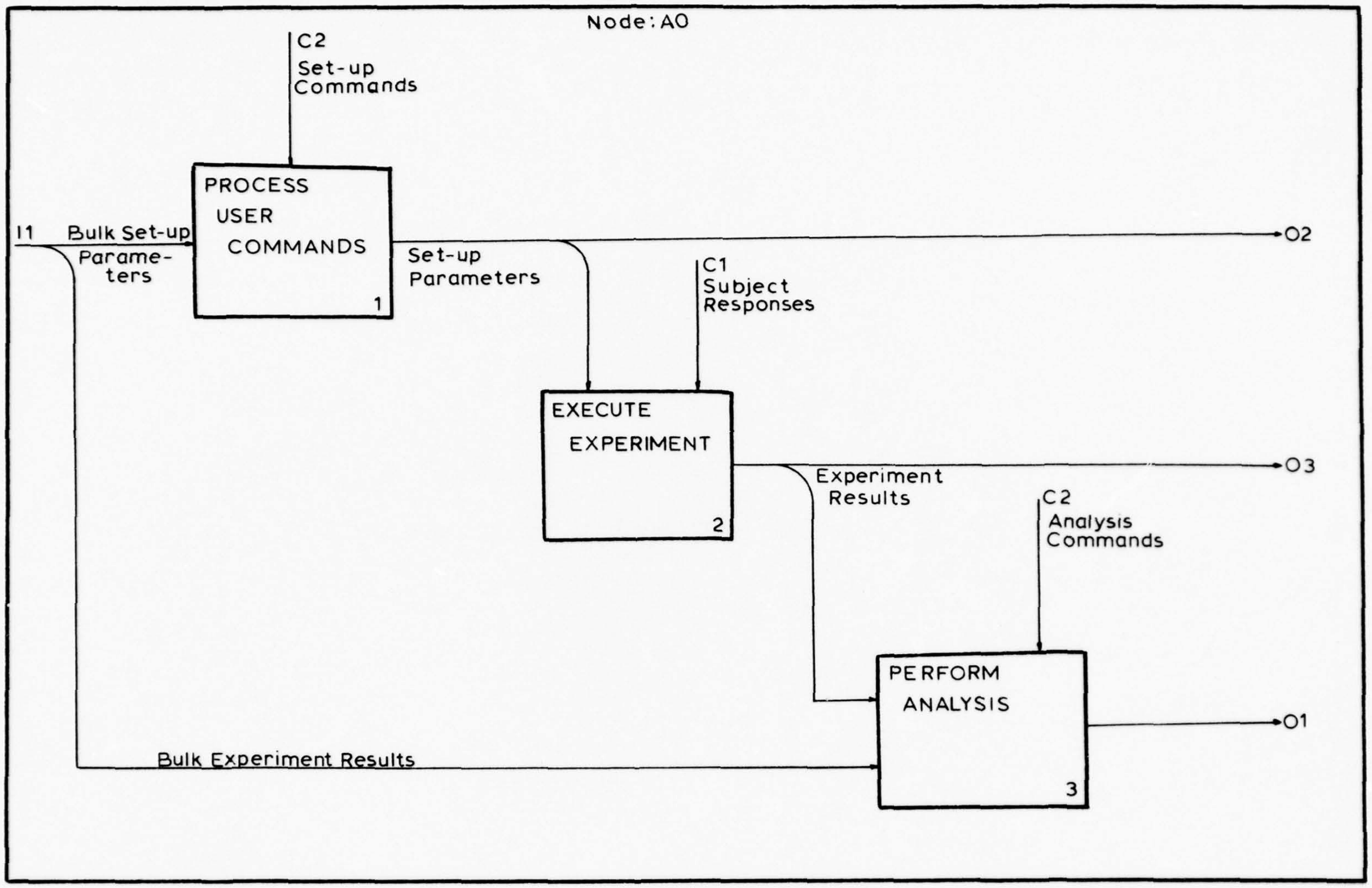


Figure 4. Perform Perception Experiment

A0 Text. The set-up parameters (1O1) are obtained from bulk input (1I1) or from the user set-up commands (1C1). Once the set-up parameters have been generated, they can be outputted (1O1) to be stored or they can be used to control the execution of an experiment (2C1). Execute Experiment (2) produces the experiment results (2O1), which can be outputted or they can be inputted (3I1) to Perform Analysis (3). Perform Analysis (3) uses the experiment results and the bulk input (3I2) to analyze the results of experiments, subject to the user commands (3C1). The bulk input (3I2) can consist of the results of other experiments and other data necessary to accomplish the analysis (e.g., a predictor matrix).

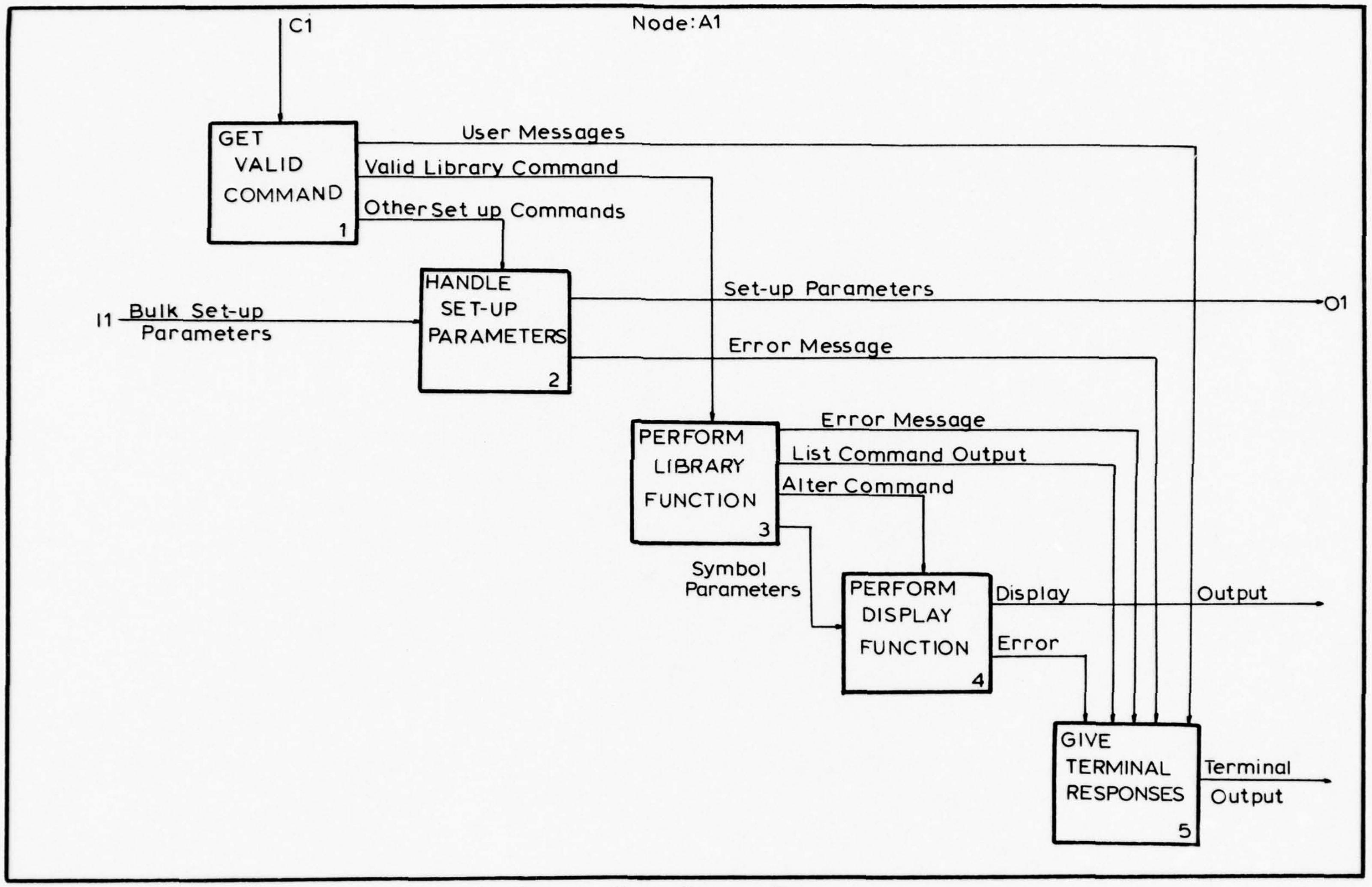


Figure 5. Process User Commands

<u>A1</u> <u>Text</u>. User set-up commands (1C1) are checked to determine if they are valid by (1). If the command is invalid, an error message is generated (102). For valid commands, (1) determines if the command parameters have been inputted, and requests the user to input these parameters (101) if they have not been entered. Get Valid Command (1) also determines if the command is a library command (103) or any of the other set-up commands (104).

The other set-up commands (2C1) are used by Handle Set-Up Parameters (2) to generate the set-up parameters (201) or to control the generation of the set-up parameters (201) from bulk inputs (2I1). If (2) receives a valid command which has invalid command parameters, an error message (202) is outputted.

Valid library commands (3C1) are used to control the performance of library functions (3). For a library <u>LIST</u> command, the requested output (302) is sent to (5). Library <u>ALTER</u> commands are performed by (4), which gets the parameters (4I1) from (3). If (3) receives invalid parameters for a command (e.g., an invalid symbol index) it outputs an error message (301).

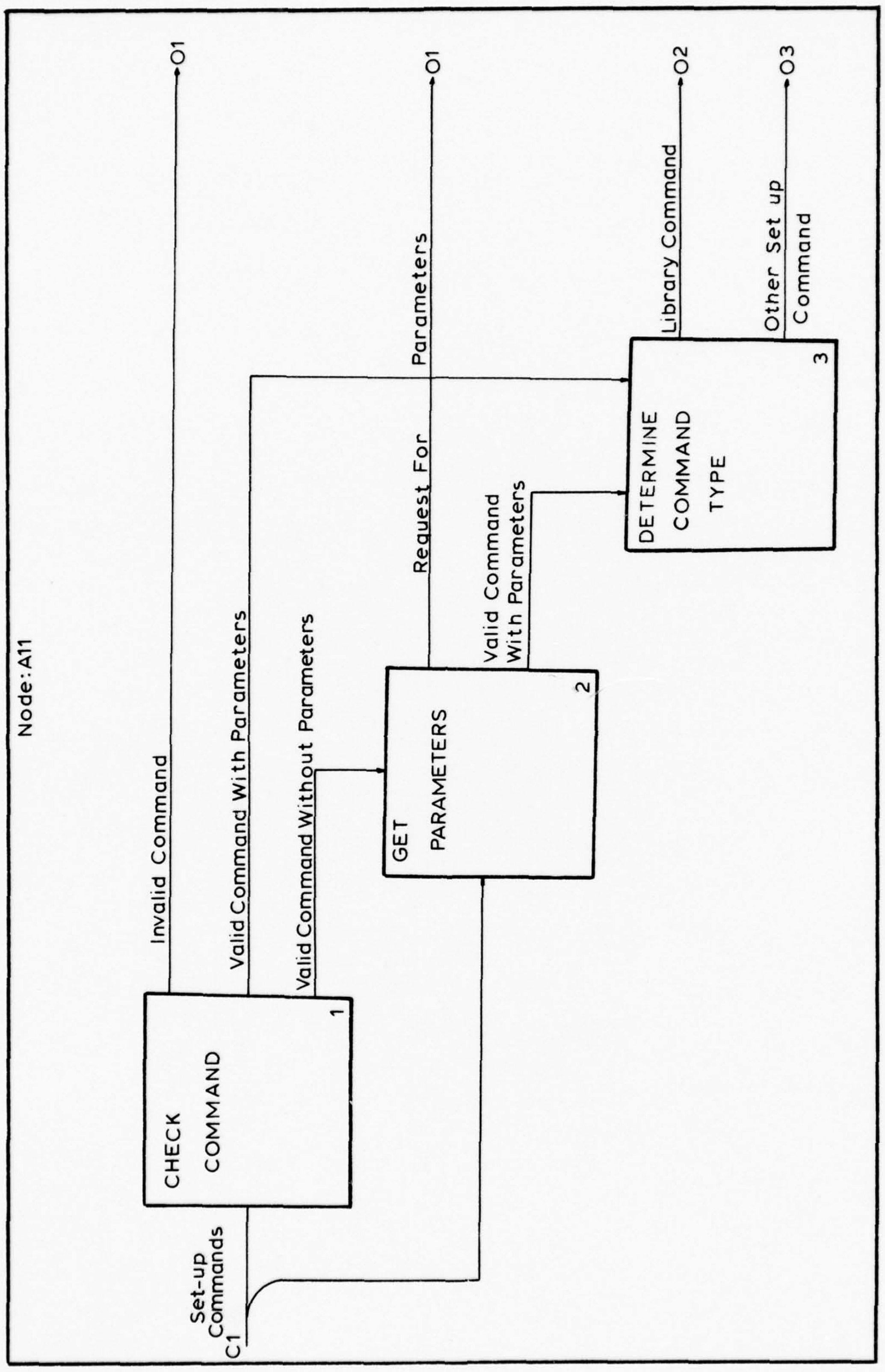


Figure 6. Get Valid Command

All Text. Check Command (1) inputs user commands (1I1) and checks if the command is valid and if the necessary command parameters have been inputted. If the command is invalid, an error message (1O1) is outputted. For a valid command which does not have the necessary command parameters (2C1), Get Parameters (2) outputs a request (2O1) for the parameters. The parameters are inputted (2I1) which results in a valid command with parameters to be outputted (2O2) to Determine Type Command (3).

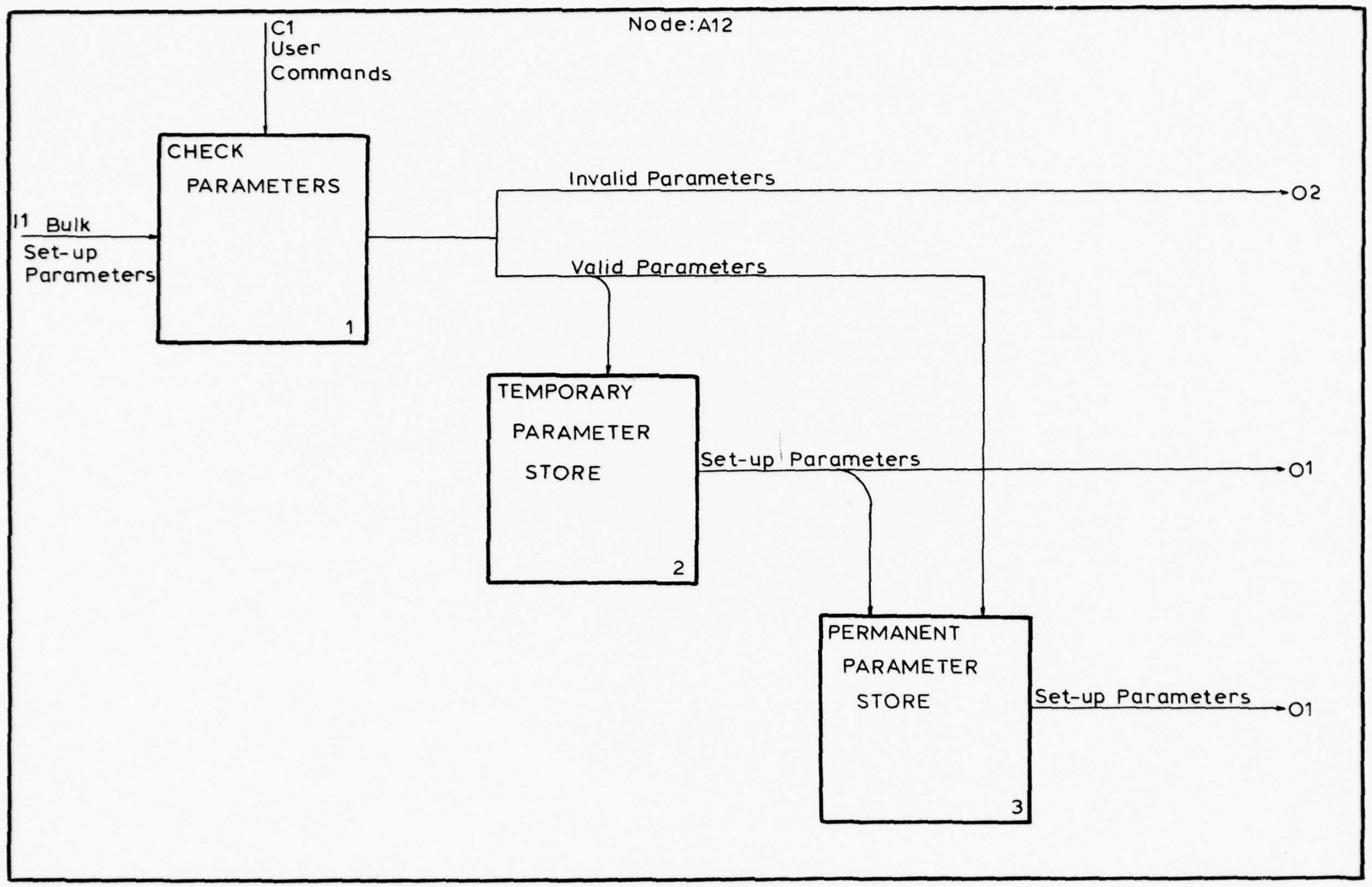


Figure 7, Handle Set-Up Parameters

A12 Text. Check Parameters (1) receives set-up commands (1C1) or bulk set-up parameters (1I1) and checks their validity. If an error is found, an error message (1O1) is generated. Valid parameters (1I2) are passed to (2) and (3). Temporary Parameter Store (2) stores the parameters for use by A2; Permanent Parameter Store stores the symbols also, and when directed by user commands (3C2), generates a permanent copy of the parameters (3O1) which can later be loaded as bulk set-up parameters.

