

AD-A055 178

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
PRELIMINARY DESIGN FOR MULTIMODE MATRIX PERCEPTION EXPERIMENT S--ETC(U)
DEC 77 S 6 WENSKA

UNCLASSIFIED

AFIT/GCS/EE/77-11

NL

1 OF 2
AD
A055 178



AD A 055178

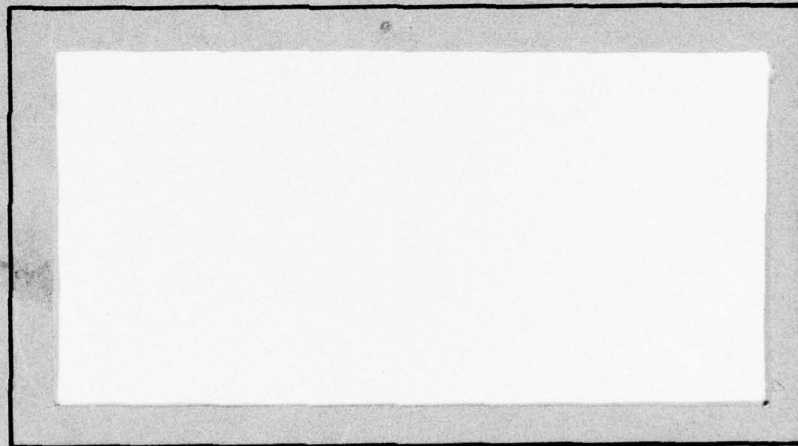
FOR FURTHER TRAN



AIR FORCE INSTITUTE OF TECHNOLOGY



AIR UNIVERSITY
UNITED STATES AIR FORCE



DDC

JUN 16 1978

AD No. _____
DDC FILE COPY

SCHOOL OF ENGINEERING

WRIGHT-PATTERSON AIR FORCE BASE, OHIO

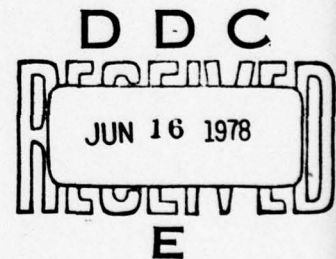
88 06 13 171

GCS/EE/77-11

PRELIMINARY DESIGN FOR MULTIMODE
MATRIX PERCEPTION EXPERIMENT
SOFTWARE

THESIS

GCS/EE/77-11 Stefan G. Wenska
 Captain USAF



Approved for public release; distribution unlimited

78 06 13 171

6 PRELIMINARY DESIGN FOR MULTIMODE
MATRIX PERCEPTION EXPERIMENT
SOFTWARE

9 Master's Thesis

THESIS

14 AFIT/GCS/EE/77-11

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

12 155 p.

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED JUSTIFICATION	<input type="checkbox"/>
BY	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. and/or SPECIAL
A	

10 by
Stefan G. Wenska B.S.
Captain USAF

Graduate Electrical Engineering

11 December 1977

Approved for public release; distribution unlimited

012 225

slt

Preface

This report provides the preliminary structure of the software for a computer-based system to perform perception tests using light-emitting diode (LED) displays. Due to the nature of the testing, the structure was developed to provide the maximum amount of ease in maintaining and modifying the system. The implementation of the system is not covered in this report. This report is written for a reader who possesses a basic knowledge of software development and of structured design techniques.

I wish to thank my loving wife, Mary, for her undying support and invaluable assistance in putting together this report.

Stefan G. Wenska

Contents

	<u>Page</u>
Preface.....	ii
List of Figures.....	v
List of Tables.....	vii
Abstract.....	viii
I. Introduction.....	1
Background.....	1
Perception Experiment.....	2
Scope.....	3
Objective.....	4
Overview.....	4
II. Software Requirements.....	6
Problem Definition.....	6
Set-up Phase.....	7
Execution Phase.....	15
Analysis Phase.....	16
III. Preliminary Design.....	18
Design Philosophy.....	18
Design Approach.....	22
Design Method.....	23
Design Criteria and Evaluation.....	27
IV. Set-up Phase Design.....	29
Identify Data Structures.....	29
Model the Problem with a Data Flow Diagram...	31
Design System Using Structured Techniques....	32
V. Execution Phase Design.....	48
Identify Data Structures.....	48
Model the Problem with a Data Flow Diagram...	50
Design System Using Structured Techniques....	50
VI. Analysis Phase Design.....	62
Identify Data Structures.....	62
Model the Problem with a Data Flow Diagram...	63
Design System Using Structured Techniques....	63

Contents

	Page
VII. Design Evaluation.....	73
Introduction.....	73
Set-up Phase Modification.....	73
Execution Phase Modification.....	74
Analysis Phase Modification.....	75
Optimization.....	76
VIII. Results and Conclusions.....	77
Results.....	77
Conclusions.....	78
Recommendations.....	78
Bibliography.....	80
Appendix A: Module Description.....	81
Appendix B: User's Guide.....	93
Appendix C: Structure Charts.....	112
Vita.....	144

List of Figures

<u>Figure</u>		<u>Page</u>
1	A Symbol Composed of Dot Elements.....	2
2	Time Intervals in a Viewing Sequence.....	11
3	Data Flow Diagram Examples.....	25
4	DFD for the Set-up System.....	31
5	Expanded DFD for Validate Parameters.....	32
6	Set-up System (First-Level Factoring).....	33
7	Get Valid Set-up Command.....	34
8	Get Valid Set-up Parameters (Transaction Center).....	35
9	Get Type I Parameters (General Form).....	36
10	Get Type II Parameter (General Form).....	38
11	Get Type III Parameter (Stub Module).....	39
12	Execute Set-up Command (Transaction Center)...	39
13	Execute Item Command (General Form).....	40
14	Execute Mask Command.....	40
15	Execute Print Parameters Command.....	41
16	Execute Clear Parameters Command.....	41
17	Execute Store Element (General Form).....	43
18	Execute Purge Command (General Form).....	43
19	Execute Reload Command (General Form).....	44
20	Execute List Command (General Form).....	44
21	Execute Create Command.....	45
22	Execute Alter Command.....	46
23	Execute Bulk Operation (General Form).....	47
24	DFD for the Execution System.....	49
25	DFD for the Test Execution.....	51

List of Figures

<u>Figure</u>		<u>Page</u>
26	Execution System (First-Level Factoring).....	52
27	Get Valid Test.....	53
28	Test Execution (First-Level Factoring).....	55
29	Get Viewing Sequence.....	56
30	Control Display Sequence.....	58
31	Store Sequence Results.....	60
32	Store Test Results.....	61
33	DFD for the Analysis System.....	63
34	Analysis Executive (First-Level and Afferent Branch Factoring)...	64
35	Execute Analysis Command (Transaction Center).....	66
36	Execute Load Data Command.....	67
37	Execute Enter Command.....	68
38	Execute Confusion Command.....	69
39	Execute Collapse Command.....	70
40	Execute Print Command.....	71
41	Output Analysis Results.....	72
42	PES System Executive (First-Level Factoring).....	81
43	Afferent Group Structure.....	82

List of Tables

<u>Table</u>		<u>Page</u>
I	Test Composition.....	8
II	Library Operations.....	10
III	Symbol Creation.....	14
IV	Abstract Data Types - Set-up Phase.....	30
V	Set-up Commands.....	30
VI	Abstract Data Types - Execution Phase.....	49

Abstract

↓

The Multi-Mode Matrix Display Program is testing the acceptability of using light-emitting diode displays in USAF aircraft. The preliminary design for the software was done by following a method which enhances software maintainability. The method uses abstracts data types, data flow diagrams, and structured design techniques to produce a complete design for the system. The software design is presented using structure charts together with functional descriptions of all modules and definitions of the interfaces between modules.

↑

I. INTRODUCTION

Background

The Air Force is currently investigating ways to improve aircraft radar display equipment. The present display unit, the cathode-ray tube (CRT), has two major liabilities. First, it is bulky and takes up a lot of space. Cockpit space is already at a premium due to the increased amount of avionics equipment required for flight. Second, if any component of the CRT fails, the entire unit fails. This may cause critical problems during flights within hostile environments where many types of damage (not to mention normal component failure) to the unit might occur.

To correct these deficiencies, the Multi-Mode Matrix Display (MMM) project office of the Flight Dynamics Laboratories at Wright-Patterson Air Force Base is developing a light-emitting diode (LED) display unit to replace the CRT unit that is now in use. The screen of the LED display is composed of an array of LED chips connected together in a checkerboard fashion. The display is controlled by a microprocessor that receives input signals from the navigation equipment. This display system will alleviate the problems of the CRT system. First, the LED unit requires much less space than that needed by the CRT unit. Second, each LED chip can be individually bypassed without affecting the rest of the display. Thus failure of a component can be isolated and not affect the entire display.

Concurrent with the hardware development is the effort by the MMM office 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 and evaluating human perceptions of modified LED display images. These modifications are done through image rotation, image vibration, and the introduction of extraneous elements to the image definition.

Perception Experiment

In order to determine the acceptability of LED displays, the MMM office must measure the effect that symbols, composed of dots, have on human recognition (Fig. 1). To accomplish this end, the MMM office has developed a dot-matrix symbology perception test. This test measures the difference of recognition difficulty between symbols composed of dot elements instead of line elements. Along with measuring the recognition of basic symbols, an evaluation of how dynamic factors (rotation, vibration, and symbol degradence) affect recognition must be made.

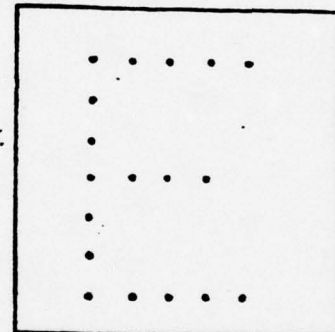


Fig 1. A Symbol Composed of Dot Elements

The test consists of showing a number of symbols to a subject and recording his responses as to their identification. After an acceptable amount of data has been re-

corded, the results are analyzed to determine the degree of recognition. For the test, the symbols will be limited to alphanumeric characters.

To provide a statistical soundness for the test, each symbol is presented in a viewing sequence. This viewing sequence consists of four displays: an acknowledgement symbol, a mask display, a test symbol display (stimulus), and another mask display. The acknowledgement symbol is a predetermined pattern alerting the subject to be ready for the next viewing sequence. The mask display is a random pattern of dot elements. Thus, the perception test consists of a series of viewing sequences to which the subject responds to the stimulus within each sequence.

The data to be analyzed is the test results. This consists of sequence results, which are recorded during the test execution, and the test parameters. The sequence results are the subject's response, his reaction time, the stimulus identification, and the dynamics, if any, that affected the stimulus. This data is recorded for each viewing sequence. The test results are converted to an intermediate form using confusion matrices and then collapsed using predictor matrices. The data is then analyzed to determine the degree of recognition.

Scope

The life cycle of software development has several stages: conceptual, requirements definition, design, coding

and debugging, testing, and operational. The first two stages of the development have already been accomplished. The requirements definition of the software system was derived using a structured analysis technique (Ref 5).

This thesis will begin the design stage of the software that is required to accomplish the perception experiment. This preliminary design uses the requirements definition to develop a structure for the software system. The design of the individual modules is beyond the scope of this paper.

Objective

Maintenance of software (to include modification) is a critical consideration in the development of software today. It is estimated that better than 50% of all software activity in the United States is spent maintaining existing systems (Ref 10:38). Maintenance costs are increasing annually. By 1980, it is estimated that software maintenance costs will be between 50% and 80% of the total system cost (Ref 1:2). Maintainability must be designed into the system from the start; it cannot be added on at the end (Ref 9:92). Therefore, the objective of this preliminary design is to use techniques that will produce a system structure that will enhance maintainability.

Overview

Chapter II will be an informal presentation of the software requirements. A design philosophy is defined in Chapter

III. This philosophy is developed into a design method that produces maintainable software structures. A method for evaluating the preliminary design structure is given. Chapters IV, V, and VI demonstrate the use of the design method chosen to develop the structure for the three phases of the perception experiment. Each phase of the experiment will be considered as a separate system. An evaluation of the design is given in Chapter VII. Finally, Chapter VIII provides the design results and conclusions. Recommendations for the further development of the software are also given.

Three appendixes are included in this paper. Appendix A contains the descriptions of the modules in the design. Appendix B is a User's Guide for operating the system. Appendix C contains copies of all the structure charts that are used in the design of the three phases.

II. SOFTWARE REQUIREMENTS

Before designing any software system, one must have a clear picture of what must be done by the system. This picture is provided by the software requirements. For the perception experiment, the software requirements have been determined in Reference 5. This chapter presents an informal description of these requirements.

Problem Definition

The system to be designed must perform the perception experiment. This experiment is broken into three phases: the set-up phase, the execution phase, and the analysis phase.

During the set-up phase, the user defines symbols and tests. A test consists of a group of items which specify the execution of the perception experiment. The system interacts with the user by providing prompts and error diagnostics. It also allows the user to specify several tests. This gives the user a choice in determining which test to execute.

During the execution phase, the system executes a test. The system records response data for later analysis. It allows multiple executions of the test in order to handle several subjects.

During the analysis phase, the test results from the execution phase is reduced into a more useful form and

analyzed. The system provides several different options and reports to satisfy the desires of the user.

The overall system must be easy to use. It must provide prompts to the user, giving him a choice of legal commands and the parameters associated with those commands. It must allow the frequent user to have the option to input the command and its parameters at the same time, thus saving him the time required for the individual prompting. Finally, the system must handle inputs from either a console or a secondary storage device.

Set-up Phase

The set-up phase is the first phase of the perception experiment. The endproducts of this phase are two data structures: the symbol library (SL) and the test library (TL). Two other data structures, the symbol and the test, are also defined in this phase. These are used to create the libraries.

Each of the data structures is composed of identifiable elements. A symbol is composed of symbol segments. A symbol library is composed of symbols. In this case, the data structure, symbol library, is composed of other data structures--symbols. Similarly, the test library is composed of tests. The test is made up of items, where each item is defined by a description. A description consists of a qualifier (which is optional) and one or more specifications. Table I shows the breakdown of the test into its items and their descriptions.

Table I
Test Composition

ITEM	DESCRIPTION	
	Qualifier	Specification
Masks	Static	Number to be made Maximum number of elements per mask
	Dynamic	Time interval between lighting elements Persistence time
Time Parameters	MT1	Length of mask display
	MD1	Length of interval
	ST	Length of stimulus display
	MD2	Length of interval
	MT2	Length of mask display
	PD	Length of prompting delay
Stimlist	none	Indexes from the SL
Rotate	Constant	Rotation angle
	Random	Maximum rotation angle
Degrade	Add	Maximum number of elements affected
	Delete	Maximum number of elements affected
Displace	none	Maximum distance from the center of the viewing area
Vibrate	none	Frequency
		Distance of movement
Display Options	Remove	Number of symbols
	Redisplay	YES or NO
	Reinforcement	YES or NO
	Print	YES or NO
	Ack. Symbol	Index from the SL
	Stimlist Order	Random or specified

The basic operations for the data structures are retrieval, addition, and deletion of elements. Other operations may augment these. Each of the data structures will be described in this section.

The symbol library and the test library are similar in operation. Aside from the difference between element types in these two structures, each library must maintain an index table. Each element in the library has a unique index. The index is the normal reference to the element. Moreover, each element in the SL has an identifier associated with it. The identifier "describes" the element, i.e., during test execution, if the subject's response to the symbol display and its identifier match, then the subject has identified the symbol.

The library operations are listed in Table II. Along with the basic operations are two that deal with the index table of the library. The "list" operation provides either the number of entries in the table or a list of the entries in the table. The "check" operation determines whether the specified index is in the table.

The test is made up from several items. The user can specify these items, and in doing so, determines how the test will be executed. While the user is creating a test, the items are stored in a data structure called the active test (AT). (The modifier "active" when used in this context, denotes that a data structure is being specified. Thus, an active test is the test while it is being specified. An active symbol is the symbol as it is being created.) The items

Table II
Library Operations

OPERATION	PARAMETER	RESULTS
Add	index	adds entry to library with the specified index
Delete	index	removes entry from library and updates index table
Reload	index	copies entry from library to the active structure
List	all	lists the entries in the index table
	number	counts the number of entries in the index table
Check	index	determines if an index is an entry in the index table

may be user-specified or system-specified (default values). When the user has specified what he wants, he stores the contents of the active test into the test library. The items of a test are described in the following paragraphs.

The masks-item can be qualified as static or dynamic. In a static mask, the elements are lit continuously. In a dynamic mask, the elements are lit intermittantly. The specifications for the static mask option are the number of masks to be generated by the system and the maximum number of elements that can be lit for each mask. The system generates the static masks and stores both the item with its description and the static masks in the active test. The specifications for the dynamic mask option are the time interval between lighting elements and the length of time an element is to be lit (persistence time). The system generates the dynamic masks during the execution of the test.

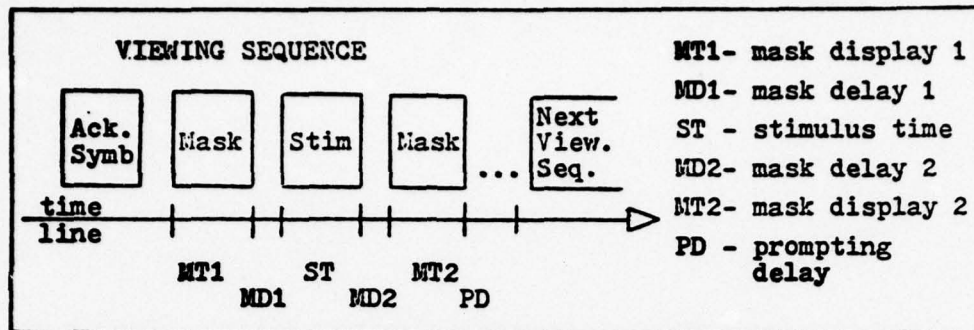


Figure 2. Time Intervals in a Viewing Sequence

The time parameters are the times associated with the viewing sequence (Fig. 2). The six times are the lengths of time to display each mask, the time intervals between each mask and the stimulus, the stimulus display time, and the maximum time the system should wait for a response before displaying a prompting message (prompting delay).

The stimlist is the list of indexes of the symbols to be displayed during test execution. An index may appear in the list more than once. The user can specify that a particular order for the symbols be followed or he may allow the system to generate a random order for their display. During execution, the system will create a stimulus library which will be composed of the symbol data structures that correspond to the indexes in the stimlist.

Symbol dynamics can be specified to determine how the symbol should be displayed. These dynamics items are rotation, vibration, degradence, and displacement. If no dynamics items is specified, the display during execution will be the basic stimulus display, i.e., displayed as the symbol was defined during symbol creation. The rotate item is

qualified as either constant rotation or random rotation. For constant rotation, the user specifies a rotation angle. During the test, each symbol will be displayed rotated by the specified angle. For random rotation, the user specifies a maximum angle of rotation. During test execution, the system will rotate each symbol by a random angle.

Degradence is the modification of the symbol's appearance through the random addition or deletion of elements. The degrade item is qualified as add or delete and is specified by the number of elements to be affected.

The vibrate item is described by the frequency and range of movement that the displayed stimulus is to be vibrated. This is the only dynamics item that affects the stimulus during execution time. Thus, this item will be the most difficult to implement.

The displace item is described by the maximum distance from the center of the viewing area that a symbol can be displaced. If displace has been selected, the symbols will appear at random locations within the viewing area. These dynamics items are not mutually exclusive. Thus a symbol might appear degraded, rotated, vibrating, and off-centered during the execution of the test.

The display options items can be used to change the order of the display sequence during execution or to provide the subject with feedback during the test. The remove and the redisplay items affect the display order. If the remove option is selected, the user specifies the number of consecu-

tive times that the subject must correctly respond to a symbol in order to cause that symbol to be removed from the stimulus library. If the redisplay option is selected, the system will redisplay the same symbol if the subject responds incorrectly.

The reinforcement option and the print option affect the feedback to the subject. The reinforcement option causes the system to display a correct/incorrect acknowledgement after each subject response. The print option causes the system to print the index of the stimulus, the subject's response, the reaction time, and the display dynamics after each viewing sequence.

The final display option is the specification of the index of the acknowledgement symbol. This symbol will be entered as an element in the SL. Typically, it is an easily recognizable pattern of elements, such as a "plus" made up of dot elements and positioned in the center of the viewing area.

There are two specific applications of the basic data structure operations which affect the active test. The first is the clearing operation. This sets all the items in the active test to their default values. This operation is used prior to specifying a new test. This clears any residual values from the previous test specified. The second operation is the listing of the items in the active test. When these items are listed, those descriptions containing default values are flagged.

An example of the use of the listing operation would be

after the retrieval of a test from the TL. When a test is retrieved from the library, it is stored in the active test. It can then be listed to determine what values are in it. If the user wanted a similar test then he could respecify those items that he desired different and store the active test to the library under a different index.

Table III
Symbol Creation

OPERATION	PARAMETER	RESULTS
Add	segment	adds segment to the active symbol and displays the change
Delete	segment	removes segment from the active symbol and displays the change
Clear	----	removes all segments from the active symbol and displays a blank screen
ALTER FUNCTIONS		
Rotate	angle	rotates the display to the specified angle
Vibrate	frequency, distance	vibrates the display
Add/Delete	number	adds/deletes the specified number of elements to the display

The final data structure in this phase is the symbol. The user specifies symbol segments which are stored in a structure called the active symbol (AS). While the user is creating a symbol the system enters a "create mode". While in this mode, the contents of the active symbol are displayed on the console screen. This enables the user to see the

symbol as he creates it.

The operations for the symbol data structure are shown in Table III. Apart from the basic operations, the system must allow the user the option to view the symbol as it is affected by symbol dynamics. During symbol creation, the user may specify rotation, vibration, or degradence to test the dynamics.

Similar to the test definition, the user has the option to start creating a symbol from an already completed one. When a symbol is retrieved from the library, it is stored in the active symbol. The user can specify addition and deletion of segments to create a different symbol. This new symbol can be added to the library under a new index.

Execution Phase

The execution phase is the second phase of the perception experiment. During this phase, the system executes the test selected by the user and records the data that is necessary for later analysis.

In order to execute the test, the system generates the displays of the viewing sequence according to the specified time parameters. Dynamic masks, if applicable, are generated during mask display time. The stimulus displays are presented dynamically. Feedback is produced according to the display options. The system maintains a test history to keep track of the results that provide for the removal and/or redisplay of symbols.

During execution, there are three different responses that the subject can give: one normal and two special. The normal response is the symbol identifier (which can be either correct or incorrect). Since the symbols are alphanumeric characters, the subject responds by pressing the key corresponding to the symbol that he believes is being displayed.

The two special responses are the wrong-key indicator and the I-don't-know indicator. If the subject inadvertently presses a key other than what he intended while responding to the stimulus, he must press the wrong-key indicator. This notifies the system to flag the response and to continue the test. The I-don't-know response is used when the subject does not recognize the symbol that is being displayed.

For each test, the system will repeatedly traverse the stimulus library until the user enters a command to stop the test. The system will then reset itself in preparation for the next subject.

Analysis Phase

The analysis phase is the final phase of the perception experiment. During this phase, the data from one or more tests is compiled and analyzed. The system can generate three types of confusion matrices from the data. These confusion matrices can be based on the subject's responses, the reaction times, or the percentage of times that a particular response was given to a stimulus.

A collapsing routine, using a predictor matrix, then reduces the confusion matrices to a more useful form. The predictor matrix is input by the user and specifies the degree of error that an incorrect response has. After the data is collapsed, it can be stored or output to the user. If it is stored, it can be read and/or augmented at a later time as more testing analysis is completed.

III. PRELIMINARY DESIGN

This chapter will discuss the general approach used in producing the preliminary design of the perception experiment software (PES). It will cover the design philosophy, the design methodology, and the design evaluation method. The actual design of the three phases of the problem will be covered in the next three chapters.

Design Philosophy

The objective of this design is to produce a structure that is easily maintainable and modifiable. The benefits of such an objective are numerous: the maintenance and modification costs will be lower; the development of the software will be easier because changes required during the latter stages of the development will be limited to specific areas and therefore be easier to change; the debugging and testing will be easier; and the program will be more reliable. The list of benefits is extensive because designing with the goal of maintainability affects most of the other program goals favorably. The most notable exception is program efficiency (optimization). This exception will be addressed later and shown to have little effect on the preliminary design (see Chapter VII).

To design maintainability into a program, it is necessary to develop a program structure that localizes changes within the program. The two components of every program--data

and functions--must be handled in a way so that they are always recognizable. The program functions should be distinct and identifiable; the program data should be determinable throughout the program. The structure should not allow communication between unrelated program parts and should minimize communication between related parts. Three concepts which are considered when designing a maintainable program are modularity, program structure-problem structure resemblance, and information hiding. These concepts provide a perspective for designing maintainable programs.

Modularity is the breaking of a program into identifiable units (called modules) which interact with one another during execution of the program. The idea behind modularity is to break complex, large problem into simpler, but more numerous, small problem. The effect of modularity is measured by module coupling and module cohesion.

Module coupling is the measure of communication between two distinct modules. If this communication is minimized, then changes to the program are less likely to propagate through the system. Coupling, therefore, affects the determinability of the program data.

Module cohesion is the measure of the internal binding within a module--the specificity of purpose. A module that performs only one function (e.g., a module that determines the sine of a number) is very cohesive. If all the modules in a program have high cohesion, an error in the program can be easily pinpointed and corrected. Thus, cohesion relates

to the distinctiveness and identifiability of program functions.

The desired design goals, considering these two measures, are the minimization of coupling and the maximization of cohesion. However, there is a trade-off here. The more specific that the modules become (higher cohesion), the greater the number of modules that will be required--thus increasing the amount of communication (higher coupling). Scales depicting the relative difference of degrees of coupling and cohesion have been derived (Refs 7:30; 3:118,179).

Another concept of importance is the program structure-problem structure resemblance. If both the problem and the program are structured alike, then modification is made easier. Changes to the system normally take the form of changes to the problem. These changes must then be converted into changes to the program. If there is a strong correlation between the two structures, this conversion can be done easily. The parts of the program that need to be changed will correspond to the parts of the problem that are changed. Weak program-problem resemblance can lead to difficulty in determining the changes that must be made to the program. Moreover, after the changes have been made to the program, it will be difficult to determine whether the changes accomplished what was desired.

The final concept in the design philosophy is information hiding. This concept impacts two areas: system interfacing and system decomposition. Normally the system commu-

nicates with other systems through the sharing of a data base. Information hiding is the defining of a set of operations (called an abstract data type) on that data base so that each system can access it independent of how the data base is implemented.

Each abstract data type has two interfaces--a system interface and a data base interface. The system interface is the set of operations that the system uses to access the data base. The data base interface is the set of operations that physically manipulate the data; it is dependent of the data base implementation. The abstract data type hides the data base from all of the systems using it. Thus, when the data base structure changes, the system interfaces remain unchanged. The rest of the abstract data type, including the data base interface, is modified to reflect the changes. Therefore, the systems are not affected by the changes to the data base structure.

This concept also applies to internal data structures which are accessed by modules within a system. In this case, the data structure is hidden from the system modules in order to facilitate changes to it.

The second area is system decomposition. In deciding how the system should be decomposed, the designer should consider the modules to be like "black boxes". Each module should perform a specific function but the details of how the function is performed is hidden from the other modules by a well-defined interface (Ref 8). In this way, a change of

the function in one module (e.g., finding an alternate method to achieve a function) does not cause any change to the other modules because the interfaces between the modules remain unchanged.

Design Approach

The perception experiment software is viewed as three separate systems. There are two reasons to view it in this manner. First, the perception experiment itself is structured in this way. At any one time, the user may devote his efforts to only one phase. The only thing relating the phases are the data structures which are outputs to one phase and inputs to another. Thus the software lends itself to being structured as three systems with one common structure for the communication of the data structures.

Second, each problem has its own peculiarities with regard to the amount of user-system interaction. The set-up phase is a continual interaction between the user and the system, where by the messages from each party direct the succeeding interchanges. The execution phase has very little interaction. The user has little control of what occurs during this phase. In the analysis phase, the user sets up the activities he desires and the system then executes them. There is interaction but only infrequently. If a designer were to design the PES as one system, conceptual integrity would be sacrificed in accommodating these differences. By breaking the PES into three systems, each system

can be designed with maximum conceptual integrity which is beneficial to system maintenance (Ref 2:48).

Before the preliminary design of the systems, the user-system interface must be designed. This is done by way of writing a User's Guide for the system (Appendix B). The specification of this interface, this early in the design, does two things. One, the designer has a clearer idea with regards to what the user expects. Two, the user can see the software development better at this stage, at a time when progress visibility is usually limited. The User's Guide communicates the user's desires to the designer. If a change is desired or required, it provides a common ground for both parties.

Design Method

The method to design each of the three systems is outlined here:

1. Identify the data structures (internal and external).
2. Model the problem with a data flow diagram (DFD).
3. Transform the DFD into a program structure using structured design techniques.

These steps are discussed in separate sections that follow.

Identify Data Structures

The first step is to identify the data structures associated with the system. The external data structures are those that are shared with other systems. The internal data

structures are those shared exclusively by modules within the system. Along with the identification of the data structures, the abstract data types, associated with the structures, should be defined. This definition of the abstract data type may not be complete at this time because every operation on the data structure may not be known. The abstract data type should be updated whenever a new operation is identified during the design.

Model Problem with a Data Flow Diagram

The second step of the design method is to model the problem with a data flow diagram. The DFD models the conceptual flow of the data through the problem. The elements of the DFD are transforms and data elements. The transforms are represented by circles (or bubbles); the data elements are represented by labelled arrows that enter and leave the bubbles (Fig. 3a). There are two operators in the DFD. The " * " (asterisk) and the " + " (circled plus) are the conjunctive and disjunctive operators, respectively. The former denotes that two arrows entering (or leaving) the same bubble are both required for the transform (Fig. 3b). The latter denotes that an exclusive-OR situation exists (Fig. 3c).

To develop a DFD, one must first consider the problem in its entirety. All of the data inputs should be listed toward one edge of a sheet of paper. Then all of the data outputs should be listed toward the other edge of the same sheet. The idea is to determine the transforms that are

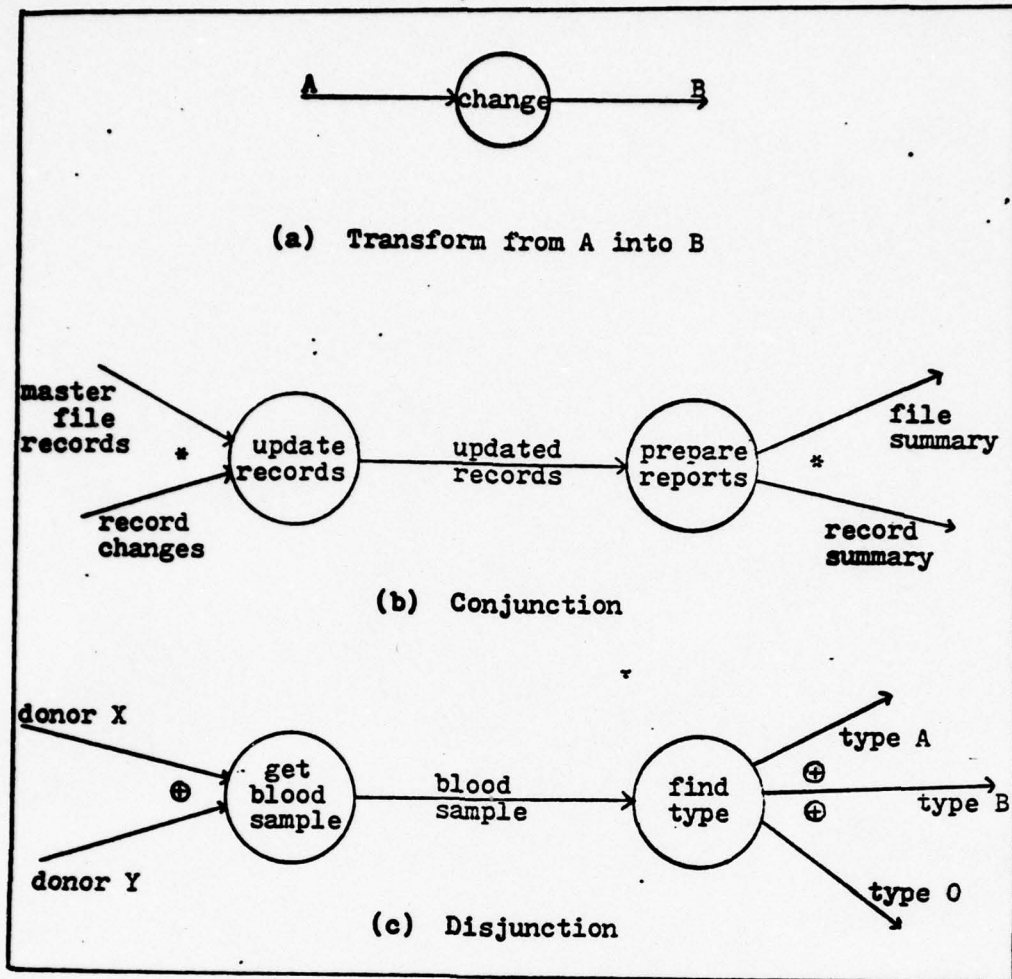


Figure 3. Data Flow Diagram Examples

necessary to get from the input edge to the output edge. How the transforms will be implemented is of no concern at this point. One then proceeds by starting at the input edge and specifying transforms that the data goes through to get to the other edge. Another method is to start at the output edge and determine the transform that has just taken place. In this manner the diagram would develop backstepping to the input edge.

Often it is not clear what the transforms are that convert the input to the output. It may then become necessary

to specify a vague, general transform(s) to go from input to output. Then by decomposing the general transforms into smaller, more specific transforms, one develops the DFD.

The process of developing a DFD is an iterative one. At any time the designer can go back and decompose a transform into several smaller, more specific ones. It is better to overspecify the transforms than to be too general. Once the DFD has been developed, it is relatively easy to convert this problem model into a program structure.

Transform DFD to Program Structure

The final step of the design method is to translate the DFD into a program structure. It has been shown that those structures whose shapes are transform-centered have been associated with low maintenance costs and low modification costs (Ref 3:254). By transform-centered is meant the identification of three types of branches to the structure: an input (afferent) branch, where the data is brought into the system and prepared for the main transformations; a central transforms branch, where the main transformations and computations of the data take place; and an output (efferent) branch, where the results of the central transforms are prepared and processed for output. To accomplish this structure, a design technique called transform analysis was used.

In transform analysis, the afferent data elements, the efferent data elements, and the central transforms must be identified on the DFD. An afferent data element is the

highest level of abstraction of a system input. On a DFD, there is a data element such that if the transform to which it is input is accomplished, the output of that transform could no longer be termed a system input. It is this data element(s) that needs to be identified. Conversely, an efferent data element is the highest level of abstraction of a system output. The transforms between the afferent and efferent data elements are termed the central transforms. These are the transforms which convert the inputs into the outputs. In some systems, the afferent and efferent data elements are the same, that is, are identified by the same labelled arrow on the DFD. These systems have no central transforms.

The identification of these elements provide the first level factoring (decomposition) of the system. Each element will be the head of a branch in the structure. Each branch must be factored fully to get the complete program structure. Reference 3 provides guidelines for the factoring of the afferent and efferent branches. The central transforms should be factored in a manner consistent with the design philosophy. The final structure will tend toward having more executive-type modules higher in the structure chart calling on the more functional modules, which tend to be lower in the structure chart.

Design Criteria and Evaluation

This design method will produce a highly maintainable

system. Step one, identifying data structures, highlights the concept of information hiding. Step two, problem modeling, aims at providing a strong program structure-problem structure resemblance. Step three, structured design, provides a modular program structure. Thus, the design method chosen for the preliminary design of the PES follows those concepts needed to achieve the objective.

To measure the effectiveness of the design, it is necessary to measure the relative ease of maintaining it. To do this, chapter seven will evaluate the design in the following manner. Changes and enhancements to each phase of the system will be proposed. A description of the changes to the system will be given. These changes will be rated as to their impact to the overall system and the difficulty in implementing them.

IV. SET-UP PHASE DESIGN

Identify Data Structures

There are two external data structures to this system: the symbol library and the test library. The symbol library is a set of symbols and its associated index table. The test library is a set of tests and its associated index table. Three functions are required to manipulate these structures: updating (that is, the deleting and adding of elements to the library), retrieval of an element, and examination of the library index table.