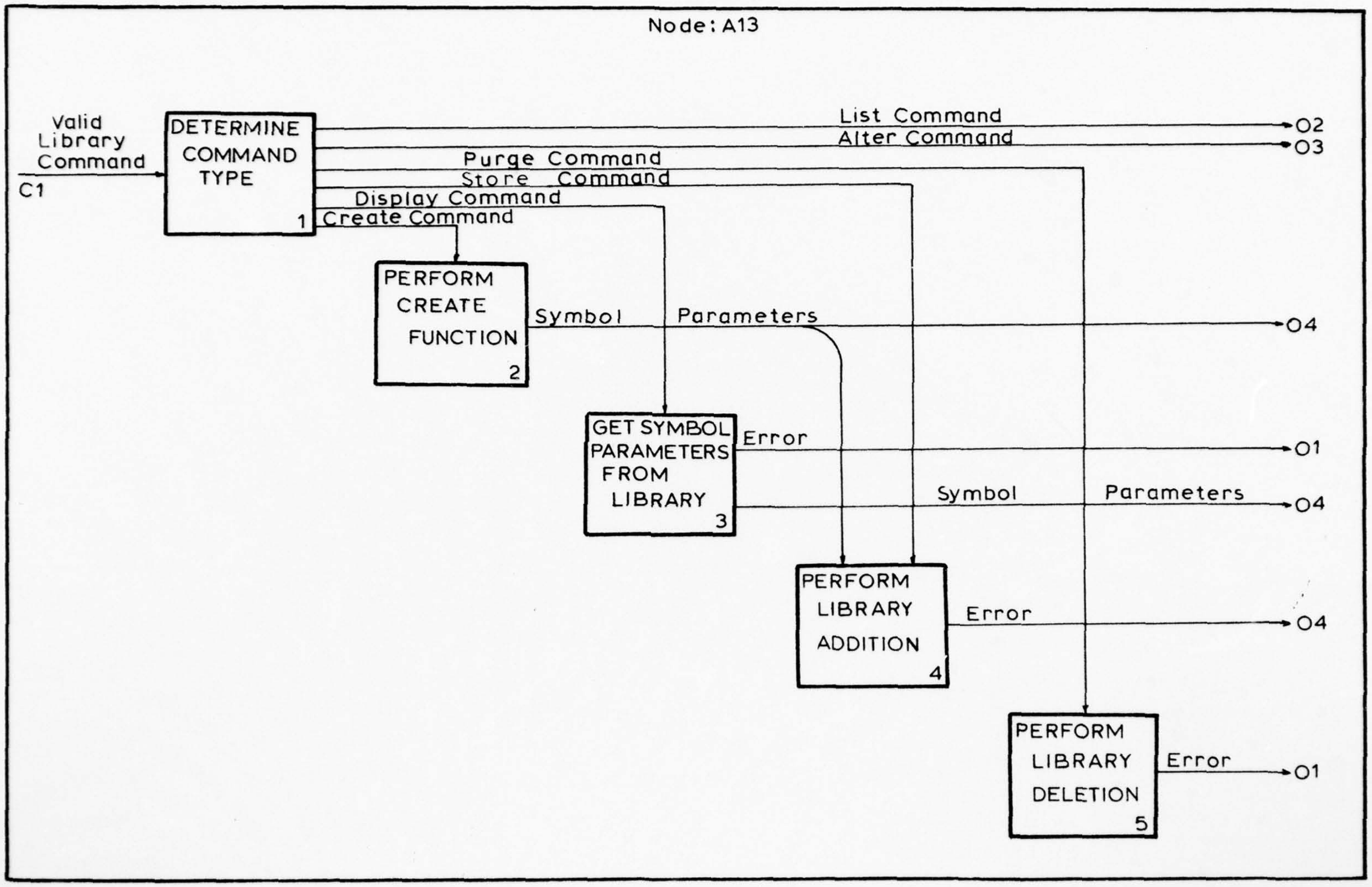


Figure 8. Perform Library Function

A13 Text. Determine Command Type (1) determines which library command has been inputted (1I1) and outputs the command to the appropriate module. Symbol parameters (201, 302) are generated by the create function (2) or by recalling the parameters (3) from the symbol library. For the STORE command, the symbol parameters generated by the create function (201) are stored in the symbol library by (4). Get Symbol Parameters From Library (3) generates an error message (301) if the index inputted by the user is not in the library. Similarly, Perform Library Addition (4) and Perform Library Deletion (5) produce error messages (401, 501) if the user inputs an invalid index.

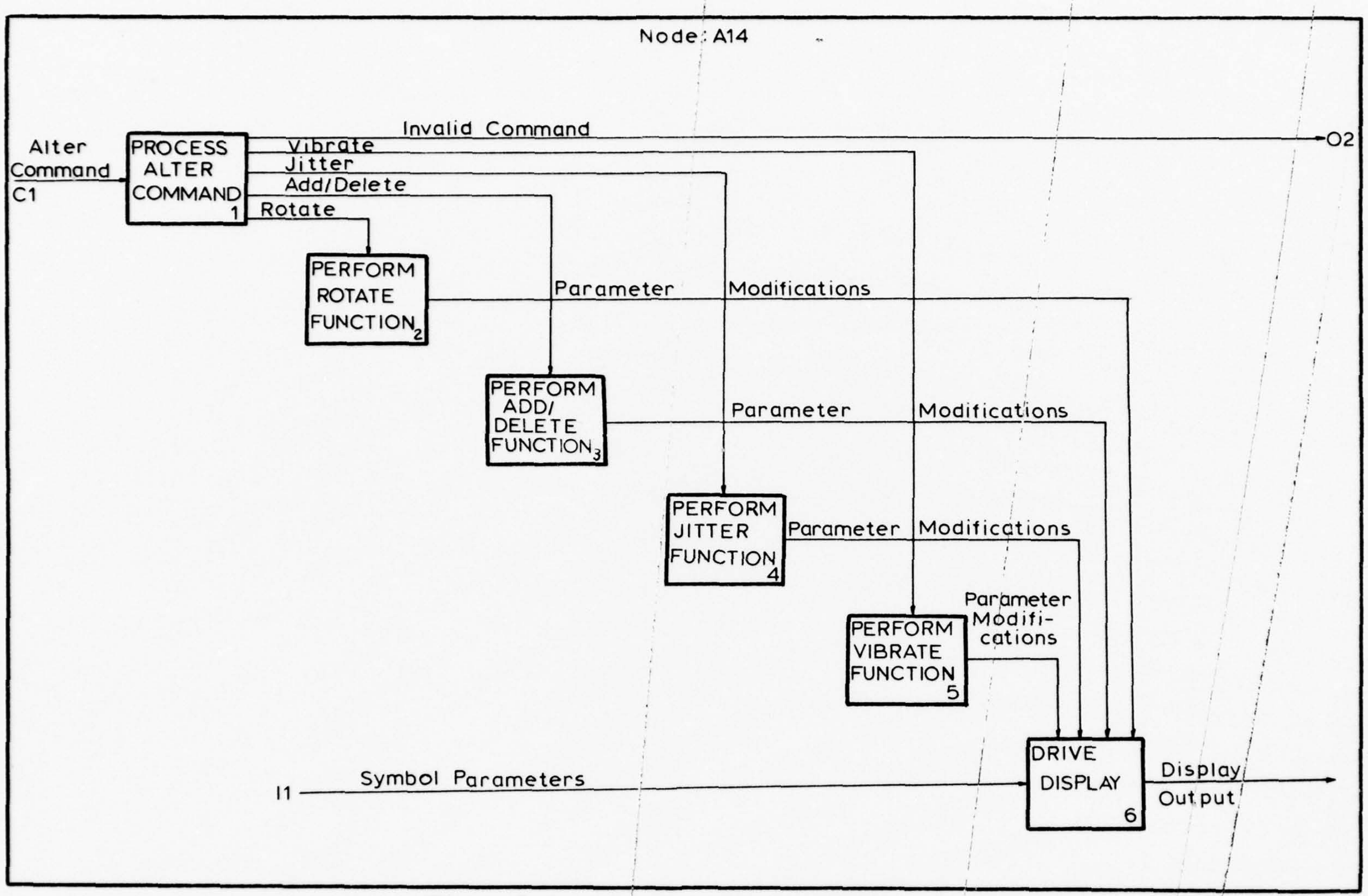


Figure 9. Perform Display Function

A14 Text. Process Alter Function (1) checks the validity of the alter command parameters and produces an error message (101) if the parameters are illegal. Alter commands with valid parameters (102, 103, 104, 105) are passed to the appropriate module which generates the necessary parameter modifications (201, 301, 401, 501). The parameter modifications (6C1, 6C2, 6C3, 6C4) are used by Drive Display (6) to manipulate the display output (601).

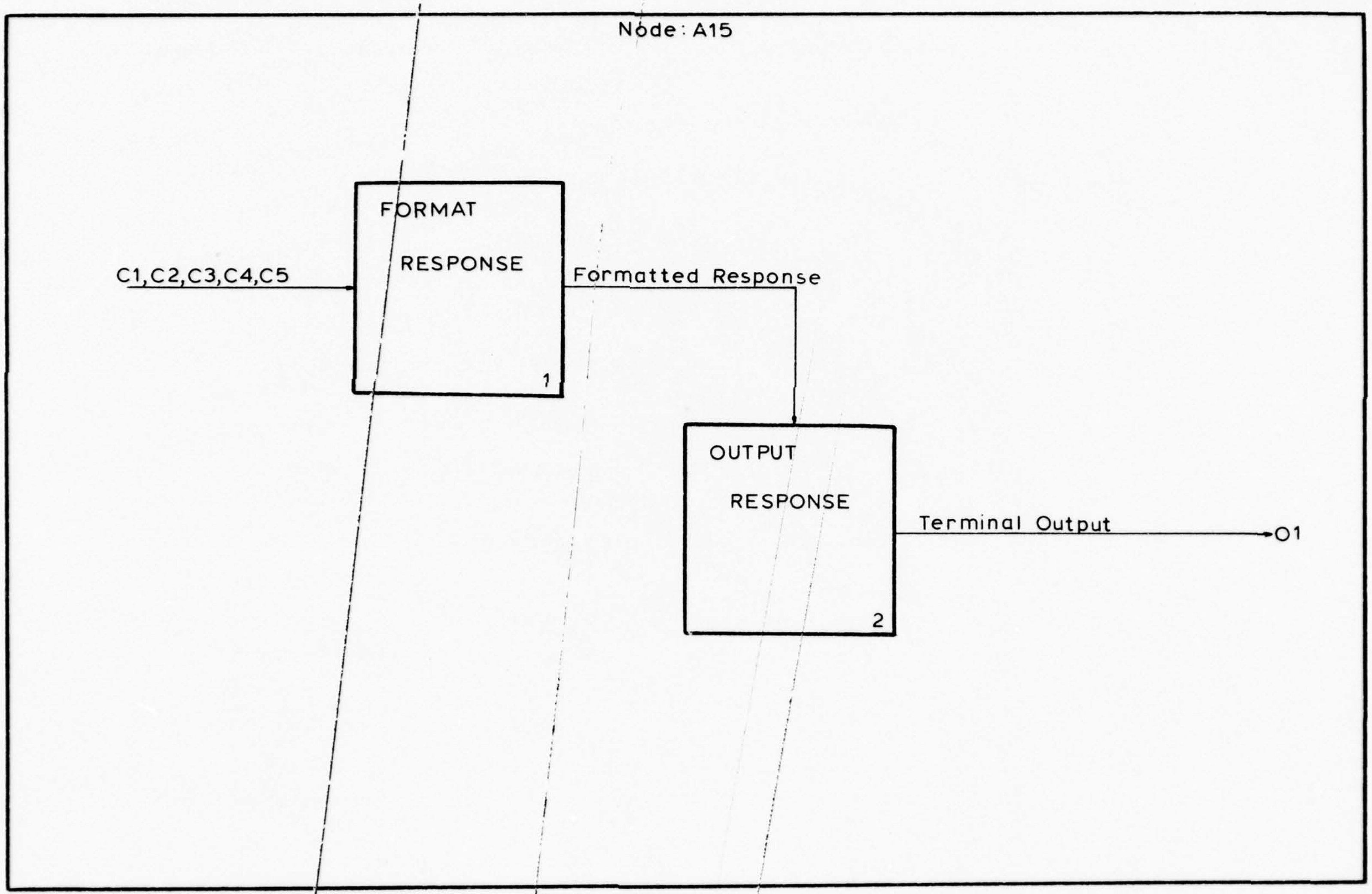


Figure 10. Give Terminal Responses

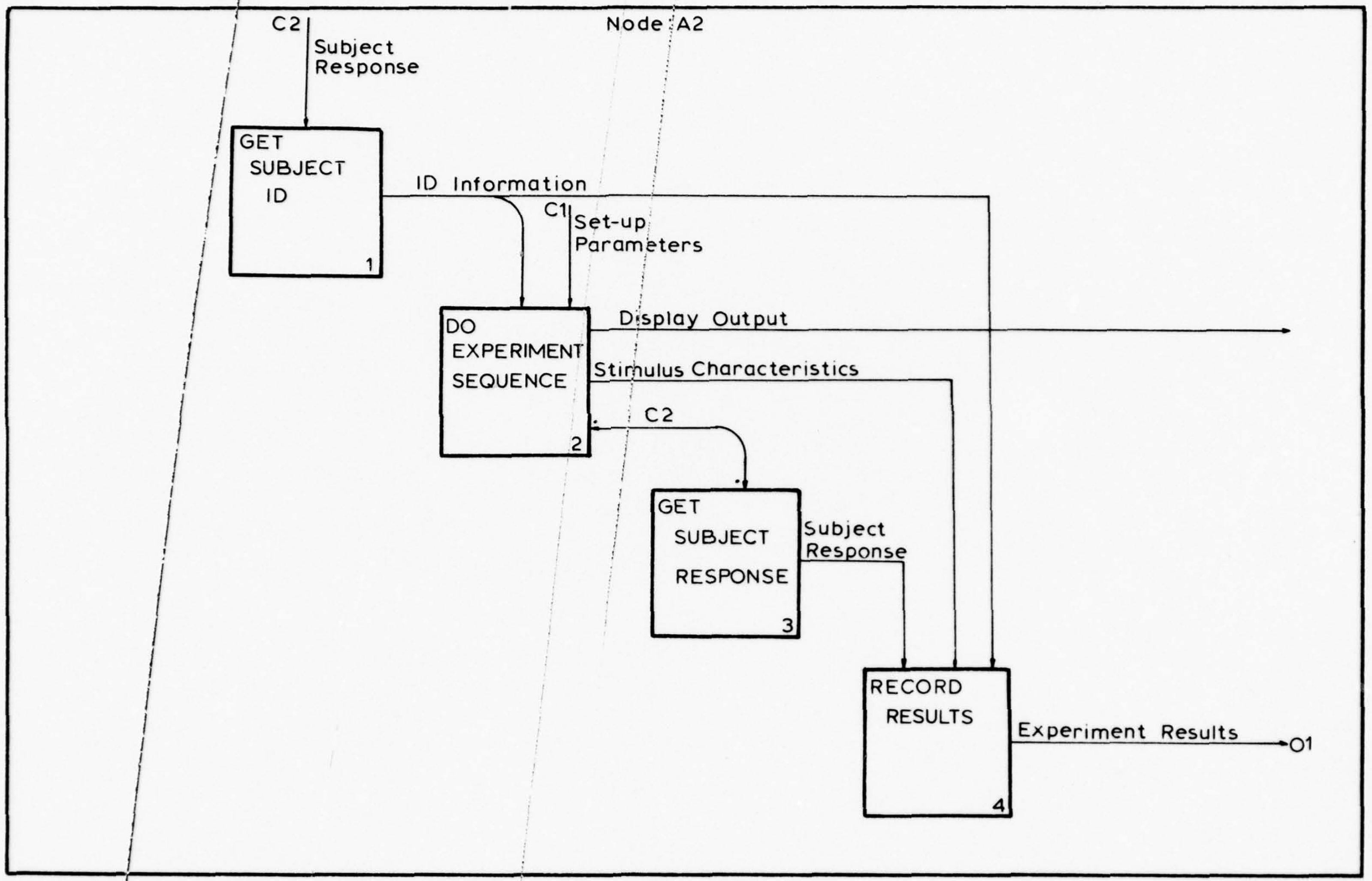


Figure 11. Execute Experiment

A2 Text. After the subject has entered his ID information (2C1), Do Experiment Sequence (2) displays a stimulus (201) according to the set-up parameters (2C2). The stimulus characteristics (202) and subject responses (301) are passed to Record Results (4) which produces the experiment results (401).

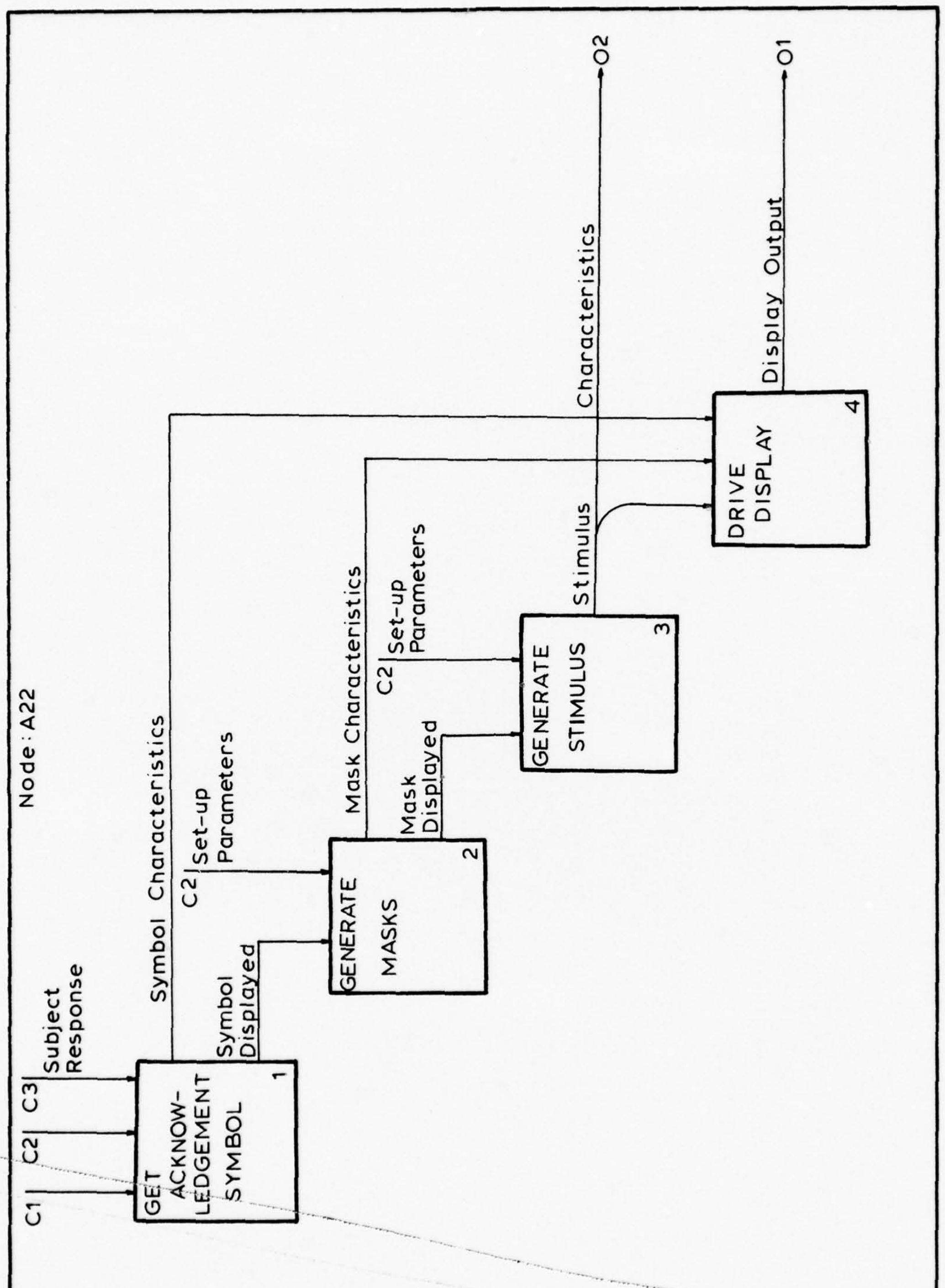


Figure 12. Do Experiment Sequence

A22 Text. Get Acknowledgement Symbol (1) uses the set-up parameters (1C2) to output the acknowledgement symbol characteristics (101) and informs Generate Masks (2) that the symbol has been displayed (2C1). Generate Masks (2) outputs the mask characteristics (201) and informs (3) that the mask has been displayed (3C1). Generate Stimulus (3) then outputs the stimulus characteristics (301) based on the set-up parameters (3C2).

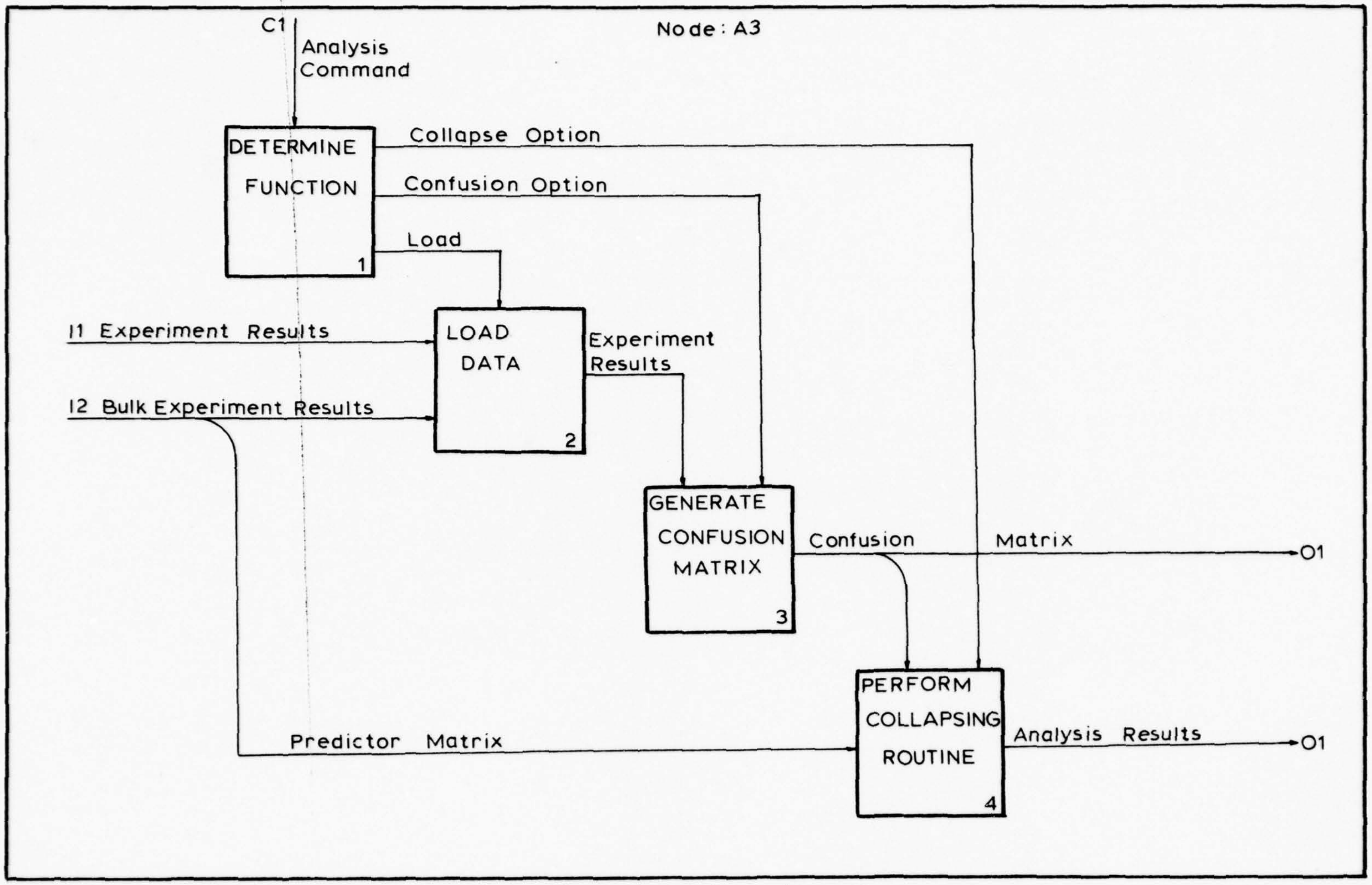


Figure 13. Perform Analysis

A3 Text. Determine Function (1) receives the user analysis commands (1C1) and passes the commands to the appropriate module. Load Data (2) inputs the experiment results (2I1, 2I2) and passes them to (3). Generate Confusion Matrix (3) produces the confusion matrix (3O1) based on the user commands (3C2). The confusion matrix (3O1) is then outputted or it is passed to (4), depending on the user commands (4C2). Perform Collapsing Routine (4) inputs the predictor matrix (4I1) and produces the desired results (4O1) based on the user commands (4C2).

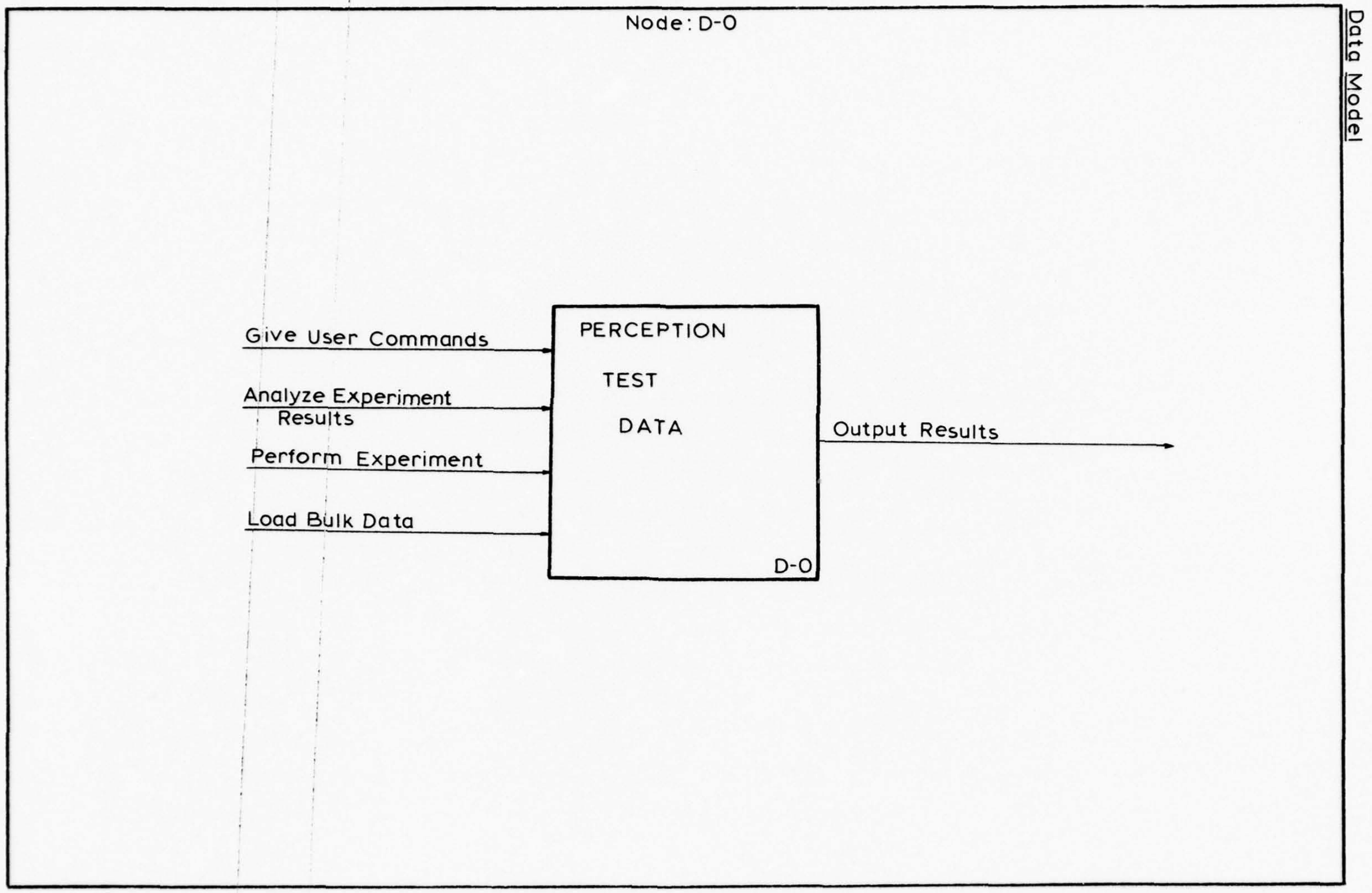


Figure 14. Perception Experiment Data (Context)

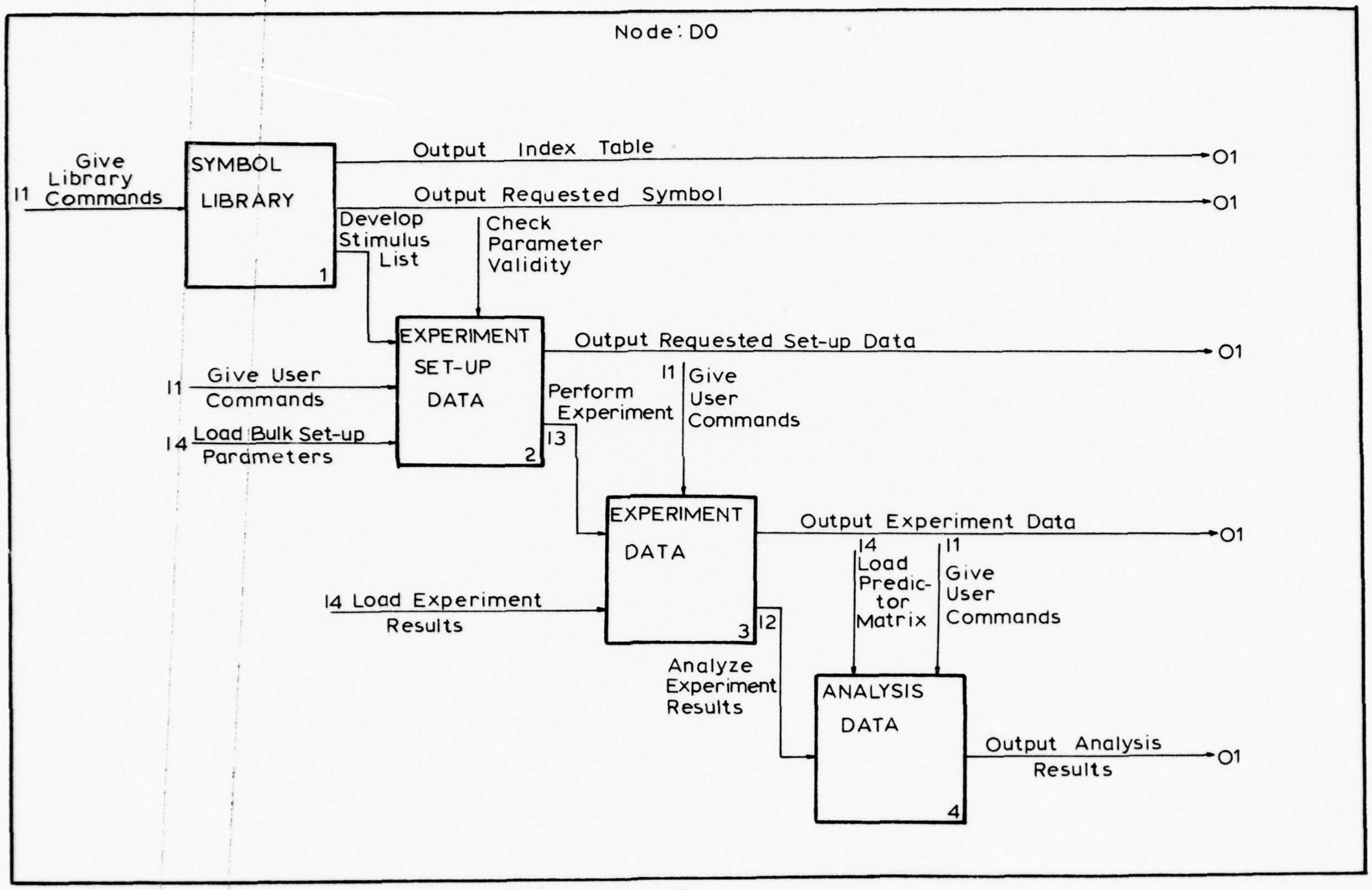


Figure 15. Perception Experiment Data

DO Text. The Symbol Library (1) is created from user library commands (1I1). The library (1) can be accessed to output the index table (1O1) or to output a symbol (1O2). The develop stimulus list activity (1O3) gets the symbol parameters for the stimulus list from (1).

Experiment Set-Up Data (2) is produced by user commands (2I1) and the develop stimulus list activity (2I1) or from the loading of bulk set-up parameters (3I3). The validity of these inputs is checked (2C1) before the inputs are entered into (2). The Experiment Set-Up Data (2) can be modified by user commands (2I2) or it can be accessed to output specific entries (2O1) which it contains. The performance of an experiment (2O2) also uses (2) to control the experiment.

The Experiment Data (3) is created by the performance of an experiment (3I1) or from loading bulk experiment results (3I2). Experiment Data (3) can be outputted (3O1) or it can be analyzed (3O2), subject to the user commands (3C1) which are inputted. The analysis of experiment results (4I1) creates the Analysis Data (4), which is manipulated according to the user commands which are given (4C2) and the predictor matrix which is inputted (4C1).

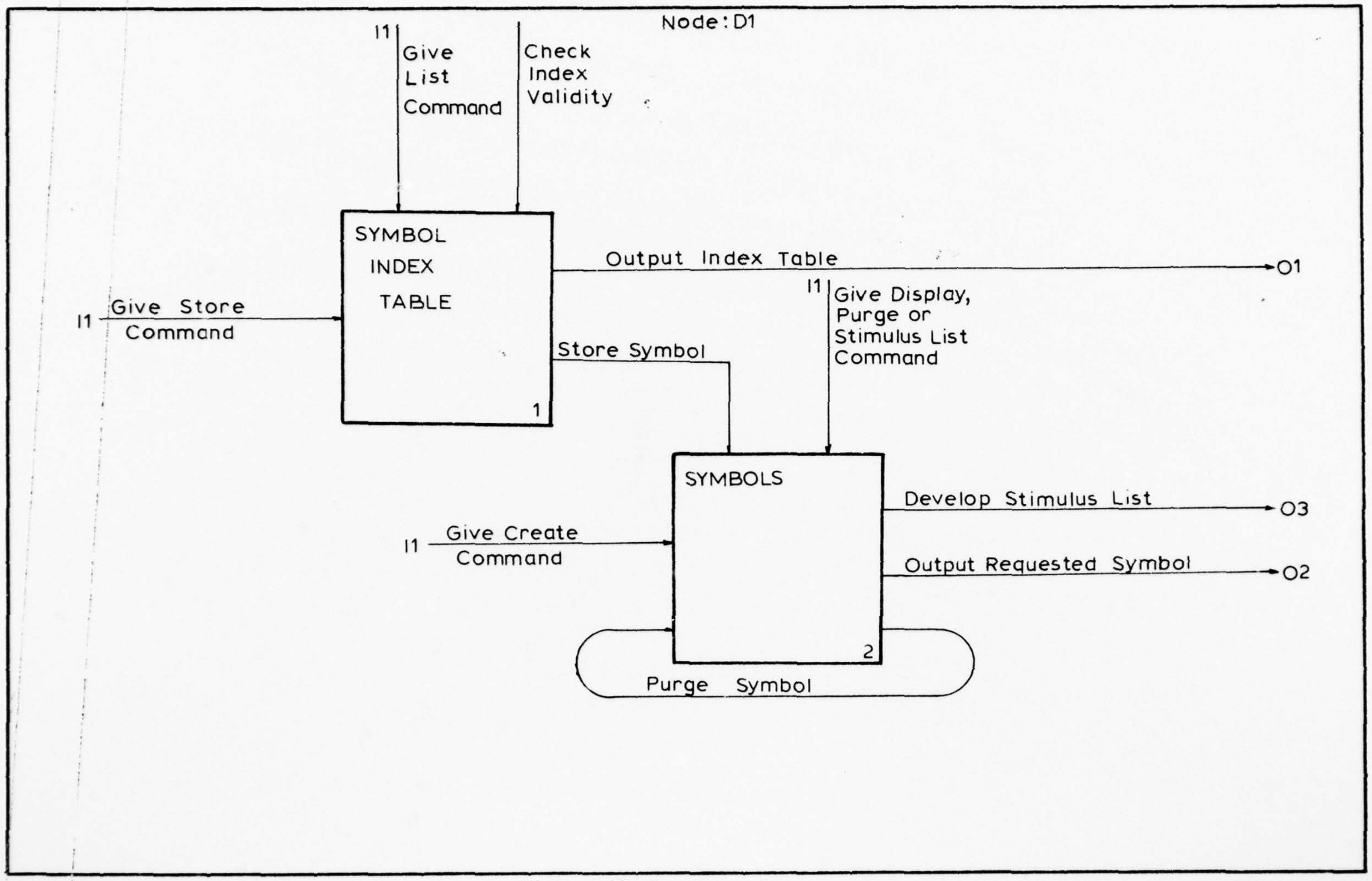


Figure 16. Symbol Library

D1 Text. The Symbol Library is composed of the Symbol Index Table (1) and the Symbols (2). The parameters associated with a STORE command are used to make entries into (1), subject to the validity of the inputted index (1C2). The LIST command (1C1) and the index validity (1C2) are used to control the outputting of the index table (1O1).

The store symbol activity (2C1) uses (1) to control the entry into (2) of symbol parameters associated with a CREATE command (2I1). The DISPLAY, PURGE and STIMULUS LIST commands constrain the use of (2) to create a stimulus list (2O1), output a symbol (2O2) or to purge a symbol (2O3). Purging a symbol (2O3) causes the contents of (2) to be modified (3I2).