There are three internal data structures: the active symbol, the active test, and the default value table. The active symbol and the active test are the working areas for the creation of symbols and tests. There are three functions which can be done to these two structures: adding elements, deleting elements, and listing the elements in the structure. The clearing of the structure is accomplished by the deletion of all the elements in the structure. In the case of the active test, the elements are set to default values. The default value table contains default values for the items in the test. The abstract data types associated with these data structures are given in Table IV.

The user-system interface is also defined in this step. The list of commands used in the set-up phase is given in Table V. The types are differentiated by the number of parameters required for each command (see Appendix B).

Table IV
Abstract Data Types - Set Up Phase

DATA STRUCTURE	OPERATION
Symbol Library	store symbol delete symbol get symbol count indexes list indexes check index
Test Library	store test delete test get test count indexes list indexes check index
Active Symbol	store segment delete segment
Active Test	store item delete item get item
Default Value Table	get item
Item	store qualifier store specification get qualifier get specification

The Type I commands require more than one parameter. The Type II commands require only one parameter and the Type III commands do not require any parameters.

Table V
Set Up Commands

TYPE I	TYPE II	TYPE III
TIME PARAMETERS	STORE PARAMETERS	CLEAR PARAMETERS
STIMLIST	PURGE PARAMETERS	CREATE SYMBOL
DISPLAY OPTIONS	RELOAD PARAMETERS	END SET UP
MASKS	LIST PARAMETERS	
ROTATE SYMBOL	PURGE SYMBOL	
VIBRATE SYMBOL	RELOAD SYMBOL	
STORE SYMBOL	LIST SYMBOLS	
PRINT PARAMETERS	DISPLACE SYMBOL	
BULK LOAD/STORE	DEGRADE SYMBOL	

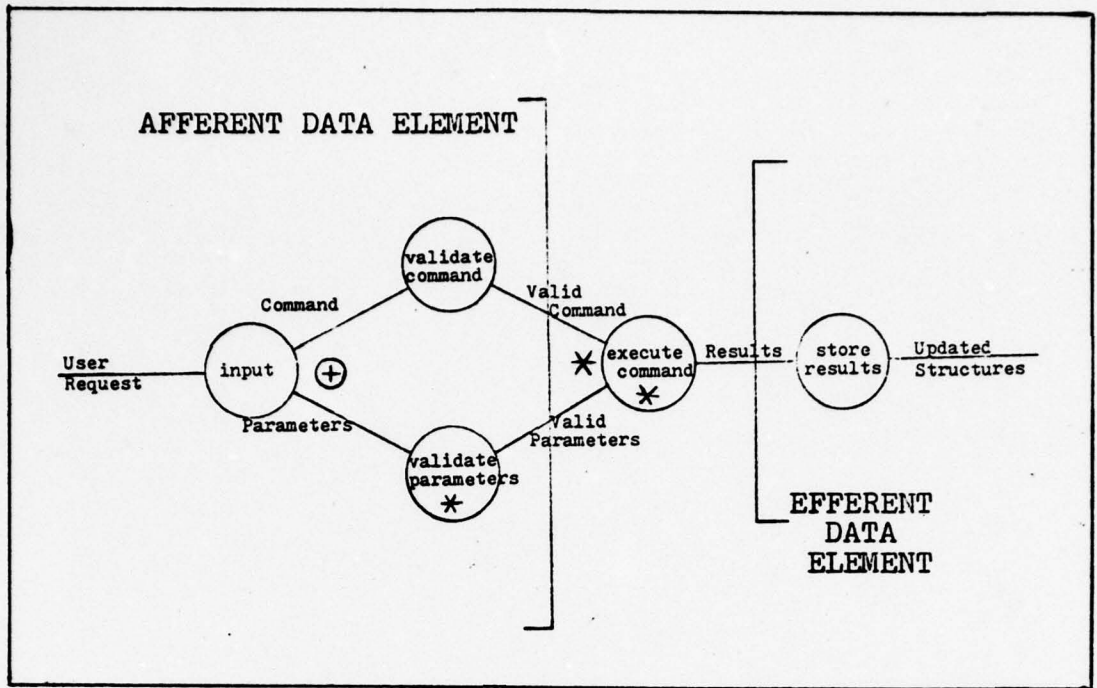


Figure 4. DFD for the Set-up System

Model the Problem with a Data Flow Diagram

The inputs for the system are the user requests--commands and parameters. The outputs are the updated libraries. The afferent data elements are the valid command and the valid parameters. The efferent data elements are the results of the executed commands (Fig. 4). The central transform is the execution of the particular command.

There are two transforms that are marked with asterisks. These asterisked transforms are transaction centers and they represent a class of similar transforms. For instance, the transform "validate parameters" is the general form for the twenty-two types of parameters possible (one for each command). A complete description of the validate function is shown in Figure 5. This figure shows all of the

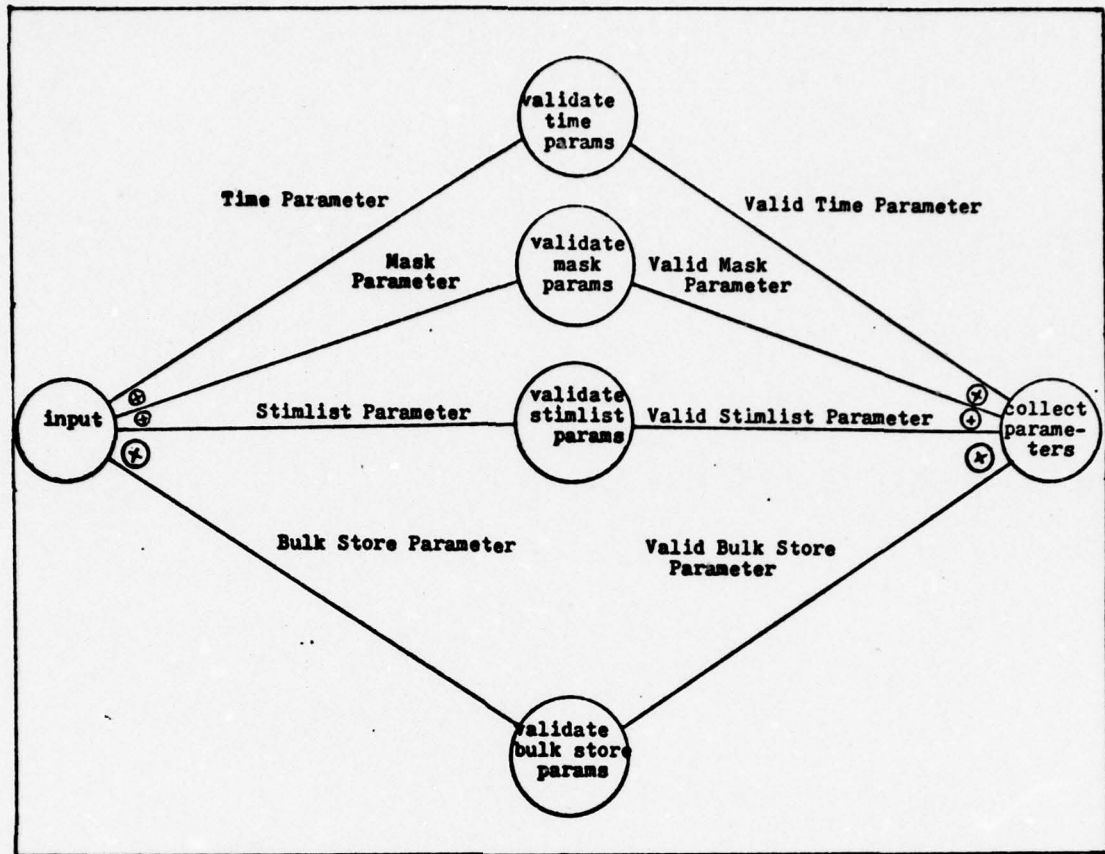


Figure 5. Expanded DFD for Validate Parameters

possible paths of validation that may be taken, depending on the command that was input previously. To develop a structure chart for this type of data flow, a design strategy known as transaction analysis was used (Ref 6:301-308).

Design System Using Structured Techniques

This section describes the development of the structure charts for the set-up phase. A description of structure charts is given in Reference 6. A symbol that is unique to this text is the small circle located in the upper right-hand corner of some of the module boxes. This designates

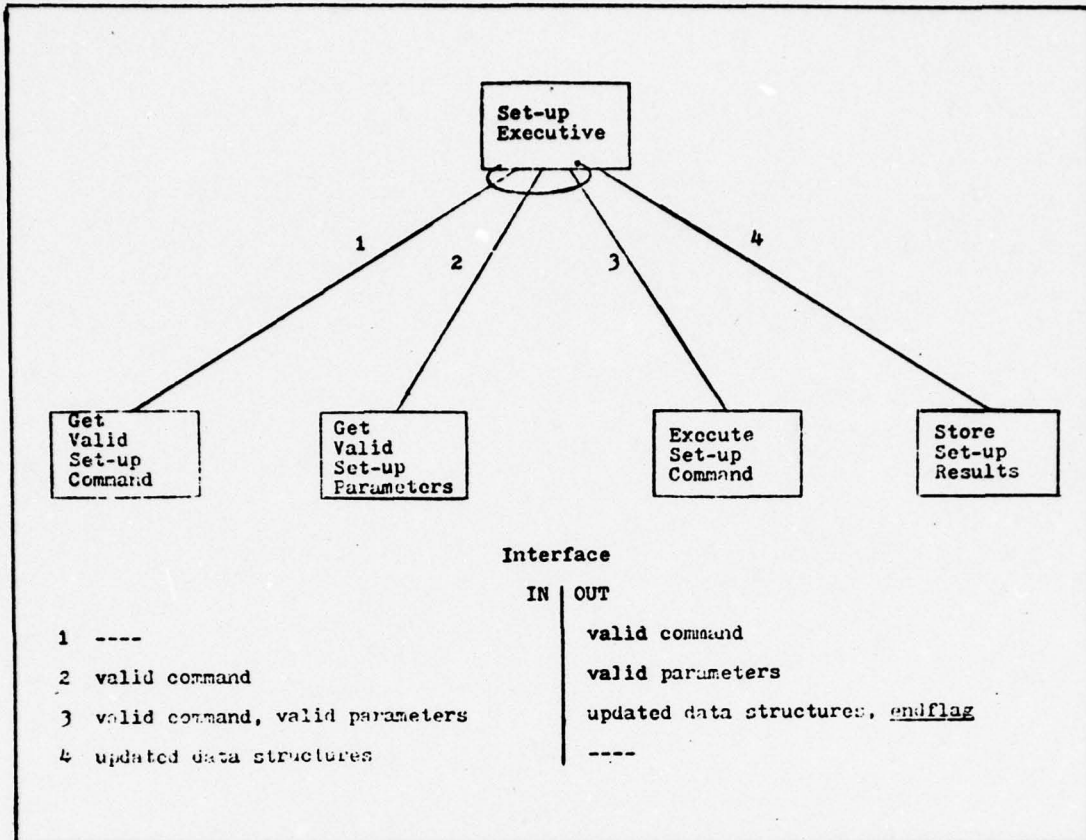


Figure 6. Set-up System (First - Level Factoring)

that the module is a function of an abstract data type.

The first-level factoring was accomplished by creating a module for each of the afferent elements, the central transform, and efferent element (Fig. 6). The next step in developing the structure chart is to factor fully each first level module.

The GET VALID SET UP COMMAND module was factored using the transform analysis technique (Fig. 7). This technique causes the higher level modules of the structure to do more decision making while the lower level modules to be more functional.

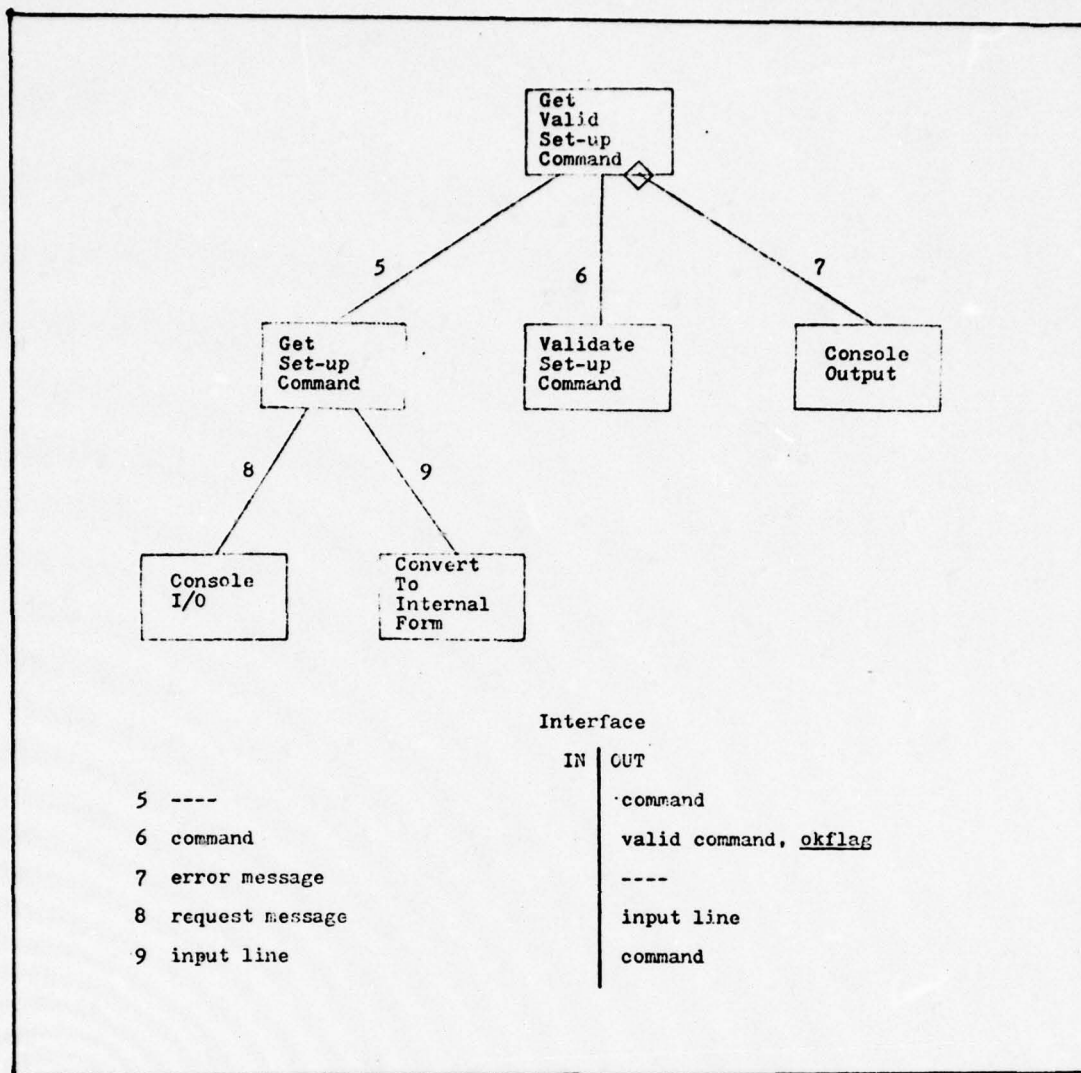


Figure 7. Get Valid Set-up Command

The GET VALID SET UP COMMAND module interacts directly with the CONSOLE OUTPUT module because it makes the decision whether to pass on the command to the SET UP EXECUTIVE or to call on GET SET UP COMMAND in order to request another command (e.g., if the command is valid, it passes the command on; if the command is invalid, it calls on CONSOLE OUTPUT to output an error diagnostic before attempting to get another command). Due to the high interactivenss of the system,

many of the modules communicate with CONSOLE OUTPUT modules.

The GET VALID SET UP PARAMETERS module is a transaction center (Fig. 8). The factoring of this module shows that it is an executive-type module that calls on one of the 22 possible get-valid-parameters modules. This structure, although it looks unwieldy, is manageable because the twenty-two branches will share many of the same lower-level functional modules.

Each of the get-parameters modules for the ten Type I commands (from Table V) are structured in the form shown in Figure 9. The six modules marked with asterisks must be tailored to the specific type of parameter being sought. All of the other modules are shared within this transaction structure. The modules, CHECK ALPHA, CHECK INTEGER, CHECK

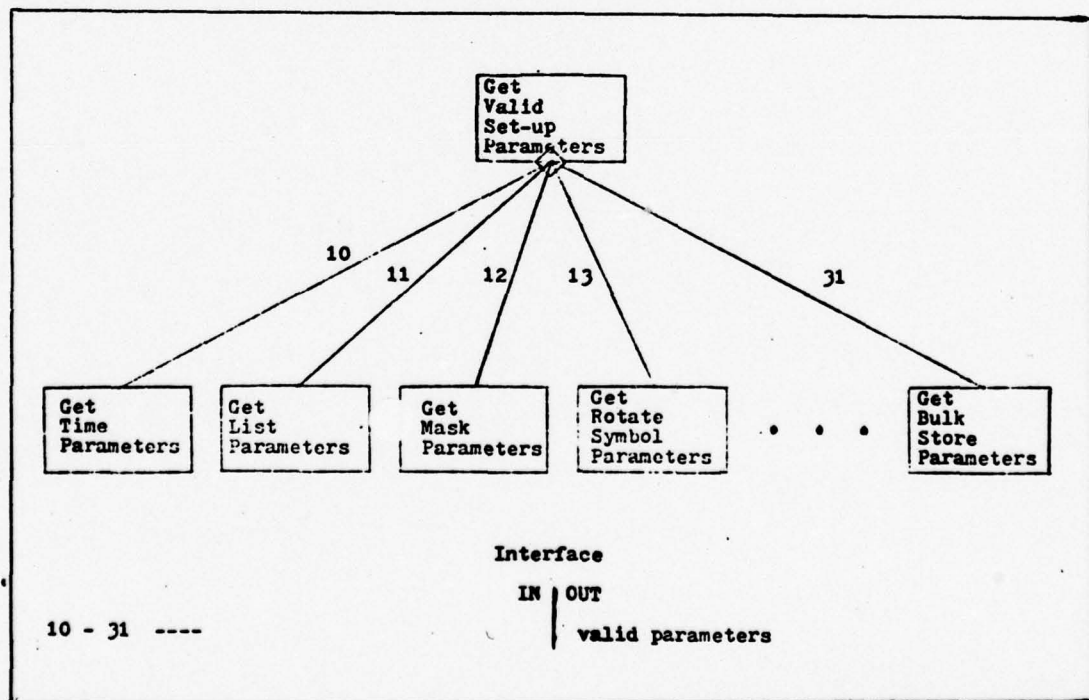


Figure 8. Get Valid Set-up Parameters (Transaction Center)

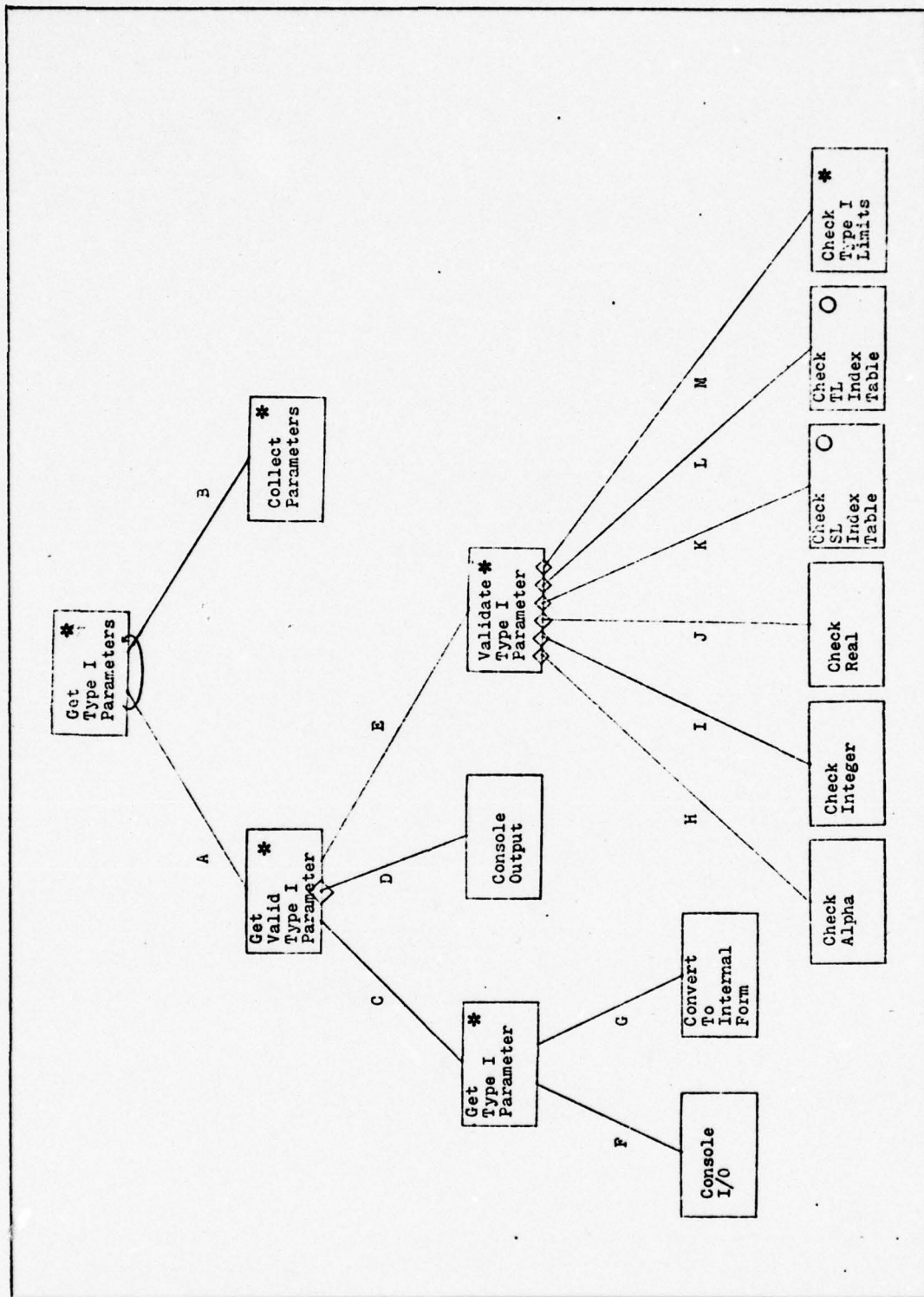


Figure 9. Get Type I Parameters (General Form)

Interface	
	IN OUT
A parameter number	valid parameter
B valid parameter	valid parameters
C Parameter number	parameter
D error message	----
E parameter number, parameter	valid parameter, <u>okflag</u>
F request message	input line
G input line	parameter
H - L parameter	<u>okflag</u>
M parameter, parameter number	<u>okflag</u>

Figure 9 continued

REAL, CHECK SL INDEX TABLE, and CHECK TL INDEX TABLE, might not appear in the structures to which they do not apply specifically. For example, the VALIDATE TIME PARAMETERS module would only call the CHECK INTEGER module since all of the time parameters are integers. In a general form structure chart, the interfaces will be labelled by alphabetic characters instead of numerals.

A Type II command has only one parameter. Therefore, the module COLLECT PARAMETERS is not needed for these commands. The general structure chart for these commands is a compressed form of the structure chart shown in Figure 9 (Fig. 10).

The Type III commands require no parameters. The structure for these commands is a stub (Fig. 11); that is, the stub immediately returns to the calling module and does no

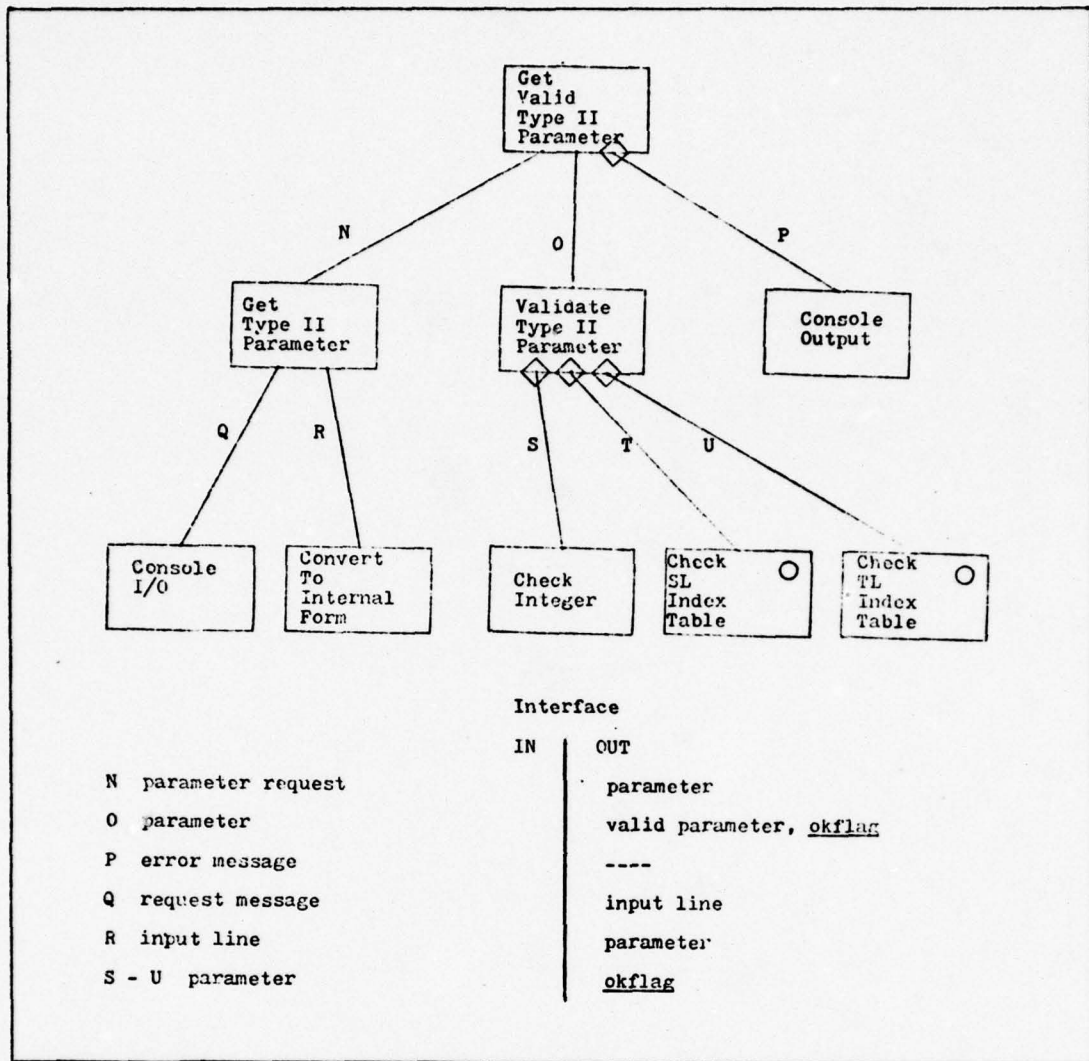


Figure 10. Get Type II Parameter (General Form)

processing. They are shown in the structure but could be eliminated by having the transaction center, GET VALID SET UP PARAMETERS, check for them.

The central transform module, EXECUTE SET UP COMMAND, is also a transaction center. The factoring of this module is handled in the same way as the afferent transaction center (Fig. 12). It has 22 branches emanating from it, one for each possible command.

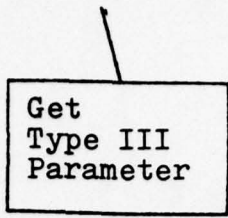


Figure 11. Get Type III Parameter (Stub Module)

The commands whose execution consists of storing items into the active test (except for the MASKS command) have the general form shown in Figure 13. The MASKS command must determine whether static masks need to be generated prior to storing the item into the active test (Fig. 14).

Other commands affecting the active test are the CLEAR PARAMETERS command (Fig. 15) and the PRINT PARAMETERS command (Fig. 16). These commands show the operations--store

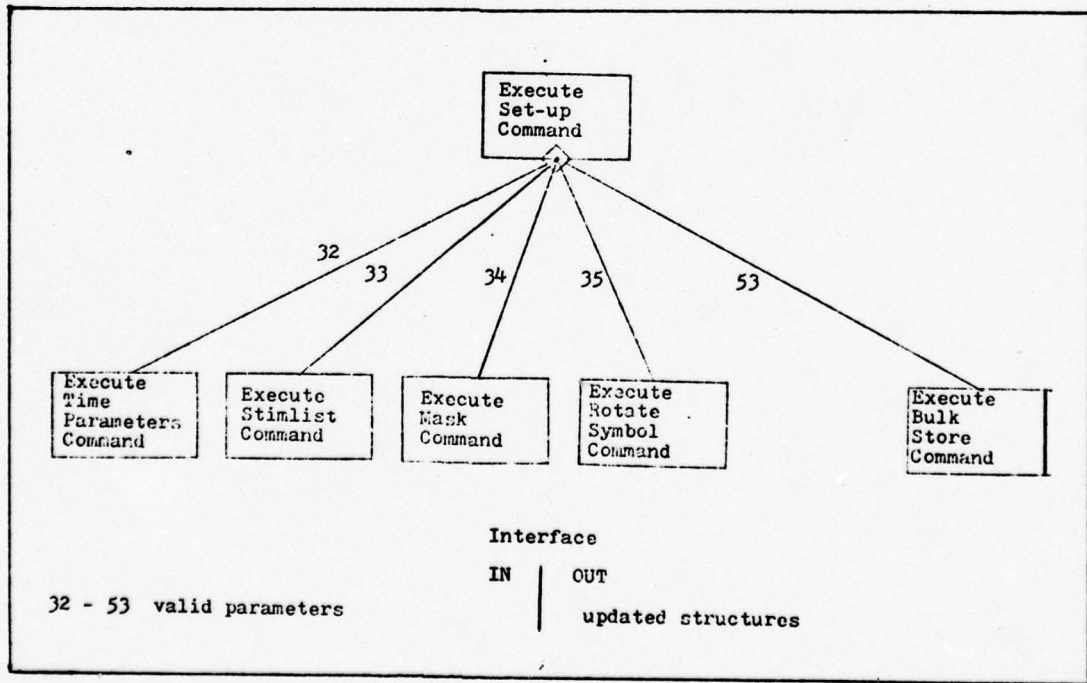


Figure 12. Execute Set-up Command (Transaction Center)

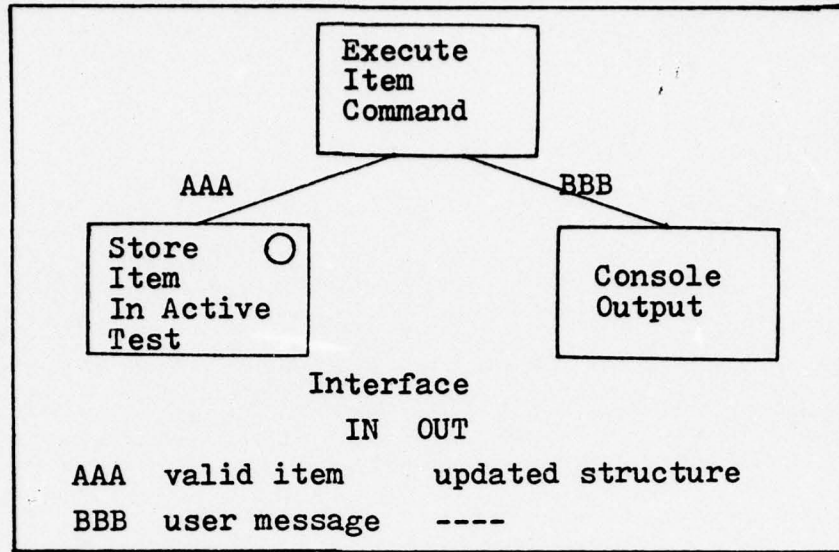


Figure 13. Execute Item Command (General Form)

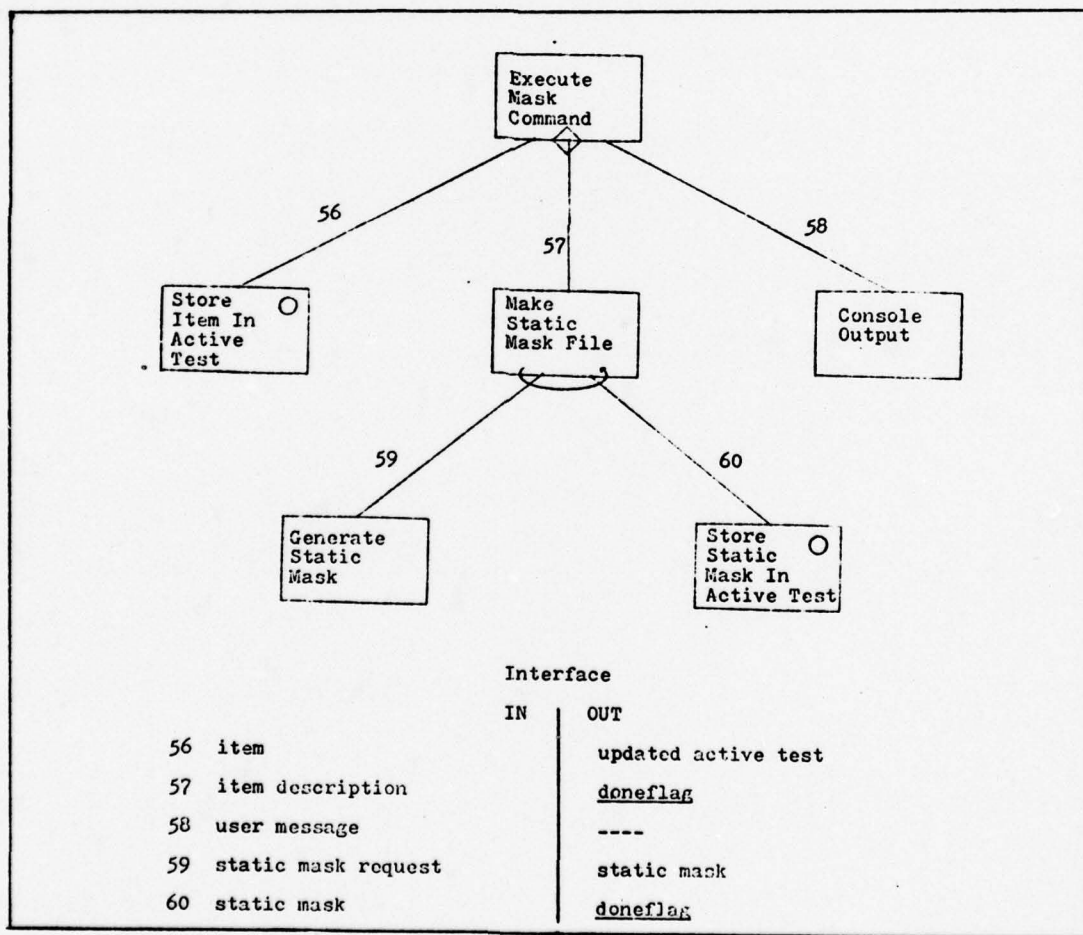


Figure 14. Execute Mask Command

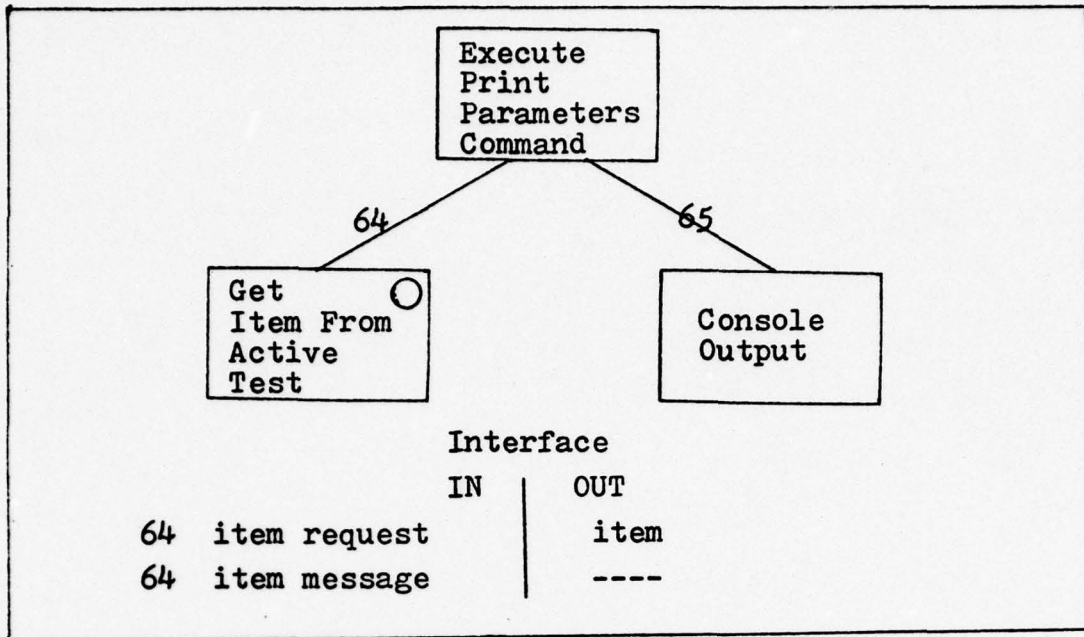


Figure 16. Execute Print Parameters Command

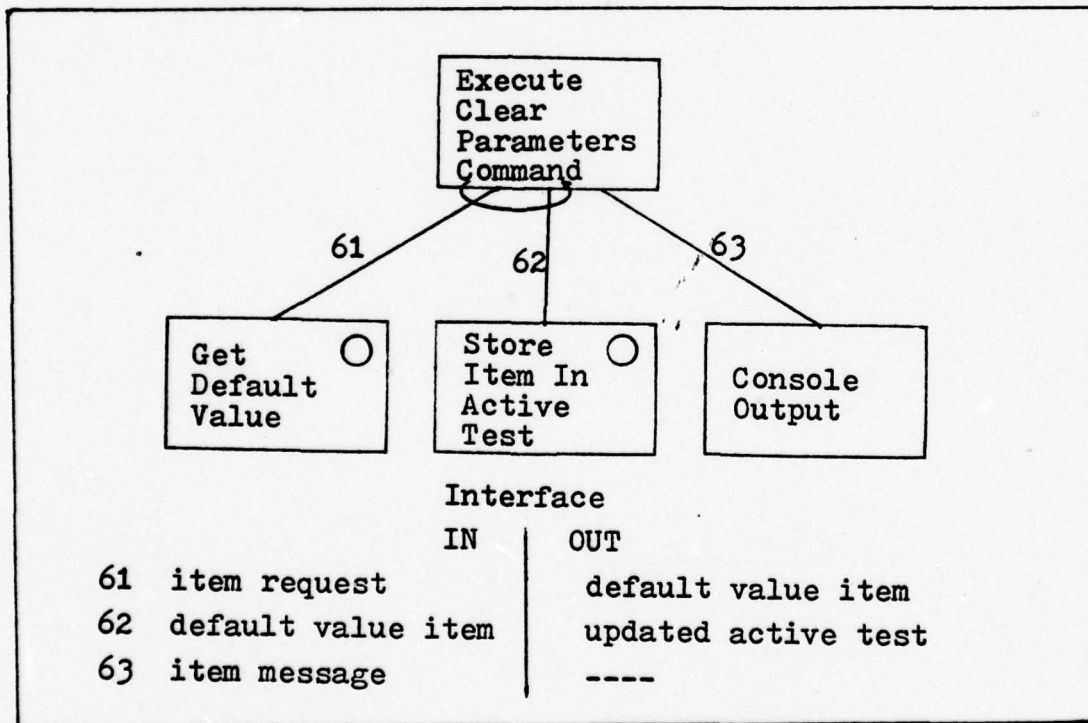


Figure 15. Execute Clear Parameters Command

and get--of the abstract data type AT.