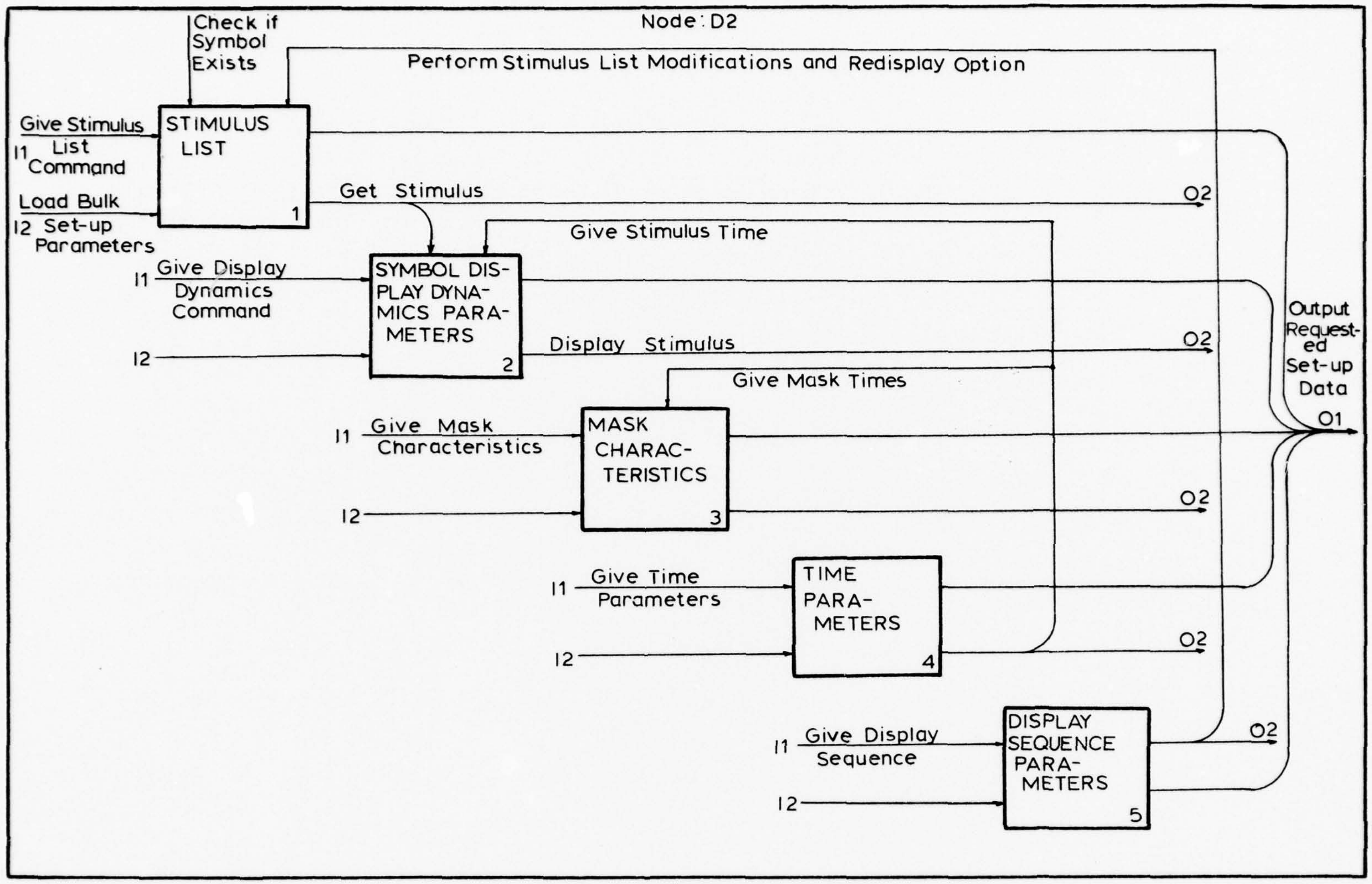

Node: D2
Check if Symbol Exists
Perform Stimulus List Modifications and Redisplay Option
Give Stimulus List Command
I1
Load Bulk Set-up Parameters
I2
STIMULUS LIST
1
Get Stimulus
O2
Give Stimulus Time
I1
Give Display Dynamics Command
I2
SYMBOL DISPLAY DYNAMICS PARAMETERS
2
Display Stimulus
O2
Give Mask Times
Output Requested Set-up Data
O1
I1
Give Mask Characteristics
I2
MASK CHARACTERISTICS
3
O2
I1
Give Time Parameters
I2
TIME PARAMETERS
4
O2
I1
Give Display Sequence
I2
DISPLAY SEQUENCE PARAMETERS
5
O2

Figure 17. Experiment Set-Up Data

D2 Text. The Stimulus List (1) is created by the STI-MULUS LIST command (1I1) or by the loading of bulk parameters (1I2), subject to the existance of the stimulus (1C1) in the symbol library. The Stimulus List (1) can be outputted (1O1) or it can be used by the get stimulus activity (1O2) during the performance of an experiment. The get stimulus activity is controlled by the stimulus list modifications (which can remove a symbol from the stimulus list) and by the redisplay option (1C2).

Symbol Display Dynamics Parameters (2) are created by the DISPLAY DYNAMICS command (2I1) or by the loading of bulk parameters (2I2). The contents of (2) can be outputted (2O1) or it can be used to display a stimulus (2O2) during the performance of an experiment, subject to the constraint of the get stimulus activity (2C1), and the stimulus time (2C2).

The Mask Characteristics (3) are created by the MASK command (3I1) or by the loading of bulk set-up parameters (3I2). Mask characteristics (3) are used to display masks (3O2) during an experiment (according to the mask times (3C1)) or they can be outputted (3O1).

Time Parameters (4) are produced by loading bulk parameters (4I2) or from user commands. The parameters can be outputted (4O1) or they can be used to control the use of (2) and (3).

The Display Sequence Parameters (5) are used to modify the stimulus list (1) or to constrain the get stimulus activity (102) by specifying that a stimulus be redisplayed. These parameters (5) are created by the DISPLAY SEQUENCE command (5I1) or by loading bulk set-up parameters (5I2).

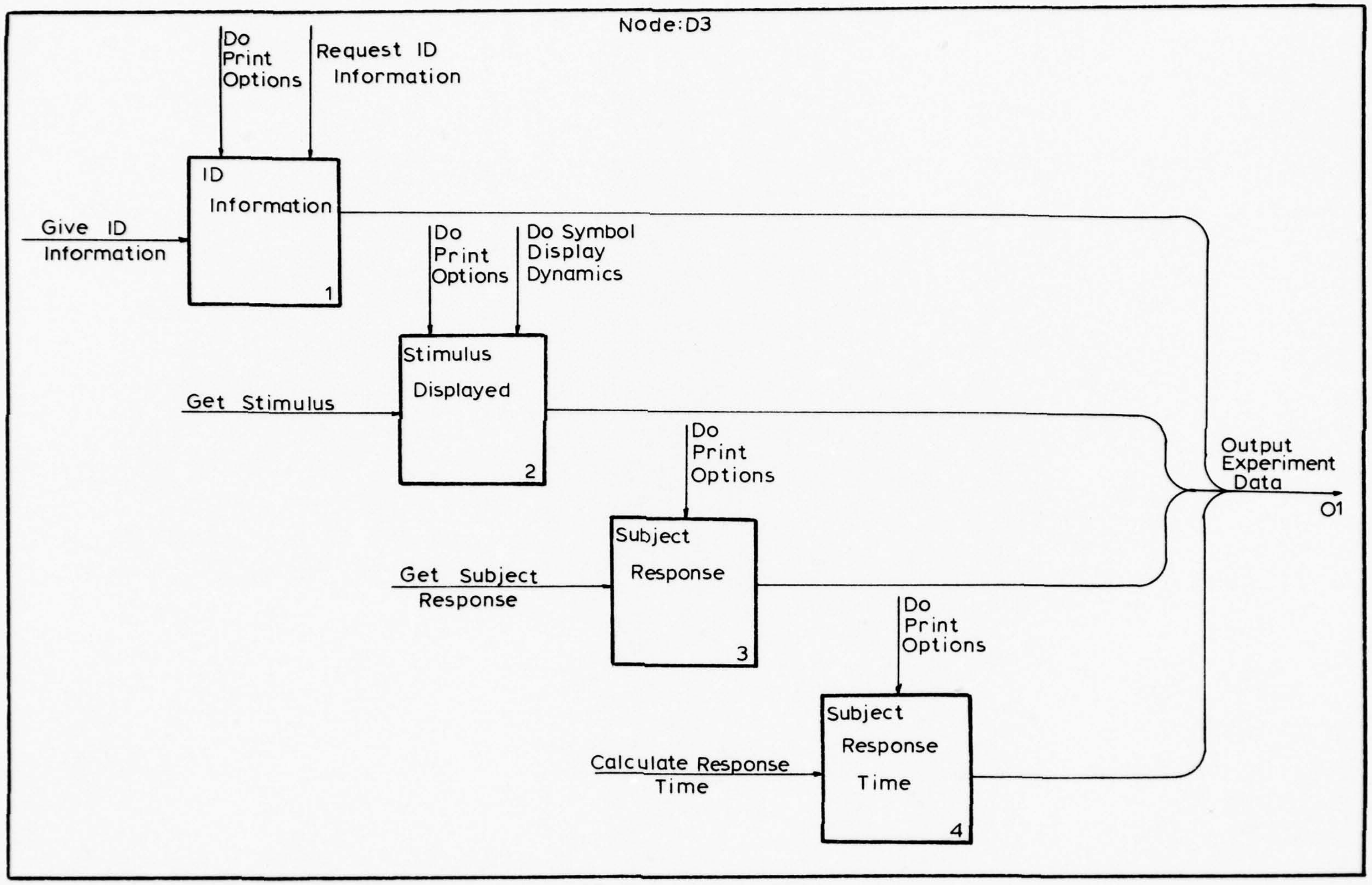


Figure 18. Experiment Data

D3 Text. ID Information (1) is produced by the subject inputs (1I1) after the information has been requested (1C3). This data (1) is outputted (1I1) if this is specified by the print option (1C1).

The displayed stimulus (2) is created by the get stimulus activity (2I1) and is modified according to the symbol display dynamics (2C2). The stimulus index and modifications (2) are outputted (2O1) according to the print option (2C1).

The Subject Response (3) is created by the subject responding to a displayed stimulus (3I1). This data (3) is outputted (3O1) when specified by the print option (4C1).

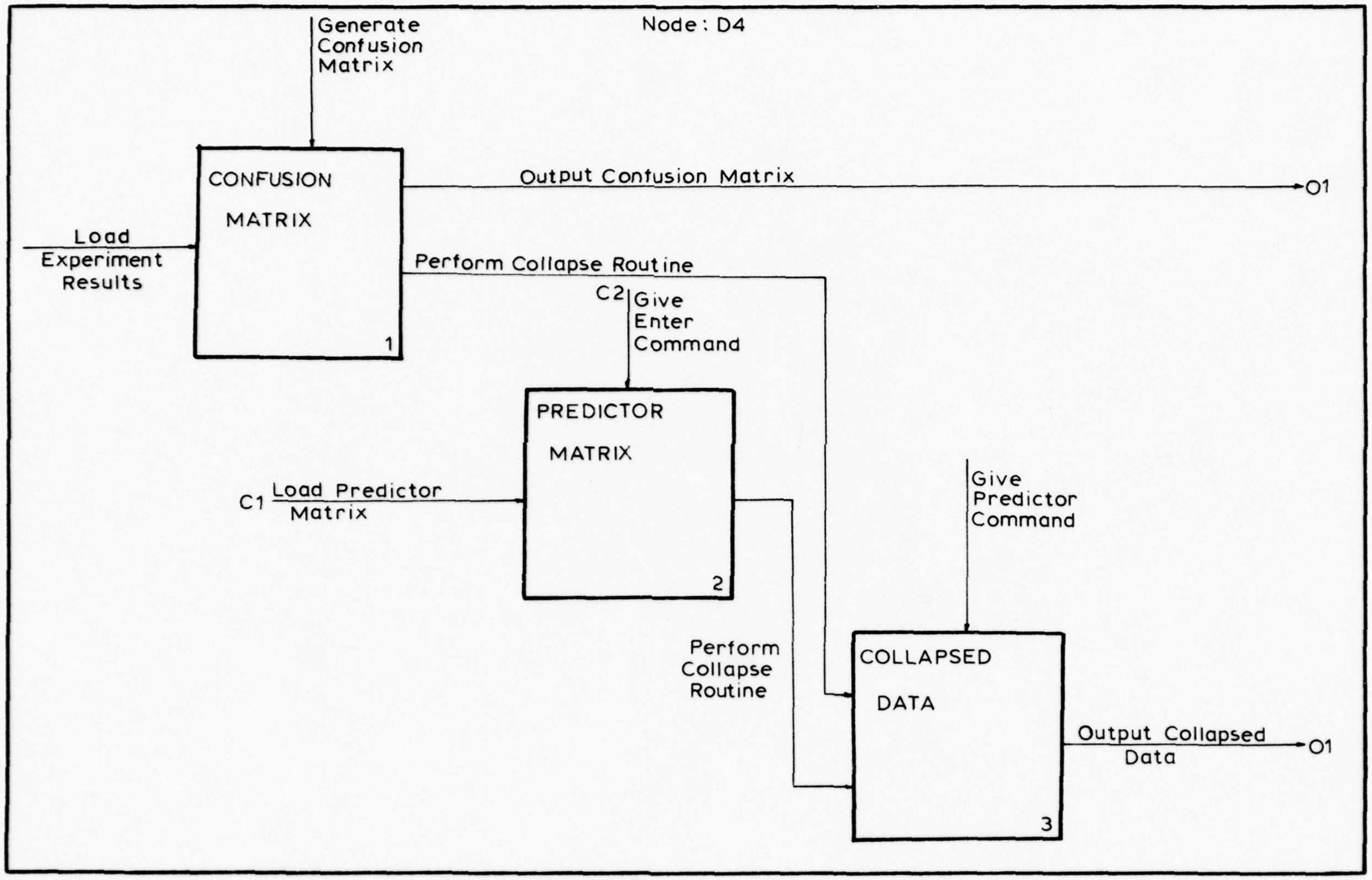


Figure 19. Analysis Data

D4 Text. Confusion Matrices (1) are created by the generate confusion matrix activity (1C1) which uses the experiment results (1I1). The resulting confusion matrix (1) can be outputted (1O1) or it can be manipulated by the performance of a collapsing routine (1O2) to produce (3).

The Predictor Matrix (2) is loaded (2I1) subject to the user ENTER command (2C1). This data is also used in performing a collapsing routine (2O1) to produce (3).

The performance of a collapsing routine using (1) and (2) produces the Collapsed Data (3) according to the option specified by the predictor command (3C1).

Summary

This chapter has presented the functional model of the system, which is the basis of the software design presented in Chapter iV.

IV. DESIGN

Introduction

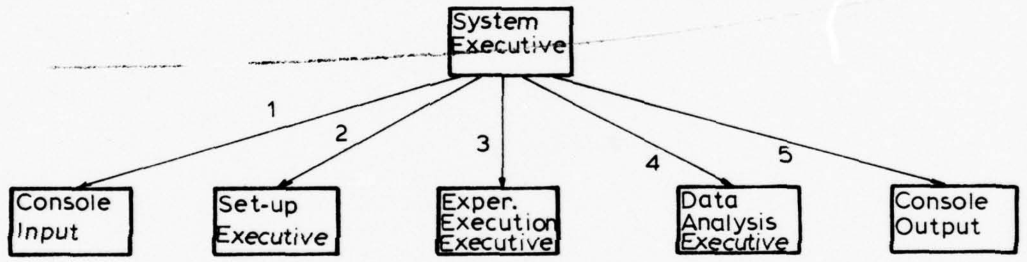
The purpose of the design phase in a development project is to definitize the functions and operations necessary to satisfy the system requirements. As discussed in Chapter I, this investigation does not present a complete design phase. Rather, the intent is to design the software structure which will be used in implementing the system.

The approach which is used in this investigation is called structured design (Ref 5). This design method attempts to reduce software complexity to make coding, debugging and modification easier, faster and less expensive. The symplicity of the design which is developed is enhanced by dividing the software into separate pieces (modules) in such a way that modules can be implemented and modified with minimal consideration or effect on the other parts of the system.

The remainder of this chapter uses a graphical tool called structured charts (Ref 7:547-587) to present the software structural design which was developed. The interfaces between the modules of the design are also given, together with a functional description of each module. Where appropriate, the module descriptions point out any anticipated problems which may be encountered in implementing the module.

Structure Charts

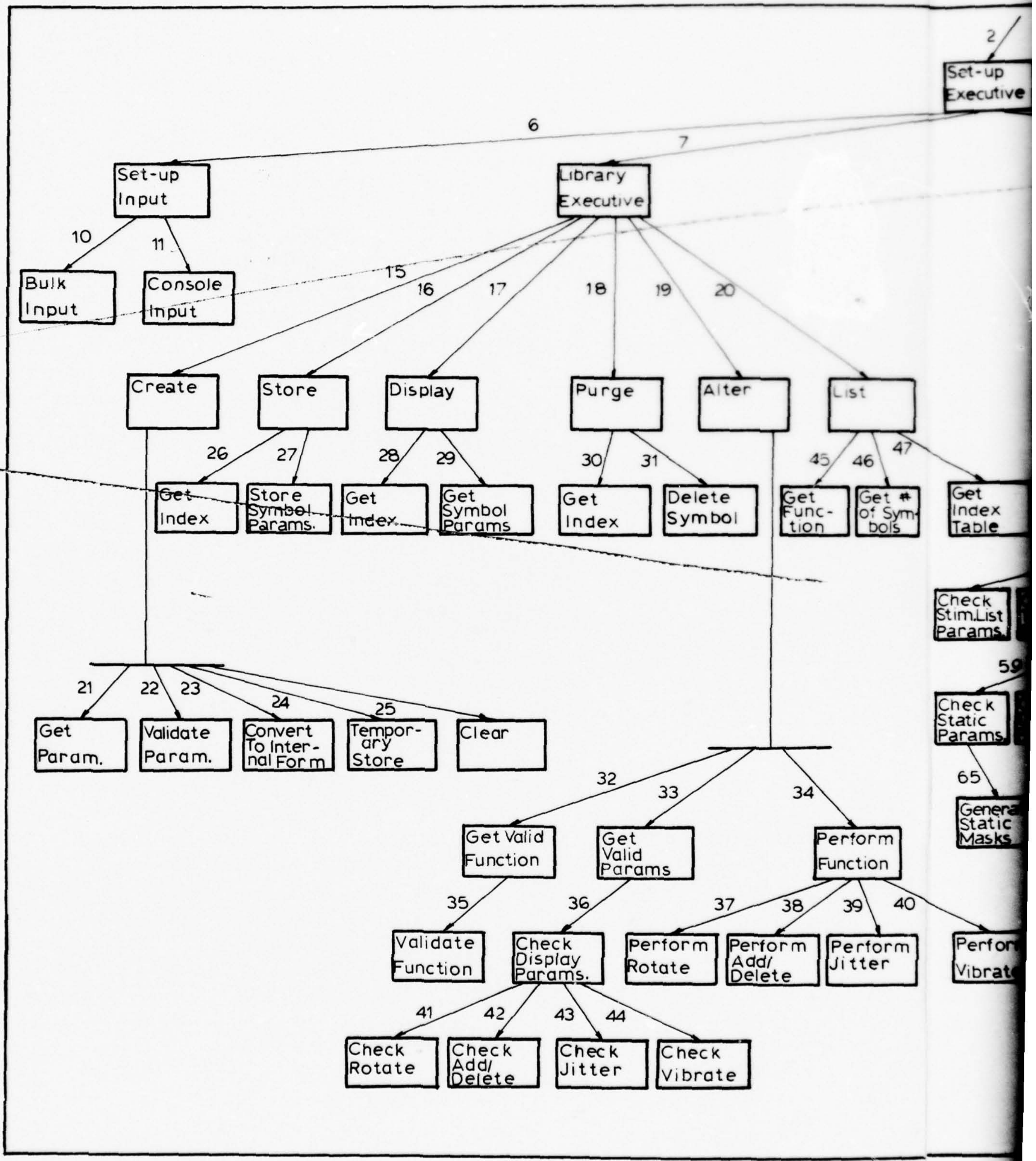
The structure charts for the complete software design are shown in Figures 20 thru 23. These figures present the structure of the system executive, together with the structure of each of the system's three basic modes of operations (Set-Up, Experiment Execution and Data Analysis). The following sections of this chapter discuss the details of each part of the software structure.



	Interface	
	IN	OUT
1	---	system commands
2	set-up command	set-up parameters
3	set-up parameters	experiment results
4	experiment results	analysis results
5	error messages	---

Figure 20. System Executive Structure

System Executive. When the system is initiated, the SYSTEM-EXECUTIVE outputs a message to the user stating that the system is executing the executive, and requesting the user to input the desired mode of operation. If the user responds with any input other than SET-UP, EXPERIMENT EXE-



2
Set-up Executive
6
7
Set-up Input
Library Executive
10
11
Bulk Input
Console Input
15
16
17
18
19
20
Create
Store
Display
Purge
Alter
List
26
27
28
29
30
31
45
46
47
Get Index
Store Symbol Params.
Get Index
Get Symbol Params
Get Index
Delete Symbol
Get Function
Get # of Symbols
Get Index Table
Check Stim.List Params.
59
Check Static Params.
65
Genera Static Masks
21
22
23
24
25
Get Param.
Validate Param.
Convert To Internal Form
Temporary Store
Clear
32
33
34
Get Valid Function
Get Valid Params
Perform Function
35
36
37
38
39
40
Validate Function
Check Display Params.
Perform Rotate
Perform Add/ Delete
Perform Jitter
Perfor Vibrat
41
42
43
44
Check Rotate
Check Add/ Delete
Check Jitter
Check Vibrate

Fi

2

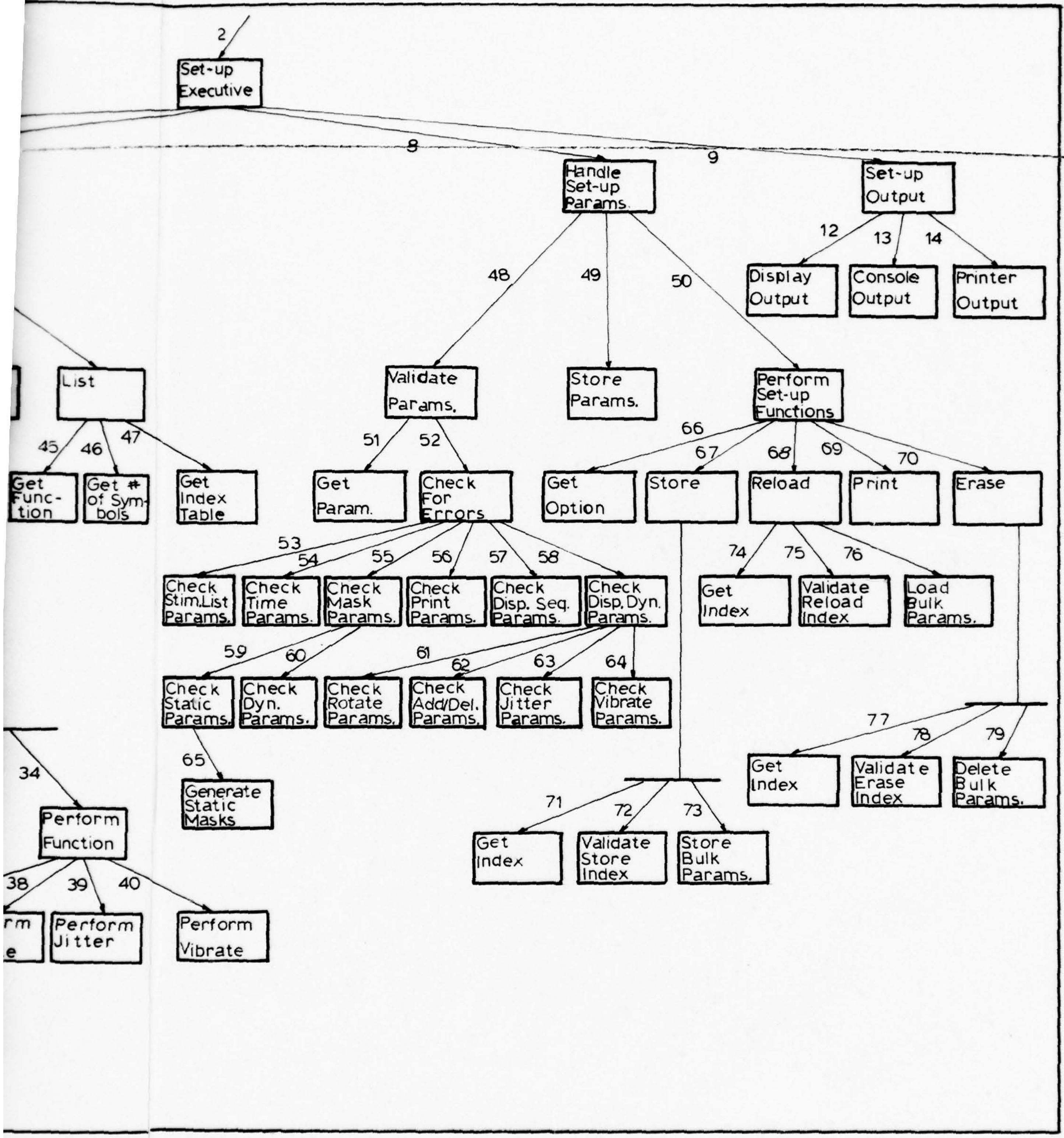


Figure 21. Structure of Set-Up Mode

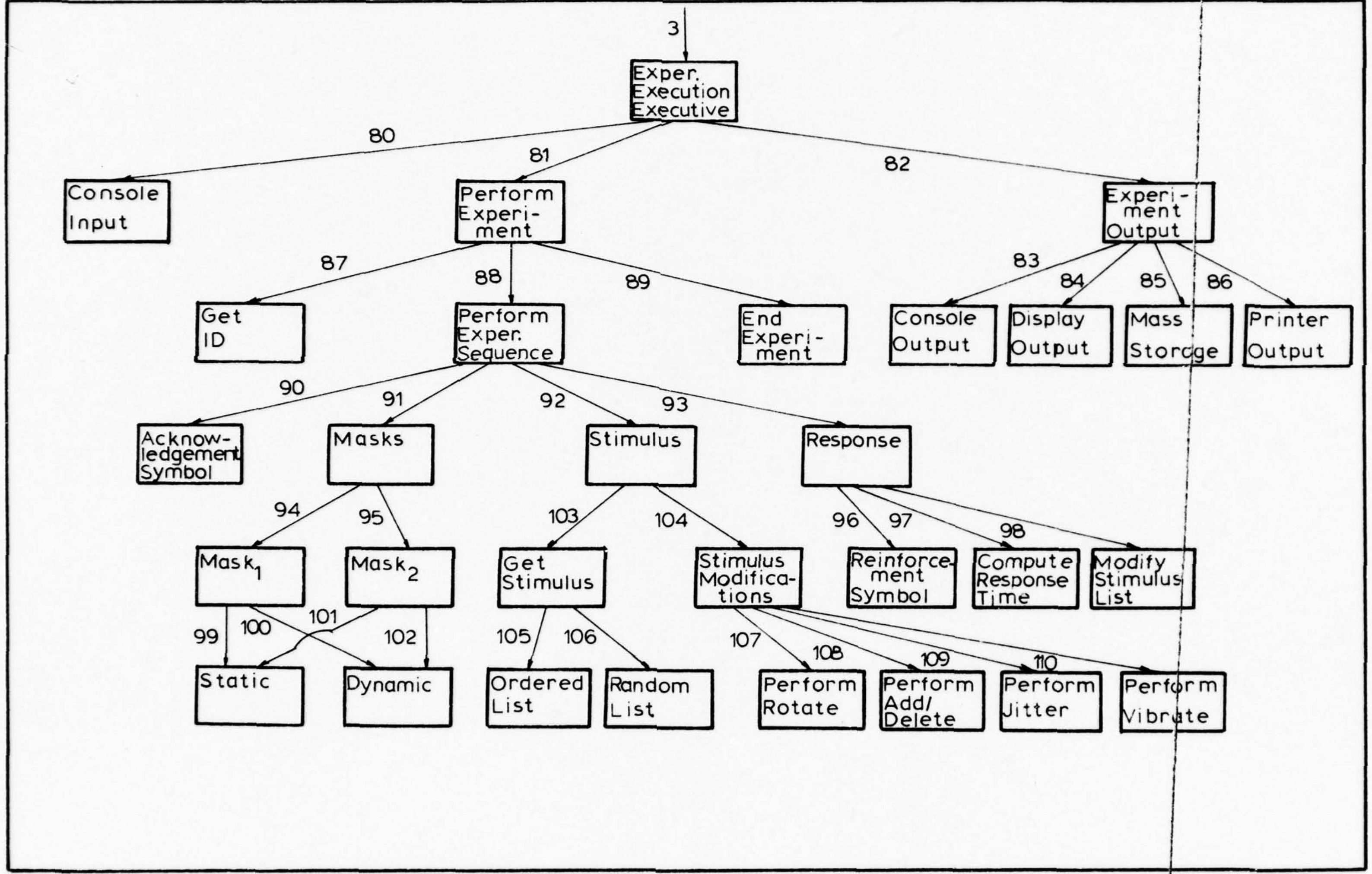


Figure 22. Structure of Experiment Execution Mode

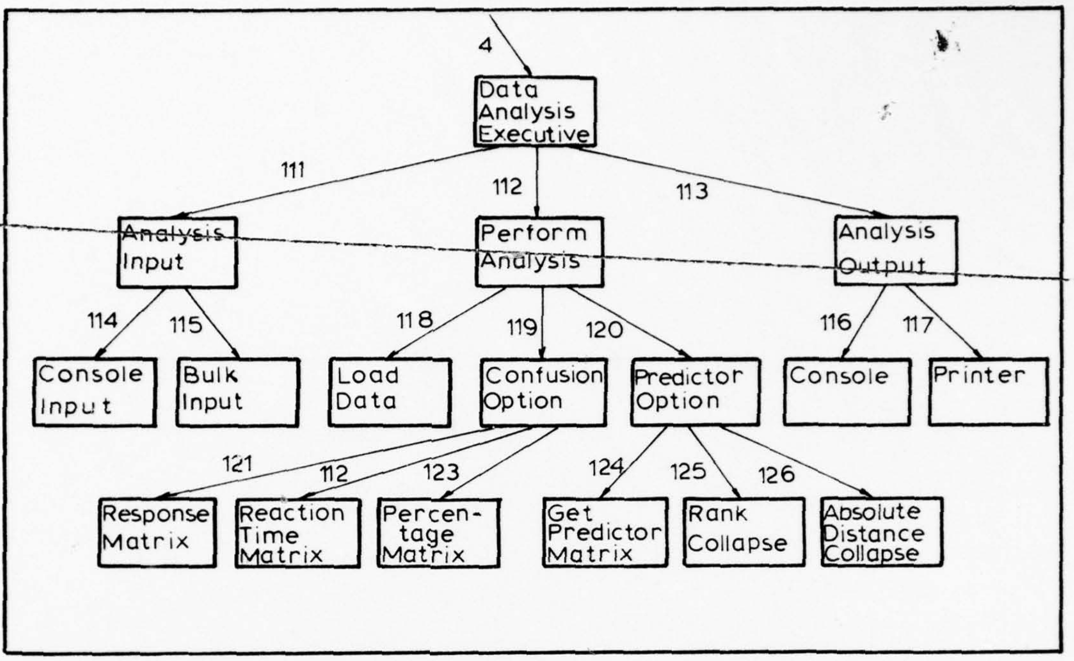
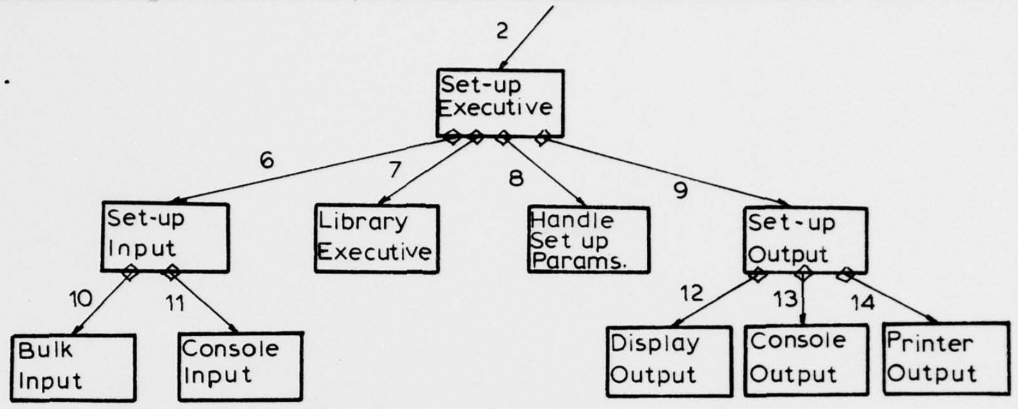


Figure 23. Structure of Data Analysis Mode

CUTION or DATA ANALYSIS, the executive will output an error message and request the user to reenter the operation mode. When a valid operation mode is inputted, the SYSTEM-EXECUTIVE will invoke (call) either the SET-UP-EXECUTIVE, the EXPERIMENT-EXECUTION-EXECUTIVE or the DATA-ANALYSIS-EXECUTIVE, depending on the user input.

The CONSOLE-INPUT and CONSOLE-OUTPUT modules consist of the routines necessary to control I/O between the user terminal and the system, while the system is executing the SYSTEM-EXECUTIVE. It is anticipated that most of the required routines will be included in the operating system or other software which is obtained with the hardware system. The SET-UP-EXECUTIVE, EXPERIMENT-EXECUTION-EXECUTIVE and DATA-ANALYSIS-EXECUTIVE are discussed separately below.

Set-Up Mode. When invoked, the SET-UP-EXECUTIVE (Figure 24) outputs a message to the user informing him that



Interface

	IN	OUT
6	input type	set-up mode commands and parameters
7	library commands and parameters	messages to user, display outputs
8	set-up commands and parameters	messages to user
9	output type, messages to user	---
10	---	bulk inputs
11	---	console inputs
12	display outputs	---
13	console outputs	---
14	printer outputs	---

Figure 24. Set-Up Executive

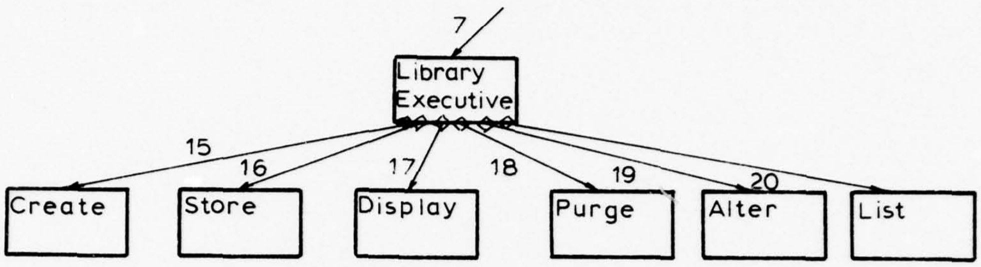
the system is in the Set-Up mode. The user may then choose one of the options shown in Table I. If the user enters any other command, the SET-UP-EXECUTIVE will output an error message and request the user to enter another command. The SET-UP-EXECUTIVE returns control to the SYSTEM-EXECU-

TIVE if the user enters the END SET-UP command. If the user enters the LIBRARY command, the SET-UP-EXECUTIVE passes control of the system to the LIBRARY-EXECUTIVE. For any of the other commands in Table I, the HANDLE-SET-UP-PARAMETERS module is invoked.

When a Set-Up mode module requires an I/O operation, an identifier for the I/O device is passed to the SET-UP-INPUT or SET-UP-OUTPUT modules. In the case of an output operation, the data to be outputted is also passed to the SET-UP-OUTPUT module. The SET-UP-INPUT and SET-UP-OUTPUT modules then invoke the appropriate subordinate modules to perform the I/O operation. The subordinate modules contain the I/O handlers for the particular device they control. The DISPLAY-OUTPUT module will manipulate the contents of a buffer whose contents will be continuously mapped onto the display device by a hardware display driver.

The LIBRARY-EXECUTIVE (Figure 25) is invoked when the SET-UP-EXECUTIVE receives the LIBRARY command from the user. The LIBRARY-EXECUTIVE responds by requesting the user to choose one of the following options: Create, Store, Display, Purge, Alter, List or End. If the user requests any other function, the LIBRARY-EXECUTIVE will output an error message and repeat the request. If the user enters the END function, the LIBRARY-EXECUTIVE will return control to the SET-UP-EXECUTIVE. For any of the other functions, the LIBRARY-EXECUTIVE invokes the appropriate subordinate module, and passes to that module any function parameters which have been inputted.

For the CREATE command, the LIBRARY-EXECUTIVE invokes



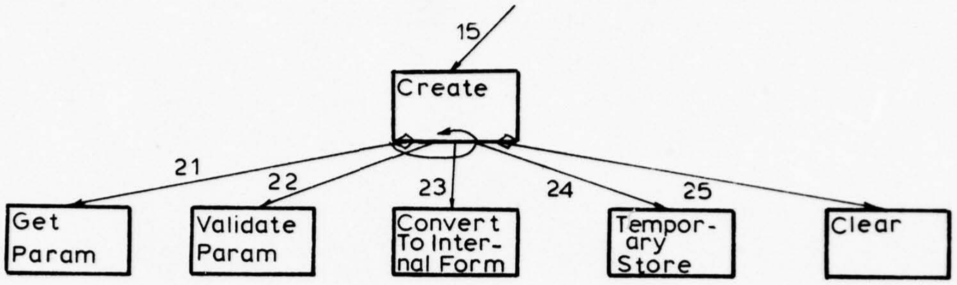
Interface

	IN	OUT
15	create parameters	display code, user message
16	index, symbol params.	user messages
17	index	symbol params, user message
18	index	user messages
19	alter function, params.	display output, user mess.
20	list function	user messages, index table, number of indexes

Figure 25. Library Mode Executive

the CREATE module. The CREATE module and its subordinates (Figure 26) allow the user to enter the parameters which define a symbol, and to output that symbol on the display as it is being created. When invoked, the CREATE module checks to determine if the user has entered any parameters with the CREATE command. If no parameters have been entered, the GET-PARAMETER module is called to request the user to enter a parameter. Only one parameter is passed to the VALIDATE-PARAMETER module each time it is called, so if the user has entered a series of parameters, the VA-

LIDATE-PARAMETER module will be invoked once for each parameter in the series. If a parameter is invalid, the VALI-



Interface

	IN	OUT
21	---	user messages
22	parameter	user message, valid param.
23	valid parameter	display driver code
24	display driver code	---
25	---	clear display command

Figure 26. Create Module and Subordinates

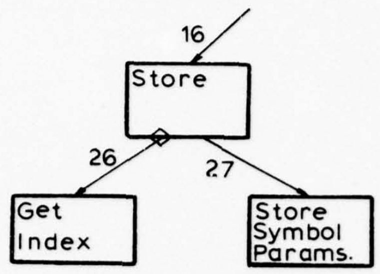
DATE-PARAMETER module returns an error message.