Figures 17, 18, 19, and 20 show the structure charts of the execution of commands affecting the libraries. The use of abstract data types in creating the AS, AT, SL, and TL simplified the designing of these structures.

The EXECUTE CREATE COMMAND module has an afferent and a central transform branch (Fig. 21). The EXECUTE ALTER module can be further decomposed (Fig. 22). In this decomposition, a new data structure is identified--the active dynamics list. This stores the dynamics that affect the active symbol during symbol creation.

The bulk operations have the general structure chart shown in Figure 23. These operations allow the loading or storing of the libraries to mass storage.

The final command to execute is the END SET UP. This command is processed by a module that returns an end-flag to the set-up system executive. This causes control to return to the overall system executive.

The efferent branch in Figure 6 (the STORE SET UP RESULTS module) was not needed. All of the results were handled as part of the execution of the various commands. This branch could be used to pass a final message to the user prior to leaving the set-up mode of operation.

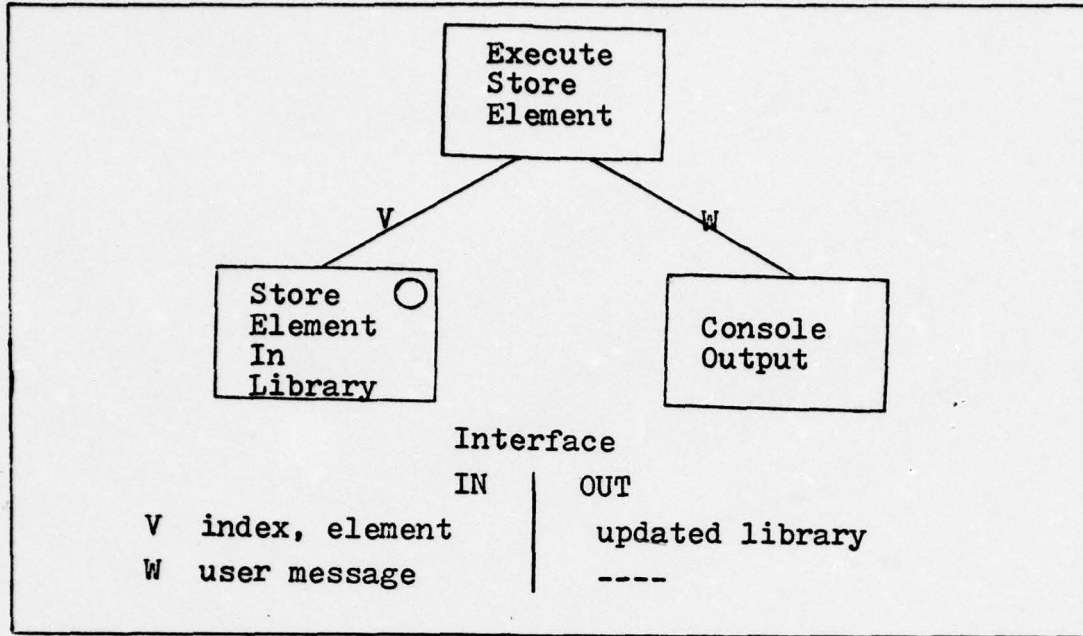


Figure 17. Execute Store Element (General Form)

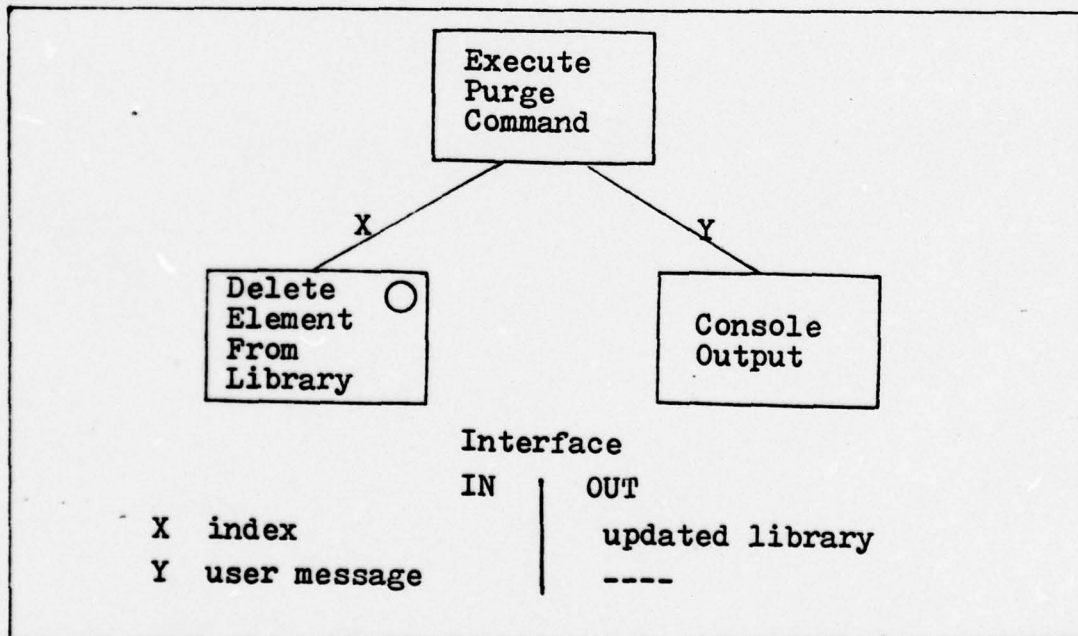


Figure 18. Execute Purge Command (General Form)

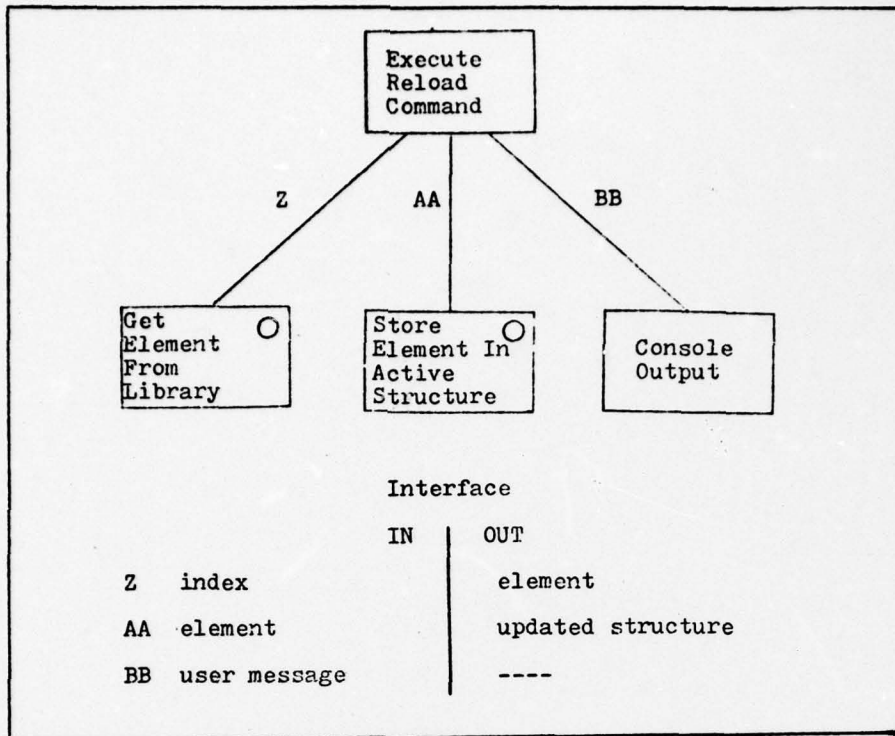


Figure 19. Execute Reload Command (General Form)

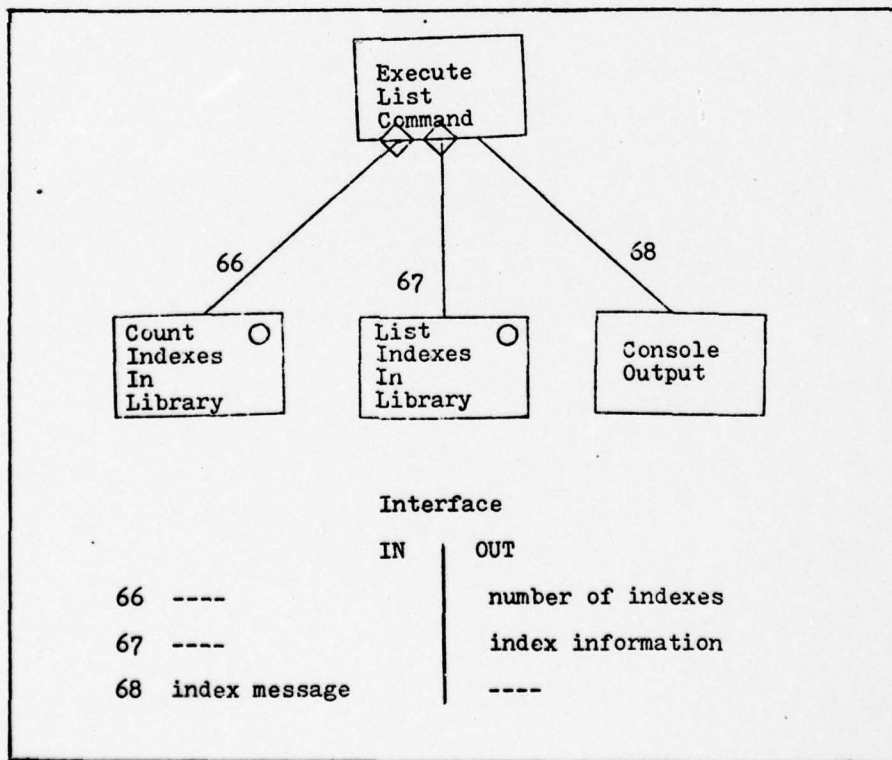
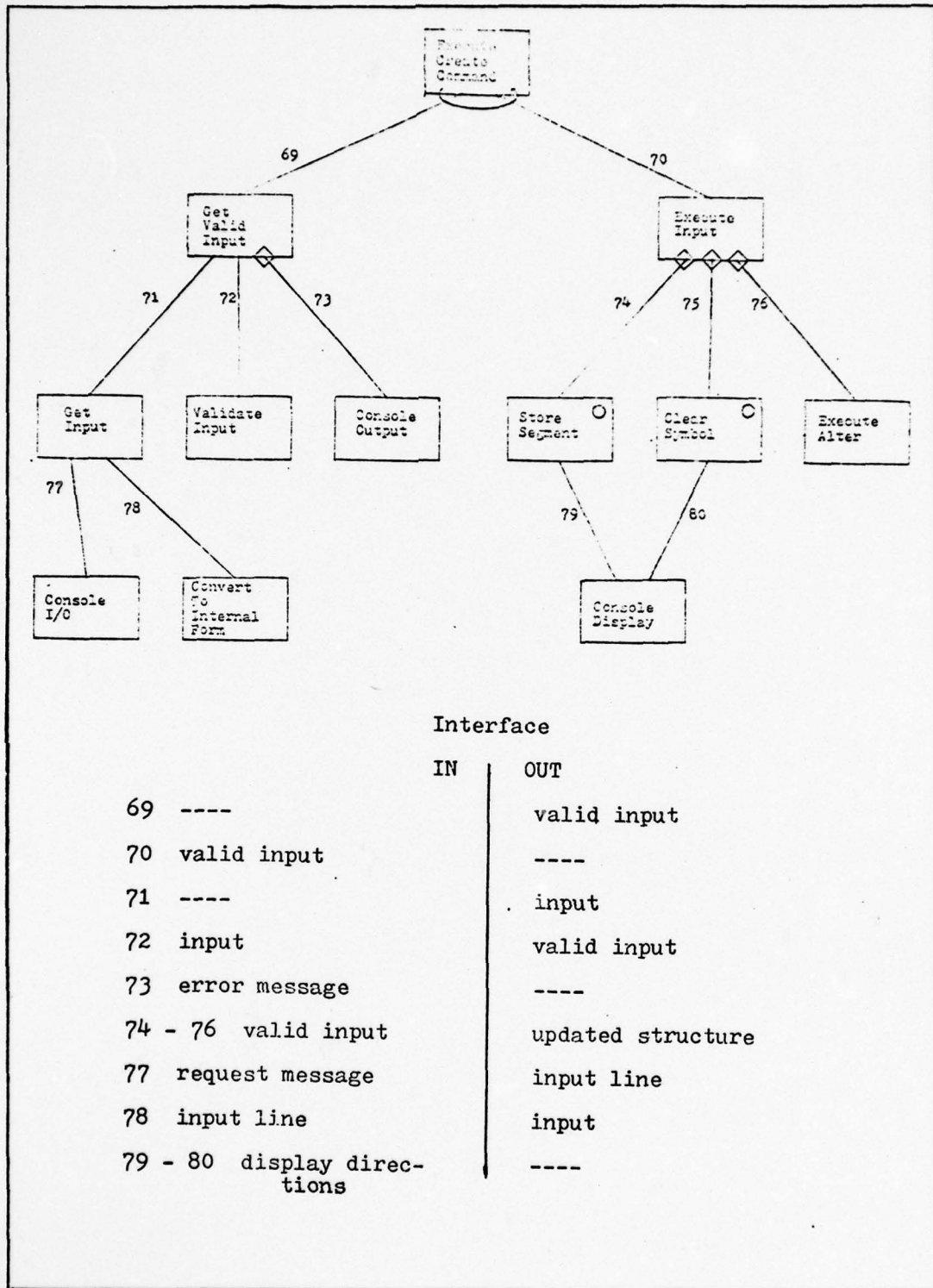


Figure 20. Execute List Command (General Form)



Interface

	IN	OUT
69	----	valid input
70	valid input	----
71	----	input
72	input	valid input
73	error message	----
74 - 76	valid input	updated structure
77	request message	input line
78	input line	input
79 - 80	display directions	----

Figure 21. Execute Create Command

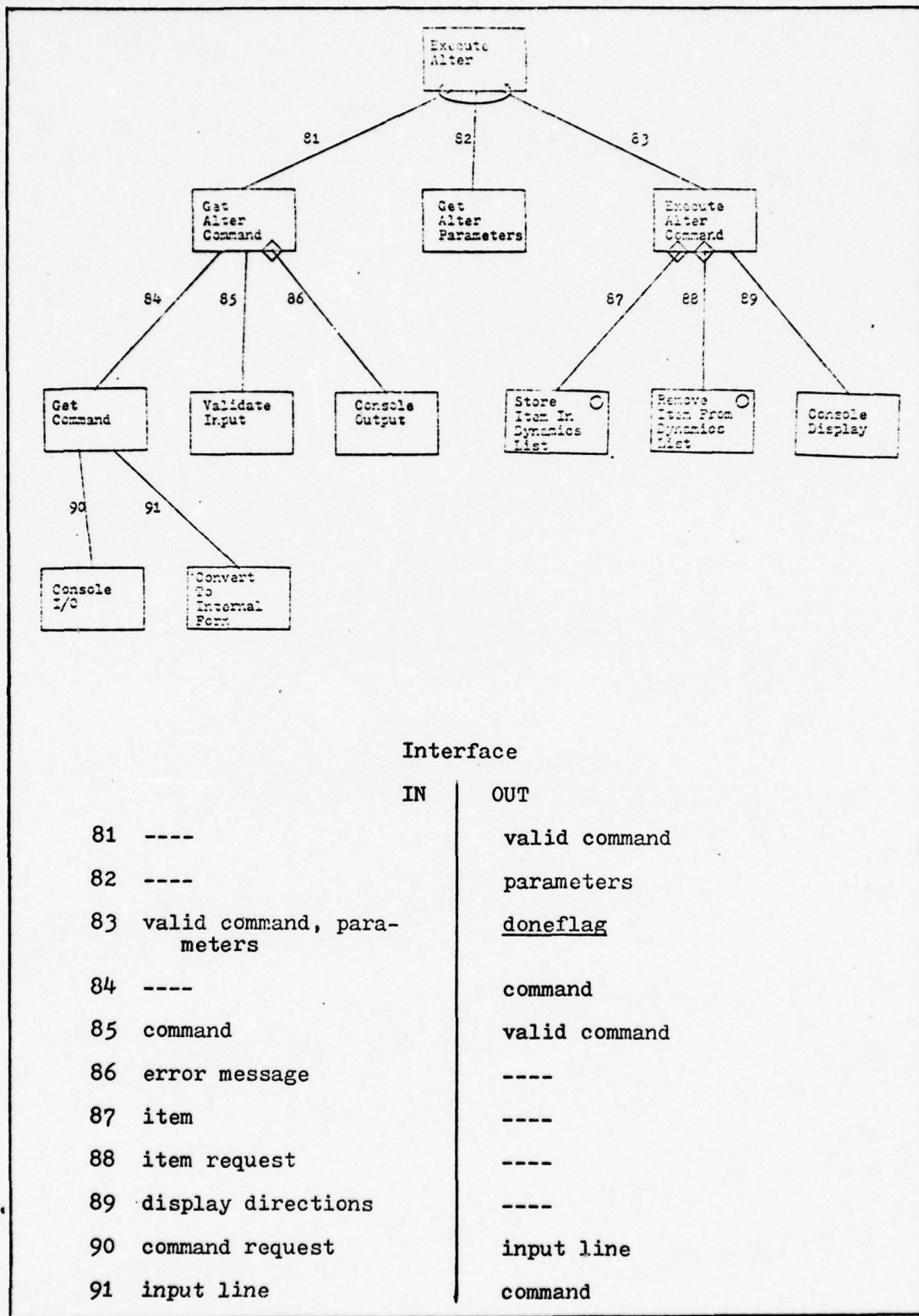


Figure 22. Execute Alter Command

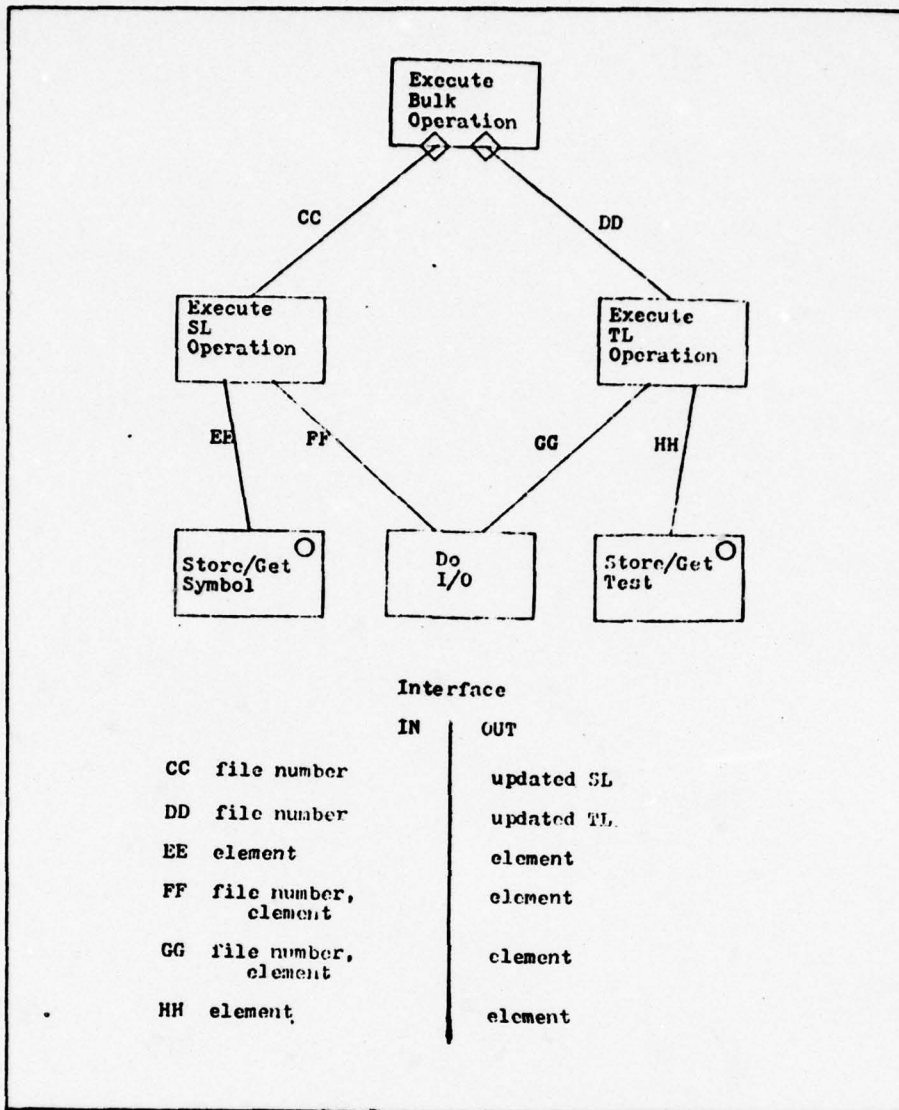


Figure 23. Execute Bulk Operation (General Form)

V. EXECUTION PHASE DESIGN

Identify Data Structures

There are three external data structures to this system: the test library, the symbol library, and the test results file. The TL supplies the test for the system to execute. The SL supplies the symbols to be used during the test. The test results file is created by the system. It contains all of the sequence results and the test used for execution.

The internal data structures are the viewing sequence, the response data, the sequence results, the individual items of the test, and the stimulus library. The viewing sequence is a data structure containing the acknowledgement symbol, two masks (or the dynamic mask parameters), and a test symbol. The response data is a structure that contains the test history for determining the removal and/or redisplay of a symbol. The items of the test are the time parameters, the mask parameters, the stimlist, the display options, and the symbol dynamics. The display options is further broken down into the acknowledgement symbol index, the redisplay option, the remove option, and the feedback options. The symbol dynamics is broken into two classes: vibrate dynamics, which affects the symbol during display time, and all the other dynamics, which are termed static dynamics because they affect the test symbol prior to the display time. The stimulus library is the set of symbols corresponding to the stimlist. The sequence results are the results of each viewing

Table VI
Abstract Data Types - Execution Phase

DATA STRUCTURE	OPERATION
Test Results	store test store sequence results get test get sequence results
Sequence Results	store/get response store/get reaction time store/get dynamics store/get stimulus identifier
Response Data	store/get stimulus identifier store/get previous symbol store/get symbol history
Viewing Sequence	store/get acknowledgement symbol store/get mask parameters store/get stimulus
Stimulus Library	store/get symbol delete symbol

sequence. The abstract data types for those structures which have not been already identified is given in Table VI.

The user-system interface is small in this system. It consists of three commands: EXECUTE, STOP TEST, and END EXECUTION MODE. Apart from these commands, the user has little

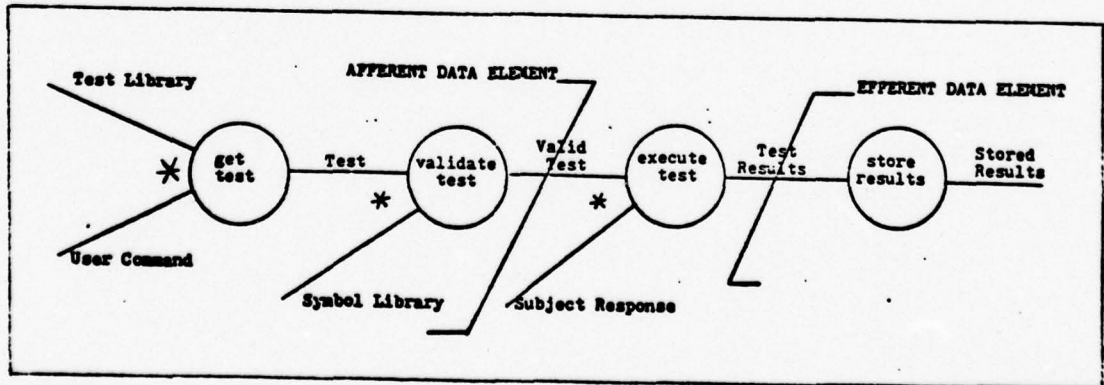


Figure 24. DFD for the Execution System

control over the execution of the test.

Model the Problem with a Data Flow Diagram

The data flow model for the execution phase system consists of two diagrams: the DFD for the entire system and the DFD for the test execution. In the entire system depiction, the afferent data element is the test that is to be executed (Fig. 24). The efferent data element is the test results. The central transform, execute-test, is a system in itself.

The data flow model for the test execution is shown in Figure 25. The model was developed by starting at a known data element in the problem and working backwards until a data structure (internal or external) was reached. The viewing sequence was the starting point. From this point, three lines of flow were derived--the acknowledgement symbol flow, the test symbol flow, and the mask parameters flow. Each line was followed until reaching the test that specifies the execution.

The afferent data elements are the viewing sequence and the time parameters. The efferent data element is the test results data. The central transforms include the displaying of the sequences and the handling of the responses.

Design System Using Structured Techniques

The system was factored into three first level modules; these represented the afferent, efferent, and the central transform branches (Fig. 26). Each branch was then factored

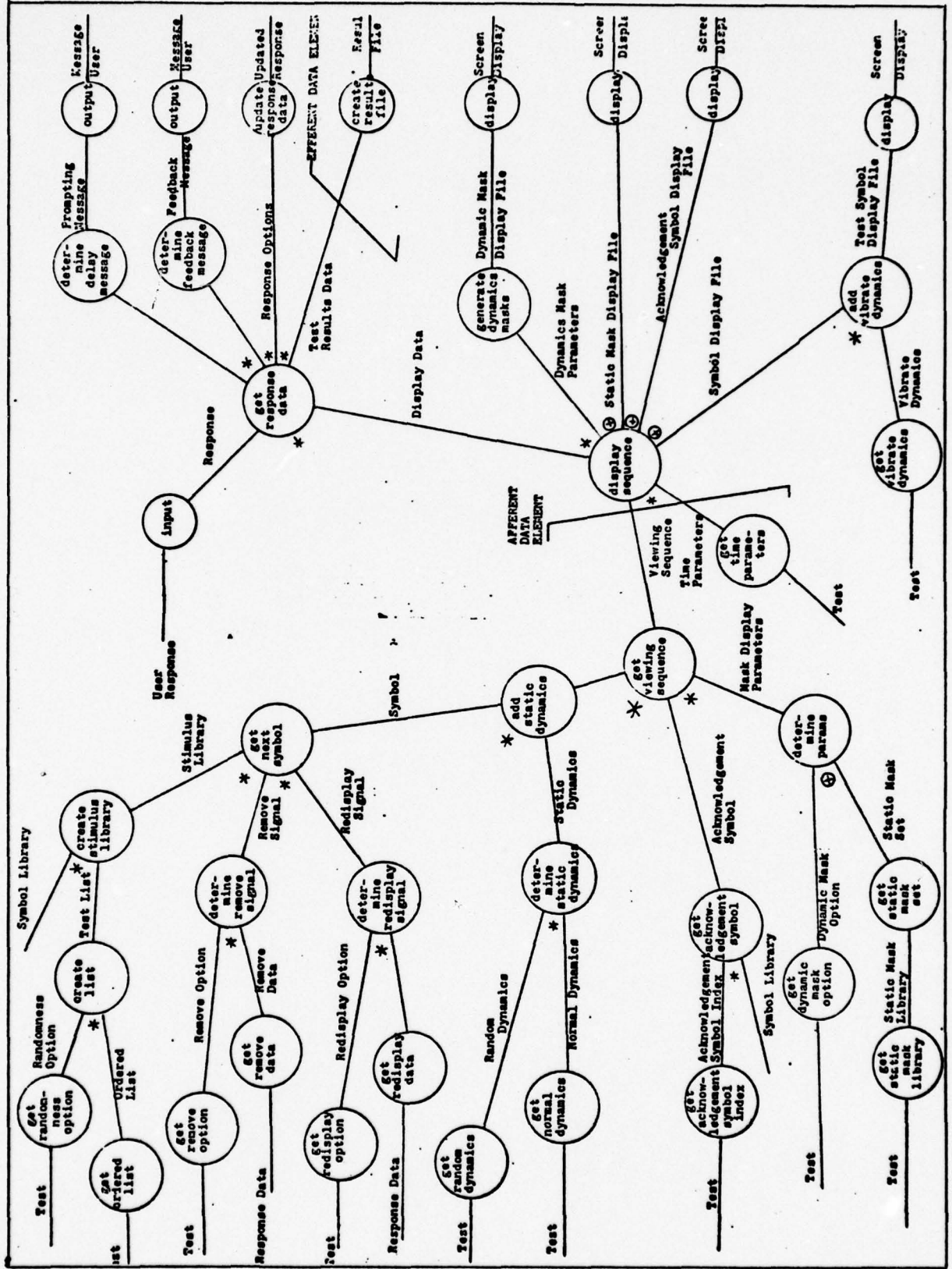


Figure 25. DFD for the Test Execution

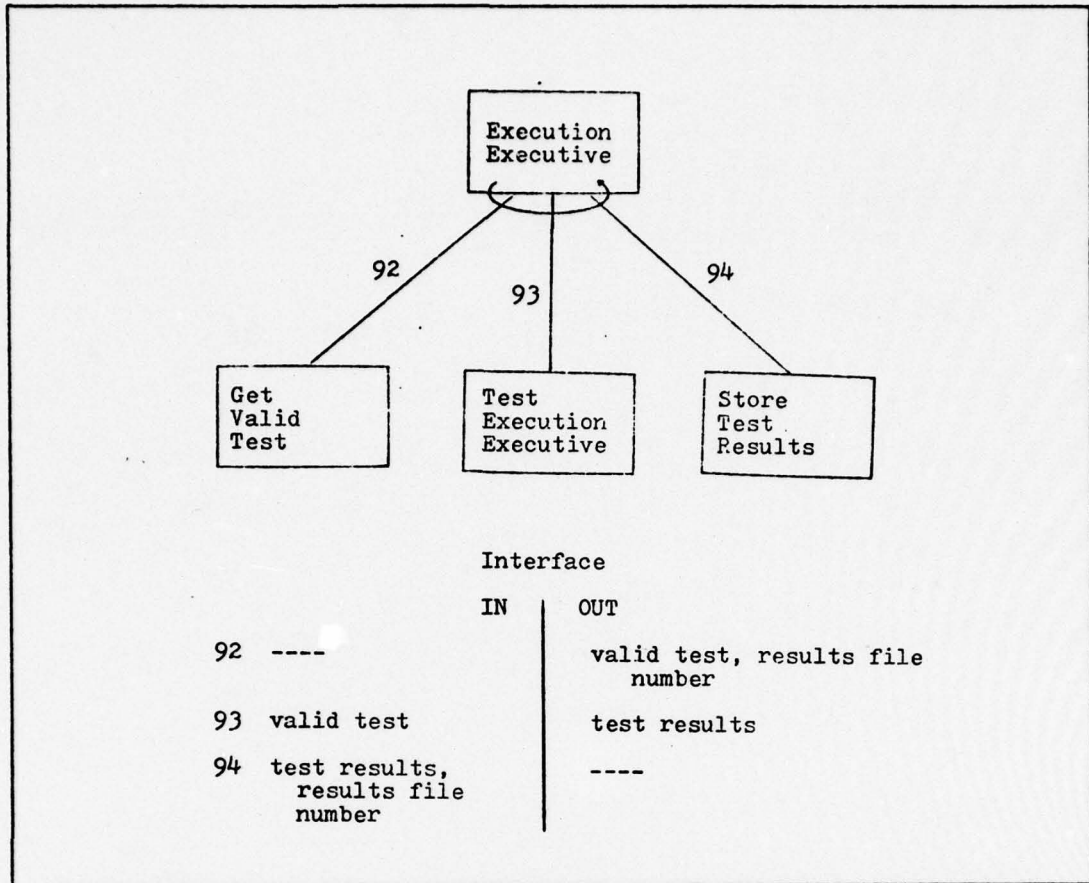


Figure 26. Execution System (First - Level Factoring)

to provide the complete system structure.

The afferent branch (GET VALID TEST) was factored by successively bringing inputs into the system (Fig. 27). At the different levels, the modules communicate with the user through the numerous CONSOLE OUTPUT modules. Most of the system-user communication is limited to this branch.

The central transform branch, being a system in itself, was factored using a transform analysis approach. The TEST EXECUTION EXECUTIVE module was factored into four modules representing the afferent, efferent, and the central transform branches (Fig. 28).