A valid CREATE command parameter is passed to the CONVERT-TO-INTERNAL-FORM module. Although the exact symbol representation which will be used is not known, it is assumed that the user inputs will have to be converted to a different form to be used by the DISPLAY-OUTPUT module, and to be stored in the symbol library. The converted parameter is passed to the DISPLAY-OUTPUT module and to the TEMPORARY-STORE module.

The TEMPORARY-STORE module accumulates the individual converted CREATE parameters to define a complete symbol. The accumulated parameters (called Symbol Data) are treated as an abstract data type (discussed in the last section of this chapter). Abstract data types use the "information hiding" approach by considering a data structure and its accessing and modifying procedures as a separate module (Ref 2:1056). Thus, the TEMPORARY-STORE module invokes the TEMPORARY-SYMBOL-STORE module (Figure 40) and passes it the symbol parameters which are to be added to the data structure.

After all of the CREATE command parameters have been processed, the CREATE module invokes the GET-PARAMETER module and repeats the series described above. If the user enters the CLEAR command, the CLEAR module is invoked to clear the Symbol Data Structure and to clear the display, in preparation for defining a new symbol. When the END command is entered, control is returned to the LIBRARY-EXECUTIVE.

The STORE module is called by the LIBRARY-EXECUTIVE when the user enters the STORE command. The purpose of this module and its subordinates (Figure 27) is to allow the user to enter into the symbol library a symbol which has been defined using the CREATE function. When the STORE module is invoked, it first checks to determine if the user has entered an index for the symbol. If an index has not been inputted, the GET-INDEX module is invoked to request the user to enter an index. After the index has been inputted, the STORE-SYMBOL-PARAMETERS module is called. This module invokes the TEMPORARY-SYMBOL-STORE



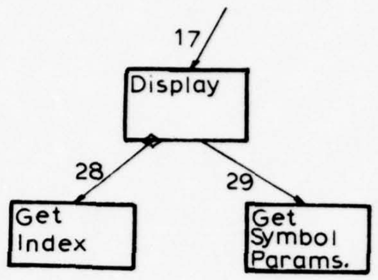
Interface

	IN	OUT
26	---	user message
27	index	error message

Figure 27. Store Module and Subordinates

module (Figure 40) to retrieve the Symbol Data. The Symbol Data and index are then passed to the LIBRARY-MANIPULATION module (Figure 41) to add the index and Symbol Data to the symbol library. (The symbol library and its index table are treated as an abstract data type). If the LIBRARY-MANIPULATION module returns an indicator that the index is already in use, the STORE-SYMBOL-PARAMETERS module will return a message to the user indicating that the STORE operation was not performed.

When the user wants to display a symbol that is stored in the symbol library, he inputs the DISPLAY command to the LIBRARY-EXECUTIVE. The LIBRARY-EXECUTIVE then invokes the DISPLAY module (Figure 28), which checks if the user has inputted a symbol index. If an index has not been inputted, the GET-INDEX module is called to output a message to the user requesting that an index be inputted. The index is passed to the GET-SYMBOL-PARAMETERS module which uses the LIBRARY-MANIPULATION module (Figure 41) to fetch the symbol parameters from the library. If the LIBRARY-



Interface

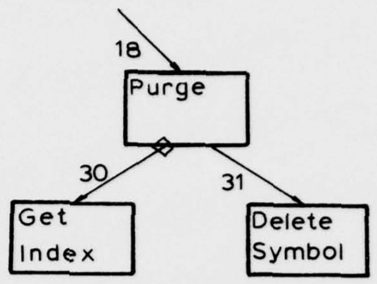
	IN	OUT
28	---	request for index
29	index	error message, symbol parameters

Figure 28. Display Module and Subordinates

MANIPULATION module returns an indicator that the index was not found in the library, the GET-SYMBOL-PARAMETERS module outputs an error message to the user. The symbol parameters are passed to the SET-UP-OUTPUT module, which displays the symbol.

The LIBRARY-EXECUTIVE invokes the PURGE module (Figure 29) when the user enters the PURGE command. This module checks to determine if the user has entered an index, and if he has not, the GET-INDEX module is called to output a request for an index. The index is passed to the DELETE-SYMBOL module which passes the index and a purge function to the LIBRARY-MANIPULATION module (Figure 41). If the index is not in the library, the DELETE-SYMBOL module returns an error message to the user.

If the user wants to test the effects of symbol dis-



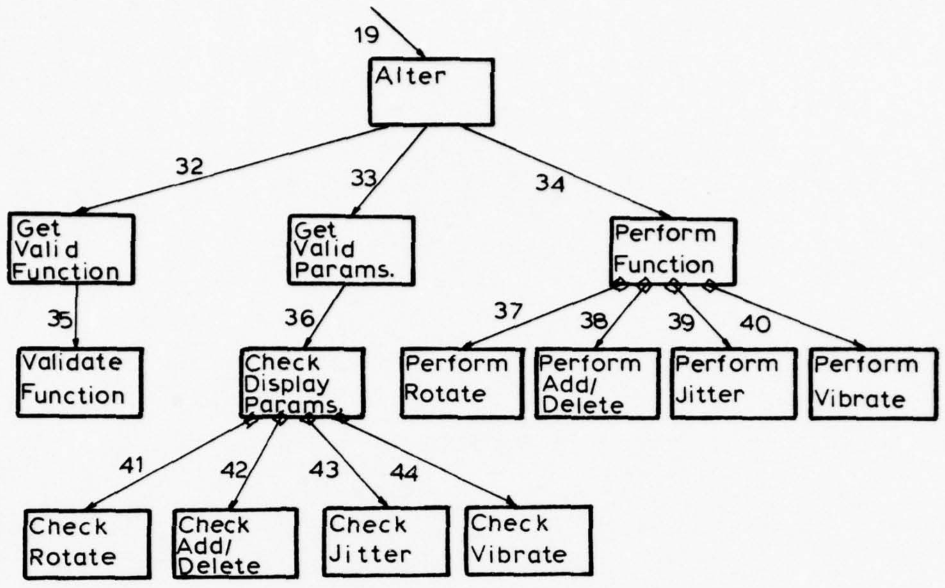
Interface

	IN	OUT
30	---	user message
31	index	error message

Figure 29. Purge Module and Subordinates

play dynamics on the symbol which is currently being displayed, he inputs the ALTER command to the LIBRARY-EXECUTIVE, which invokes the ALTER module (Figure 30). The ALTER module checks if an alter function has been inputted, and calls the GET-VALID-FUNCTION module to output a request to the user if a function has not been inputted. The GET-VALID-FUNCTION module calls VALIDATE-FUNCTION to check if a legal function has been inputted, and returns an error message if the function is invalid.

After a valid function has been inputted, the GET-VALID-FUNCTION module requests the user to enter the function parameters, and invokes the CHECK-SYMBOL-DISPLAY-DYNAMICS-PARAMETERS to determine if the parameters are legal. If the parameters are illegal, GET-VALID-PARAMETERS requests the user to reenter the parameters.



	Interface IN	OUT
32	function	user messages, valid func.
33	valid func. & params	user messages, valid params
34	valid func. & params	display driver code
35	function	error message
36	params, valid func.	error message, valid params
37	valid rotate params	display driver code
38	valid add/delete params	display driver code
39	valid jitter params	display driver code
40	valid rotate params	display driver code
41	rotate params	error message, valid params
42	add/delete params	error message, valid params
43	jitter params	error message, valid params
44	vibrate params	error message, valid params

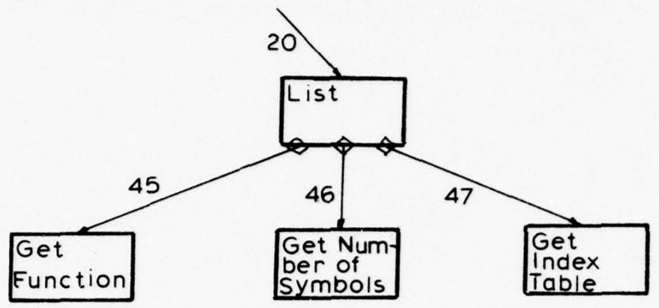
Figure 30. Alter Module and Subordinates

When the user has entered a legal function, with valid parameters, the PERFORM-FUNCTION module is invoked, which in turn calls the appropriate subordinate module to perform the function. The PERFORM-ROTATE, PERFORM-ADD/DELETE, PERFORM-JITTER and PERFORM-VIBRATE modules then output the display driver code to manipulate the display to perform the function.

After an alter function has been processed, the ALTER module repeats the above sequence, thus allowing the user to perform a combination of functions simultaneously. The ALTER module continues to request and process alter functions until the <u>END ALTER</u> command is inputted, which discontinues the alter function operations and returns control to the LIBRARY-EXECUTIVE.

The PERFORM-ROTATE, PERFORM-ADD/DELETE, PERFORM-JITTER and PERFORM-VIBRATE modules handle the operations discussed in Chapter II. The algorithms which must be performed by the modules have not been investigated, primarily because the hardware drivers which will be used have not been defined. Although each alter function is represented by a single module, these modules are very complex, requiring many subordinate modules for each function. In addition, the problems associated with performing simultaneous functions have not been investigated.

The final library command to be discussed is the <u>LIST</u> command. This command has options to output the number of symbols in the symbol library or to output the indexes of the symbols in the symbol library. When the <u>LIST</u> command is inputted, the LIBRARY-EXECUTIVE invokes the LIST module (Figure 31), which checks if the user has inputted an option. The LIST module invokes the GET-NUMBER-OF-SYMBOLS or GET-INDEX-TABLE module depending on the chosen option, or returns an error message if an invalid option is cho-



Interface

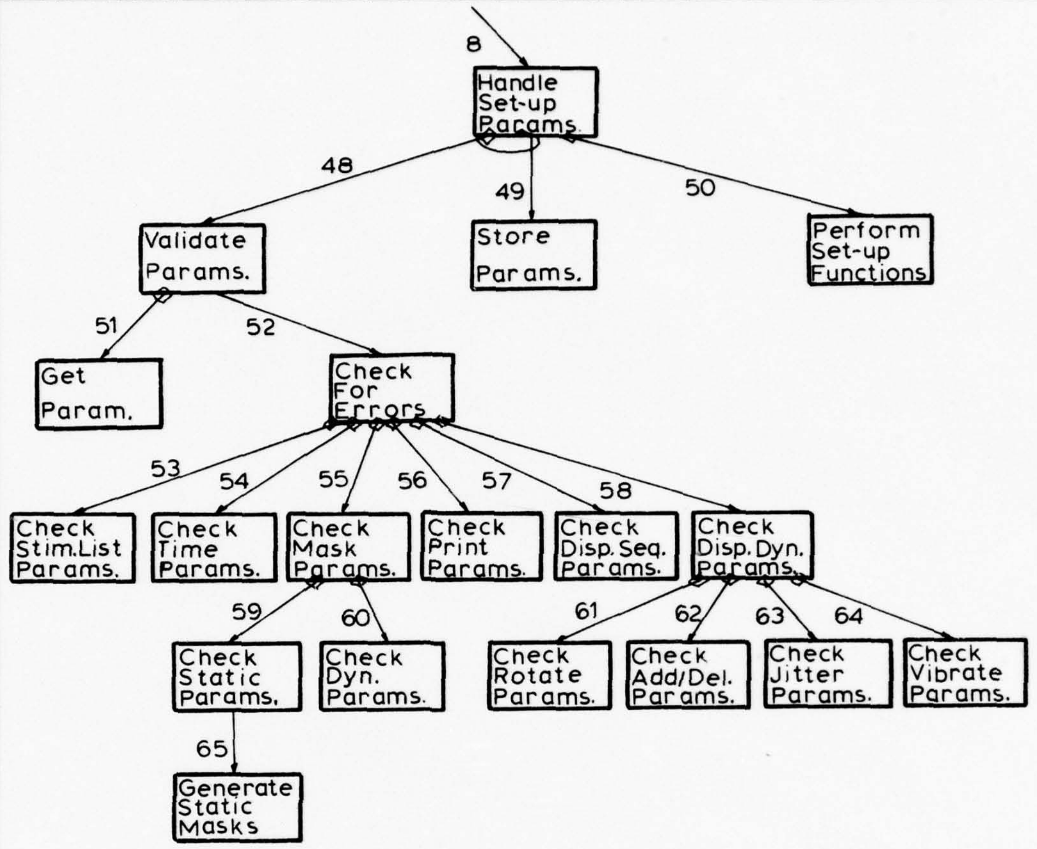
	IN	OUT
45	---	user message
46	---	number of indexes
47	---	index table

Figure 31. List Module

sen. The GET-NUMBER-OF-SYMBOLS and GET-INDEX-TABLE both use the LIBRARY-MANIPULATION module (Figure 41) to perform their functions.

The HANDLE-SET-UP-PARAMETERS module (Figure 31) is invoked when the user enters a command from Table I other than LIBRARY or END SET-UP. This module and its subordinates validate and store the set-up parameters which will be used during the execution of an experiment, and perform the set-up parameters functions.

For a command other than SET-UP-PARAMETERS, the VALIDATE-PARAMETERS module is invoked. This module first determines if the user has entered the parameters for the command. If the parameters have not been entered, the



Interface

	IN	OUT
48	valid command, params	err mess, params, stat masks
49	stat masks, comm, param	---
50	function, index	err mess, list of params
51	---	user message
52	valid command, params	err mess, params, stat masks
53	stimulus list params	err mess, valid params
54	time params	err mess, valid params
55	mask type & params	err mess, params, stat masks
56	print params	err mess, valid params
57	display seq params	err mess, valid params
58	sym disp dyn params	err mess, valid params
59	static mask params	err mess, static masks
60	dynamic mask params	err mess, valid params
61	rotate params	err mess, valid params
62	add/delete params	err mess, valid params
63	jitter params	err mess, valid params
64	vibrate params	err mess, valid params
65	valid static params	static masks

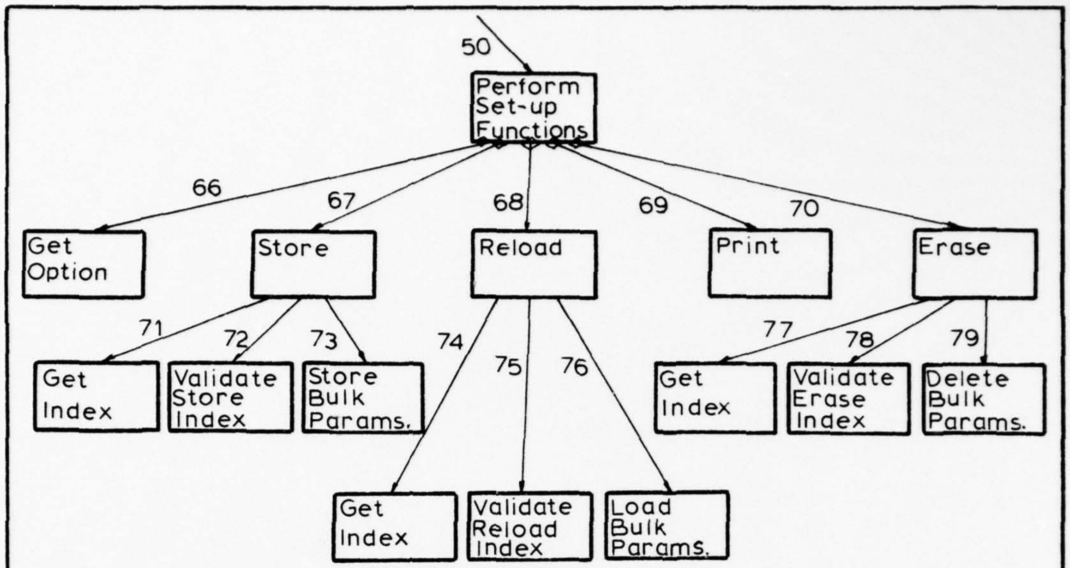
Figure 32. Handle Set-Up Parameters Module

GET-PARAMETERS module is called to request the user to input the parameters. The command and parameters are passed to the CHECK-FOR-ERRORS module. This module invokes the appropriate subordinate module to validate the command parameters.

Valid set-up parameters are treated as an abstract data type (called Current Set-Up Parameters) as discussed at the end of this chapter. The valid commands and parameters are passed to the STORE-PARAMETERS module which invokes the SET-UP-PARAMETERS-MANIPULATION module to enter the parameters into the Current Set-Up Parameters data structure.

The CHECK-STIMULUS-LIST-PARAMETERS module uses the LIBRARY-MANIPULATION module to check that the indexes which are inputted for the stimulus list are contained in the symbol library. At this point in the design it has not been determined if the stimulus list will only contain the indexes for the symbols, with the appropriate parameters being retrieved from the symbol library when the stimulus is needed.

As discussed in Chapter II, the system is required to generate and store static masks before beginning execution of an experiment. Thus, CHECK-STATIC-PARAMETERS invokes the GENERATE-STATIC-MASKS module if the mask parameters are valid, and this module produces the masks. The static masks are then passed to the STORE-PARAMETERS module which enters the masks into the Current Set-Up Parameters data structure.



Interface

	IN	OUT
66	---	user message
67	index	error message
68	index, bulk params	error message
69	---	current set-up params list
70	index	user message
71	---	user message
72	index	error message, valid index
73	valid index	---
74	---	error message
75	index	error message, valid index
76	valid index	---
77	---	user message
78	index	error message, valid index
79	valid index	---

Figure 33. Perform Set-Up Parameters Function Module

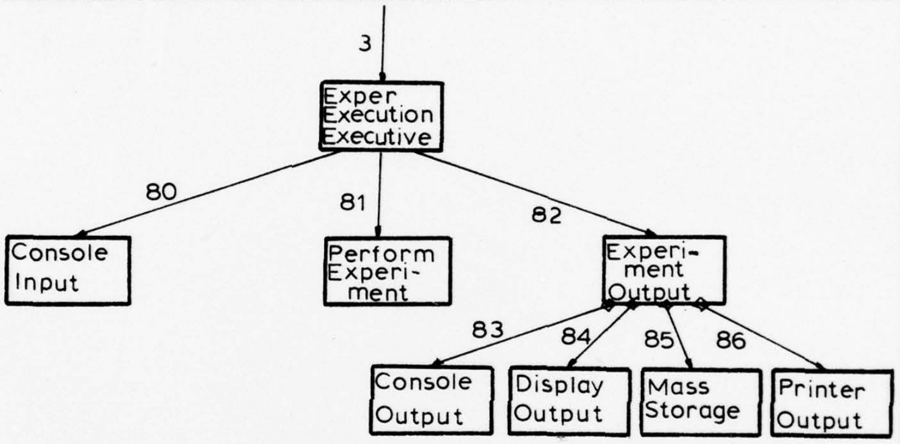
The PERFORM-SET-UP-FUNCTIONS module and its subordinates (Figure 33) execute the SET UP PARAMETERS commands which were discussed in Chapter II. When this module is invoked by the HANDLE-SET-UP-PARAMETERS module, it first checks if the user has inputted a command option. If an option has not been entered, GET-OPTION is invoked to output a request to the user.

The STORE module is called if the user chooses the STORE option. This module and its subordinates store the Current Set-Up Parameters in permanent storage. When called, the STORE module checks if the user has entered an index and uses the GET-INDEX module to request an index if one has not been entered. The index is passed to the VALIDATE-STORE-INDEX module which determines if the index is not already in use (this will probably be done using operating system file management routines). If the index is invalid, an error message is outputted to the user. For a valid index, the STORE-BULK-SET-UP-PARAMETERS module is invoked to store the parameters in a permanent file (again using operating system routines).

If the user enters the SET-UP PARAMETERS print option, the PRINT module is invoked. This module uses the SET-UP-PARAMETERS-MANIPULATION module to output the Current Set-Up Parameters to the printer. The RELOAD and ERASE modules recall and purge set-up parameters which were saved using the STORE option. Both of these modules (and their subordinates) function in a manner similar to the STORE module: they check if an index has been inputted, validate the index and then the operation is performed using operating system routines. In addition, the RELOAD module uses the

SET-UP-PARAMETERS-MANIPULATION module to enter the retrieved parameters into the Current Set-Up Parameters data structure.

Experiment Execution Mode. When the user wants to execute an experiment, he inputs the EXPERIMENT EXECUTION command to the SYSTEM-EXECUTIVE, which then invokes the EXPERIMENT-EXECUTION-EXECUTIVE (Figure 34). This module



Interface

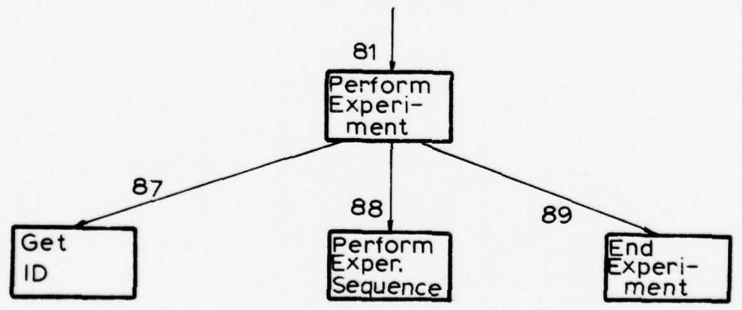
	IN	OUT
80	---	subject ID & response
81	subject response	subject messages, experiment results
82	messages, ID info, disp driver code, exp results	---
83	messages	---
84	display driver code	---
85	ID info, exper results	---
86	ID info, exper results	---

Figure 34. Experiment Execution Executive and Subordinates

outputs a message to the user stating that the system is in the Experiment Execution mode. The PERFORM-EXPERIMENT module is then called to handle execution of the experiment. While the system is in the Experiment Execution mode, the CONSOLE-INPUT and EXPERIMENT-OUTPUT modules manage the system I/O.

In addition to user messages, the EXPERIMENT-OUTPUT module is also passed the experiment results, which consists of the stimulus displayed, subject response and subject response time information. This information is passed from the PERFORM-EXPERIMENT module to the EXPERIMENT-OUTPUT module for each stimulus displayed during the experiment. The EXPERIMENT-OUTPUT module passes the experiment result information to the MASS-STORAGE module and to the PRINTER-OUTPUT module (if this option is specified in the set-up parameters).

When the PERFORM-EXPERIMENT module (Figure 35) is in-



Interface

	IN	OUT
87	---	ID request
88	subject response	stimulus, masks, experiment results
89	---	end experiment message

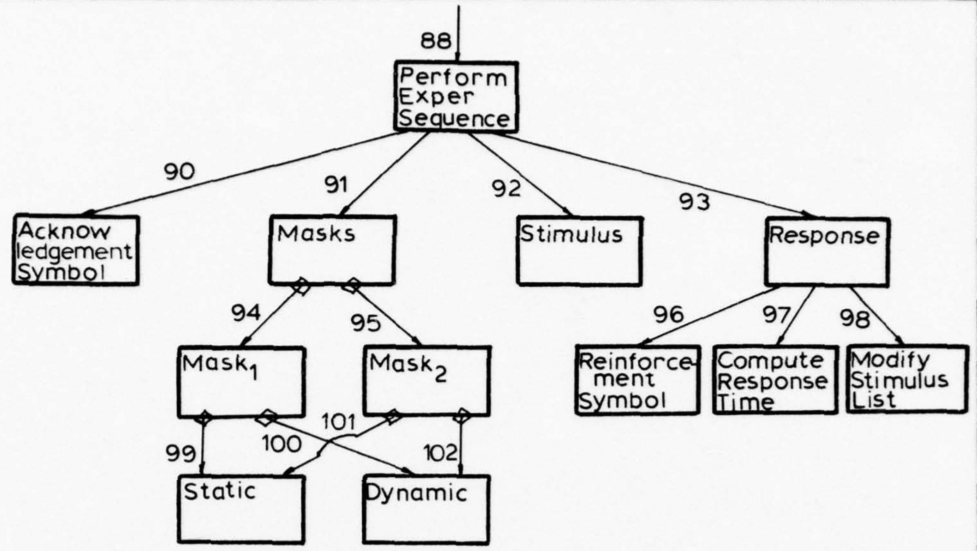
Figure 35. Perform Experiment Module

voked, it calls the GET-ID module to request the subject to input ID information (name, date, etc.). This request will be displayed on the console until the ID information is entered, or the Experiment Execution mode is terminated by inputting the END EXPERIMENT MODE command. After the subject enters the ID information, the PERFORM-EXPERIMENT-SEQUENCE module is invoked. This module controls the experiment until the subject enters the STOP command, at which time the END-EXPERIMENT module will be invoked to output an "end of experiment" message. The GET-ID module will then be invoked again, and the above sequence will be repeated for the next subject.

The PERFORM-EXPERIMENT-SEQUENCE module and its subordinates (Figure 36) use the set-up parameters (which were created in the Set-Up mode) to control the experiment. These modules all have access to the Current Set-Up Parameters data by using the SET-UP-PARAMETERS-MANIPULATION module which is discussed at the end of this chapter.

When invoked by the PERFORM-EXPERIMENT module, the PERFORM-EXPERIMENT-SEQUENCE module calls the ACKNOWLEDGEMENT-SYMBOL module to output the acknowledgement symbol, indicating that the experiment sequence is about to begin. The MASKS module is then called to output the first mask. The MASKS module and its subordinates recall the stored static masks or generate dynamic masks, depending on the set-up parameters.

After the first mask is displayed, the STIMULUS mo-



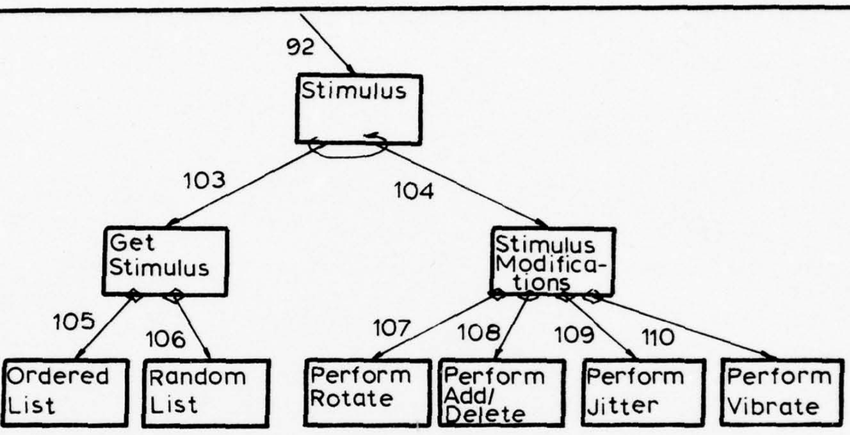
Interface

	IN	OUT
90	---	acknowledgement symbol
91	mask required (1 or 2)	$mask_1$, $mask_2$
92	---	stimulus
93	response, stimulus	response time, reinforce-ment symbol
94	---	$mask_1$
95	---	$mask_2$
96	---	reinforcement symbol
97	response	response time
98	stimulus, response	---
99	---	static mask
100	---	dynamic mask
101	---	static mask
102	---	dynamic mask

Figure 36. Perform Experiment Sequence and Subordinates

dule is invoked to output the appropriate stimulus. The MASKS module is then called to output the second mask. Next, the RESPONSE module is called to output the reinforcement symbol (using the REINFORCEMENT-SYMBOL module), record the subject's response and response time (by invoking COMPUTE-RESPONSE-TIME), and to make changes to the stimulus list according to the set-up parameters (by calling MODIFY-STIMULUS-LIST). The ACKNOWLEDGEMENT-SYMBOL module is then invoked to begin the next display sequence.

The STIMULUS module (Figure 37) invokes the GET-STI-



Interface

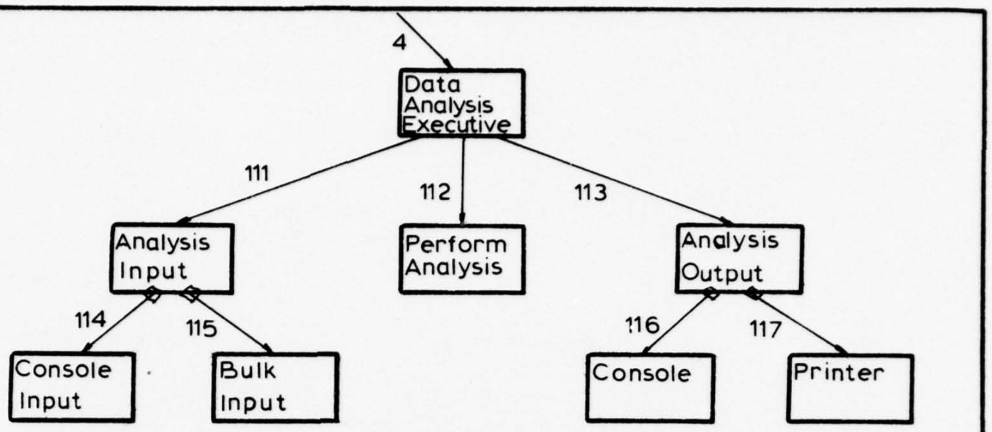
	IN	OUT
103	---	stimulus
104	stimulus	modified stimulus
105	---	stimulus
106	---	stimulus
107	stimulus	rotated stimulus
108	stimulus	add/deleted stimulus
109	stimulus	jittered stimulus
110	stimulus	vibrated stimulus

Figure 37. Stimulus Module and Subordinates

MULUS module, which checks the set-up parameters to determine the algorithm to use for selecting a stimulus from the stimulus list. The ORDERED-LIST or RANDOM-LIST module is then invoked to choose the stimulus. The stimulus is passed to the STIMULUS-MODIFICATIONS module. This module checks the set-up parameters to determine which symbol display dynamics should be performed.

Although the algorithm to be used by the RANDOM-LIST module (to select a stimulus from the stimulus list) has not been developed, this module should not be difficult to implement. However, as mentioned previously, the ROTATE, ADD/DELETE, JITTER and VIBRATE modules will probably be very complex and the problems associated with performing simultaneous display dynamics functions have not been investigated.

Data Analysis Mode. The Data Analysis mode allows the user to perform data reduction operations on experiment results. When the user inputs the DATA ANALYSIS command to the SYSTEM-EXECUTIVE, the DATA-ANALYSIS-EXECUTIVE (Figure 38) is invoked. This module then outputs a message to the user informing him that the system is in the Data Analysis mode. The ANALYSIS-INPUT module handles the input operations for the analysis mode. While in the Data Analysis mode, the system receives user commands from the console (handled by the CONSOLE-INPUT module) and also inputs experiment results and predictor matrices (handled by the BULK-INPUT module) as discussed in Chapter II. Outputs from the Data Analysis mode are handled by the ANALYSIS-OUTPUT module. These outputs consist of requests to the user and bulk outputs which are handled by the CONSOLE and PRINTER modules, respectively.



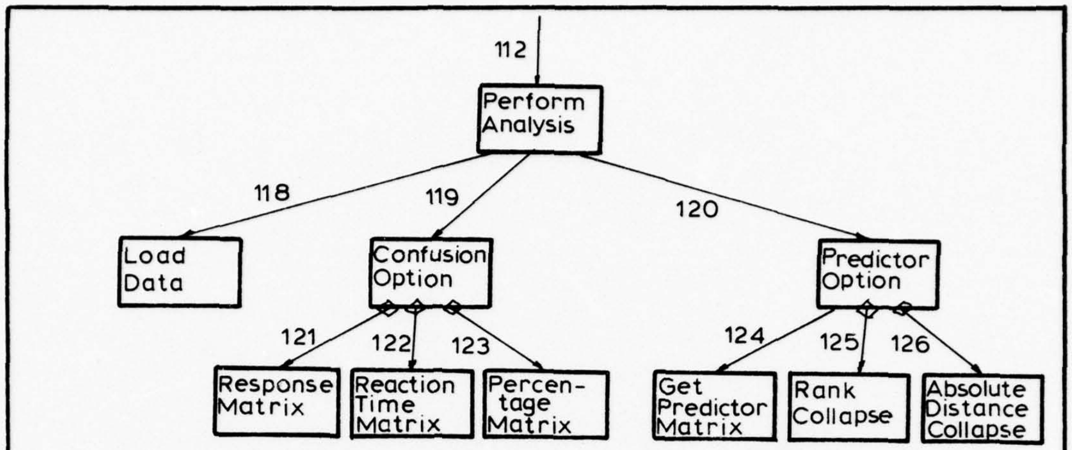
Interface

	IN	OUT
111	input type	console & bulk inputs
112	exp results, predictor matrix, user commands	user mess, analysis results
113	analysis results, user messages	---
114	---	console input
115	---	bulk input
116	user messages	---
117	analysis results	---

Figure 38. Data Analysis Executive

After the DATA-ANALYSIS-EXECUTIVE has informed the user of the system operation mode, it invokes the PERFORM-ANALYSIS module (Figure 39). This module invokes the LOAD-DATA module which requests the user to input the ID for the experiment results to be analyzed. The LOAD-DATA module then retrieves the appropriate experiment results. The CONFUSION-OPTION module is then invoked.

The CONFUSION-OPTION module requests the user to input the function to be performed, and returns an error message if an invalid input is given. The CONFUSION-OPTION



Interface

	IN	OUT
118	results ID	user mess, exper results
119	exper results, command parameters	confusion matrix, user messages
120	confusion & predictor matrix, command params	collapse results, user messages
121	command params, experiment results	response confusion matrix
122	command params, experiment results	reaction time confusion matrix
123	command params, experiment results	percentage confusion matrix
124	---	user mess, predict matrix
125	command params, predict & confusion matrix	collapse results
126	command params, predict & confusion matrix	collapse results

Figure 39. Perform Analysis Module and Subordinates

module then calls the appropriate subordinate module to perform the desired function. These subordinate modules contain the routines to generate the confusion matrices discussed in Chapter II. If the user inputs the PRINT command, the generated confusion matrix is outputted to the printer.

If the user enters the PREDICTOR command, the PREDICTOR-OPTION module is invoked. This module invokes the GET-PREDICTOR-MATRIX module to request the user to input a predictor matrix. After the predictor matrix is inputted, the PREDICTOR-OPTION module requests the user to input the type of collapsing routine to be performed, and the associated parameters. The RANK-COLLAPSE or ABSOLUTE-DISTANCE-COLLAPSE module is called to perform the collapsing routine and output the results.

Abstract Data Types. This section presents the structure charts and descriptions of the abstract data types introduced in the preceding sections. An abstract data type combines the data structure and its accessing and modifying modules so that all the operations on the data structure are performed through a single module.

The three abstract data types used in this design are the Symbol Data, the Symbol Library and the Current Set-Up Parameters. Symbol Data contains the symbol parameters which are accumulated using the CREATE command during the Set-Up mode. The Symbol Library contains the indexes and symbol parameters of the symbols which the user has saved with the library mode STORE command. The Current Set-Up Parameters data structure contains the parameters which are inputted during the Set-Up mode for use during the execution of an experiment. A summary of the operations (and the corresponding input parameters) which can be performed on the abstract data types appears in Table II.

Table II

ABSTRACT DATA TYPE OPERATIONS

Symbol Data

OPERATION	REQUIRED INPUT PARAMETERS
Add	symbol parameters
Fetch	---
Clear	---

Symbol Library

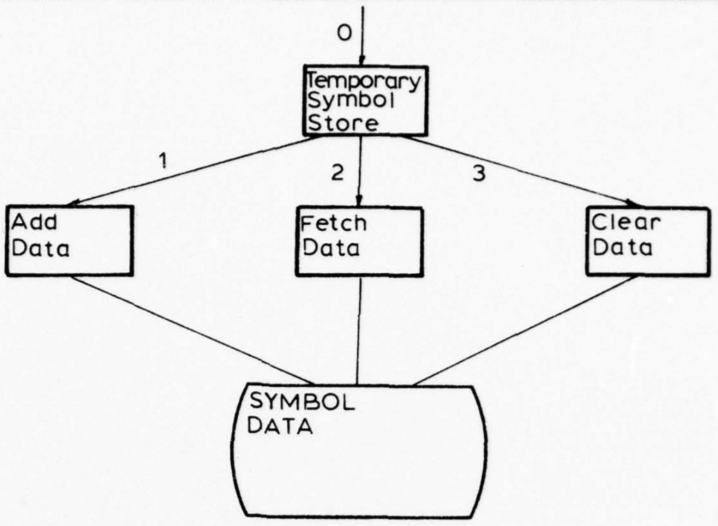
OPERATION	REQUIRED INPUT PARAMETERS
Store	symbol index, symbol parameters
Purge	symbol index
Recall	symbol index
Count	---
Fetch	---

Current Set-Up Parameters

OPERATION	REQUIRED INPUT PARAMETERS
Add	set-up parameters, parameter type
Fetch	parameter type
Reload	set-up parameters

The Symbol Data is accessed through the TEMPORARY-SYMBOL-STORE module (Figure 40). This module performs three operations on the Symbol Data: Add, Fetch and Clear. When the TEMPORARY-SYMBOL-STORE is called to perform the Add operation, the parameters which are passed from the calling module are entered into the data structure by the ADD-DATA module. The Fetch operation is performed by the FETCH-DATA module, which returns the contents of the data structure to the calling module. The Clear operation allows the calling module to delete the contents of the Symbol Data.