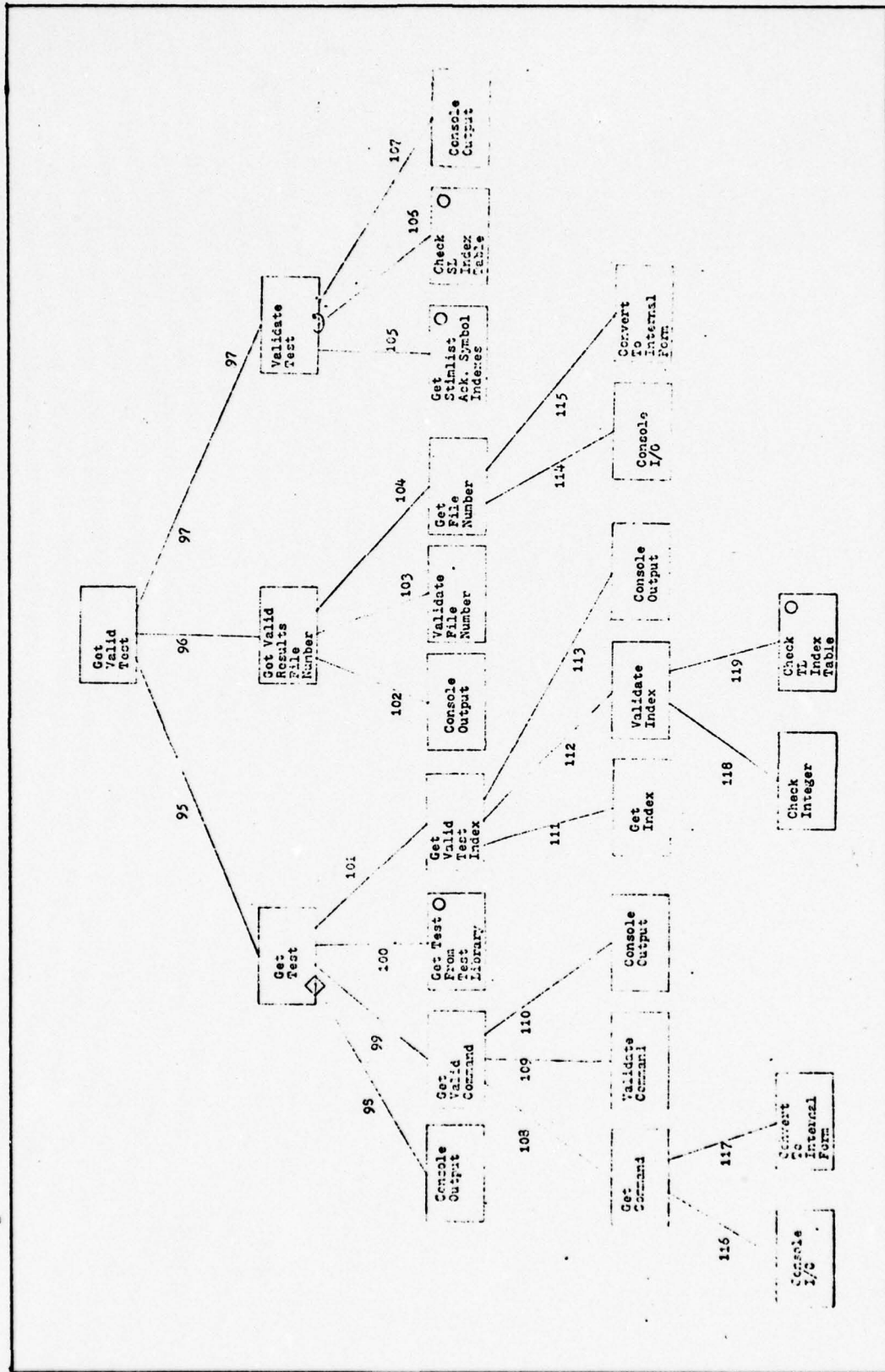


Figure 27. Get Valid Test

		Interface	
		IN	OUT
95	----		test, <u>endflag</u>
96	----		valid results file number
97	test		valid test, <u>okflag</u>
98	error message		----
99	----		valid command, <u>endflag</u>
100	index		test
101	----		valid index, <u>okflag</u>
102	error message		----
103	file number		valid file number
104	----		file number
105	test		indexes
106	indexes		<u>okflag</u>
107	error message		----
108	----		command
109	command		valid command, <u>okflag</u>
110	error message		----
111	----		index
112	index		valid index, <u>okflag</u>
113	error message		----
114	file number request		input line
115	input line		file number
116	command request		input line
117	input line		command
118	index		<u>okflag</u>
119	index		<u>okflag</u>

Figure 27 continued

The factoring of the first afferent branch (GET TIME PARAMETERS) was trivial because it is an operation of the abstract data type--test (i.e., get-item). This branch was

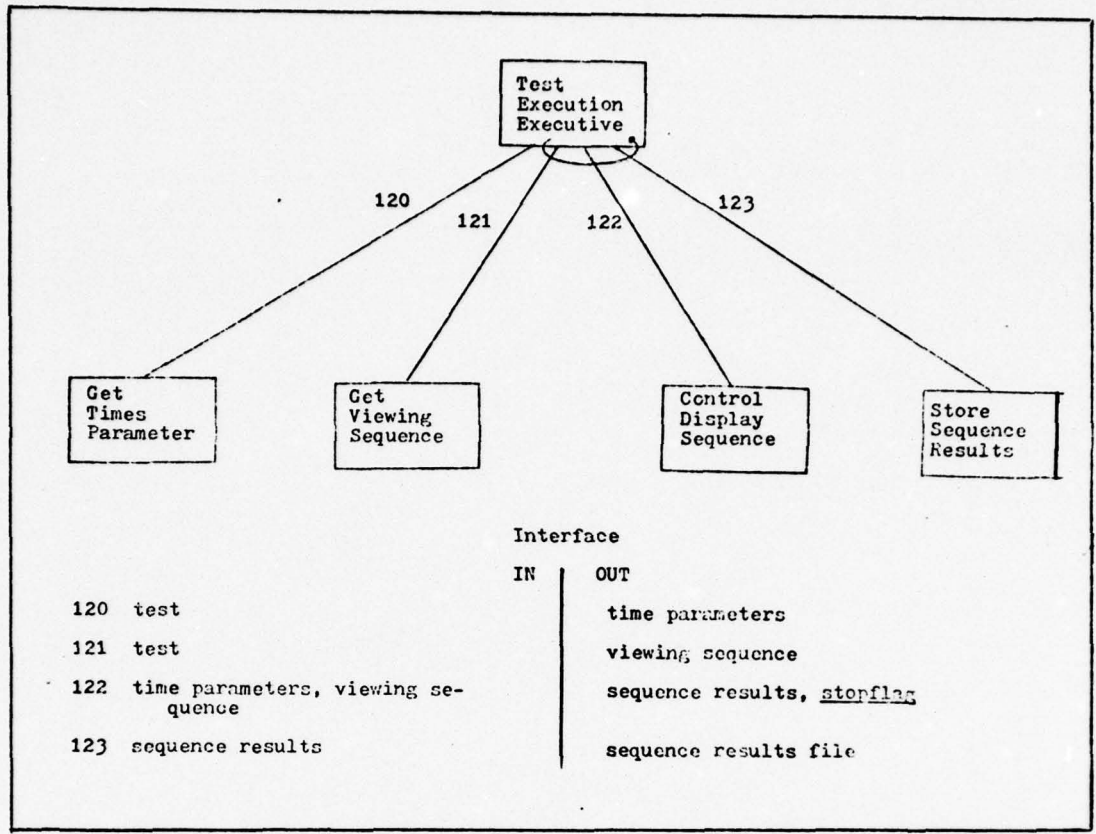


Figure 28. Test Execution (First - Level Factoring)

not factored any further.

The factoring of the GET VIEWING SEQUENCE module demonstrates the relative ease that transform analysis provides when decomposing an afferent branch (Fig. 29). There is almost a one to one correspondence between the data elements and transforms of the DFD to the modules in the structure chart.

The central transforms were designed into one module, CONTROL DISPLAY SEQUENCE (Fig. 30). The two functions of the central transforms--displaying sequences and handling responses--needed to be coordinated. If the display is one that needs updating (i.e., due to the vibration option), then

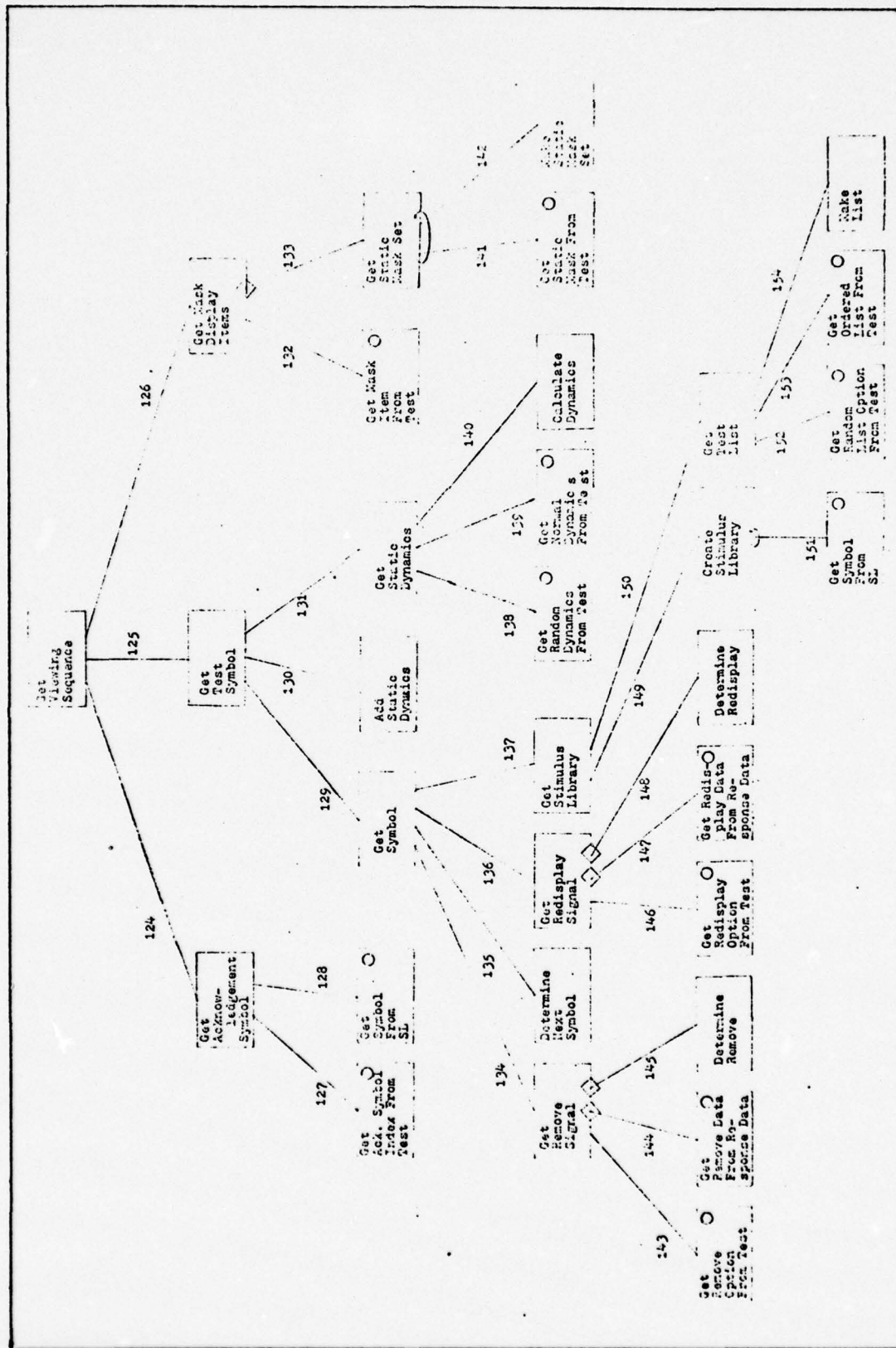


Figure 29. Get Viewing Sequence

		Interface	
		IN	OUT
124	----		acknowledgement symbol
125	----		test symbol
126	----		mask display items
127	----		index
128	index		acknowledgement symbol
129	----		symbol
130	static dynamics symbol		test symbol
131	----		static dynamics
132	----		mask item
133	----		static mask set
134	----		remove signal
135	remove signal, redisplay signal, stimulus library		symbol
136	----		redisplay signal
137	----		stimulus library
138	----		random dynamics
139	----		normal dynamics
140	random dynamics, normal dynamics		static dynamics
141	----		static mask
142	static mask		static mask set
143	----		remove option
144	----		remove data
145	remove option, remove data		remove signal
146	----		redisplay option
147	----		redisplay data
148	redisplay option, redisplay data		redisplay signal
149	test list		stimulus library
150	----		test list
151	index		symbol
152	----		random list option
153	----		ordered list
154	random list option, ordered list		test list

Figure 29 continued

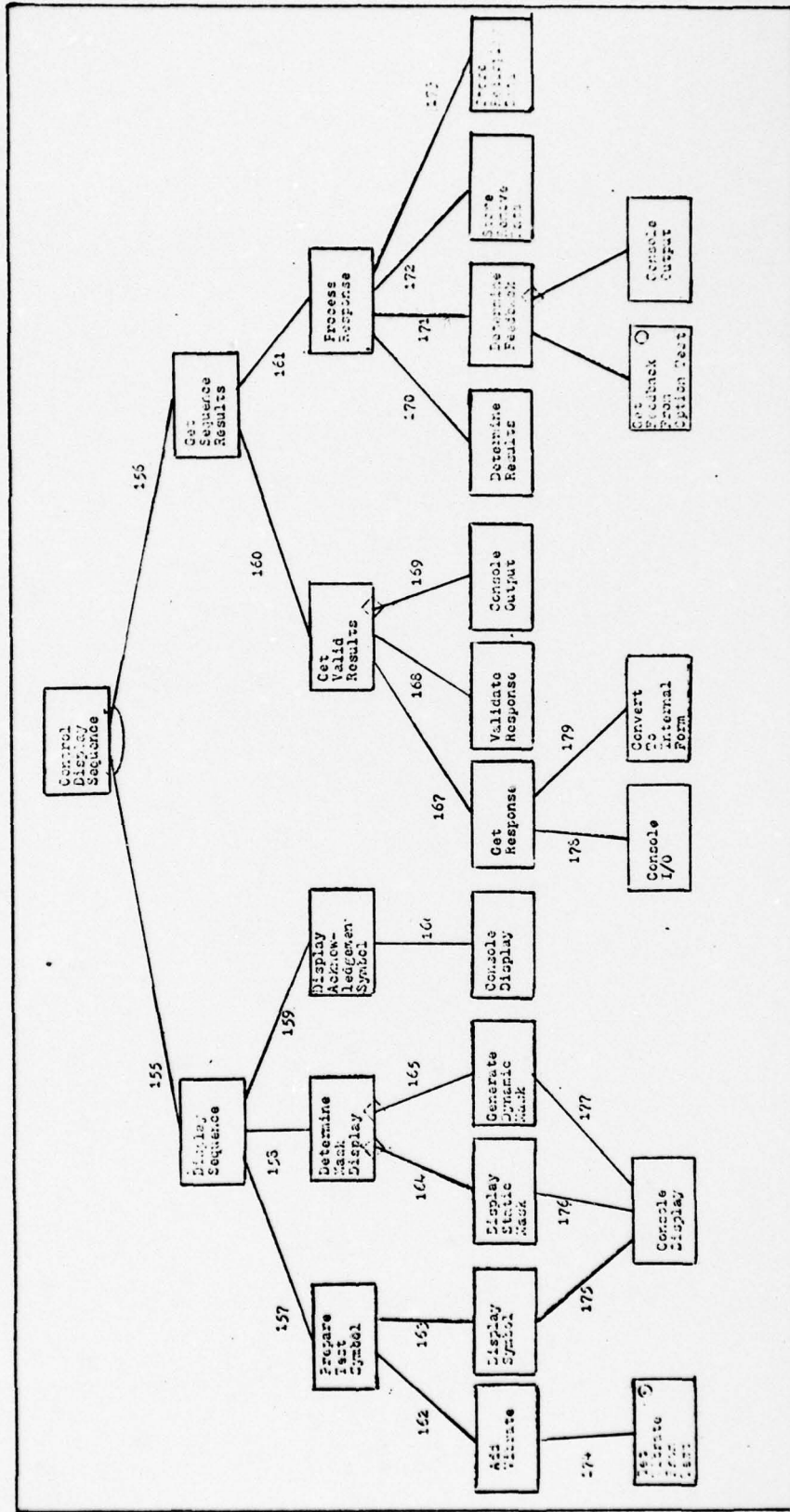


Figure 30. Control Display Sequence

Interface		
	IN	OUT
155	viewing sequence, time parameters	----
156	----	sequence results, <u>stopflag</u>
157	test symbol, stimulus time	----
158	mask item, mask times	----
159	acknowledgement symbol	----
160	----	valid response, <u>stopflag</u>
161	valid response	sequence results
162	symbol	display symbol
163	display symbol	----
164	static mask set	----
165	mask item	----
166	acknowledgement symbol display	----
167	----	response
168	response	valid response, <u>okflag</u> , <u>stopflag</u>
169	error message	----
170	valid response	sequence results
171	sequence results	----
172	remove data	----
173	redisplay data	----
174	vibrate option	----
175	stimulus display	----
176	static mask display	----
177	dynamic mask display	----
178	prompting message	input line
179	input line	response
180	----	feedback option
181	feedback message	----

Figure 30 continued

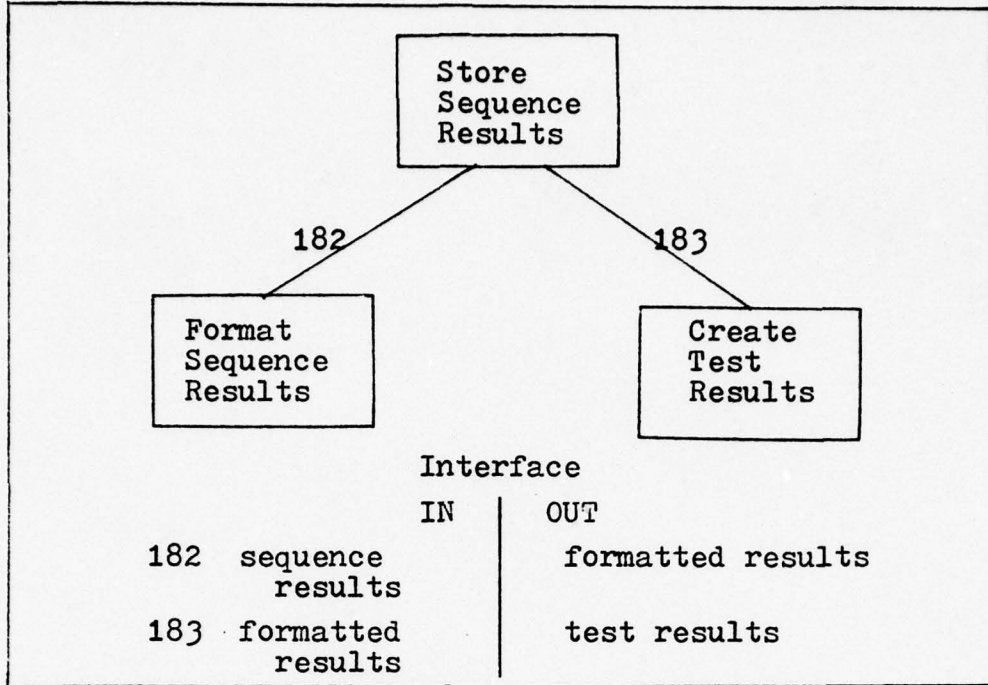


Figure 31. Store Sequence Results

the control must continually shift between the waiting for a response and the updating of the display file.

The efferent branch of the test execution is the storing of the sequence results to form the test results. When the command STOP TEST is input by the user, the test results will be passed on to the execution executive (Fig. 31).

The efferent branch of the execution system stores the test results in a file to be output later (Fig. 32). The results from several executions of the test can be stored in the same file.

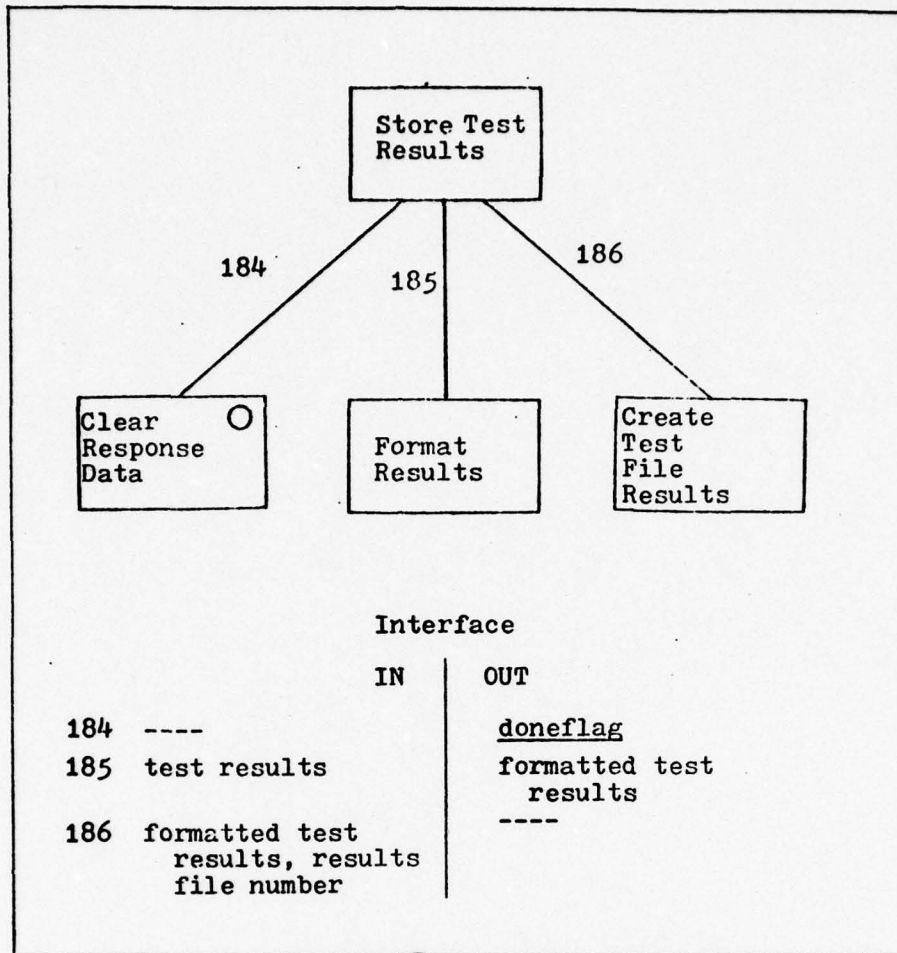


Figure 32. Store Test Results

VI. ANALYSIS PHASE DESIGN

Identify Data Structures

There are two external data structures to this system: the test results file and the collapsed data file. The test results file was output from the execution system. It is used in the analysis phase to provide the necessary data for the creation of the confusion matrices. The collapsed data file stores the data after it has been processed with the predictor matrices.

There are two types of matrices used during this phase: the confusion matrix and the predictor matrix. The confusion matrix can be of three types: response based, reaction time based, and percentage based. The values in the matrix are determined from the test results. The predictor matrix has user-determined values that specify the degree of error of an incorrect response. The information in these two matrices are combined using a collapsing function. This determines an overall degree of error in the recognition of the test symbols. Reference 5 contains examples of confusion and predictor matrices which will not be presented here (Ref 5:18-19).

The user-system interface consists of six commands: LOAD DATA, CONFUSION, PRINT, COLLAPSE, ENTER, and END ANALYSIS. These commands and their options are explained in Appendix B, User's Guide.

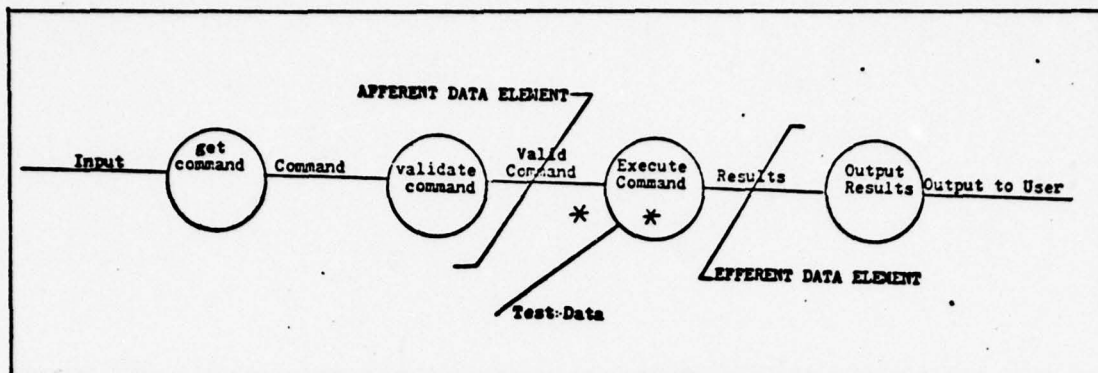


Figure 33. DFD for the Analysis System

Model the Problem with a Data Flow Diagram

The inputs for this system are the user commands and the test results. The outputs are the results from the collapsing routine. This produced a simple model for the problem (Fig. 33). This model is similar to the model for the set-up phase. The afferent data element is the valid user command. The efferent data element is the results from the execution of the commands. The central transform is a transaction center, controlling the execution of the commands.

Design System Using Structured Techniques

The first level factoring of the system executive and the factoring of the afferent branch are shown in Figure 34. The procedure for deriving the first level modules is the same procedure that has been used in the previous designs-- one module for each of the different branches. Further, the afferent module has been decomposed using the transform analysis technique.

The EXECUTE ANALYSIS COMMAND module (transaction center)

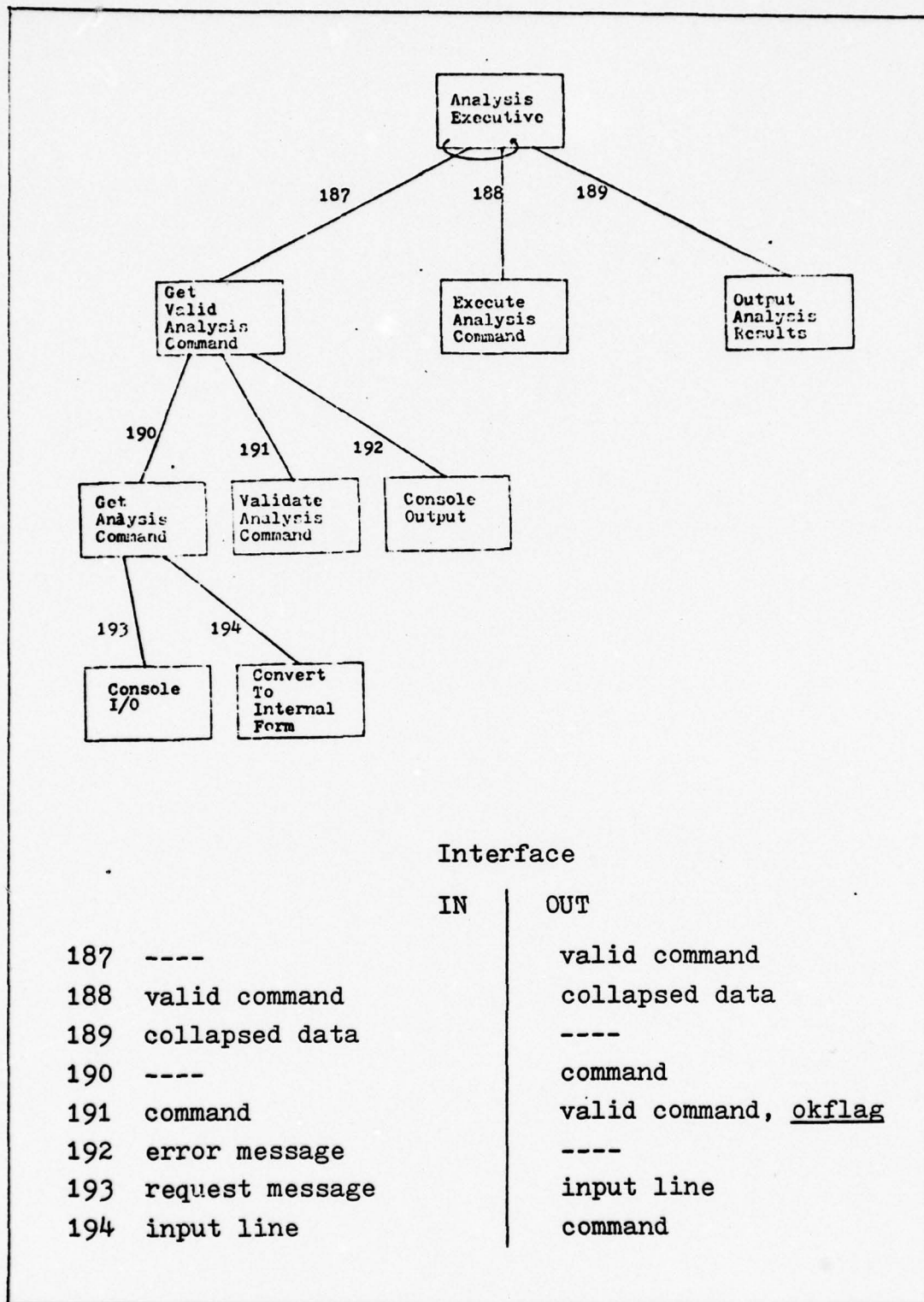


Figure 34. Analysis Executive (First - Level and Afferent Branch Factoring)

controls the entry into the specific execute-modules (Fig. 35). The reason for not having an afferent get-parameters module for the system (as was done in the design of the set-up phase system) is that some of the commands are not executable until others have been completed. For example, the CONFUSION command cannot be done until after the LOAD DATA command has been done. Thus the transaction center screens the commands prior to the calling for their execution.

The decomposition of each of the execution modules was very similar. There was an afferent branch in each which was used to get the parameter needed to perform the function. The central transform branch handled the function and returned its results to the transaction center. These decompositions are shown in Figures 36, 37, 38, 39, and 40. The EXECUTE END ANALYSIS COMMAND is a stub module and does not perform any function other than returning an end-flag .

The decomposition of the efferent branch (OUTPUT ANALYSIS RESULTS) is shown in Figure 41. The results can be stored or output to the user.

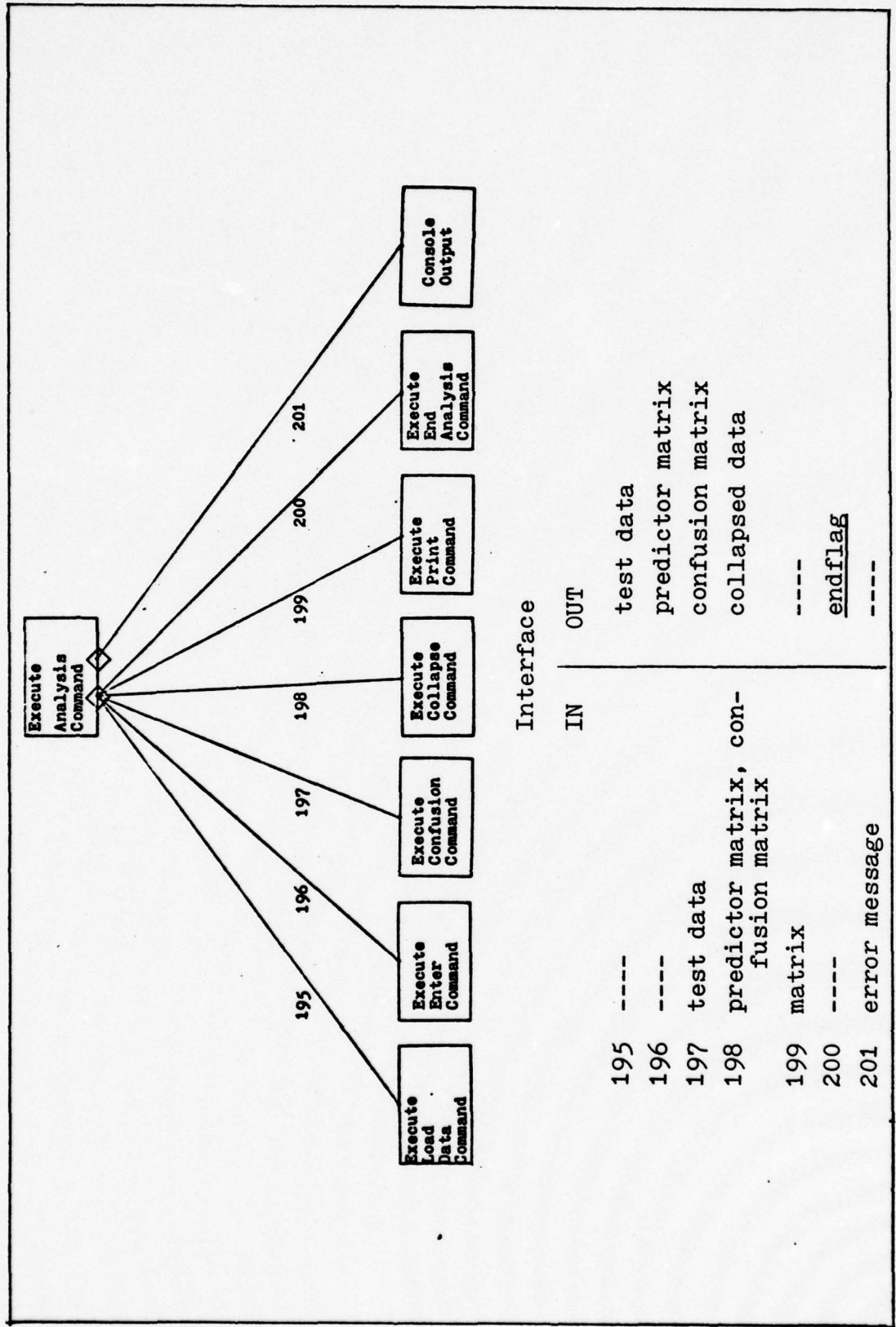
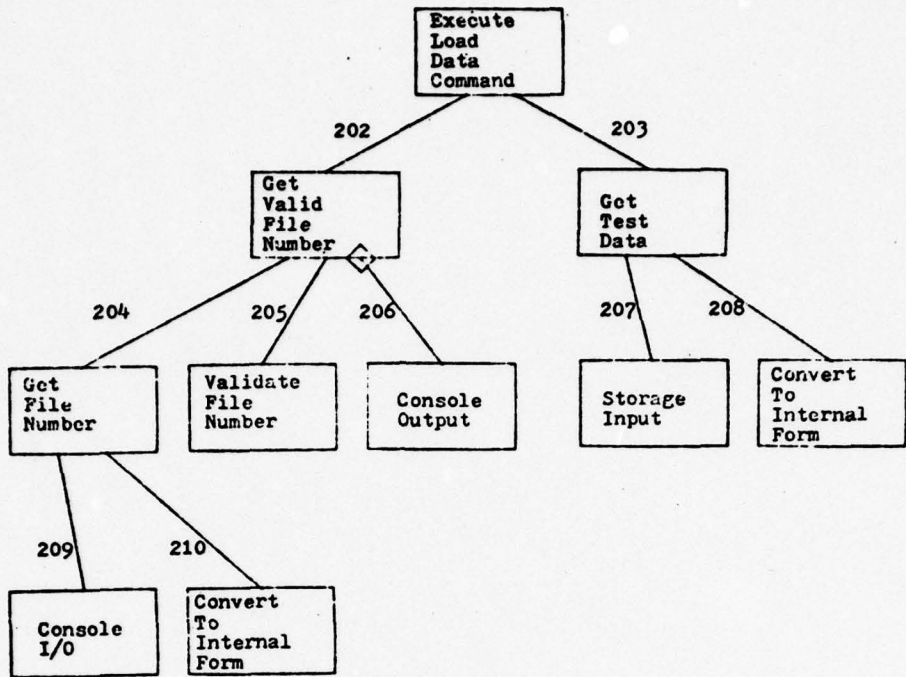