The LIBRARY-MANIPULATION module and its subordinates (Figure 41) manage the operations which can be performed



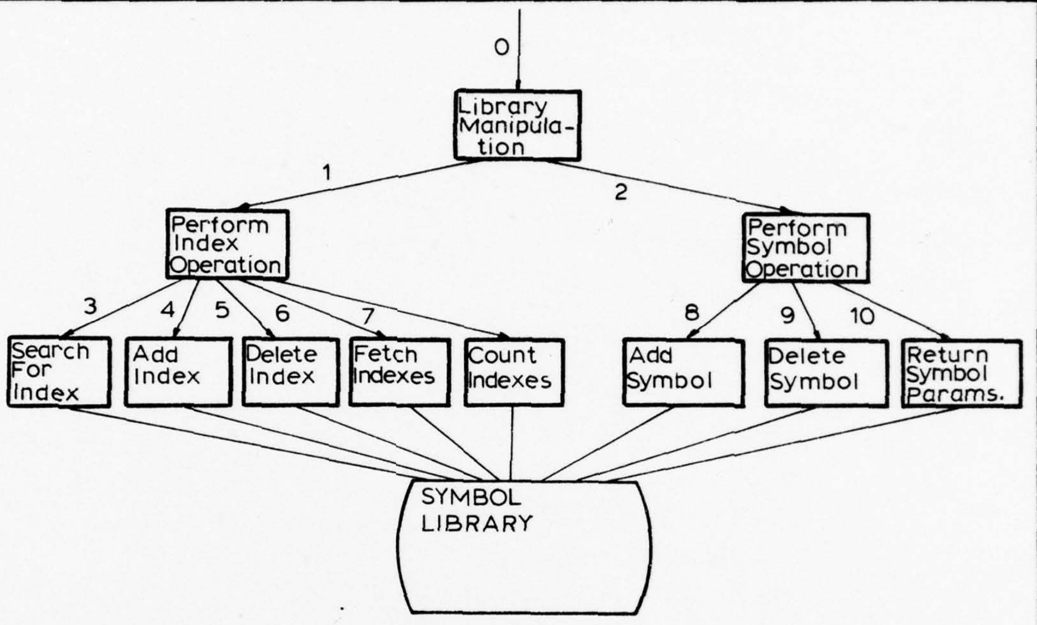
Interface

		IN	OUT
0	operation, parameters		symbol data
1	symbol parameters		---
2	---		symbol data
3	---		---

Figure 40. Temporary Symbol Store Module

on the Symbol Library and its index table. The operations that a calling module can request are: Store or Purge a symbol index and corresponding symbol parameters, Recall the symbol parameters for a given index, Count the number of indexes in the index table and Fetch the index table.

For the Store operation, the calling program passes the symbol index and symbol parameters to LIBRARY-MANIPULATION. The LIBRARY-MANIPULATION module invokes the PERFORM-INDEX-OPERATION module with a request that the index be added to the index table. PERFORM-INDEX-OPERATION first searches the index table (using SEARCH-FOR-INDEX) and, if the index is found, returns an error condition to LIBRARY-



Interface

	IN	OUT
0	operation, index, params	error cond, params, indexes
1	operation, index	error cond, param location
2	operation, params locat.	parameters
3	index	found/not found, value
4	index, value	---
5	index, index location	---
6	---	index table
7	---	number of indexes
8	location, parameters	---
9	location	---
10	location	symbol parameters

Figure 41. Library Manipulation Module

MANIPULATION. If the index is not found, ADD-INDEX is invoked to enter the index into the index table. Next, LI-

BRARY-MANIPULATION invokes PERFORM-SYMBOL-OPERATION to enter the symbol parameters into the symbol library (using ADD-SYMBOL).

The Purge operation is performed in a manner similar to the Store operation: first the index table is searched for the inputted index and if it is not in the table, an error condition is returned. If the index is found, PERFORM-INDEX-OPERATION invokes DELETE-INDEX to remove the index table entry. LIBRARY-MANIPULATION then invokes PERFORM-SYMBOL-OPERATION to delete the symbol parameters from the library.

To Recall symbol parameters for a given index, LIBRARY-MANIPULATION invokes PERFORM-INDEX-OPERATION to search the index table. If the index is found, its value (the location of the symbol parameters) is returned to LIBRARY-MANIPULATION; if the index is not found, an error condition is returned. The index value is then passed to PERFORM-SYMBOL-OPERATION which uses RETURN-SYMBOL-PARAMETERS to fetch the parameters from the library.

When the calling module requests a Count of the number of indexes in the table, the PERFORM-INDEX-OPERATION module is invoked, which then invokes COUNT-INDEXES. The contents of the index table can be returned to the calling module by using the Fetch operation. This operation is also handled by PERFORM-INDEX-OPERATION, which uses FETCH-INDEXES to perform the operation.

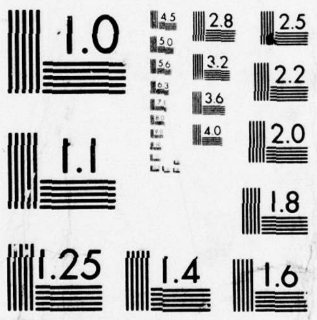
The Current Set-Up Parameters data structure is ma-

AD-A034 273   AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
SOFTWARE DESIGN FOR MULTIMODE MATRIX DISPLAY PERCEPTION TESTS.(U)
DEC 76 R A LIEBACH

UNCLASSIFIED   GE/EE/76-30   NL

2 OF 2
AD
A034273

END
DATE
FILMED
2-77



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

naged by the SET-UP-PARAMETERS-MANIPULATION module and its subordinates (Figure 42). These modules perform the following operations: Add parameters to the data structure, Fetch parameters and Reload the contents of the data structure.

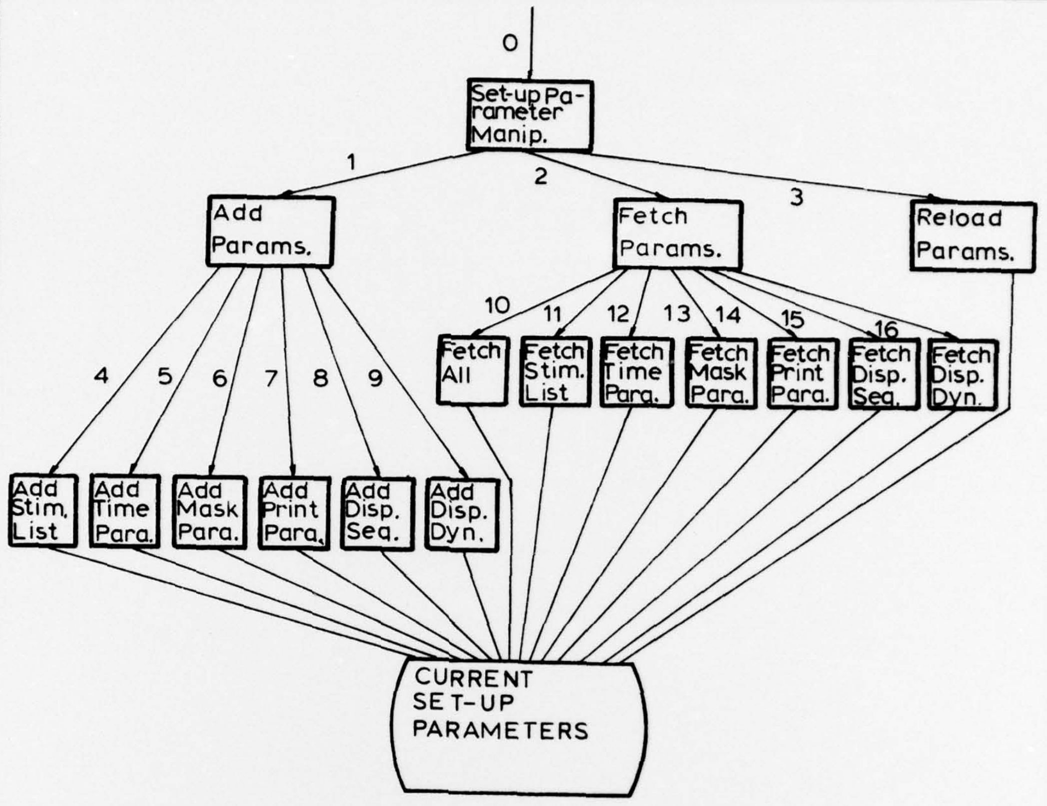
When a calling module wants to enter new parameters into the data structure, it calls the SET-UP-PARAMETERS-MANIPULATION module and requests an Add operation. SET-UP-PARAMETERS-MANIPULATION invokes the ADD-PARAMETERS module which uses its subordinate modules to enter the parameters into the data structure. No error checking is performed on the parameters, and the new parameters are written over any parameters which may have been previously entered.

The FETCH-PARAMETERS module is used to perform the Fetch operations. A calling module can request a particular type of set-up parameters (e.g., the stimulus list or mask parameters) or it can request that all of the parameters in the data structure be returned.

The RELOAD-PARAMETERS module is used to enter an entire set of set-up parameters into the data structure. The Reload operation is requested by the STORE-BULK-SET-UP-PARAMETERS module (Figure 33) to perform its function.

Summary

This chapter has presented the design of the system software architecture using structure charts. The functional descriptions of each module and the interfaces between modules was also presented.



Interface

	IN	OUT
0	operation, params	set-up parameters
1	operation, param type	---
2	operation, param type	set-up parameters
3	set-up params	---
4	stimulus list params	---
5	time params	---
6	mask params	---
7	print params	---
8	display seq params	---
9	display dyn params	---
10	---	set-up parameters
11	---	stimulus list params
12	---	time params
13	---	mask params
14	---	print params
15	---	display seq params
16	---	display dyn params

Figure 42. Set-Up Parameters Manipulation and Subordinates

V. RESULTS AND CONCLUSIONS

Introduction

The purpose of this chapter is to discuss the results obtained from the software structure design and the conclusions drawn from these results.

Results

The requirements definition for the Multimode Matrix Display Perception Tests was accomplished in both an informal manner (Chapter II) and in a rigorous manner (Chapter III). Requirements definition is one of the most difficult and most important phases of a development project (Ref 1: 5). The requirements definition presented in Chapters II and III provide concise and thorough functional specifications for the system, which allowed the development of the design presented in Chapter IV.

The application of structured design techniques resulted in a sound modular design which will be easy to implement, modify and maintain. Most of the modules in the design are relatively small, allowing the function of each module to be easily understood. The use of abstract data types for the major system data structures provides the capability to make modifications in the implementation of any module without affecting the interface to the data structure.

Conclusions

The application of structured analysis and structured design techniques during the early phases of a development project helps to prevent a lot of problems frequently encountered in the implementation of a system. Avoiding the tendency to begin coding software early in the project not only results in a better implementation of the system, but it also avoids a lot of coding modifications which might have to otherwise be made. For instance, there were many changes made to the functional specifications and design presented in this thesis. These changes were easier to make at this early stage in the development cycle, and they avoided the necessity of making any corresponding changes in software code, since no code had been produced at this point.

Recommendations

When this thesis was started, no decisions had been made concerning the hardware which would be used to implement the system. This caused no serious problems with accomplishing the work in this thesis, since it is usually better to perform the requirements definition and the initial design phase of a project and use the results of this work to decide on the best hardware to use. However, the MMM project office recently ordered a PDP-11/45 minicomputer system to be used to implement the perception test system. Therefore, it is recommended that the system which was purchased be studied to determine the extent to which this existing system can be used to directly support the perception test system. For example, the operating system should be studied to determine how much of its I/O support can be used to implement the I/O modules described in Chapter IV.

The symbol display dynamics modules (to perform the Rotate, Add/Delete, Jitter and Vibrate functions) have not been specified in detail. It is necessary to develop the details of the interface between these modules and the display unit. After the interface has been defined, the algorithms for the symbol display dynamics functions should be developed. This will allow the design of the software structure to be completed.

The next major step in the design phase is to develop flow charts for each of the modules in the design. These flow charts will provide a more rigorous specification for the modules than the functional description presented in Chapter IV. The flow charts should be developed in enough detail so that the software code can be produced from them in the chosen programming language.

Before coding is started, it is necessary to develop the design specifications for the system. These specifications will consist of constraints on the system memory size, response time and other constraints.

It is recommended that formal test procedures be developed. These test procedures will assure that the software performs as intended and that the system requirements are met. The test procedures should provide for testing on the subprogram level and on the full program.

Bibliography

1. Logicon, Inc. Management Guide to Avionics Software Acquisition, Volume I - An Overview of Software Development and Management. Dayton, Ohio: June 1976.

2. Parnas, D. L., "On the Criteria to be Used in Decomposing Systems Into Modules". Communications of the ACM, 15: 1053-1055 (December 1972).

3. Ross, D. T. and K. E. Schoman, Jr. Structured Analysis for Requirements Definition. Waltham, Massachusetts: Softech, Inc., April, 1976.

4. Softech, Inc. Structured Analysis Reader Guide. Waltham, Massachusetts: May 1975.

5. Stevens, W. P., et al. "Structured Design". IBM Systems Journal, 2: 115-139 (1974).

6. Teichroew, D. PSL/PSA A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. Ann Arbor, Michigan: University of Michigan, March 1975.

7. Yourdon, E. and L. L. Constantine. Structured Design. New York: Yordon, Inc., 1976.

Appendix A

READING STRUCTURED ANALYSIS DIAGRAMS

Introduction

This appendix contains a brief discussion of how to read structured analysis diagrams. For a more detailed treatment of this subject, the reader is referred to Reference 7.

Diagram Syntax

Structured analysis models are created by decomposing a subject (in a top-down fashion) into smaller and smaller pieces. The subject matter is treated twice: once from an activity aspect (activity model) and once based on the subject's data (data model).

A structured analysis model consists of labeled boxes and arrows. Inside a box, the name of the activity (for an activity model) or data item (for a data model) is written. In the activity case, the name contains a verb (to express action) and in the data case it is a noun or noun phrase (to express a data item).

The boxes of a diagram are connected together with arrows, where the arrows represent interfaces between the boxes. The four sides of a box are assigned a specific meaning which defines the kind of arrow which may enter or leave that side of a box. This is illustrated in Figure 43.

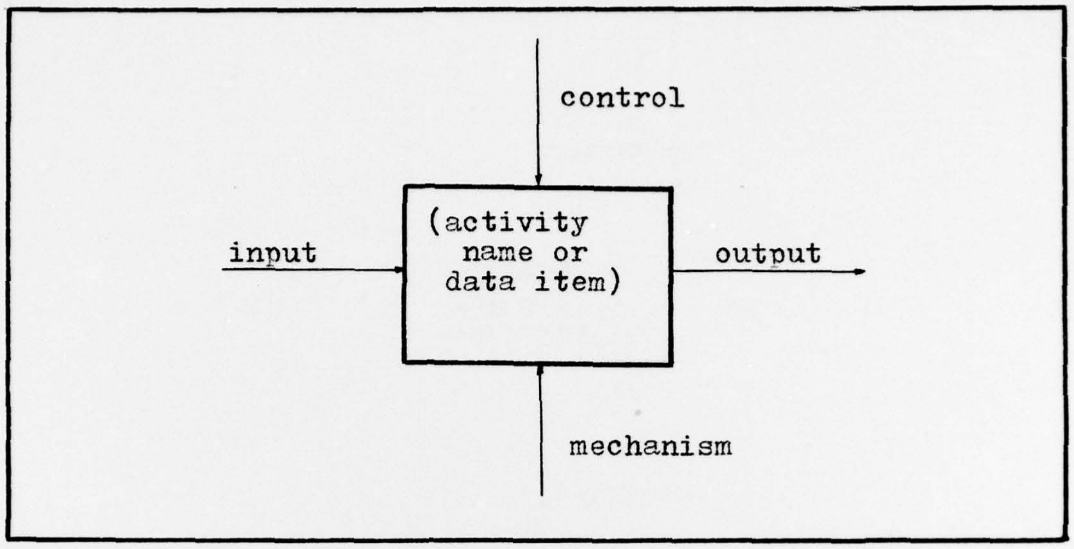


Figure 43. Structured Analysis Box Labeling

Each of the four types of interface arrows represents a particular kind of interface. For an activity diagram, the interfaces between activity boxes are data items, since data is commonly passed back and forth between activities. The interfaces between data boxes for a data diagram are activities, since the data item is created and used by activities. The "mechanism" arrow is an exception to these rules, since it represents the mechanism necessary to "realize the box". The mechanism arrow is usually not shown on the diagram, since the mechanism is usually evident from the title of the box.

The boxes of a diagram (called the parent) are decomposed into smaller boxes in another diagram (called the child). This parent-child relationship is represented by the node number assigned to a diagram. For example, if diagram "AO" contains three boxes numbered from one to three, then diagrams "A1", "A2" and "A3" are the decompositions (children) of the respective boxes of diagram "A0". In the same manner, the boxes of diagram "A1" (which can now be considered a parent) can be decomposed in diagrams "A11", "A12", etc., depending on the number of boxes contained in diagram "A1".

Basic Reading Sequence

The following reading sequence is recommended, taking the diagrams in a top-down order:

1. Scan only the boxes of the child diagram to gain an impression of the module decomposition.

2. Using the parent diagram, rethink the message of the parent module, observing the arrows feeding to and from the parent module.

3. Referring back to the child module, see how and where each arrow from the parent context attaches to the factors of the child module.

4. Next consider the internal arrows of the child module to see the details of how it works. Consider the boxes from top to bottom and from left to right. The arrows should be examined in a clockwise manner, going around each box.

5. Finally, read the text for the child diagram to confirm or alter the interpretation gained from considering the diagrams themselves.

VITA

Robert A. Liebach was born on 7 September 1950 in Brooklyn, New York. He graduated from Venice Senior High School, Venice, Florida in 1967. After completion of two years of college at Manatee Junior College in Bradenton, Florida and six months at the University of Florida at Gainesville, he enlisted in the U. S. Air Force in June, 1970. He was accepted into the Airmen Education and Commissioning Program and returned to the University of Florida in June 1971 to complete undergraduate studies in the electrical engineering program. After receiving his Bachelor of Science in Electrical Engineering in August 1972 and successful completion of the USAF Officer Training School in Texas, he was commissioned as a Second Lieutenent in the USAF on 30 November 1972. He served two and one half years at HQ SAMSO in Los Angeles, California before coming to the Air Force Institute of Technology in June 1975.

Permanent Adress:	103 South Portia St. Nokomis, Florida 33555

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(14) REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. ~~REPORT NUMBER~~ GE/EE/76-30	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. ~~TITLE (and Subtitle)~~ (6) Software Design for Multimode Matrix Display Perception Tests.		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. ~~AUTHOR(s)~~ (10) ROBERT A./LIEBACH Captain USAF		8. CONTRACT OR GRANT NUMBER(s) (9) master's thesis,
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Flight Dynamics Laboratory (AFFDL/FGR) Wright-Patterson AFB, Ohio 45433		12. ~~REPORT DATE~~ (11) Dec[redacted] 1976
		13. ~~NUMBER OF~~ PAGES (12) 109p. 107
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. ~~SECURITY~~ CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

[illegible]

JERRAL F. GUESS, Captain, USAF
Director of Information

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Design
Requirements Definition
Structured Analysis
Structured Design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Multi-Mode Matrix Display Program is attempting to test the acceptability of using light-emitting diode displays in USAF aircraft. The informal and formal functional specications for the software necessary to perform the desired tests have been developed. Formal functional specifications were developed using a technique called structured analysis. The functional specifications were used to design the software

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

012225

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

system. This design was accomplished using structured design techniques. The software design is presented using structure charts together with functional descriptions of all modules and definitions of the interfaces between modules.