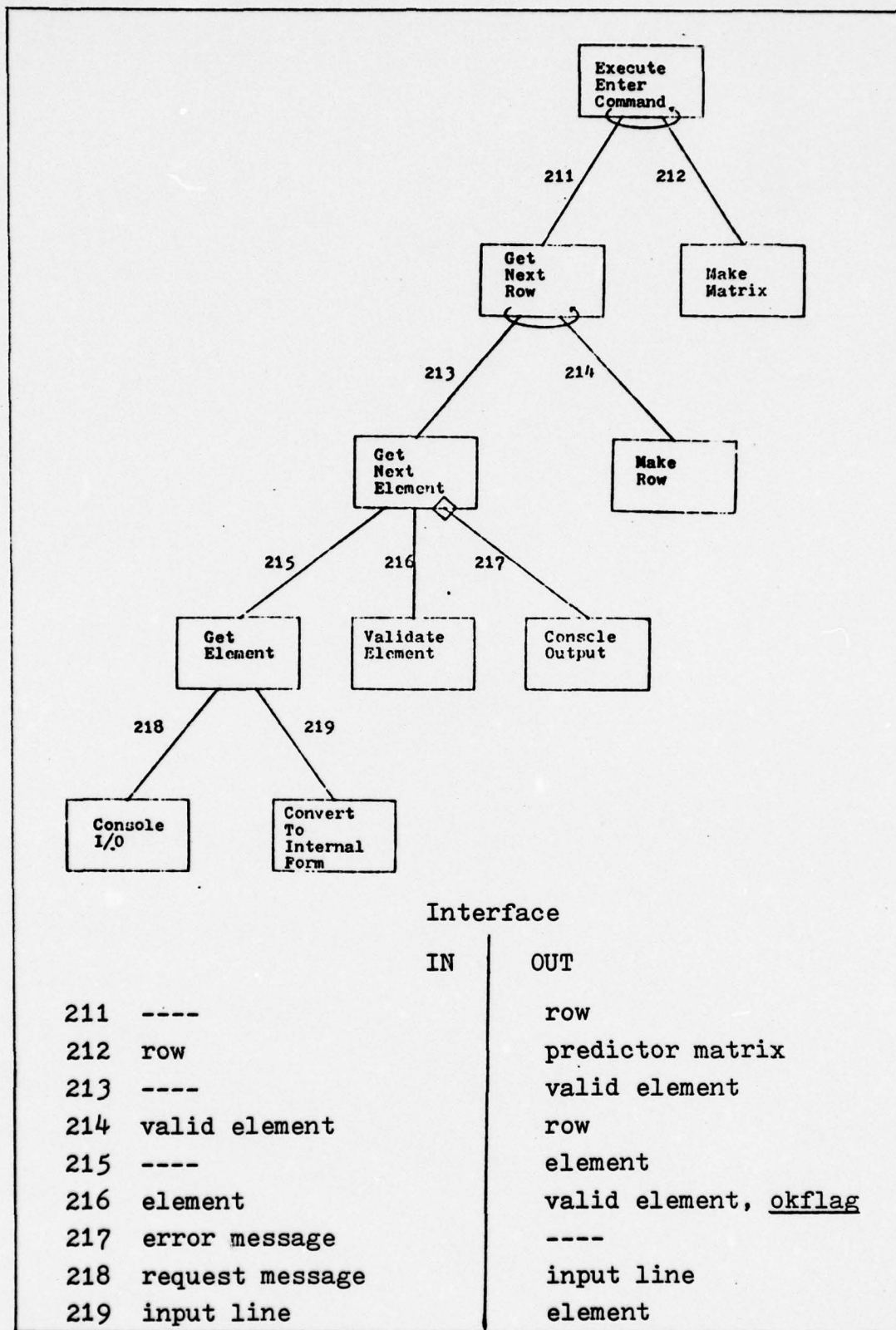
Figure 35. Execute Analysis Command (Transaction Center)



Interface

	IN	OUT
202	----	valid file number
203	valid file number	test data
204	----	file number
205	file number	valid file number, <u>okflag</u>
206	error message	----
207	file number	input data
208	input data	test data
209	request message	input line
210	input line	file number

Figure 36. Execute Load Data Command



Interface

	IN	OUT
211	----	row
212	row	predictor matrix
213	----	valid element
214	valid element	row
215	----	element
216	element	valid element, <u>okflag</u>
217	error message	----
218	request message	input line
219	input line	element

Figure 37. Execute Enter Command

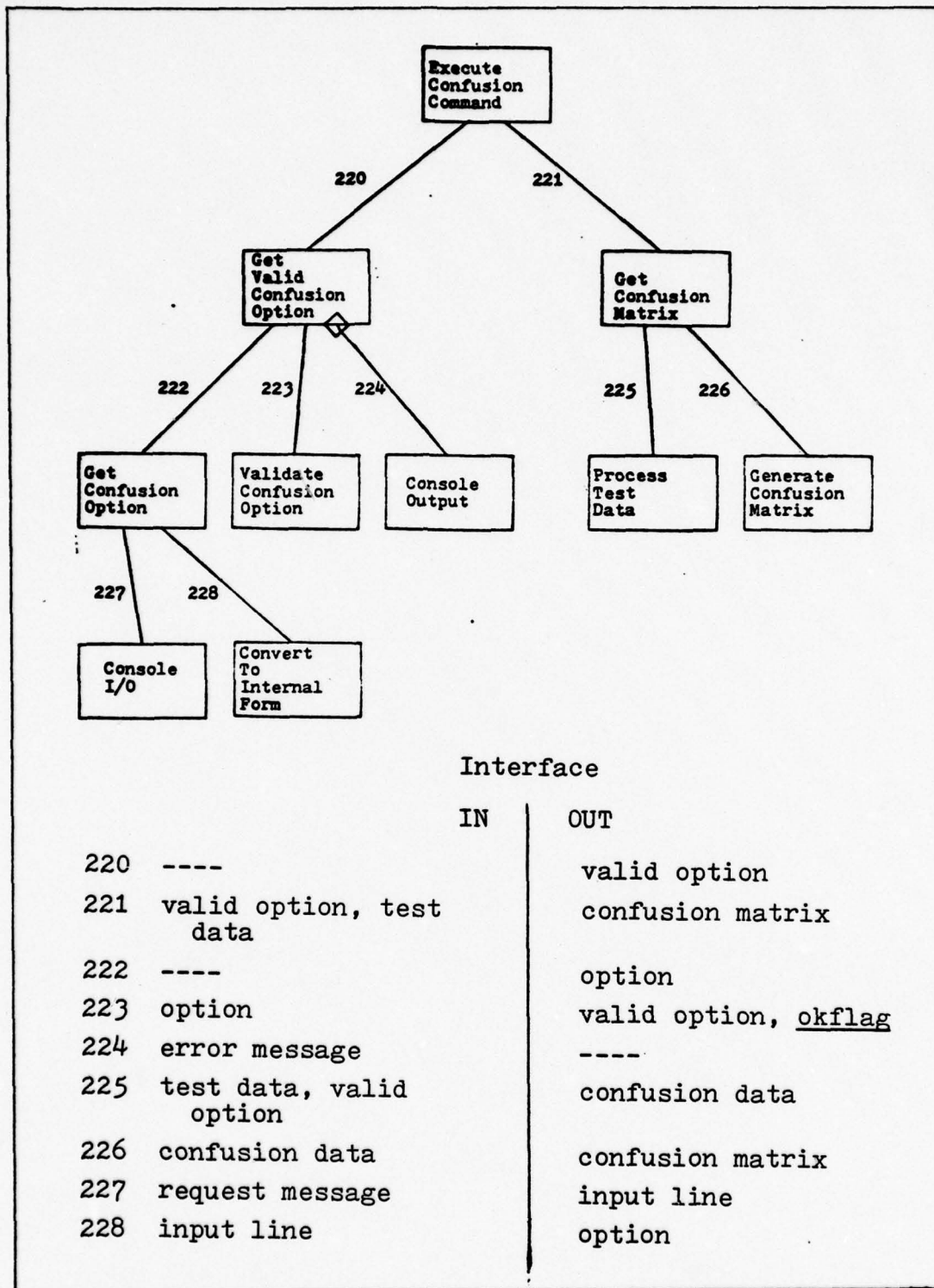


Figure 38. Execute Confusion Command

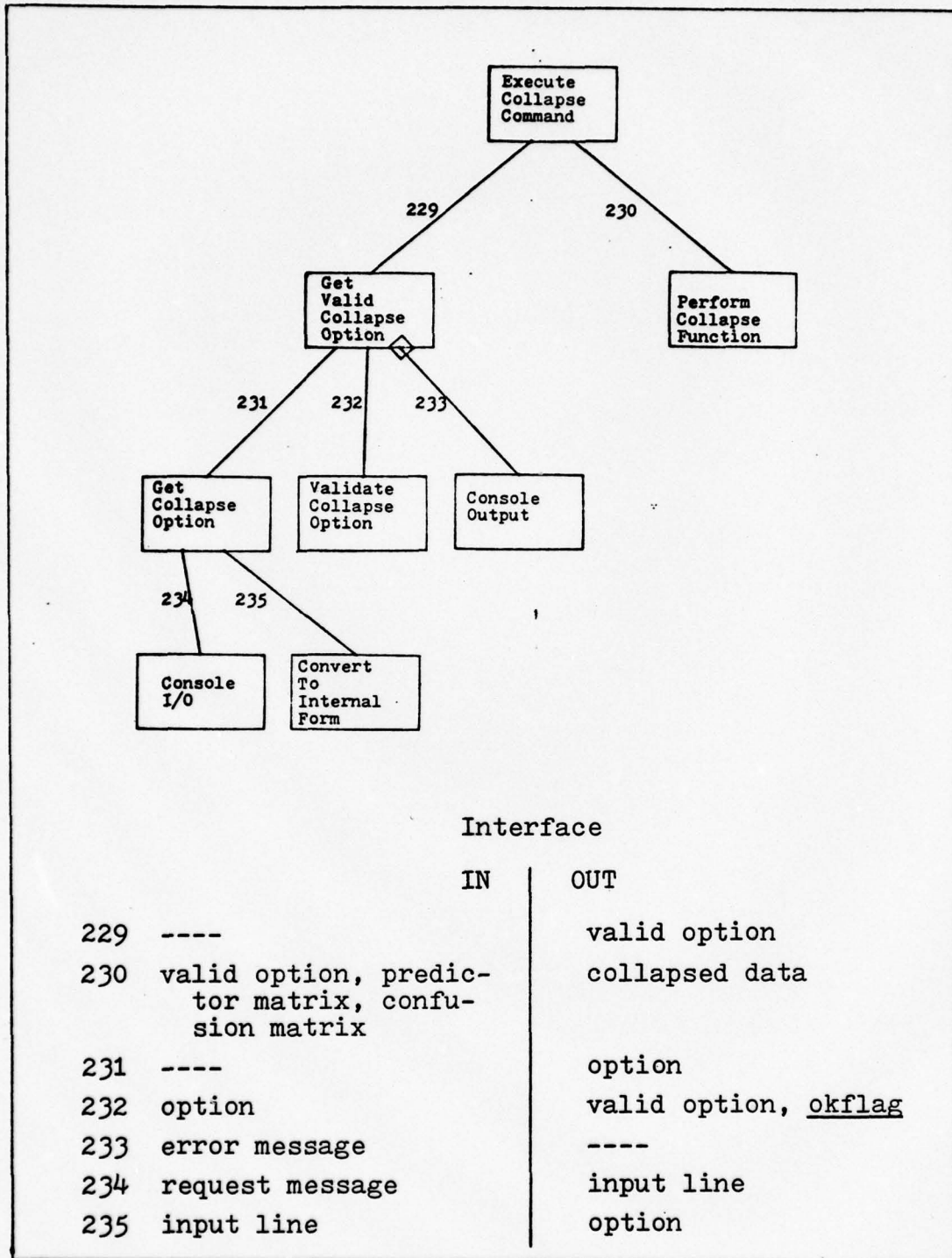


Figure 39. Execute Collapse Command

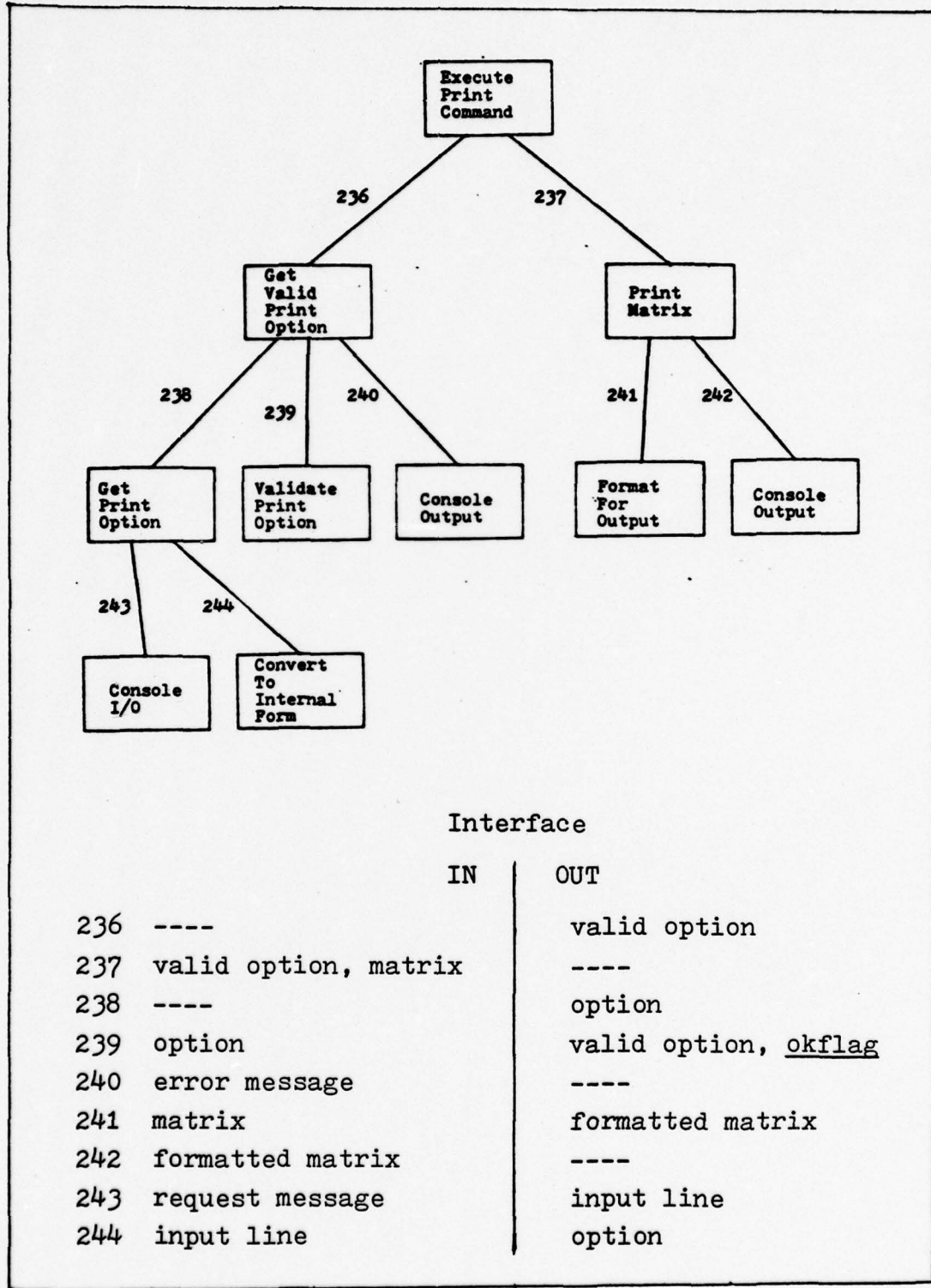


Figure 40. Execute Print Command

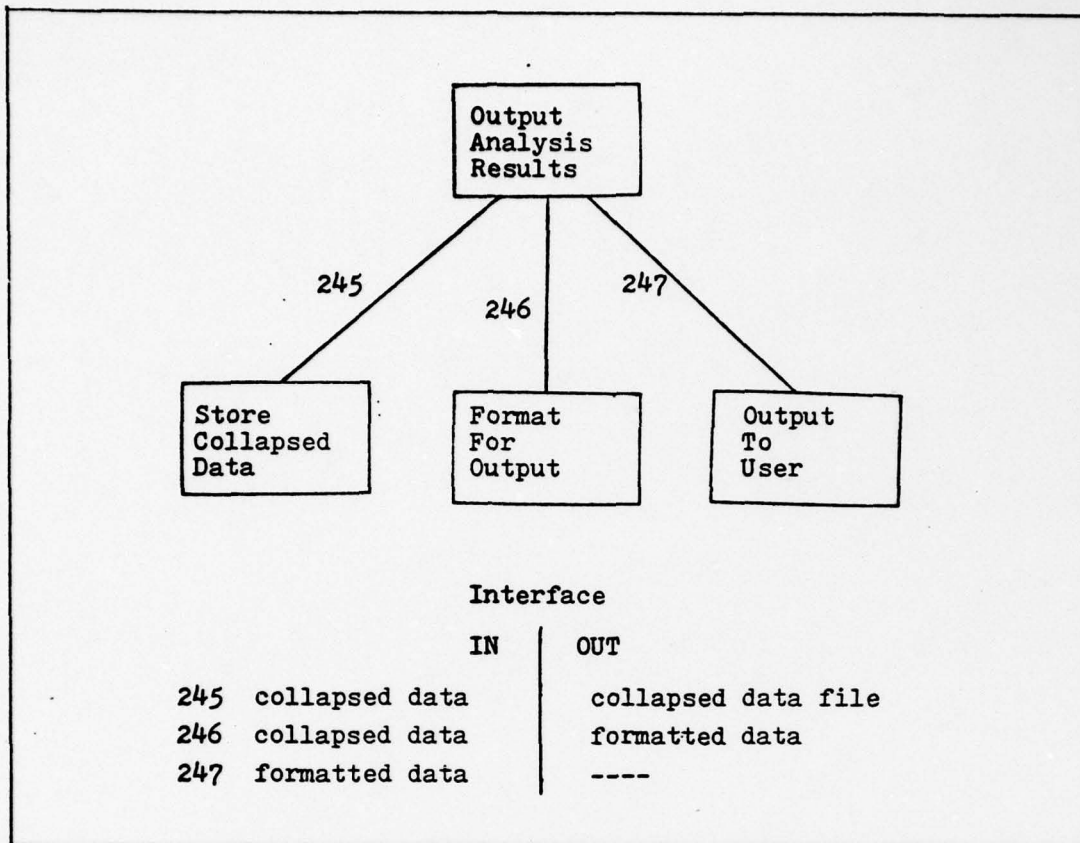


Figure 41. Output Analysis Results

VII. DESIGN EVALUATION

Introduction

The perception experiment software is being developed as a system for use in perception testing. Thus it will undoubtedly undergo many changes and enhancements throughout its lifetime. An interactive system, such as this one, must be easily maintainable.

The method that was used to produce the preliminary design of the PES produces a structure that is maintainable. To measure this maintainability, several changes and enhancements will be proposed for each phase. The amount of change required in the system will be assessed. Finally, the question of optimization will be addressed.

Set Up Phase Modification

The kinds of changes to this phase of the problem would be changes such as adding a new command, deleting a command, changing the function of a command, changing the limits for the parameters of a command, and changing the structure of any of the data bases.

The transaction centered strategy used to design this system, allows the first three types of changes to be handled quite easily and with little effect to the rest of the system. To add a new function would entail changing the VALIDATE SET UP COMMAND module in the first afferent branch of the system (to allow this new command to be valid) and

changing the transaction centers (GET VALID SET UP PARAMETERS and EXECUTE SET UP COMMAND) to reflect the addition. The new function can be placed into the existing structure without affecting any other module. The same three modules are the only ones that will realize that a command has been deleted.

The changing of the function of a command and of the limits of its parameters will be isolated entirely within the two branches coming from the transaction centers. In fact, the changed limits can be handled by changing only one module (CHECK LIMITS).

To add or delete a command, or to change its function, will probably cause changes to the data structures. Since these structures are to be designed as abstract data types, the effect of the changes, except for data operation procedures, will be imperceptible to the rest of the system.

Execution Phase Modification

The types of changes to the execution phase would be the expanding of the response-directed options, the elimination of the dynamically visible options (vibrations and dynamic masks) and the allowing of the user to specify the number of test symbols to be shown.

The expanding of the response-directed options would cause changes in two areas. First, the data structure, response data, would have to be expanded to maintain more test history. This will have no effect to the system

because response data is an abstract data type. Also in this area will be an addition of a module to the PROCESS RESPONSE module. This new module would determine the data for the response data file. Second, assuming the test has been modified, modules would have to be added to the afferent branch of the GET TEST SYMBOL module. This case would only be done if the new option would affect the symbol list. In both cases, the modularization of the system has prevented the change from causing widespread changes.

The elimination of the dynamics options is a definite possibility. The changing of the display while waiting for a response is a problem that has not yet been investigated. To eliminate them from the system would entail deleting the associated modules and modifying the one or two modules with which they communicated. There should be no other changes to the system.

If the user is provided the option to specify the number of test symbols to be shown, the only module affected would be the CONTROL DISPLAY SEQUENCE module. Since the transform has been structured so that this module coordinates all activities associated with the display sequence, the rest of the system will not be affected by the new option.

Analysis Phase Modification

The changes to the analysis phase would be similar to those in the set-up phase. These changes would be in the

adding of new capabilities to the analysis. The addition of functions, as in the set-up phase modifications, would cause changes only to the transaction center and to the validate module. The only other type of change would be a requirement for more, and varied, types of output reports. Here, the change to a system would be limited to the efferent branch of the system.

Optimization

This preliminary design provides a structure that allows the designer the maximum flexibility in decision-making during the rest of the design and implementation stages. It provides a complete structure chart in which the modules are as functional as possible.

If the designer must optimize (and there are many that are against such a practice (Ref 4:vii)), this structure will make him aware of how he is affecting the system. This allows him the maximum choice of areas to optimize. If he is optimizing space, he can see the modules that can be most easily recomposed without affecting the system. If he is optimizing for speed, he can determine those critical areas that are executed the most and redesign those areas. Since the areas have been designed for minimal communication, careful redesign of those areas should cause only a few changes.

VIII. RESULTS AND CONCLUSIONS

This chapter discusses the results obtained from the software design and the conclusions drawn from these results. It also provides recommendations for the further design and implementation of the perception experiment software.

Results

The design for the perception experiment software was accomplished by following a well-defined structured method in a rigorous, disciplined manner. The requirements definition for the problem was presented informally in Chapter II. Knowledge of the problem requirements is necessary to form a sound foundation on which to design a program structure. A design philosophy emphasizing program maintenance was used to develop the tools with which to build the program structure. The tools were in the form of a design method which, when rigorously followed, produces an easily maintainable structure (Chapter III). The program structure for the three phases of the problem was built using this design method (Chapters IV, V, and VI). The structure was evaluated in Chapter VII and shown to be easily maintained and modified.

The application of structured design techniques resulted in a program design which will be easy to implement, maintain, and modify. Most of the modules in the design are singly functional and easily understood. The use of

abstract data types for the data structures within the system greatly facilitated the design effort as well as enhanced the maintainability of the system.

Conclusions

The most important factor in designing a system is to have a clear idea of what is required. Without this knowledge, it is nearly impossible to develop a workable program. The next most important factor is to develop a good design prior to any coding being done. This design must be accomplished in a methodical manner. Typically, a designer cannot haphazardly design a program and then expect someone else to have an easy time maintaining it. Structured design provides a methodical approach to producing programs that anyone can maintain. It is only after the complete program structure has been designed that detailed module design and coding of the program should begin.

Recommendations

The perception experiment software is in the design stage of development. The following recommendations are provided as guidelines for the further development of the system.

The abstract data type interfaces for all of the data structures in the system must be defined prior to coding. Without this definition, it will be impossible to produce any meaningful code. The three systems can be coded concurrently

if the interfaces of the data structures that are shared between the systems are defined first.

Finally, the software should be implemented using top-down methods in coding and testing. In this way, the system interface testing is accomplished throughout the coding stage. Implementation using a top-down approach starts at a single point, but parallel coding becomes possible as the lower-level components are developed.

Bibliography

1. Boehm, B. W. Software Engineering. TRW Systems Engineering and Integration Division, Redondo Beach, Calif.: (October 1976).
2. Brooks, F. P. The Mythical Man-Month. Reading, Mass.: Addison-Wesley Publishing Company, 1975.
3. Constantine, L. L. and E. Yourdon. Structured Design. New York: Yourdon Inc., 1975.
4. Jackson, M. A. Principles of Program Design. New York: Academic Press Inc., 1975.
5. Liebeck, R. A. Software Design for Multimode Matrix Display Perception Tests (Thesis). (December 1976).
6. McGowan C. L. and J. R. Kelly. Top-Down Structured Programming Techniques. New York: Mason/Charter Publishers, Inc., 1975.
7. Myers, G. J. Reliable Software Through Composite Design. New York: Mason/Charter Publishers, Inc., 1975.
8. Parnas, D. L. "On The Criteria To Be Used In Decomposing Systems Into Modules". Communications of the ACM, 15: 1053-1055 (December 1972).
9. Ross, D. T., et al. "Software Engineering: Process, Principles, and Goals." Computer: 89-99 (May 1975).
10. Yourdon, E. "Structured Maintenance--Approach Trains User to Read Alien Code." Computerworld, 37: 38 (12 September 1977).

Appendix A
MODULE DESCRIPTION

System Description

The perception experiment software was developed as three separate systems. Implicit in this development is the idea that an overall system executive would coordinate the phase executives (Fig. 42). The SYSTEM EXECUTIVE, after receiving a valid command, passes control to one of the three phase executives. This module and its subordinates control the entire perception experiment system.

The GET VALID COMMAND module has a structure that appears

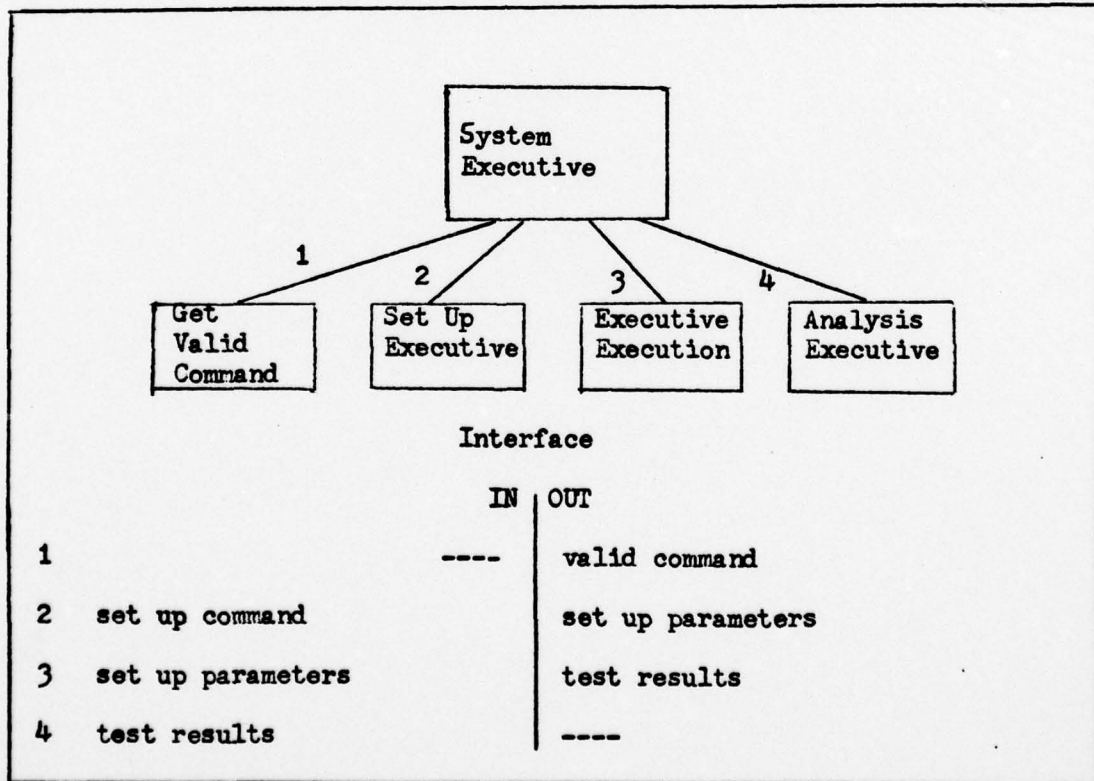


Figure 42. PES System Executive (First - Level Factoring)

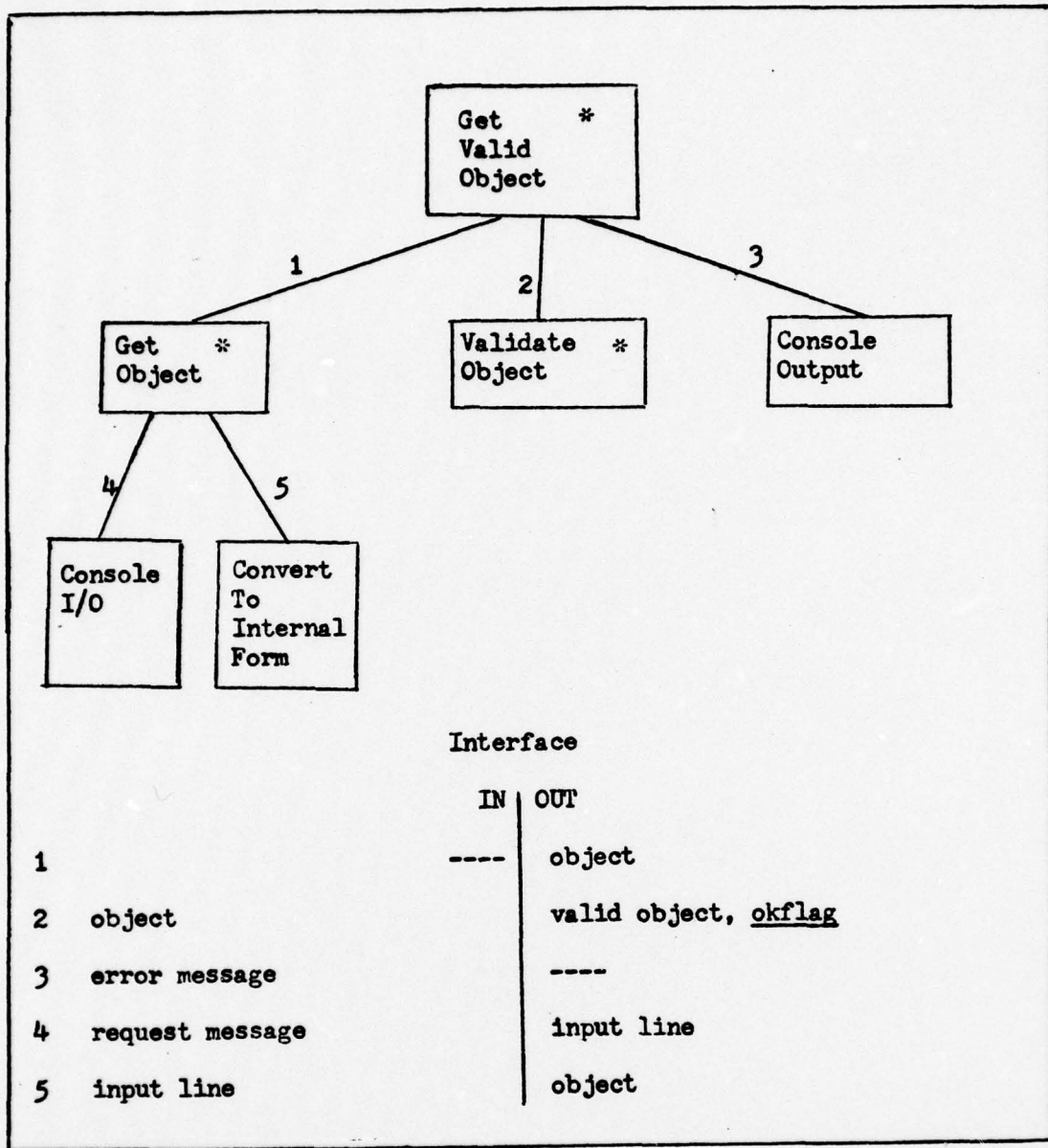


Figure 43. Afferent Group Structure

throughout the entire system (Fig. 43). This structure will be termed an afferent group. It will be described in general terms which will apply to all afferent groups in the system. The modules marked with asterisks are the modules that change for a particular application.

The GET VALID OBJECT module provides a valid object to its calling module. It gets an object from the GET OBJECT module and determines its validity. If the object is invalid, an error message will be output through CONSOLE OUTPUT. In this case, the GET VALID OBJECT module will call the GET OBJECT module again, in an attempt to get a valid object.

The GET OBJECT module provides the appropriate request-object message through the CONSOLE I/O module. The input line will be sent to CONVERT TO INTERNAL FORM. This takes the input line and converts it to a form recognized throughout the system.

The VALIDATE OBJECT receives the object and, depending on the application, determines its validity. It then returns the object to GET VALID OBJECT with a flag as to its validity.

This afferent group, as it applies to the SYSTEM EXECUTIVE, would be described like this. The GET COMMAND module would request the user to enter a system command. This command is converted to a usable form (by CONVERT TO INTERNAL FORM) and passed to the GET VALID COMMAND module. This module would pass the input command to the VALIDATE COMMAND module. This module would ensure that the command was either a SET UP, EXECUTE, or an ANALYZE command as defined in Appendix B. If it was, an okflag would be returned and GET VALID COMMAND would pass the valid command upwards. If it was not valid, a not-okflag would be returned. GET VALID COMMAND would provide the user with an error message through CONSOLE OUTPUT. It would then invoke GET COMMAND to request another

input.

Set-up Phase Description

The SET UP EXECUTIVE invokes its subordinates to execute the set-up commands (Fig. 6). A valid command would be any one of the SET UP commands in Appendix B. When it receives a valid command from the afferent group, GET VALID SET UP COMMAND, no validation has been done on the parameters that might have been entered with the command. These are passed to the transaction center GET VALID SET UP PARAMETERS. After getting the parameters, the SET UP EXECUTIVE invokes the EXECUTE SET UP COMMAND transaction center. The EXECUTE SET UP COMMAND module will return an updated data structure to the phase executive. An end-flag will be returned when the command END SET UP is entered by the user. The SET UP EXECUTIVE, upon receiving the end-flag will output an appropriate message to the user and return control to the SYSTEM EXECUTIVE.

The GET VALID SET UP COMMAND module (Fig. 7) is an afferent group. When it is invoked by the SET UP EXECUTIVE, it will return one of the valid set-up commands. The GET SET UP PARAMETERS (Fig. 8) and the EXECUTE SETUP COMMAND (Fig. 12) are transaction centers that are almost purely executive in function. When invoked, they determine the appropriate module to call on to execute the command.

Those commands that have more than one parameter associated with them have COLLECT PARAMETERS modules that are shown in Figure 9. The GET TYPE I PARAMETERS creates the test-item

by successively getting valid parameters from the afferent group, GET VALID TYPE I PARAMETER and passing them to COLLECT PARAMETERS. The GET VALID TYPE I PARAMETER module will request specific parameters from the GET TYPE I PARAMETER module. This will cause the output of the appropriate parameter-request message. The VALIDATE TYPE I PARAMETER module can call on numerous submodules to determine validity. The CHECK ALPHA module checks to see that the parameter consists of alphabetic characters and returns a flag if it is not. The CHECK INTEGER and CHECK REAL modules ensure that the parameter is an integer or a real number, respectively. The CHECK SL INDEX TABLE and the CHECK TL INDEX TABLE are operations of the abstract data types symbol library and test library, respectively. The CHECK TYPE I LIMITS module will check that the parameter is within the limits prescribed by the user.

The structure of the GET VALID TYPE II PARAMETER module is similar to the structure of the commands with more than one parameter (Fig. 10). The main difference is that the GET VALID TYPE I PARAMETER (of Figure 9) functions have been incorporated into the head of the structure. This was possible because there is only one parameter needed.

For the commands without any parameters, the structure of the GET TYPE III PARAMETER consists of a stub (Fig. 11). In this case, control is returned immediately to the calling module.

The EXECUTE SET UP COMMAND module (Fig. 12) calls

on one of 22 modules to execute a particular command. The modules affecting the active test are depicted in Figure 13. The EXECUTE ITEM COMMAND module will call on the STORE ITEM IN AT module. This is a function of the abstract data type AT. The user message is a review of the parameters that were just stored.

The EXECUTE MASK COMMAND module determines which option, static or dynamic, has been chosen (Fig. 14). If static was selected, then MAKE MASK FILE is invoked. This stores the static masks as items in the AT. If dynamic was selected, the item is immediately stored in the AT. The user message details the action taken. The MAKE STATIC MASK FILE calls GENERATE STATIC MASK and STORE STATIC MASK IN ACTIVE TEST until the specified number of masks are made.

The EXECUTE CLEAR PARAMETER COMMAND module will make repeated calls to GET DEFAULT VALUE and STORE ITEM IN AT (Fig. 15). In each call, it will request a different default value for the different item in the AT. GET DEFAULT VALUE and STORE ITEM IN ACTIVE TEST are operations of the abstract data types, default value table and active test, respectively. The user message will denote that default values are in the AT.

EXECUTE PRINT PARAMETERS COMMAND will request the items by the options (i.e., MASKS, STIMLIST, ALL) from the GET ITEM FROM ACTIVE TEST (Fig. 16). The item message that is output contains the requested items.

EXECUTE STORE ELEMENT will store either a symbol from

AD-A055 178

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
PRELIMINARY DESIGN FOR MULTIMODE MATRIX PERCEPTION EXPERIMENT S--ETC(U)
DEC 77 S G WENSKA

UNCLASSIFIED

AFIT/GCS/EE/77-11

NL

2 OF 2
AD
A065 178



the AS or a test from the AT into the respective library (Fig. 17). The user message will reflect the action being taken.

The EXECUTE PURGE COMMAND invokes the DELETE ELEMENT FROM LIBRARY module to remove an element from the library (Fig. 18). The user message reflects this change.

EXECUTE RELOAD COMMAND takes an element from the appropriate library (using GET ELEMENT FROM LIBRARY) and stores it in the active symbol or active test (Fig. 19). The message to the user will notify him that the reloading operation is complete.

EXECUTE LIST COMMAND searches the library index table to determine the response to the user's request (Fig. 20). The abstract data type functions are shown by COUNT INDEXES IN LIBRARY and LIST INDEXES IN LIBRARY. The message to the user is the information desired by the user.

The EXECUTE CREATE COMMAND module with its subordinates enables the creation and display of the active symbol (Fig. 21). The afferent group GET VALID INPUT will return a legal create mode command or a symbol segment. The EXECUTE INPUT will receive the input and determine the action to be taken. EXECUTE INPUT calls on STORE SEGMENT, which stores a segment into the active symbol and causes the system to display the change; CLEAR SYMBOL, which removes all segments from the AS and clears the display screen; and EXECUTE ALTER, which processes the ALTER commands.

The EXECUTE ALTER module has its own transform-centered

design (Fig. 22). The afferent group GET ALTER COMMAND provides a valid ALTER command. The GET ALTER PARAMETERS is a transaction center sharing modules with the branches of the set-up phase dynamics commands. EXECUTE ALTER COMMAND directs the changes in the active dynamics list to determine the display.

The EXECUTE BULK OPERATION shows that the same structure is valid for both the BULK LOAD and BULK STORE operations (Fig. 23). This module calls on EXECUTE SL OPERATION or EXECUTE TL OPERATION which does the mass loading/storing of the libraries.

Execution Phase Description

The EXECUTION EXECUTIVE calls on three modules to execute a test (Fig. 26). As an executive, it acts as a communication center for passing data structures and coordinating test execution activities.

The GET VALID TEST module calls on the GET TEST module and the VALIDATE TEST module in order to determine a valid test (Fig. 27). The GET TEST module uses two afferent groups--GET VALID COMMAND and GET VALID TEST INDEX-- to determine which test is to be executed. It retrieves the test through the GET TEST FROM TEST LIBRARY module. The VALIDATE TEST checks to see that all the indexes used in the test (the stimulus list and the acknowledgement symbol index) are present in the symbol library index table. If not, an error message is output and control will return to GET VALID TEST.

This module will then attempt to get another test.

The TEST EXECUTION EXECUTIVE coordinates the actions during the test itself (Fig. 28). It uses the GET TIME PARAMETERS module to get the time values that will be used to time the viewing sequences. GET VIEWING SEQUENCE, CONTROL DISPLAY SEQUENCE, and STORE SEQUENCE RESULTS are invoked in a loop to provide for test execution.

GET VIEWING SEQUENCE is an afferent module which gets the displays needed in a viewing sequence (Fig. 29). The GET ACKNOWLEDGEMENT SYMBOL gets the symbol by first calling GET ACK SYMB INDEX FROM TEST to get the index. It then invokes GET SYMBOL FROM SL to get the acknowledgement symbol.

The GET TEST SYMBOL calls on the GET SYMBOL module to get the next symbol to be shown; it then applies the static dynamics to it. The GET SYMBOL module chooses the next symbol to be shown. Depending on the redisplay and removal options, this module will provide either the next symbol in the stimulus library (unless it is a removal symbol) of the previous symbol shown (according to the redisplay option).

The GET STIMULUS LIBRARY creates the library by getting the list of symbols (already modified by randomness, if applicable) and getting those symbols from the symbol library.

The GET TEST LIST module gets the list to be used for the test by getting the ordered list and the randomness option from the test. These are passed to MAKE LIST which applies randomness as required.

The GET REMOVE SIGNAL and the GET REDISPLAY SIGNAL will

get the respective option from the test. If there is an option, the DETERMINE REMOVE/DETERMINE REDISPLAY module will be invoked to check the response data to determine if the option applies to the next symbol to be shown.

The GET STATIC DYNAMICS module gets the dynamics that affects the symbol prior to display time.

GET MASK DISPLAY PARAMETERS will get either the two static masks that are needed (GET STATIC MASK SET) or the dynamic masks parameters needed to generate the masks (GET DYNAMIC PARAMETERS).

The CONTROL DISPLAY SEQUENCE coordinates the actions of displaying a sequence and handling the subject response (Fig. 30). This must be done to ensure that a display is maintained while waiting for a subject to respond.

The DISPLAY SEQUENCE module coordinates three subordinates to display a viewing sequence. The DETERMINE MASK DISPLAY shows either a static or a dynamic mask. PREPARE TEST SYMBOL displays the symbol while applying the vibrate dynamics to it.

GET SEQUENCE RESULTS uses the GET VALID RESPONSE afferent group to determine the action to be taken. A valid response may be either a response to the symbol, a special response, or the STOP TEST command. If one of the responses is received, PROCESS RESPONSE updates the response data structure. If the STOP TEST command is received, a flag is sent to the TEST EXECUTION EXECUTIVE and the appropriate actions will be taken. DETERMINE FEEDBACK provides the

feedback message to the subject based on the reinforcement option and the print option.

The STORE SEQUENCE RESULTS module is shown in Figure 31. This creates a file for each test. The STORE TEST RESULTS (Fig. 32) for the test execution is activated after the stop-flag is received by the EXECUTION EXECUTIVE. The modules take the stored sequence results and stores them as a test. If the test is executed several times, the module will store the results of each test in the test results file.

Analysis Phase Description

The ANALYSIS EXECUTIVE uses the GET VALID ANALYSIS COMMAND afferent group to get a command for execution (Fig. 34). These commands are listed in Appendix B. Collapsed data results can be outputted via the OUTPUT ANALYSIS RESULTS module. EXECUTE ANALYSIS COMMAND is a transaction center which calls on the various execute-command modules to perform the function (Fig. 35). This command structure ensures that a command can be executed prior to relinquishing control (e.g., attempting to generate a confusion matrix prior to loading the data is not allowed). Error messages to this effect are output to the user.

EXECUTE LOAD DATA COMMAND uses the afferent group GET VALID FILE NUMBER to determine the file designator of the test results file (Fig. 36). It then accesses the data through the GET TEST DATA module.

EXECUTE ENTER COMMAND allows the user to specify a

predictor matrix (Fig. 37). This module calls on GET NEXT ROW which provides the row data for the matrix. MAKE MATRIX takes each row and forms the matrix. GET NEXT ELEMENT and MAKE ROW function in a like manner but on a smaller scale.

EXECUTE CONFUSION COMMAND uses the afferent group GET VALID CONFUSION OPTION to get the type of matrix to be generated--response, reaction time, or percentage (Fig. 38). The GET CONFUSION MATRIX module then gets the data required in order to generate the confusion matrix. This data is passed to GENERATE CONFUSION MATRIX.

EXECUTE COLLAPSE COMMAND handles the collapsing of the confusion matrix data (Fig. 39). GET VALID COLLAPSE OPTION is an afferent group which gets either a rank option or an absolute distance option. The PERFORM COLLAPSE FUNCTION then handles the collapsing of the confusion matrix based on the option.

EXECUTE PRINT COMMAND uses the afferent group GET VALID PRINT OPTION to determine what type of matrix the user desires output (Fig. 40). It calls on PRINT MATRIX to handle the actual formatting and outputting of the matrix.

OUTPUT ANALYSIS RESULTS formats and outputs the collapsed data from the execution of the COLLAPSE COMMAND (Fig.41). This module also has the option of storing the data for later use.

Appendix B

USER'S GUIDE

Introduction

The purpose of this appendix is to define the user-system interface. In this way both the designer/implementor and the user have a definite document to refer to when discussing the system. This appendix is not meant to be an endproduct but strictly an interface definition. In this way, the system becomes visible to the user.

Since the perception experiment software system is interactive, the commands listed in this appendix are those which are indicated as necessary to implement the system at this time. The appendix should be updated at any time that the user or designer feels that an update is warranted.

Syntax

The commands are divided into four main areas: system mode, set-up mode, execution mode, and analysis mode. A brief description of each mode will be given. This will be followed by the commands allowed during that mode. Examples are given for clarification. In most cases, it should be remembered that the system will be outputting messages to the user, providing him with the most options available to him. Also, for the frequent user, the commands and the parameters may be entered at the same time. To denote this, the portion of the input line which must be entered will be

underlined.

For example, if the user wants to store a particular symbol in the symbol library, he inputs a command which has the form

STORE SYMBOL, index

The underlined portion of the command indicates that the user must input that first part of the command--STORE SYMBOL. Should he stop there, the system will request that the user enter an index. However, the user has the option to enter the command with an index, that is, STORE SYMBOL, 99.

If there is more than one parameter for a particular command, the user can enter all, or some, of the parameters when he enters the command. The system will execute the command if all of the parameters are entered and if they are valid. If a parameter is found invalid, the system will output an error message and begin outputting prompts to get the rest of the parameters for the command.

System Mode

This is the basic mode of the system. When the program is run, the system enters this mode which controls the entry into the other three modes. There are four possible commands in this mode:

SET UP, setcom, parms

This enters the system to the set-up mode to begin executing set-up commands.

where

setcom is a legal set-up mode command.

parms are the parameters for the command.

EXECUTE, testno, fileno This enters the system to the execution mode to execute test and store the results in a file.

ANALYZE, fileno This enters the system to the analysis mode to begin analyzing the results in a file.

where

testno is the test index of the test to be executed.

fileno is the identification of the file containing the test results, i.e., responses, response times, etc.

EXAMPLES

SET UP This enters the system into the set-up mode.

SET UP, PURGE SYMBOL This enters the system to the set-up mode and commands that a symbol (to be specified) be deleted from the symbol library.

EXECUTE This enters the system into the execution mode.

EXECUTE, 3, TAPE7 This enters the system to the execution to begin executing test 3 and store the results in file TAPE7.

ANALYZE, TAPE7 This enters the system to the analysis mode to begin analyzing the data in file TAPE7.

Set-up Mode

This section describes the commands necessary to create and manipulate the data structures involved in setting up a perception test. The commands are divided into five categories: symbol, symbol library, test, test library, and bulk loading. The only command in this mode that does not fall into one of these categories is the following:

END SET UP

This returns control to the system mode.

Symbol

This section contains the commands necessary to create and manipulate symbols. The user creates a data base which will be called the active symbol (AS). The system keeps track of the development of the active symbol and displays it throughout its creation.

CREATE SYMBOL

This places the system into the "create mode". This must be input prior to creating a symbol.

CLEAR SYMBOL

This initializes the AS in order to begin creating a new symbol. The system will place a null entry into the AS and display a blank screen.

ADD elem, ...

This adds elements to the AS and displays them.

DELETE elem, ...

This deletes elements from the AS and removes them from the display.

END CREATE

This terminates the create mode.

where

elem is the element description.

The above commands are the basic commands used in symbol creation. While in the create mode, the system will continually display the contents of the AS. The description of the parameter "elem" depends on the implementation of the system. The elements could be described as ordered pairs, position numbers on a grid, or some other scheme for identifying them. In the following examples, it is assumed that the elements are described by position number.

EXAMPLES

```
CREATE SYMBOL
CLEAR SYMBOL
ADD 1
ADD 2, 3, 4
DELETE 2
4
END CREATE
```

This sequence of commands creates a symbol composed of two elements, located at positions 1 and 3. At the end of the fourth input line, the AS will contain four elements which will be displayed. After the sixth input line, the AS will contain only two elements.

During the create mode, the user has the option of viewing the AS as it is affected by display dynamics. This allows him to view the symbol as a subject might see it during an actual test. The commands which allow display dynamics during symbol creation are:

ALTER SYMBOL

This places the system into the dynamics mode. This command must be input prior to inputting a dynamics command.

ROTATE, ang

This rotates the AS display.

ADD, elemno

This adds elements randomly to the display.

DELETE, elemno

This deletes elements randomly from the display.

VIBRATE, freq, dist

This vibrates the display.

where

ang is the angle of rotation in degrees.

elemno is the number of elements that the user wishes to affect.

freq is the frequency of vibration.

dist is the distance in grid units.

The dynamics commands affect the display only; that is the commands do not affect the contents of the AS. Except for the ADD and DELETE commands, the commands are not mutually exclusive; that is the display can be vibrated while it is rotated. The display will be affected until the appropriate end-command is input.

END ROTATE

This returns the symbol display to its unrotated state.

END ADD

This removes the extra elements.

END DELETE

This replaces the elements that were removed.

END VIBRATE

This causes the vibration to stop.

END ALTER

This command returns control from the dynamics mode. This command must be input prior to resuming normal create mode operation.

EXAMPLES:

```
ALTER SYMBOL  
ROTATE, 30  
ADD  
5  
END ROTATE  
END ALTER
```

This sequence of commands will cause the display to be affected in the following way: first, the display will be rotated 30 degrees counter-clockwise; then 5 elements will be randomly lit; then the display (with the 5 additional elements) will be rotated back to its original state; finally, the 5 elements will be removed and the display will appear as it did before the ALTER command.

Symbol Library

This section describes the commands necessary to manipulate the symbol library. These commands allow the user to modify the data structure as well as interrogate it as to its contents.

STORE SYMBOL, indx, id

This enters a copy of the AS into the symbol library, with a unique index and a descriptive identifier.

PURGE SYMBOL, indx

This deletes the symbol identified by the index from the symbol library.

RELOAD SYMBOL, indx

This takes a copy of the referenced symbol from symbol library and places it into the active symbol.

where

indx is an integer representing an entry to the symbol library index table.

id is an alphanumeric character representing the description of the entry in the symbol library. This character will be the correct response when the symbol is displayed during test execution.

For those commands, the system will insure that the index is valid prior to executing the command. For the STORE SYMBOL command, the index will be checked to see that it has not been used by a symbol already in the symbol library. If the user attempts to enter the symbol with an index already in use, the system will produce an error diagnostic and require the user to enter the command again. For the PURGE SYMBOL and the RELOAD SYMBOL commands, if the index is not of a symbol in the library, an error diagnostic is output and the command is ignored. The purpose of the reload is to view a particular symbol in the library or to provide a starting base for a new symbol to be created. When the RELOAD SYMBOL command is given, the system clears the AS prior to copying the reloaded symbol.

EXAMPLES: (assuming that the symbol library contains the following index-identifiers: 1-A, 2-B, 3-C, 4-E)

STORE SYMBOL, 4, A

This causes the diagnostic of "duplicate indexes" to be output and the command must be re-entered.

STORE SYMBOL, 5, A

This enters the AS into the library with an index, 5, and an identifier, A. There is no problem with symbols sharing the same identifier.

RELOAD SYMBOL, 2
CREATE SYMBOL
DELETE 5, 7
ADD 4
6, 8
END CREATE
STORE SYMBOL, 6, E

This sequence demonstrates the use of the RELOAD command to provide a starting base for a new symbol. First, symbol 2 is placed into the AS after it has been cleared. Then after entering the create mode, the AS is modified. Finally

the AS is entered into the library as a new symbol with an index, 6, and an identifier, E.

In order to manipulate the symbol library, it is necessary to keep track of what is in the library. The system maintains an index table and an identifier table for this purpose. The user can interrogate these tables with the following command:

LIST SYMBOLS, opt

This causes the desired information to be output.

where

opt is the option desired. It can be either NUMBER or ALL. If it is NUMBER, the system will output the number of symbols currently stored in the library. If it is ALL, the system will output a list of the indexes and the corresponding identifiers of all the symbols currently in the symbol library.

EXAMPLES: (assume that the situation stands as it would be immediately after the last example.)

<u>Command</u>	<u>Output</u>
LIST SYMBOL, NUMBER	6
LIST SYMBOLS, ALL	1-A, 2-B, 3-C, 4-E, 5-A, 6-E

(note: the output shown here reflects the substance of what is output by the system. The format of the output depends on implementation.)

Test

This section contains the commands necessary to create a test. The user creates a data structure which will be

called the active test. The system keeps track of the development of the active test.

CLEAR PARAMETERS

This initializes the AT in order to begin creating a new set. The system will place default values into the AT. This should be the first command before specifying any items. However, like symbol creation, the user can begin specifying a test item using a starting base of a reloaded test.

TIME PARAMETERS, intv

This specifies the time intervals in the viewing sequence.

where

intv are integer values representing the time intervals in milliseconds. The interval options are:

MT₁=int₁

where int₁ is the first mask display time.

MD₁=int₂

where int₂ is the time between the first mask and the stimulus.

ST=int₃

where int₃ is the stimulus display time.

MD₂=int₄

where int₄ is the time between the stimulus and the second mask.

MD₂=int₅

where int₅ is the second mask display time.

PD=int₆

where int₆ is the prompting delay. This is the maximum time that the system should wait for the subject to

respond before displaying a prompting message.

STIMLIST, indxs, ENDLIST

This specifies the symbols desired for the test.

where

indxs are valid indexes of symbols in the symbol library.

ENDLIST denotes that the last symbol index has been entered.

DISPLAY OPTIONS, opt

This allows the user to specify execution time display options.

where

opt are parameters denoting various display options. The options are:

ASI=opt₁

where opt₁ is the index of the symbol used as the acknowledgement symbol in the viewing sequence. (type integer)

RMV=opt₂

where opt₂ is the number of consecutive times that a symbol must be correctly identified before it is removed from the stimulus list. (integer)

RDP=opt₃

where opt₃ determines the immediate redisplay of an incorrectly identified symbol. (YES or NO)

RIF=opt₄

where opt₄ determines the use of reinforcement feedback after each response. (YES or NO)

SLO=opt₅

where opt₅ determines the randomness of the stimulus list order. (YES or NO) If YES, the stimulus list will be displayed in a random

order. If NO, the list will be displayed in the order specified in the command STIMULUS LIST.

PNT=opt₆

where opt₆ determines if the print option is used. (YES or NO)

MASKS, mkopt

This allows the user to specify the type of mask used during the viewing sequence.

where

mkopt determines the mask factors. These factors are:

TYP=opt₁

where opt₁ is the mask type. (STATIC or DYNAMIC)

PR1=opt₂

where opt₂ is one of the following: if TYP=STATIC, the number of masks to be generated; if TYP=DYNAMIC, the interval, in milliseconds, between lighting elements. (integer)

PR2=opt₃

where opt₃ is one of the following: if TYP=STATIC, the maximum number of elements lit in each mask; if TYP=DYNAMIC, the duration, in milliseconds, of the lit elements. (integer)

ROTATE SYMBOL, rdm, ang

This specifies the rotation of the symbols as they are displayed during test execution.

DEGRADE, option, maxno

The specifies the modification of the symbol display by the addition/deletion of extraneous elements.

VIBRATE SYMBOL, freq, dist

This specifies the vibration of each symbol as it is displayed during test

execution.

DISPLACE SYMBOL, rng

This specifies the maximum displacement of the symbols from the center of the viewing area during test execution.

where

rdm is a randomness option. (YES or NO)

ang is one of the following: if rdm is YES; the maximum angle of rotation in degrees; if rdm is NO, the constant angle of rotation for each symbol. (integer)

option denotes the addition/deletion of elements. (ADD or DELETE)

maxno is the maximum number of elements that can be affected. (integer)

freq is the frequency of the vibration in cycles per second. (integer)

dist is the distance of the vibration in grid units. (integer)

rng is the maximum range that the symbol can be displaced from the center of the viewing area in grid units. (integer)

PRINT PARAMETERS, option

This allows the user to determine the contents of the AT.

where

option is the item desired: MASK, TIME, STIMLIST, DYNAMICS, OPTIONS, and ALL. More than one item can be selected, i.e.,
PRINT PARAMETERS, MASK, OPTIONS

EXAMPLES:

CLEAR PARAMETERS

This sets all the items in the AT to their default values.

TIME PARAMETERS, ST=3000,
PD=6000

This sets the stimulus display time to 3 seconds and the prompting delay to 6 seconds. If this were the

TIME PARAMETERS, MT1=3000,
ST=4500

only TIME command, the other values would remain at their default values.

This sets the first mask display time to 3 seconds and changes the stimulus display time to 4.5 seconds.

STIMLIST, 1
2, 4, 3
6, 5, 1
ENDLIST

This sequence will create a stimulus list of seven symbols, with symbol 1 appearing twice.

DISPLAY OPTIONS
ASI=9
SLO=YES
RMV=3, RDP=YES

This sequence sets the acknowledgement symbol to symbol 9; the stimulus list order will be random; a symbol will be removed from the list if it is correctly identified 3 consecutive times; and if a symbol is misidentified, it will be redisplayed immediately.

MASKS, TYP=STATIC
PR1=14

This sequence will generate 14 static masks.

ROTATE SYMBOL, NO, 45

This specifies that each symbol will be displayed rotated at the same angle, 45 degrees counter-clockwise.

PRINT PARAMETERS, ALL

This will cause a display of all the items in the active test. Default values will be flagged.

PRINT PARAMETERS, TIME,
DYNAMICS

This will cause a display of the time parameters and the dynamics parameters currently in the AT. Default values will be flagged.

Test Library

This section describes the commands necessary to manipulate the test library. These commands allow the user to modify this data structure as well as examine its contents.

<u>STORE PARAMETERS</u> , indx	This enters a copy of the AT into the test library with a unique index.
<u>PURGE PARAMETERS</u> , indx	This deletes the test referenced from the test library.
<u>RELOAD PARAMETERS</u> , indx	This copies the referenced test from the library into the AT.

where

indx is an integer representing an entry in the test library.

For these commands, the system will insure that the index is valid prior to executing the command. For the STORE PARAMETERS command, the index will be checked to see that it has not been used by a test in the library already. If the user attempts to enter the test with an index already in use, the system will produce an error diagnostic and require the user to enter the command again. For the PURGE PARAMETERS and RELOAD PARAMETERS commands, if the index is not in use, an error diagnostic is outputted and the command is ignored. The purpose of the reload is to view a particular test in the library or to provide a starting base for a new test to be created. When the RELOAD PARAMETERS command is given, the system clears the AT prior to copying the reloaded test.

In order to manipulate the test library, it is necessary to keep track of what is in the library. The system maintains an index table for this purpose. The user can examine this table with the following command:

LIST PARAMETERS, opt This causes the desired information to be output.

where

opt is the option desired. It can be either NUMBER or ALL. If it is NUMBER, the system will output the number of tests currently stored in the library. If it is ALL, the system will output a list of the indexes of all the tests currently in the test library.

Bulk Loading and Storing

This section contains the commands necessary for the use of secondary storage devices in creating the symbol and test libraries. This provides the user with the capability of creating the libraries and storing them on a secondary storage device. Then the libraries could be read from the device at a later time.

BULK STORE, typ, fileno This stores a library in a file on a secondary storage device.

BULK LOAD, typ, fileno This loads a library from a file on a secondary storage device.

where

typ is the type of library being transferred.
(SYMBOL or TEST)

fileno is the file number on the storage device.

EXAMPLES:

BULK LOAD, SYMBOL, FILE5	This sequence loads the two libraries. Then a check is made to determine which sets are in the test library.
BULK LOAD, PARAMETER, FILE6	
LIST PARAMETERS, ALL	

Execution Mode

This section describes the commands required to execute the perception test. There are few commands in this section because this mode is primarily concerned with the execution of tests. The commands for this mode follow.

STOP TEST

This terminates the test just given. The command is input prior to the viewing of the first mask of a viewing sequence (that is, during the acknowledgement symbol). The system then outputs a request for ID information in preparation for another test.

END EXECUTION MODE

This returns the system to the system mode.

Analysis Mode

This section describes the commands required to analyze test results. The following is a summary of the commands used during the analysis mode.

LOAD DATA, fileno

This allows the user to specify the results to be analyzed.

CONFUSION, opt

This allows the user to specify the type(s) of confusion matrix to be generated. Opt can be RESPONSE, REACTION TIME, or PERCENTAGE which will cause the corresponding confusion matrix to be generated.

PRINT, opt

This allows the user to specify which confusion will be output. Opt has the same values as in the CONFUSION command.

COLLAPSE, op

This allows the user to specify a predictor matrix and perform a collapsing routine based on a given rank number or a given absolute distance range, depending on the value of "op".

ENTER, matval

This is used to enter the matrix values of the predictor matrix.

END ANALYSIS

This returns control to the system mode.

where

fileno is the file number of the results to be analyzed.

opt is the option for the type of confusion matrix which the user desires to be generated or printed.

op specifies the desired collapsing routine. The values of op are the following:

RANK, num where num specifies the rank number on which to collapse the confusion matrix.

ABSDIS, rng where rng specifies the

absolute distance range
on which to collapse the
confusion matrix.

matval are the matrix values of the predictor ma-
trix.

Appendix C

STRUCTURE CHARTS

This appendix contains a list of all of the structure charts developed in the preliminary design.

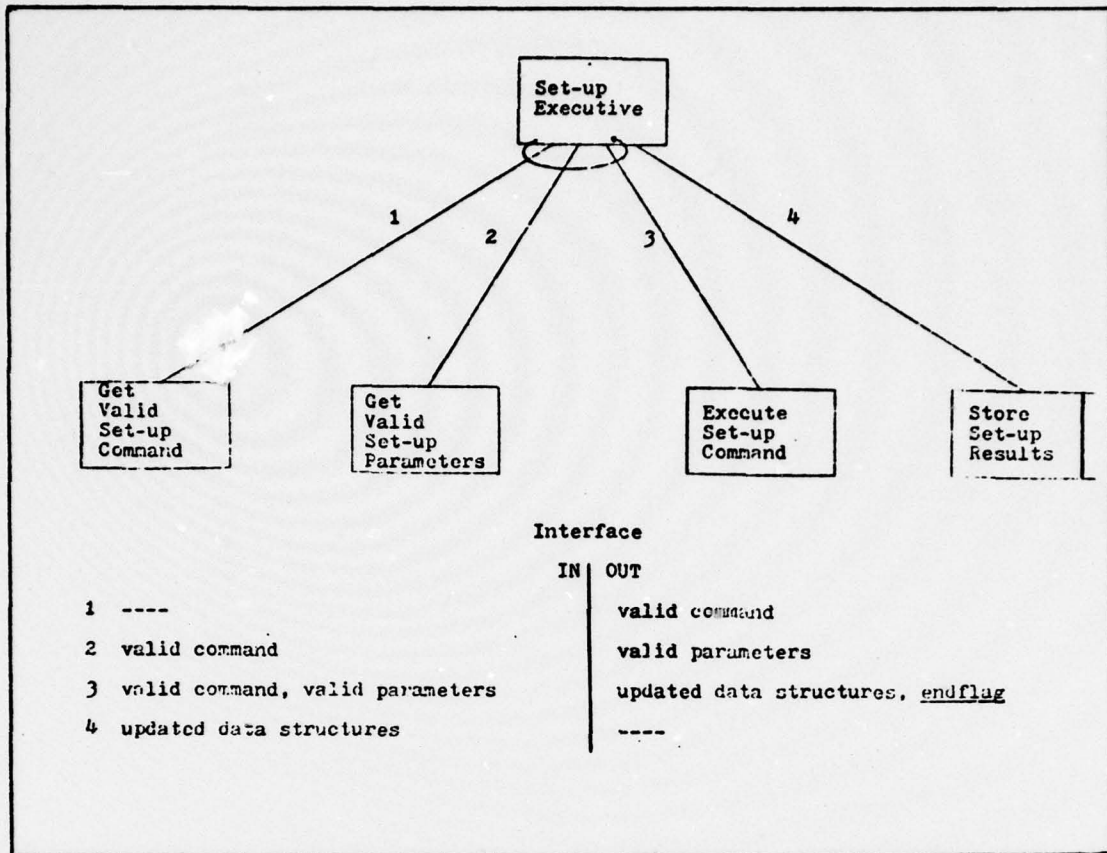


Figure 6. Set-up System (First - Level Factoring)

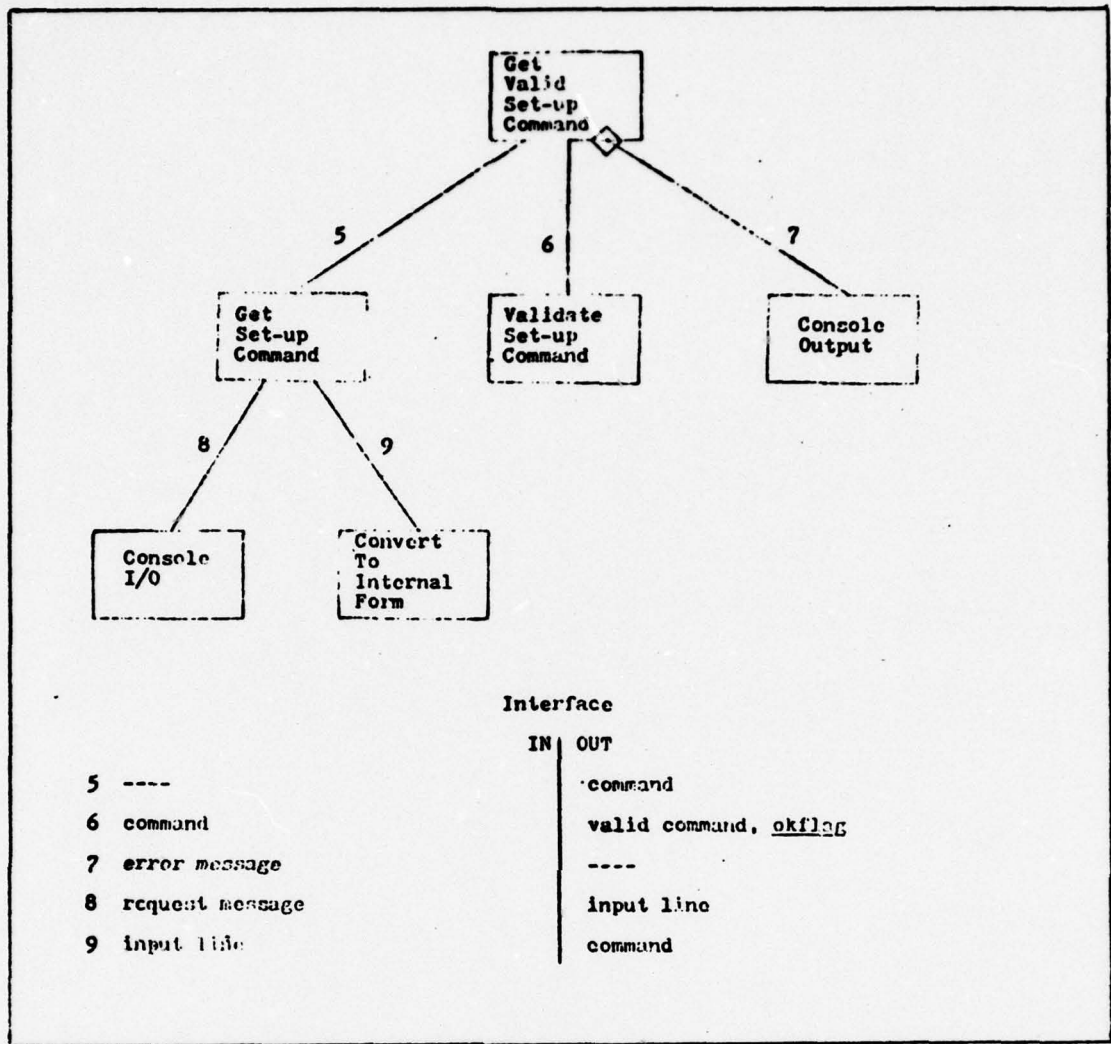


Figure 7. Get Valid Set-up Command

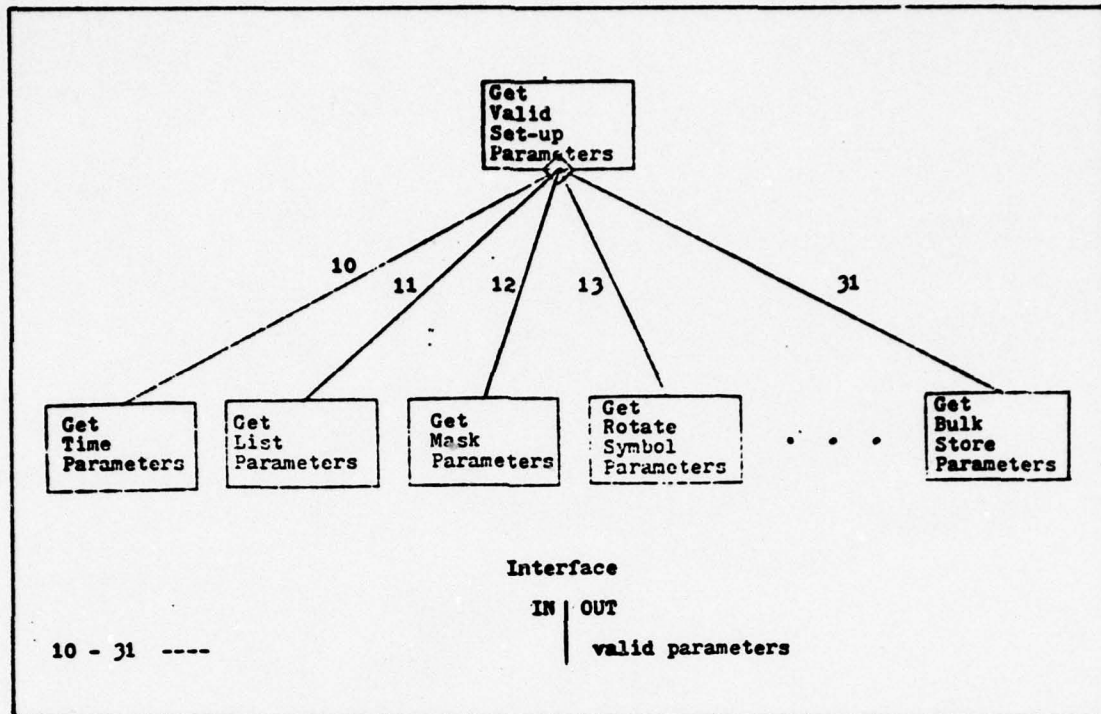


Figure 8. Get Valid Set-up Parameters (Transaction Center)

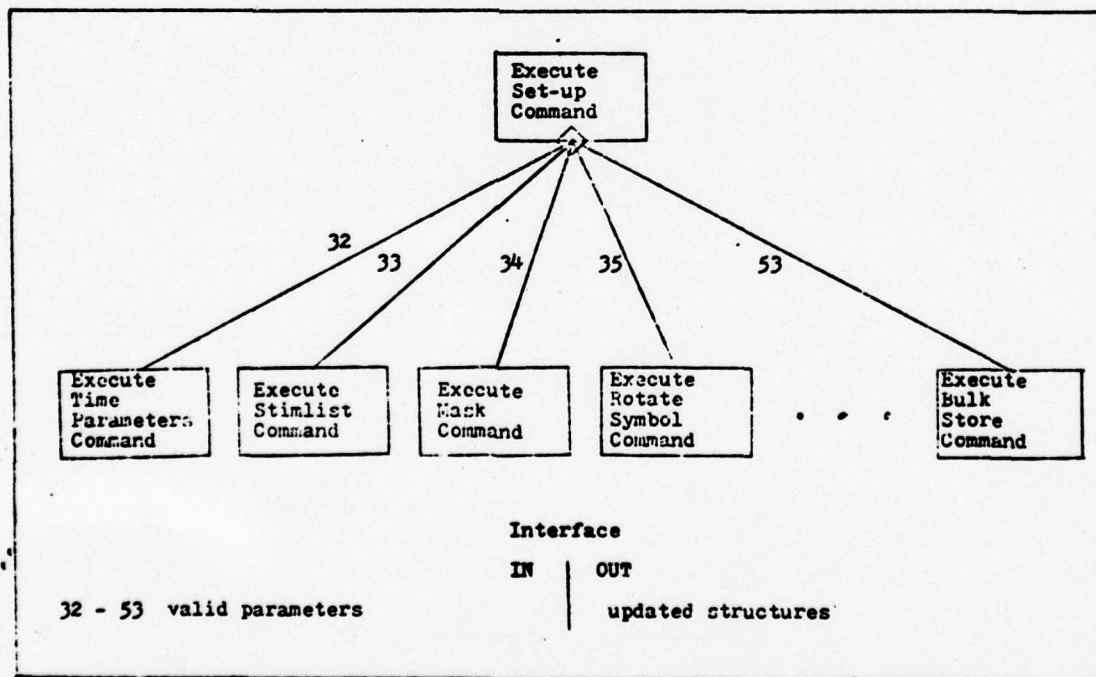


Figure 12. Execute Set-up Command (Transaction Center)

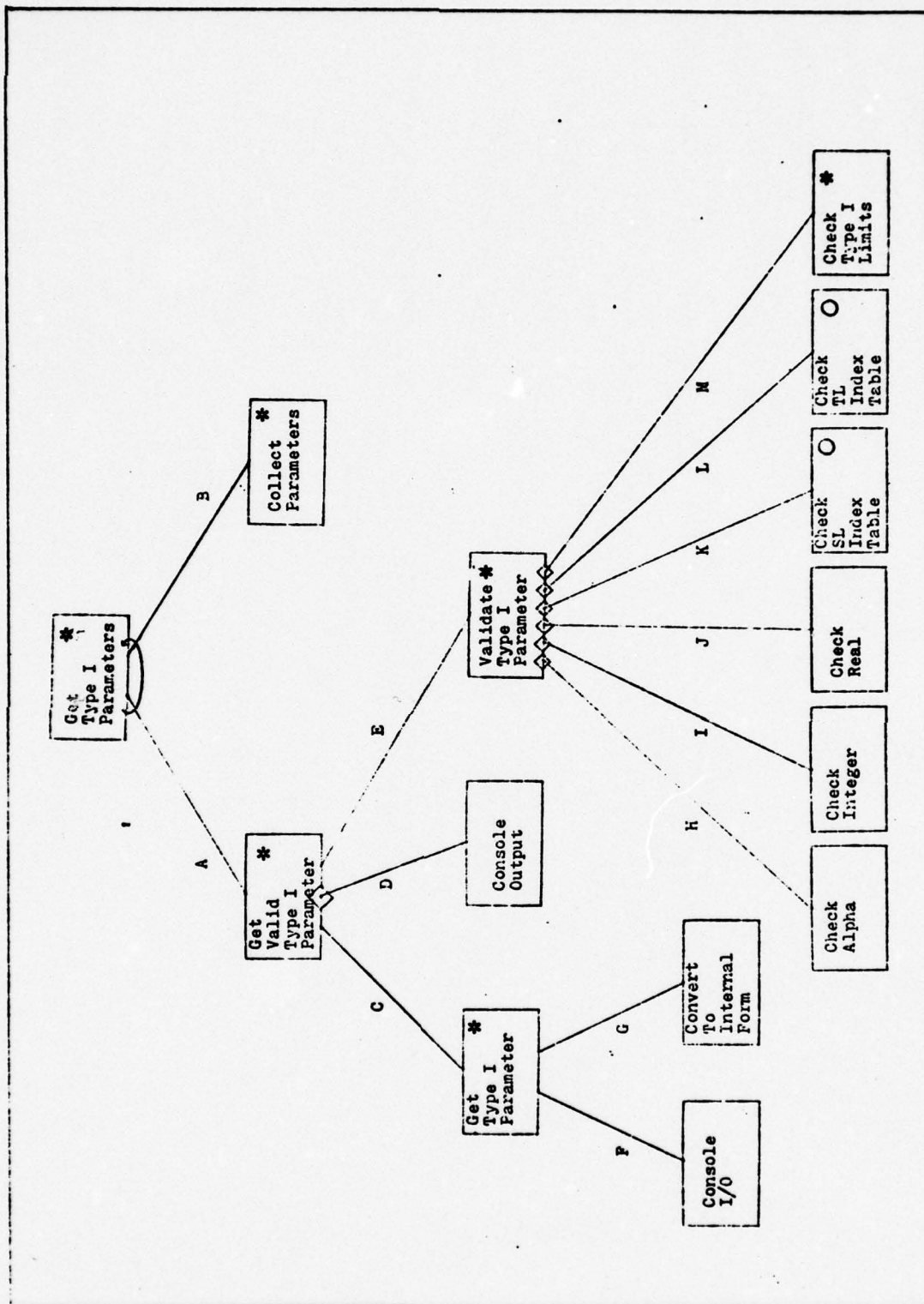


Figure 9. Get Type I Parameters (General Form)

Interface

IN OUT

A	parameter number	valid parameter
B	valid parameter	valid parameters
C	parameter number	parameter
D	error message	----
E	parameter number, parameter	valid parameter, <u>okflag</u>
F	request message	input line
G	input line	parameter
H	parameter	<u>okflag</u>
I	parameter	<u>okflag</u>
J	parameter	<u>okflag</u>
K	parameter	<u>okflag</u>
L	parameter	<u>okflag</u>
M	parameter, parameter number	<u>okflag</u>

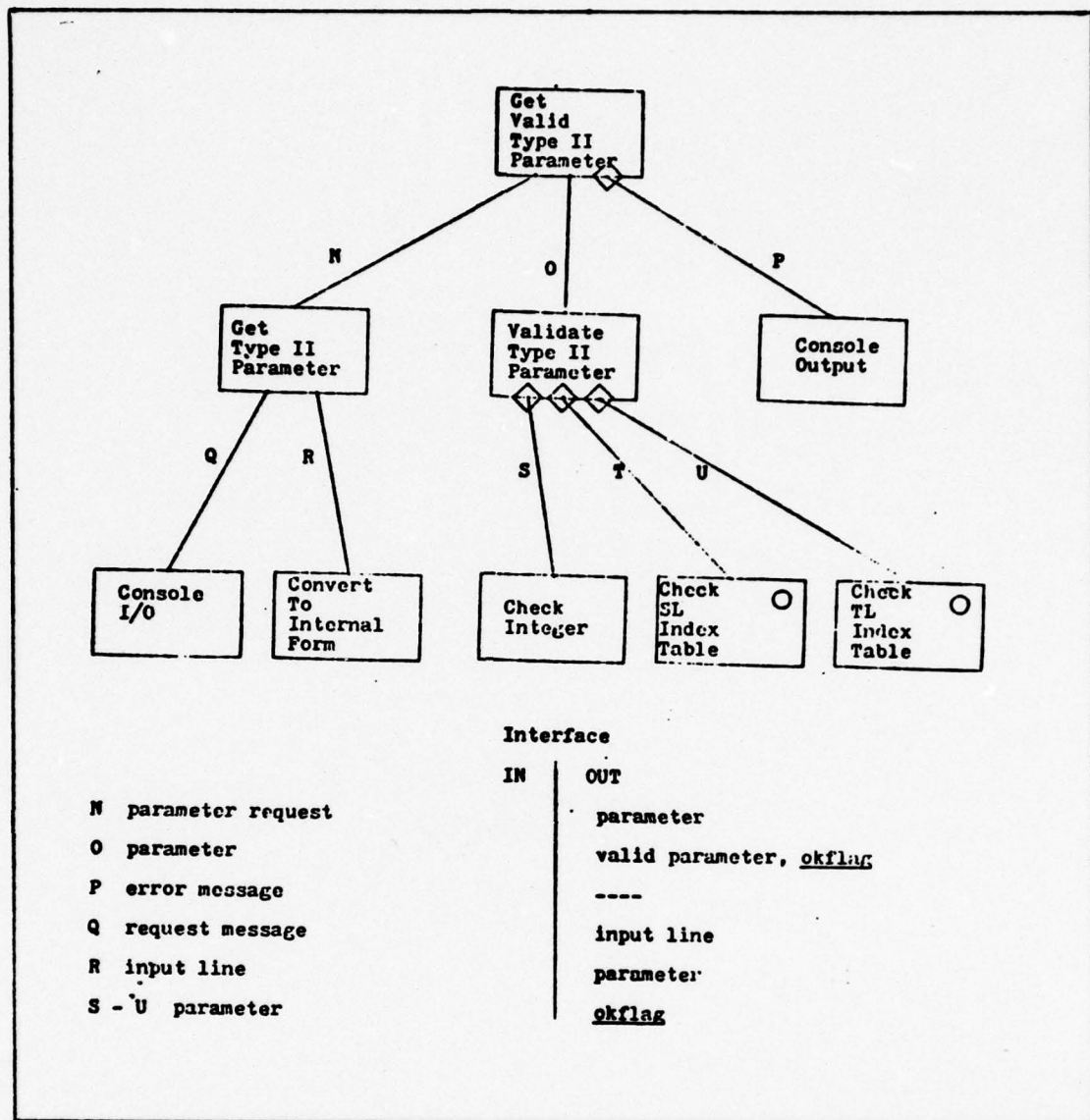


Figure 10. Get Type II Parameter (General Form)

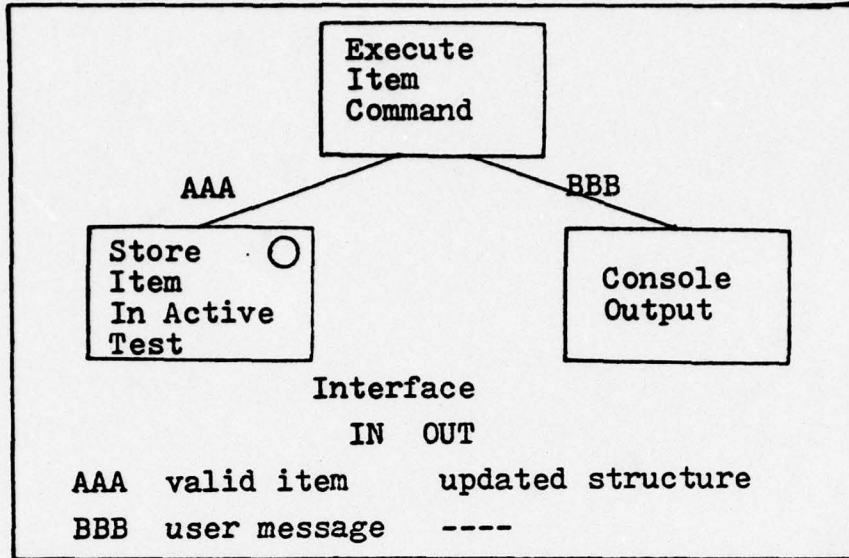


Figure 13. Execute Item Command (General Form)

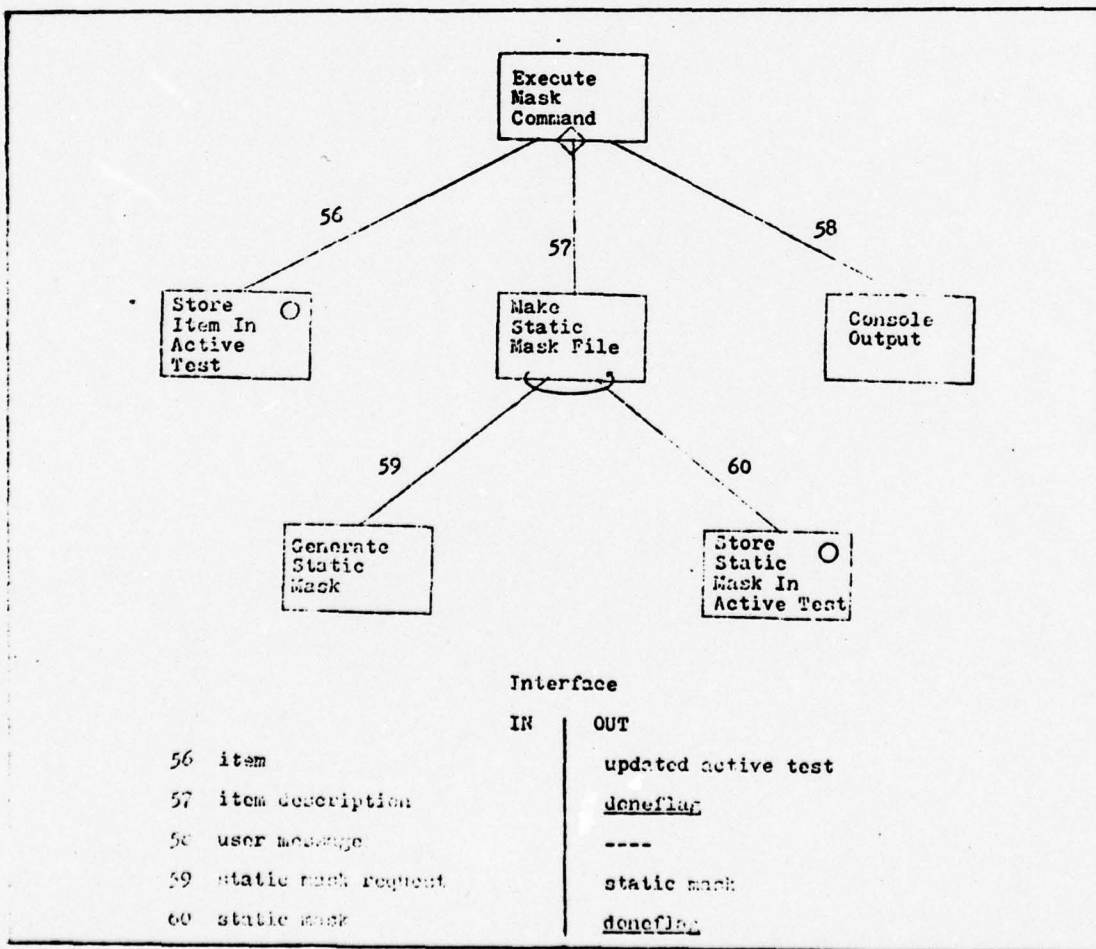


Figure 14. Execute Mask Command

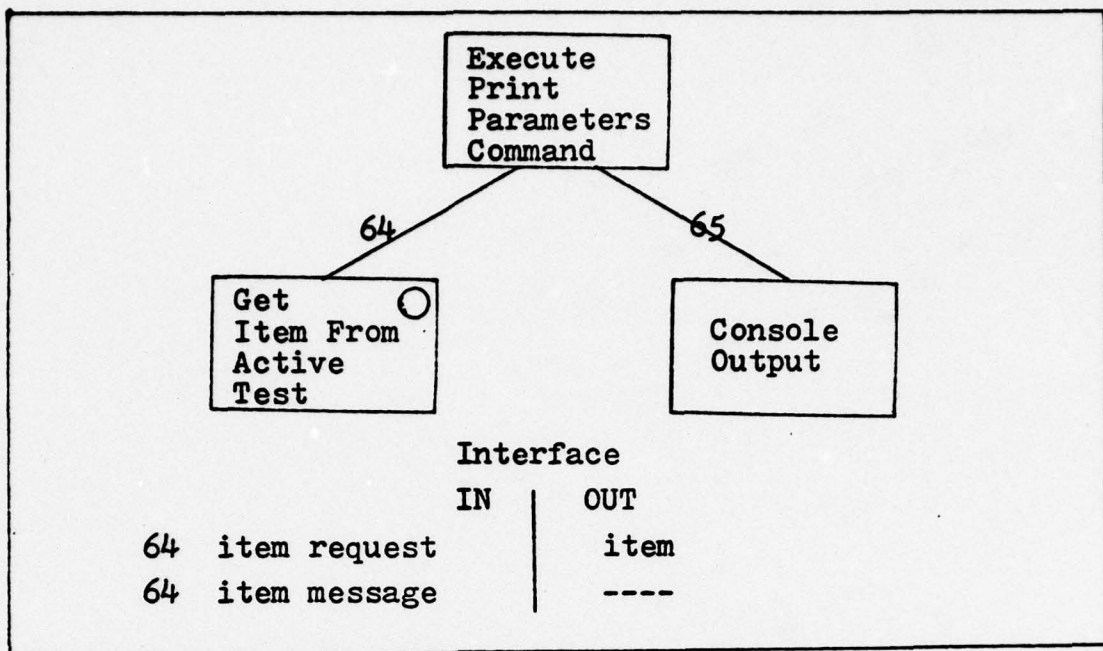


Figure 16. Execute Print Parameters Command

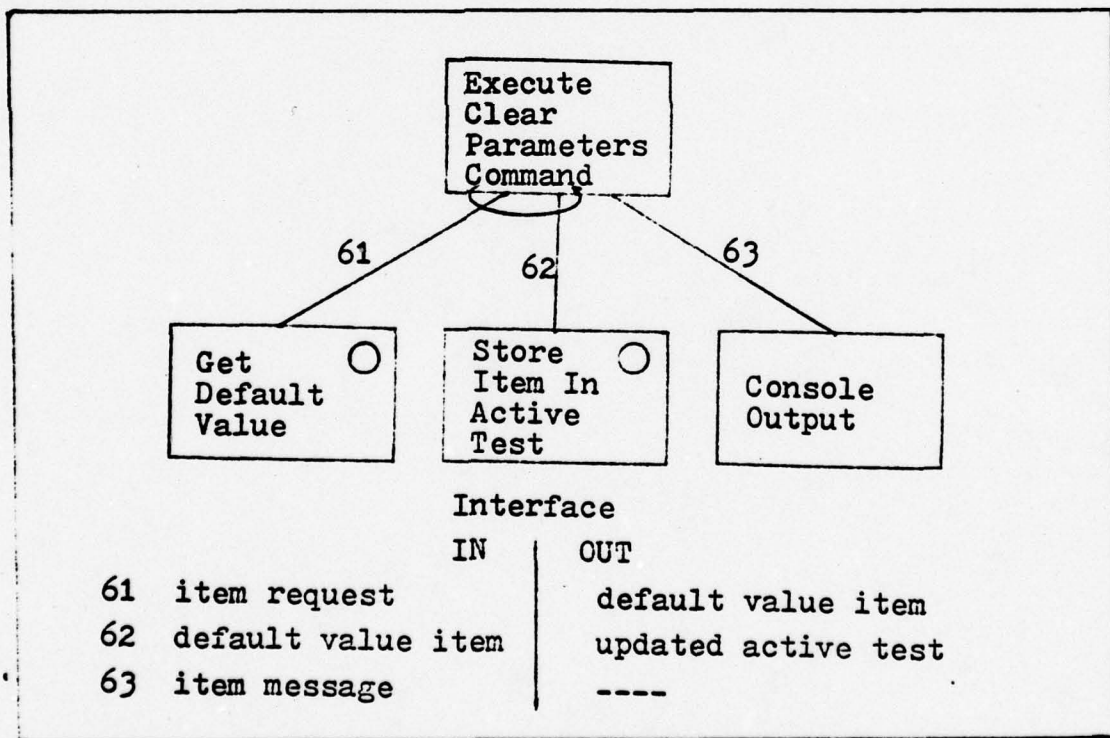


Figure 15. Execute Clear Parameters Command

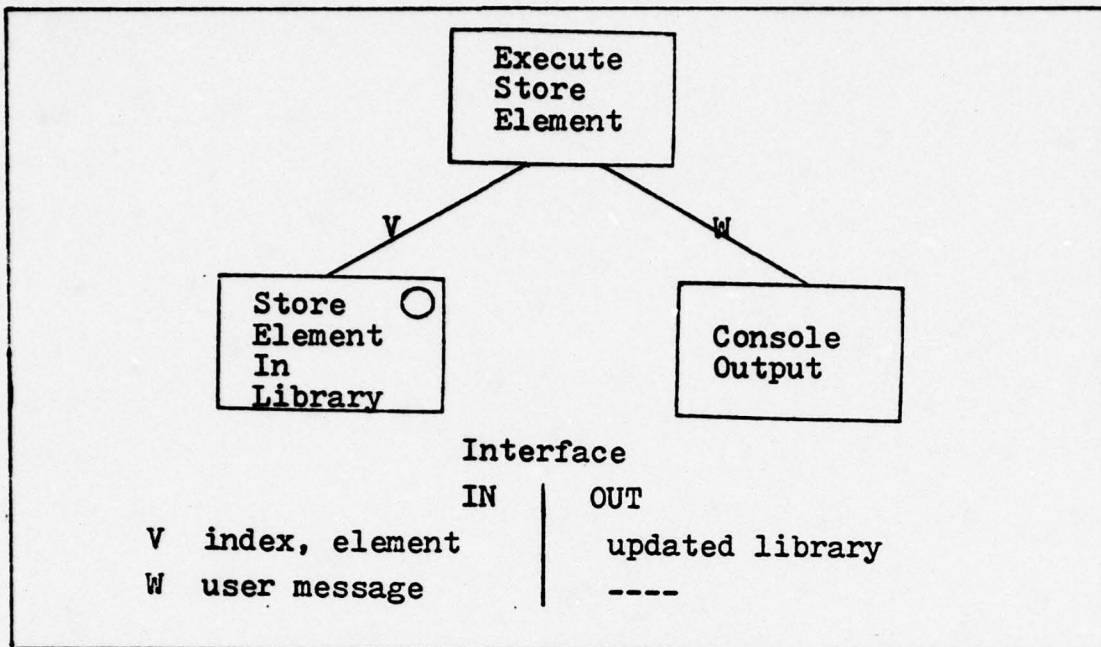


Figure 17. Execute Store Element (General Form)

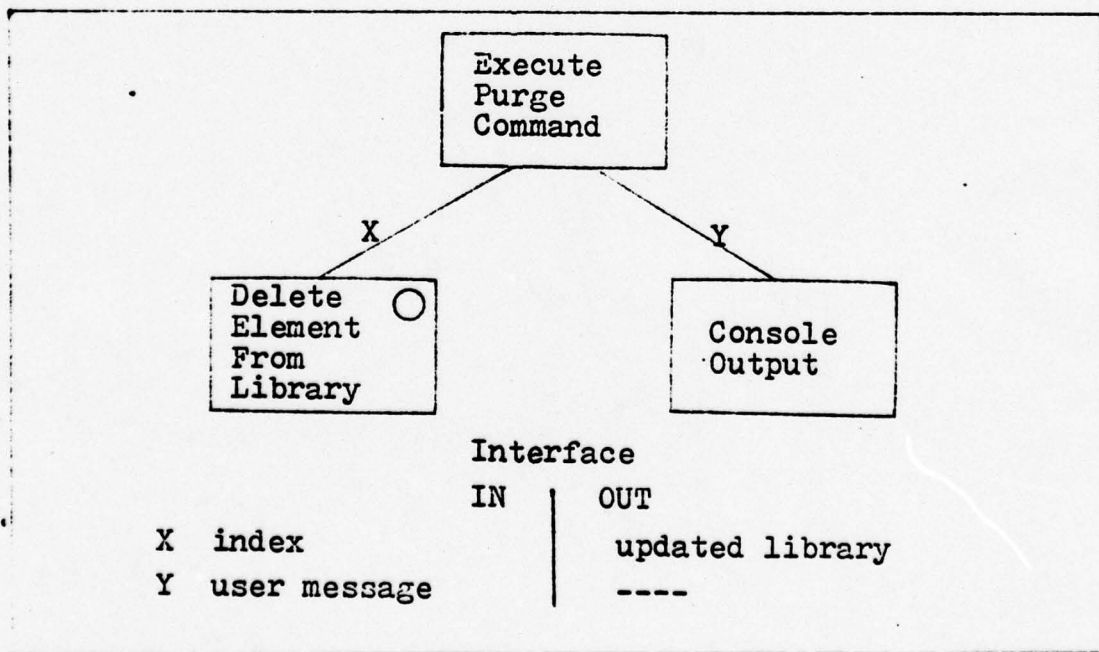


Figure 18. Execute Purge Command (General Form)

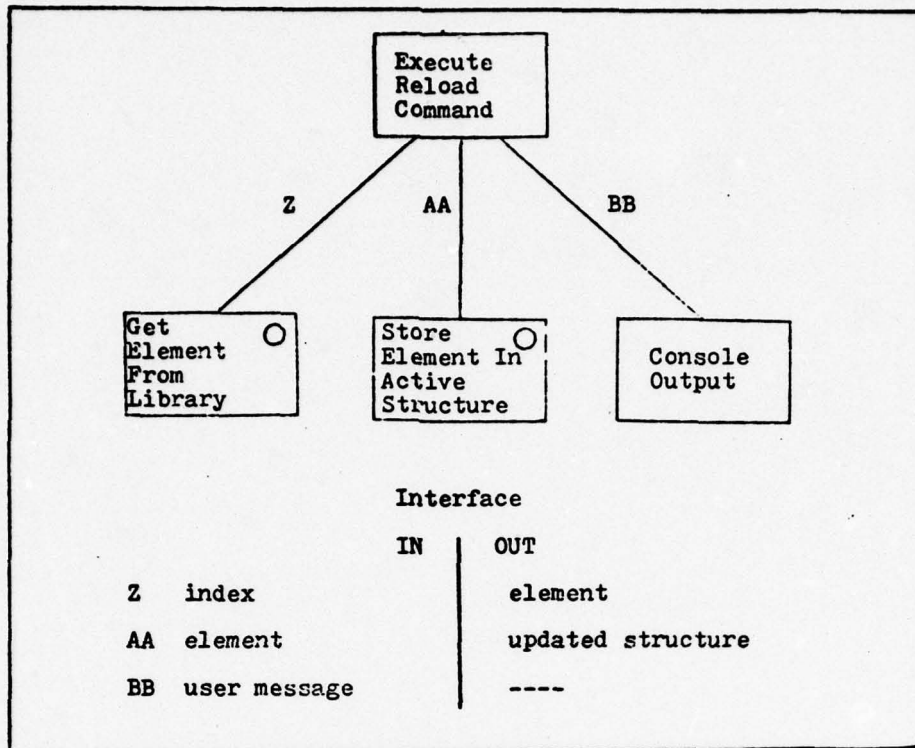


Figure 19. Execute Reload Command (General Form)

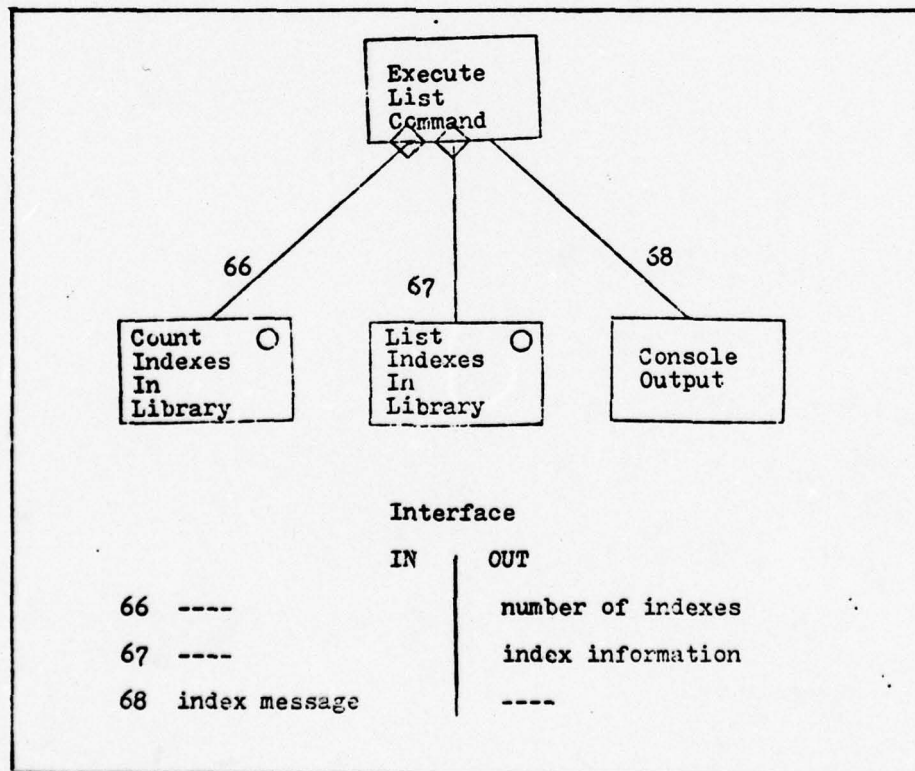


Figure 20. Execute List Command (General Form)

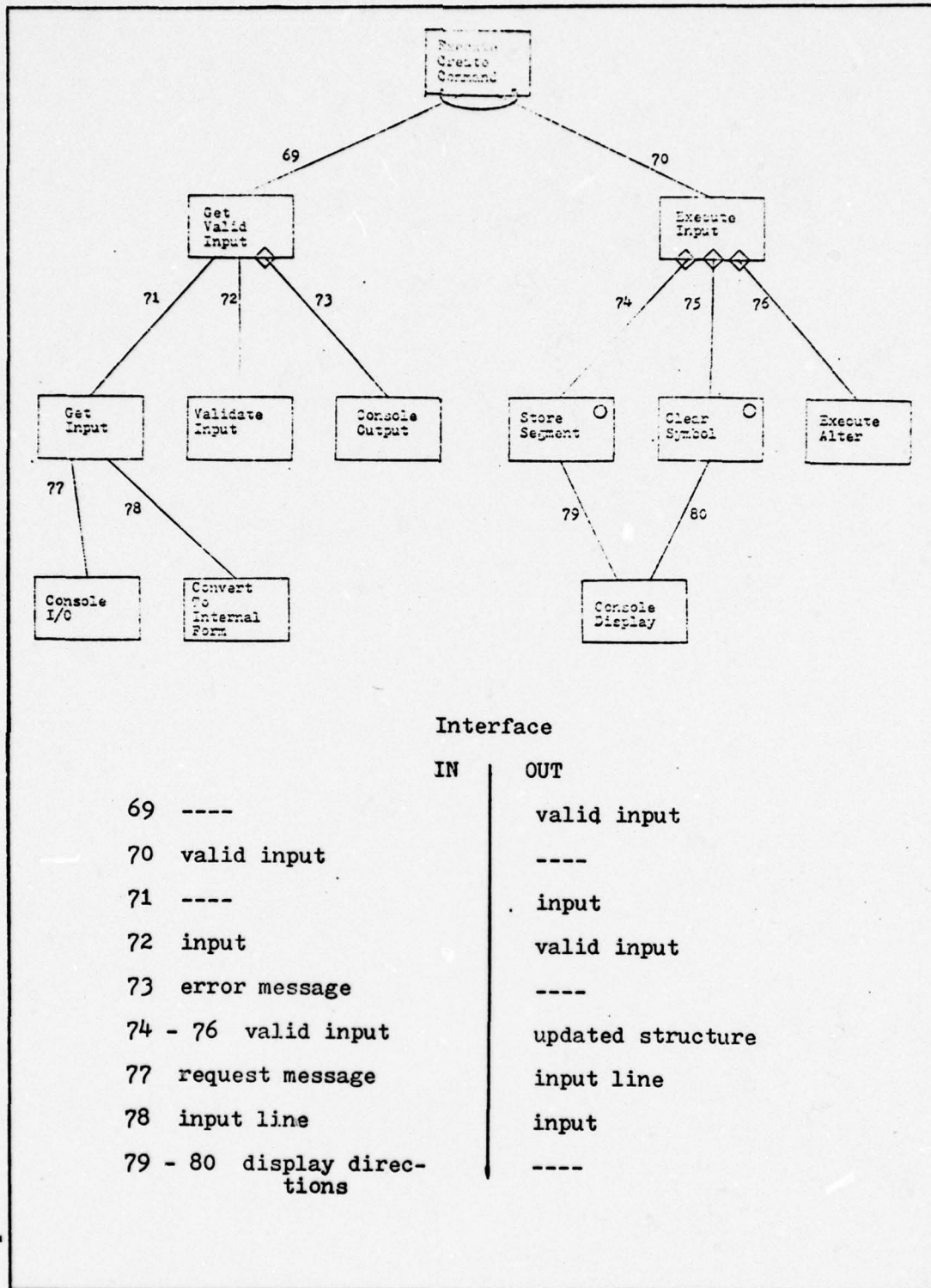


Figure 21. Execute Create Command

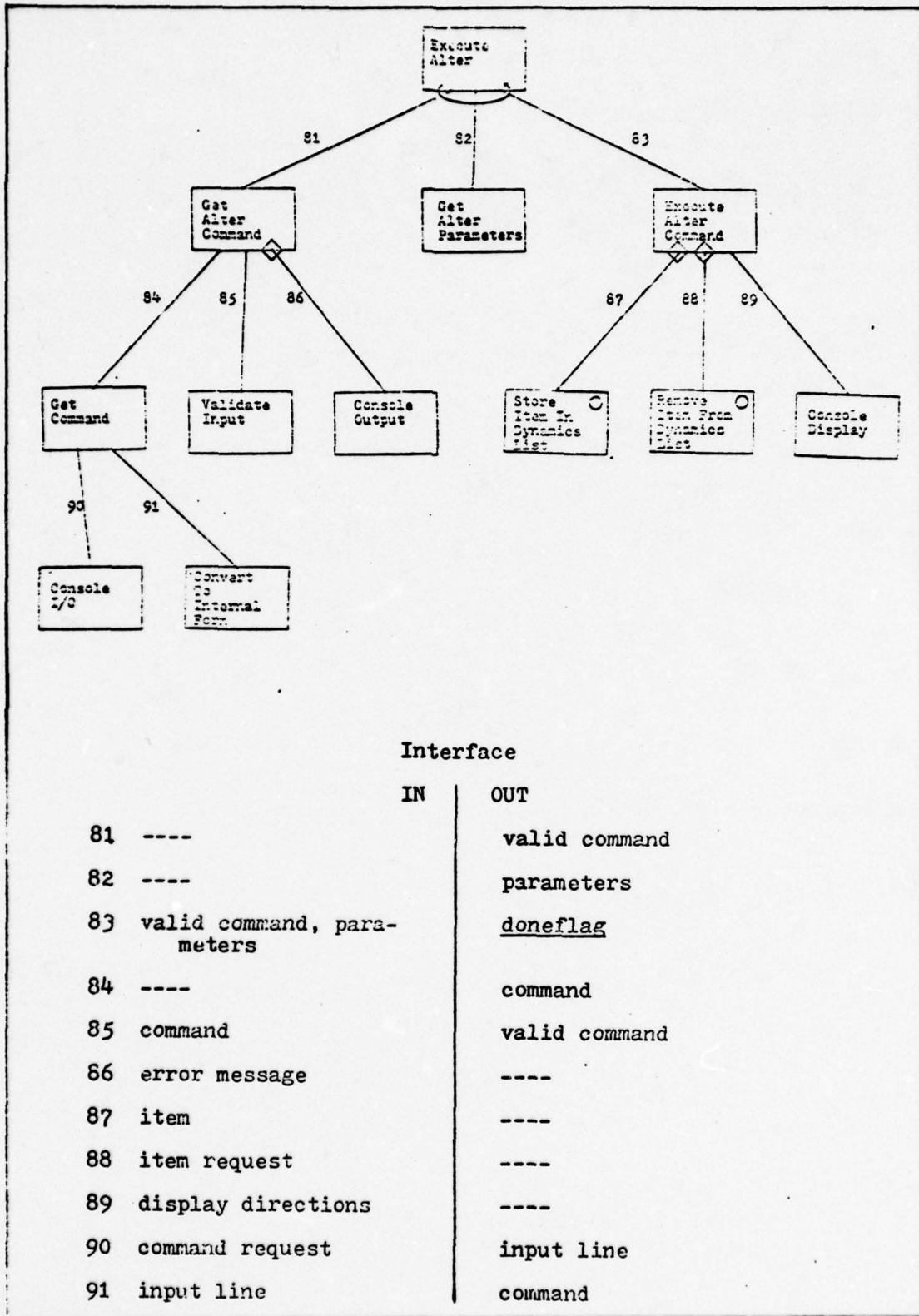


Figure 22. Execute Alter Command

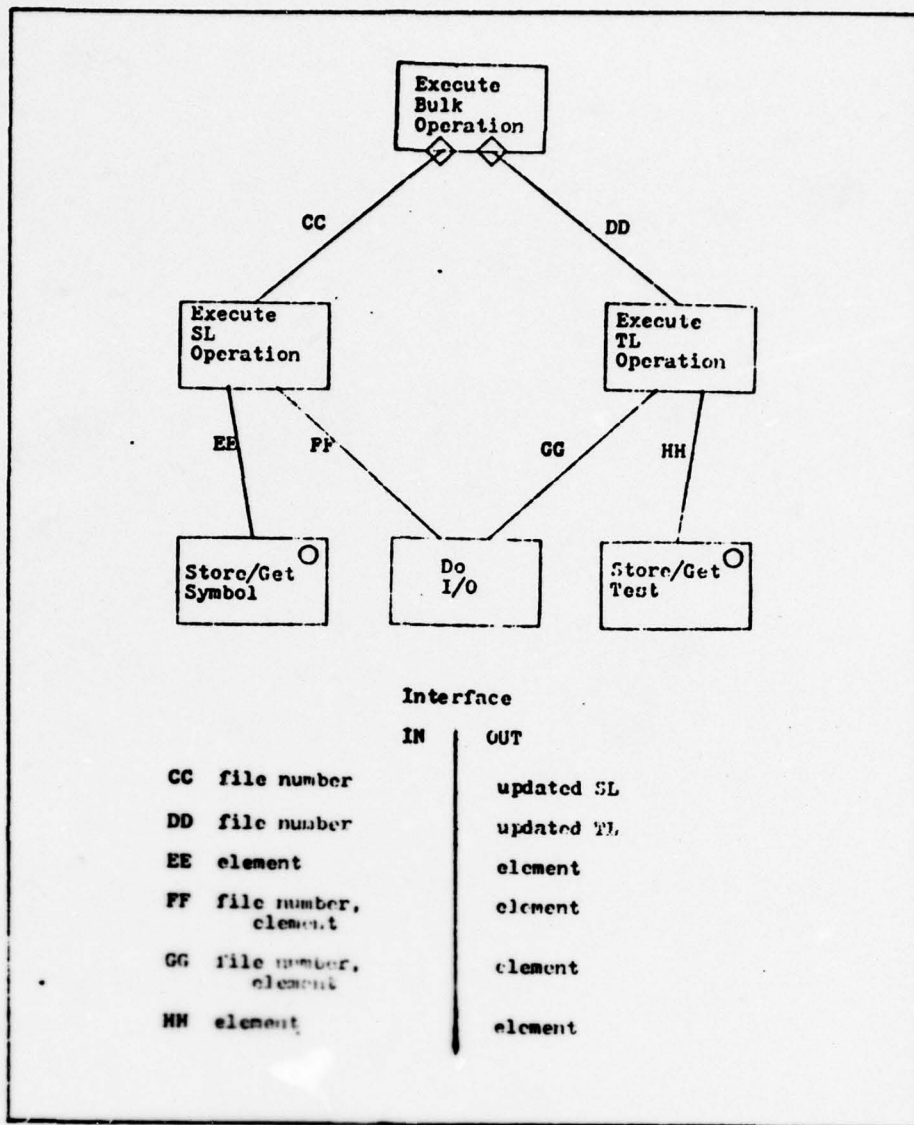


Figure 23. Execute Bulk Operation (General Form)

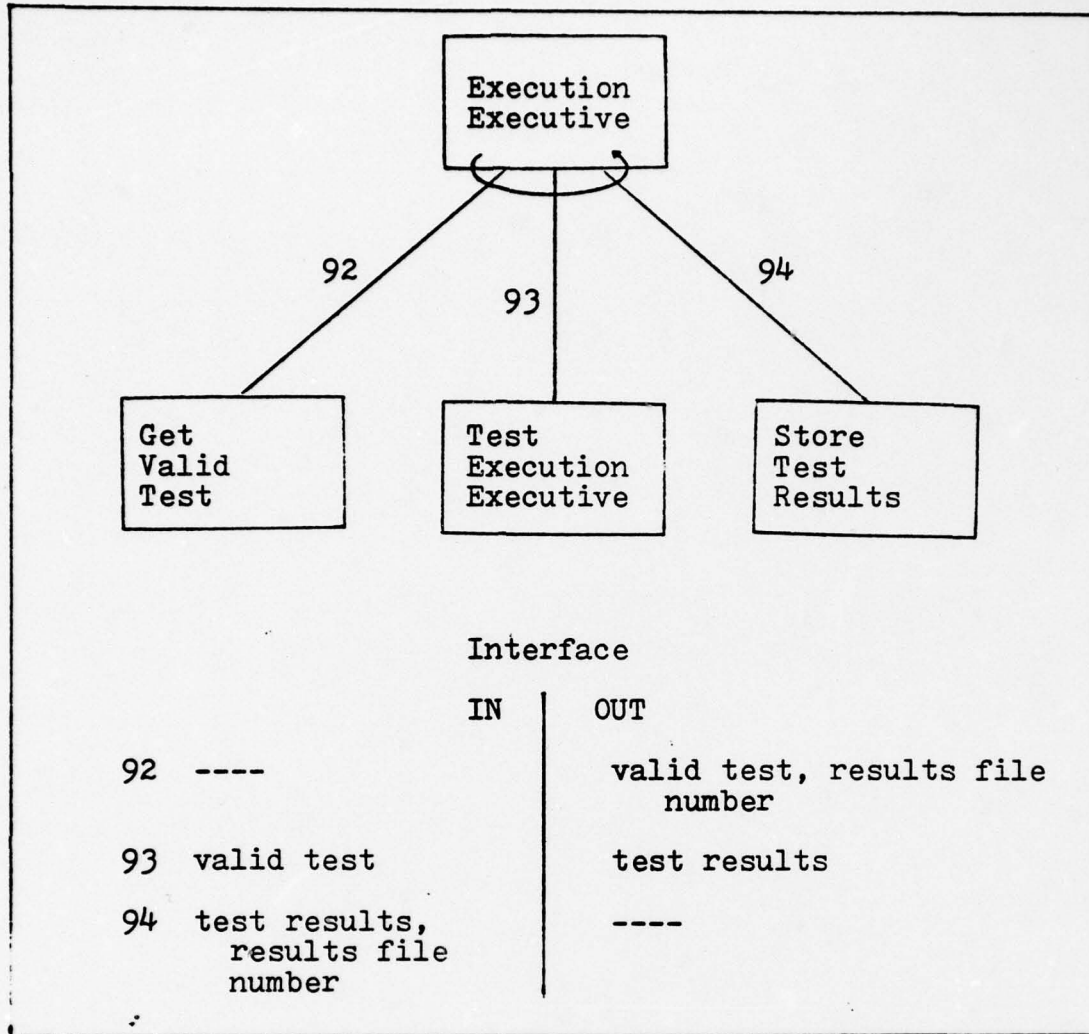


Figure 26. Execution System (First - Level Factoring)

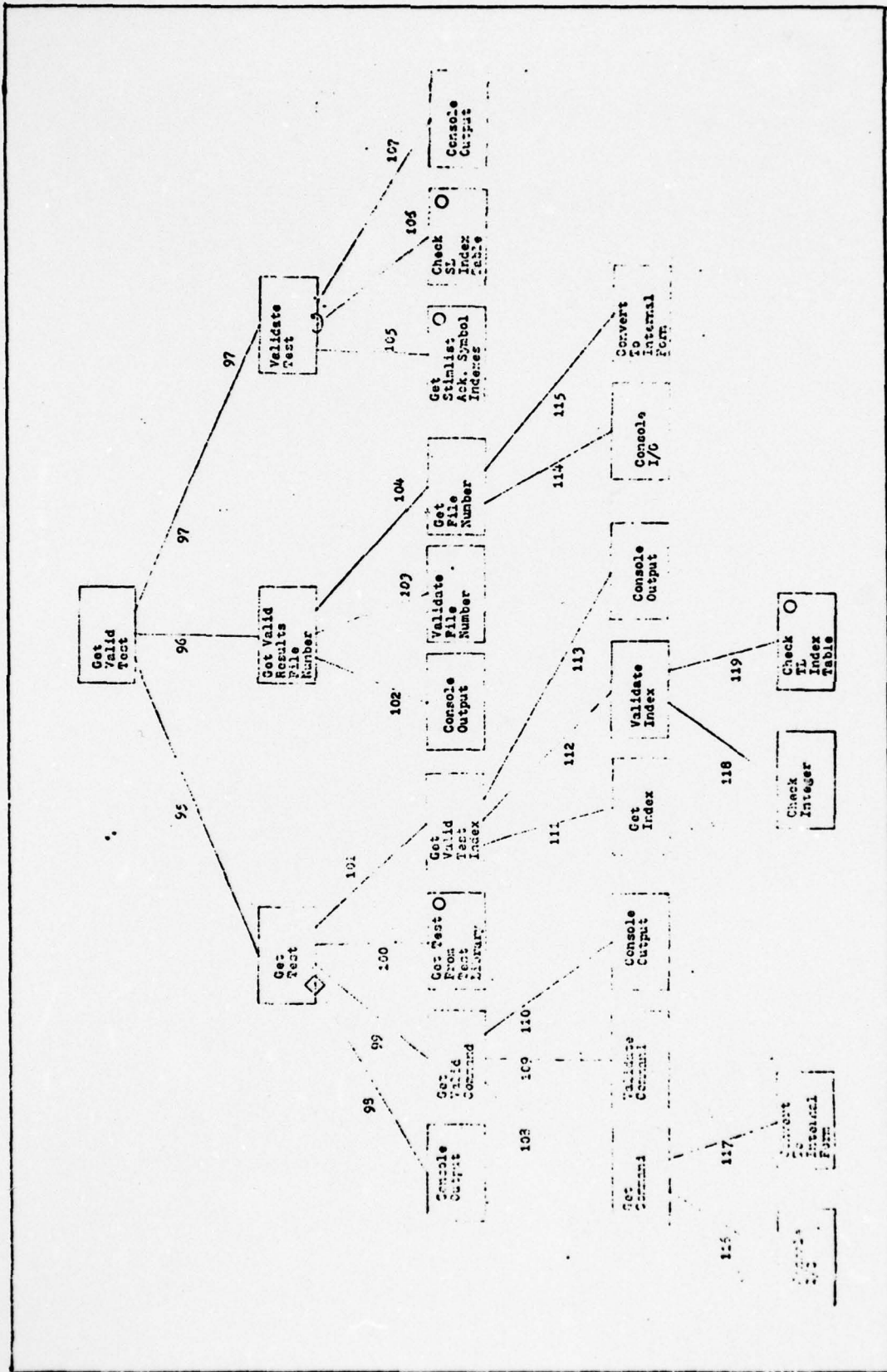


Figure 27. Get Valid Test

Interface

	IN	OUT
95	----	test, <u>endflag</u>
96	----	valid results file number
97	test	valid test, <u>okflag</u>
98	error message	----
99	----	valid command, <u>endflag</u>
100	index	test
101	----	valid index, <u>okflag</u>
102	error message	----
103	file number	valid file number
104	----	file number
105	test	indexes
106	indexes	<u>okflag</u>
107	error message	----
108	----	command
109	command	valid command, <u>okflag</u>
110	error message	----
111	----	index
112	index	valid index, <u>okflag</u>
113	error message	----
114	file number request	input line
115	input line	file number
116	command request	input line
117	input line	command
118	index	<u>okflag</u>
119	index	<u>okflag</u>

Figure 27 continued

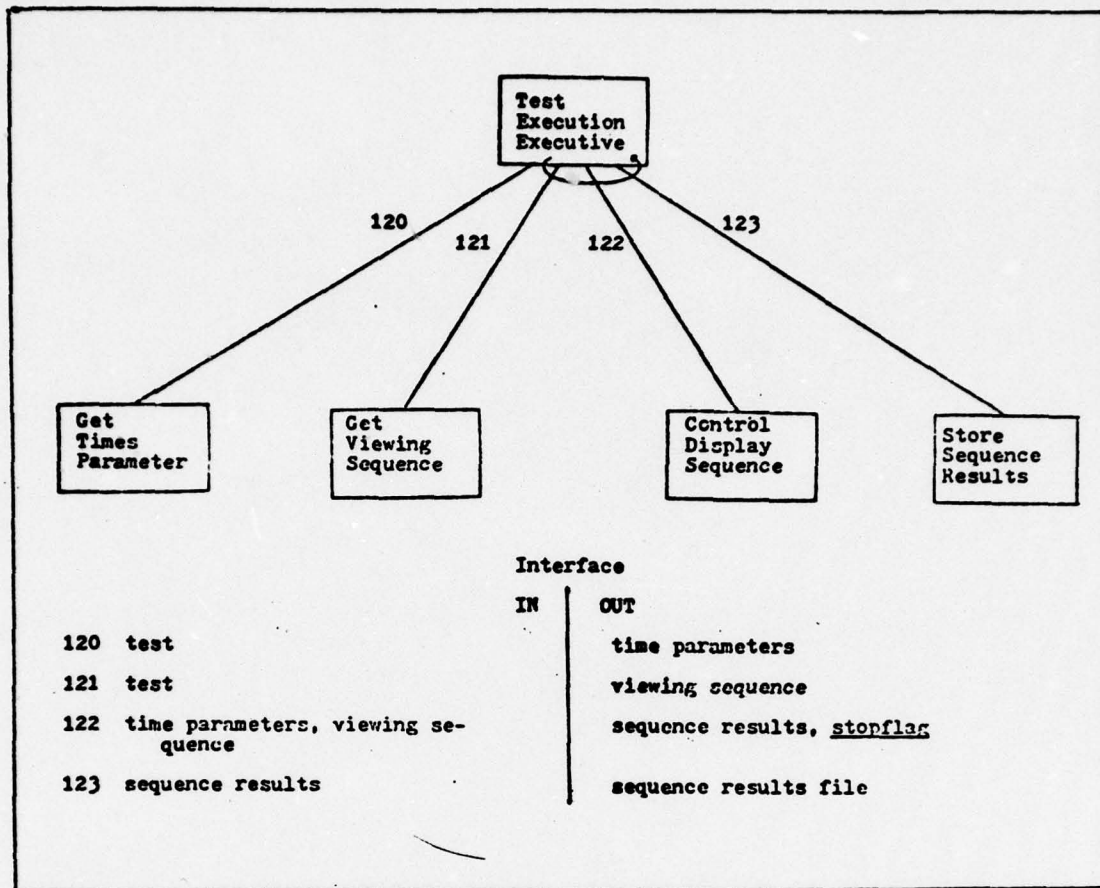


Figure 28. Test Execution (First - Level Factoring)

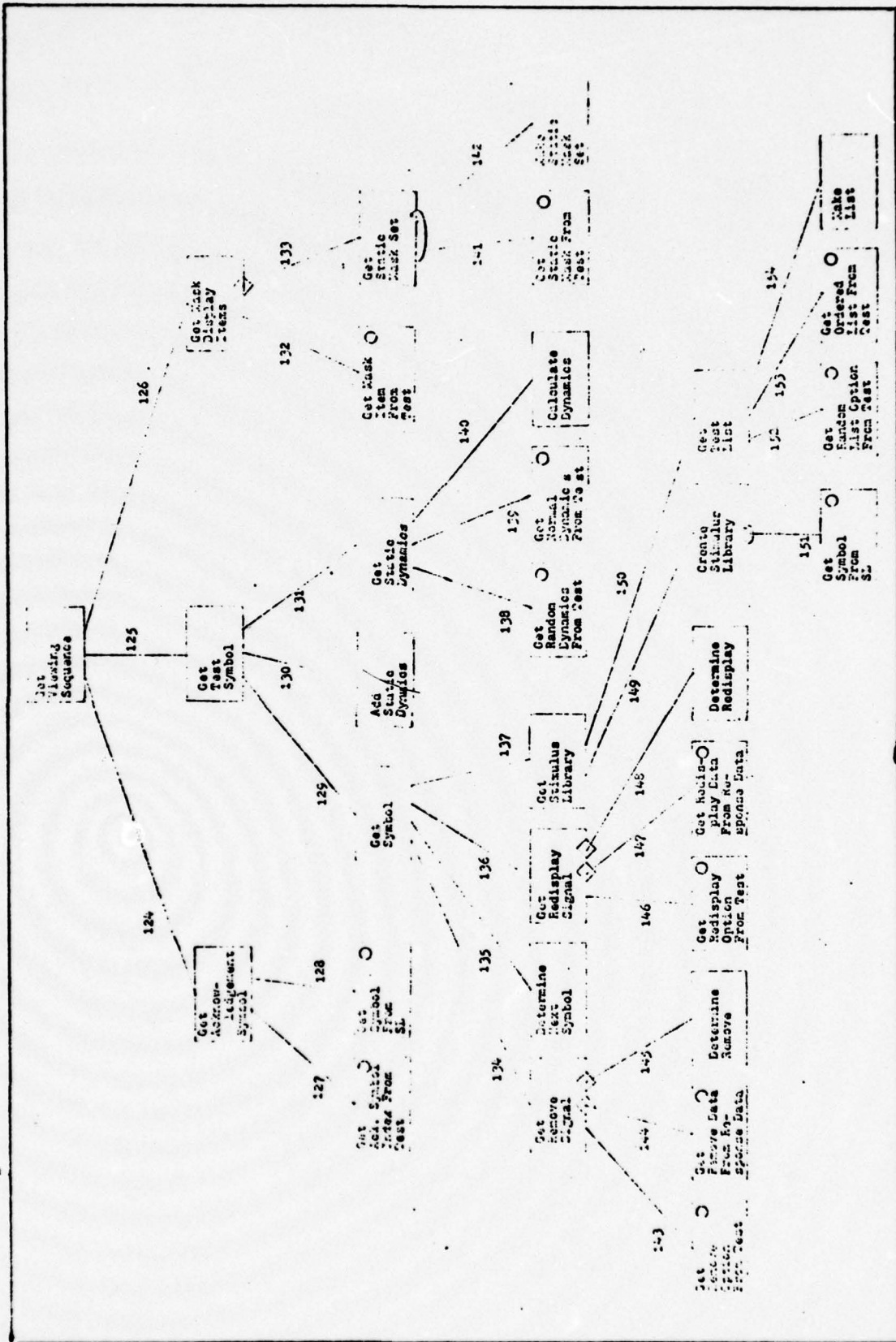


Figure 29. Get Viewing Sequence

Interface

IN	OUT
124 ----	acknowledgement symbol
125 ----	test symbol
126 ----	mask display items
127 ----	index
128 index	acknowledgement symbol
129 ----	symbol
130 static dynamics symbol	test symbol
131 ----	static dynamics
132 ----	mask item
133 ----	static mask set
134 ----	remove signal
135 remove signal, redisplay signal, stimulus library	symbol
136 ----	redisplay signal
137 ----	stimulus library
138 ----	random dynamics
139 ----	normal dynamics
140 random dynamics, normal dynamics	static dynamics
141 ----	static mask
142 static mask	static mask set
143 ----	remove option
144 ----	remove data
145 remove option, remove data	remove signal
146 ----	redisplay option
147 ----	redisplay data
148 redisplay option, redisplay data	redisplay signal
149 test list	stimulus library
150 ----	test list
151 index	symbol
152 ----	random list option
153 ----	ordered list
154 random list option, ordered list	test list

Figure 29 continued

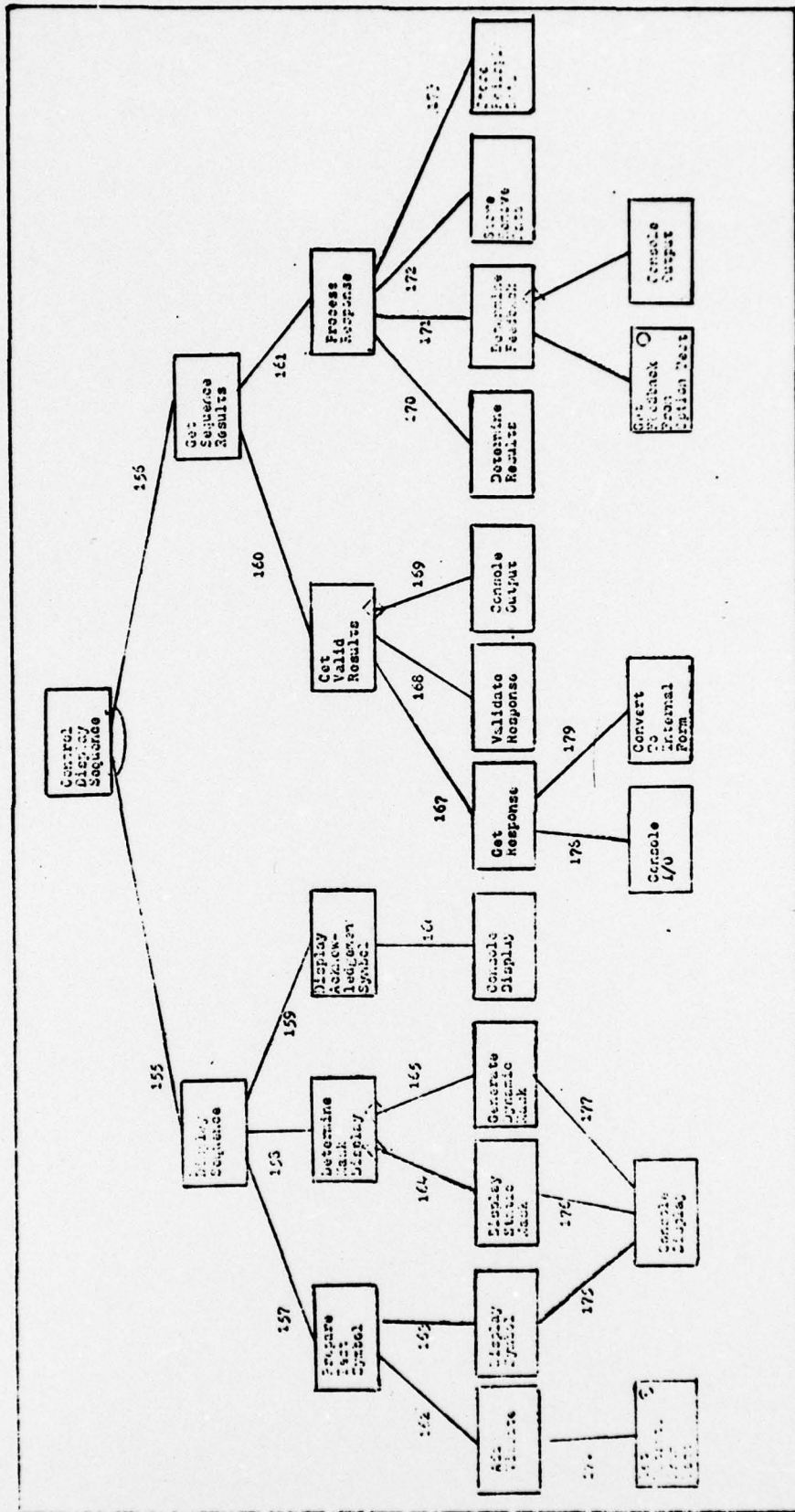


Figure 30. Control Display Sequence

Interface		
	IN	OUT
155	viewing sequence, time parameters	----
156	----	sequence results, <u>stopflag</u>
157	test symbol, stimulus time	----
158	mask item, mask times	----
159	acknowledgement symbol	----
160	----	valid response, <u>stopflag</u>
161	valid response	sequence results
162	symbol	display symbol
163	display symbol	----
164	static mask set	----
165	mask item	----
166	acknowledgement symbol display	----
167	----	response
168	response	valid response, <u>okflag</u> , <u>stopflag</u>
169	error message	----
170	valid response	sequence results
171	sequence results	----
172	remove data	----
173	redisplay data	----
174	vibrate option	----
175	stimulus display	----
176	static mask display	----
177	dynamic mask display	----
178	prompting message	input line
179	input line	response
180	----	feedback option
181	feedback message	----

Figure 30 continued

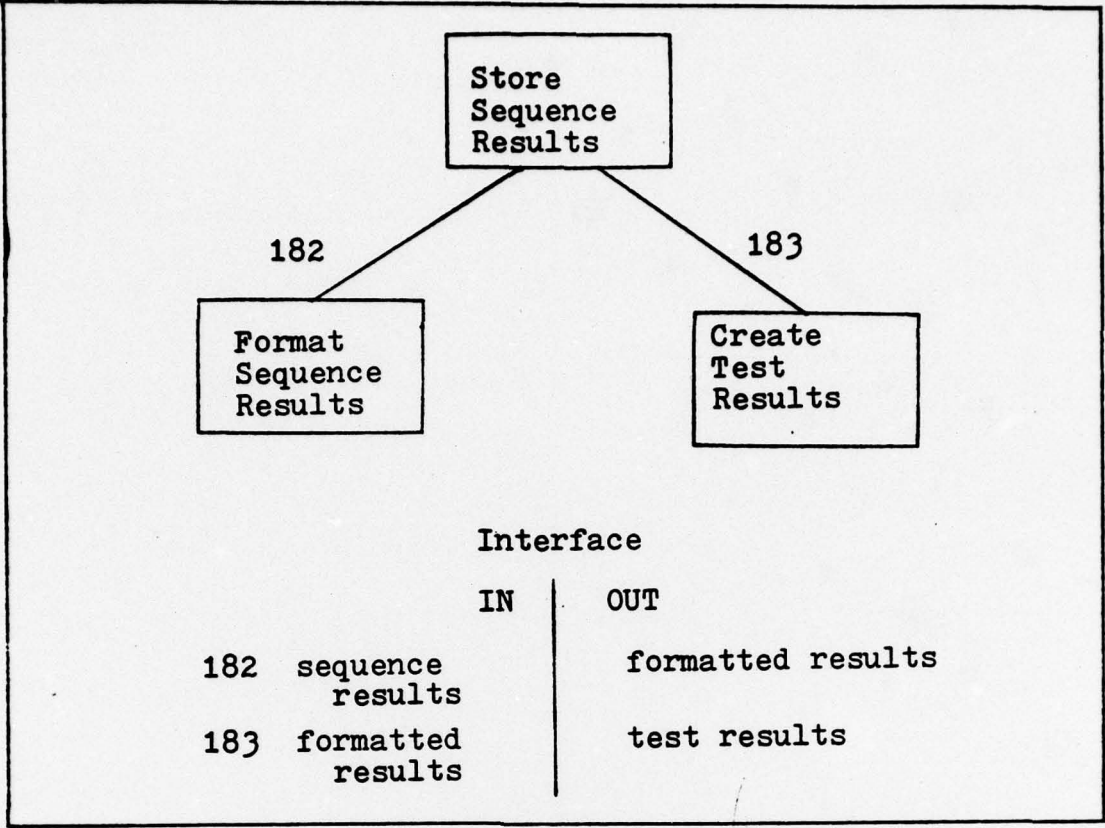


Figure 31. Store Sequence Results

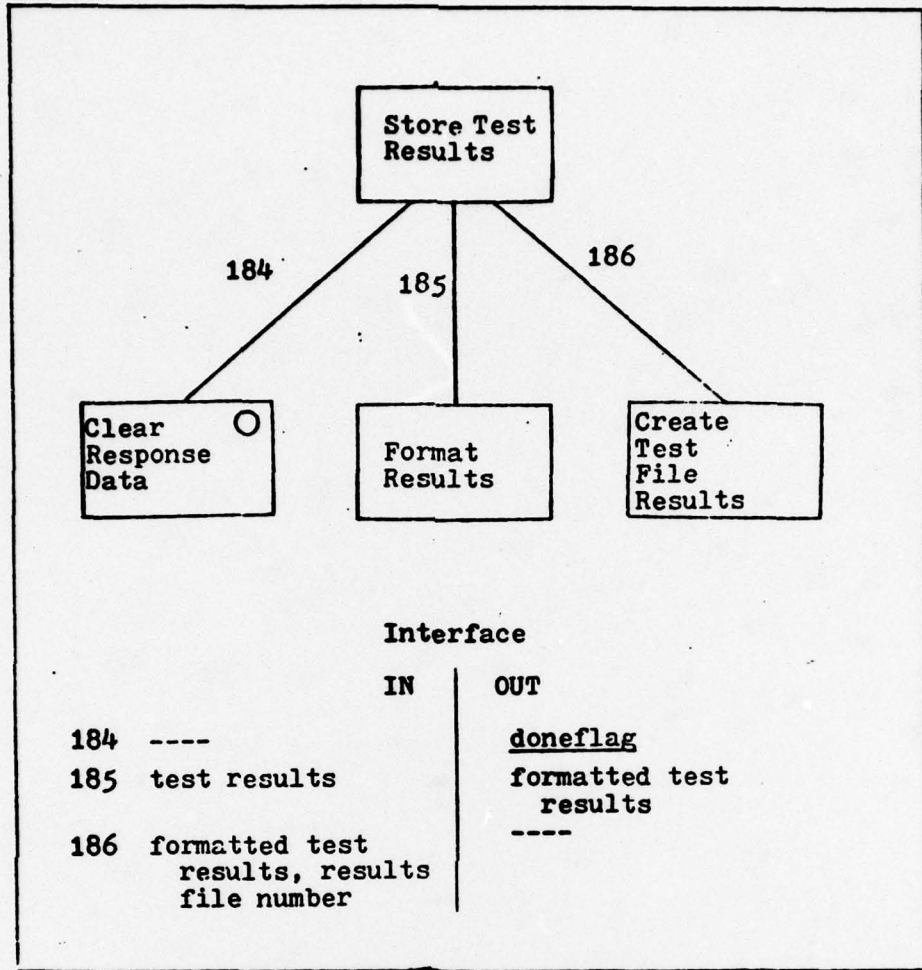
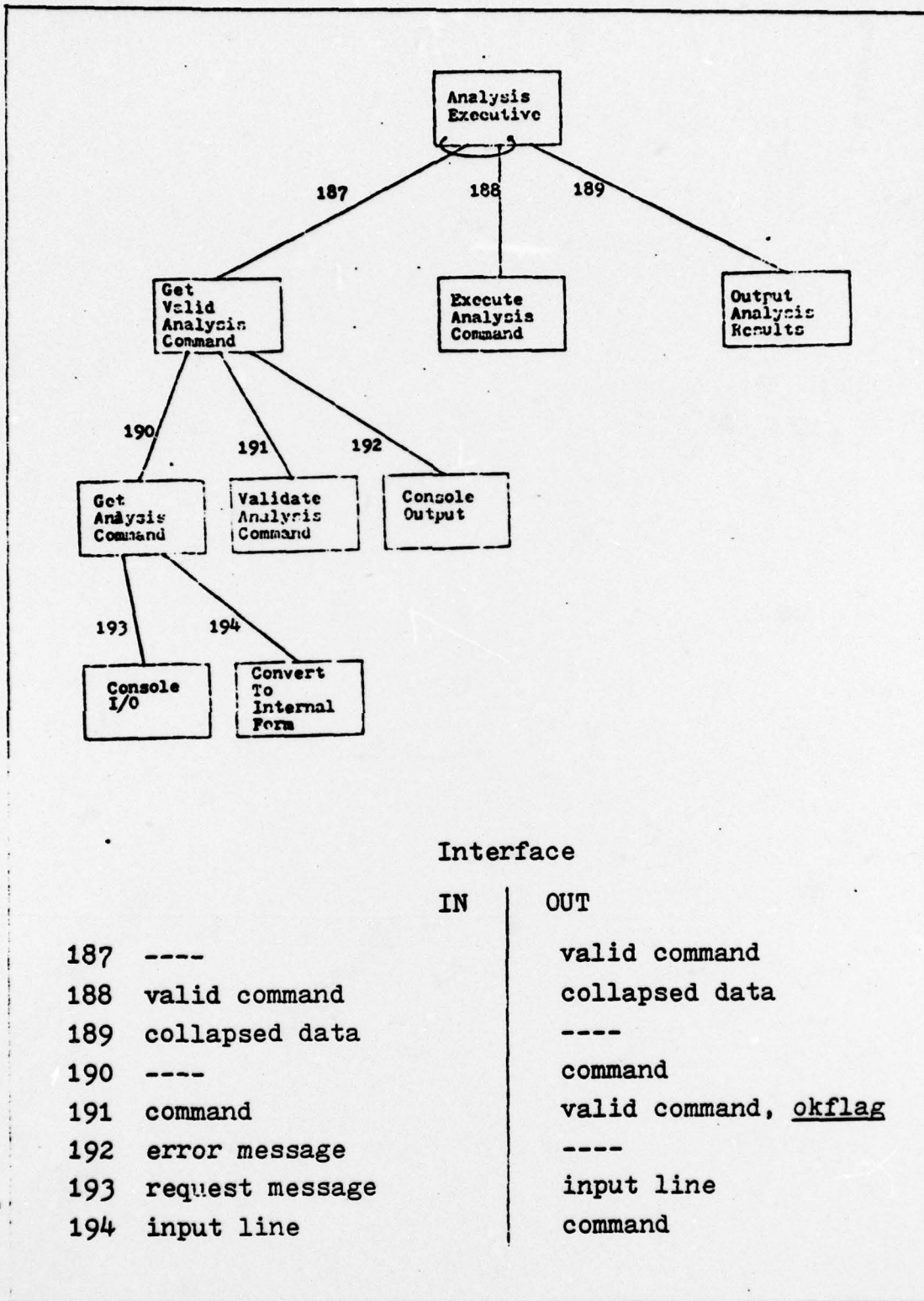


Figure 32. Store Test Results



Interface

IN	OUT
187 ----	valid command
188 valid command	collapsed data
189 collapsed data	----
190 ----	command
191 command	valid command, <u>okflag</u>
192 error message	----
193 request message	input line
194 input line	command

Figure 34. Analysis Executive (First - Level and Afferent Branch Factoring)

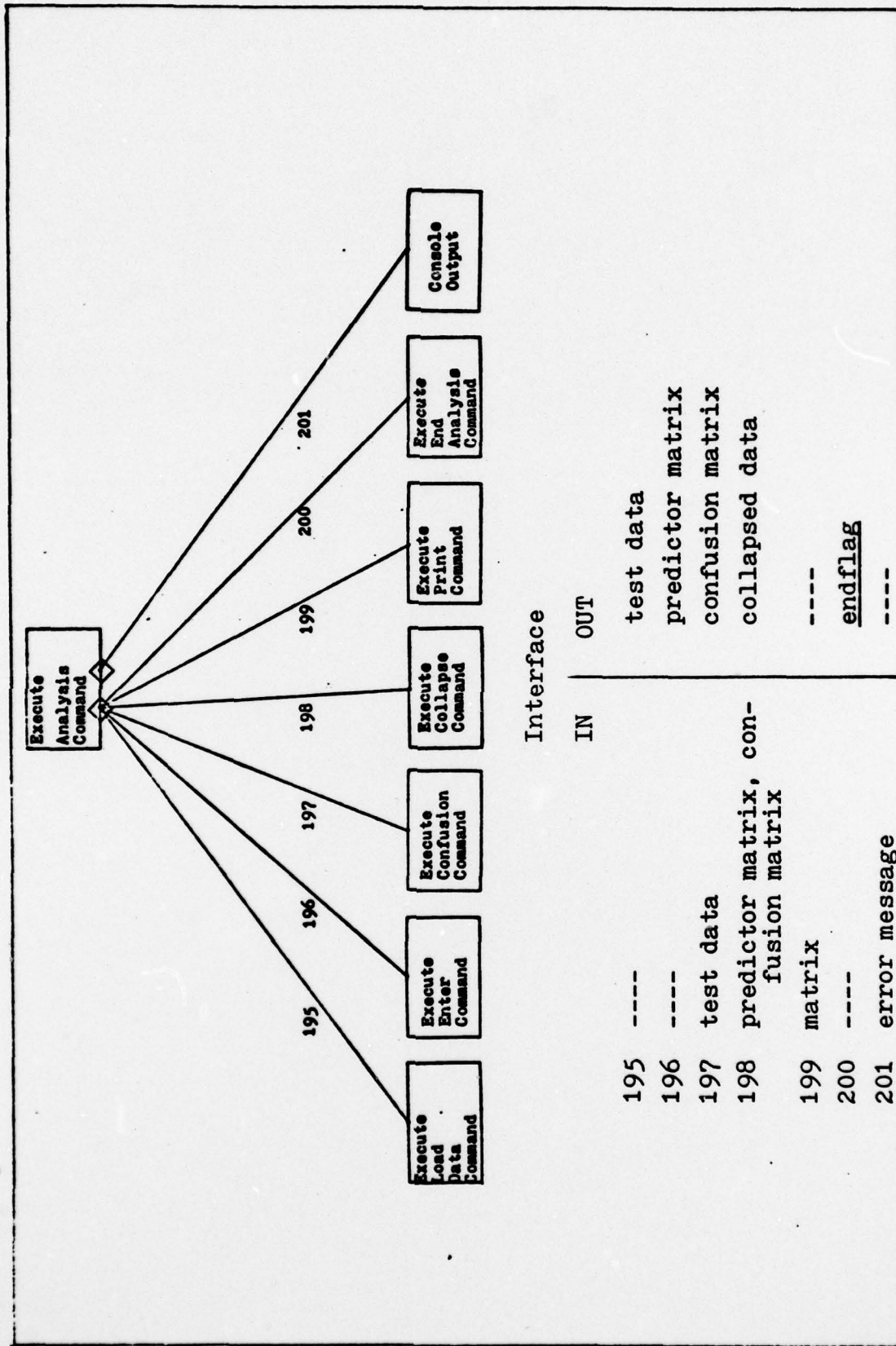
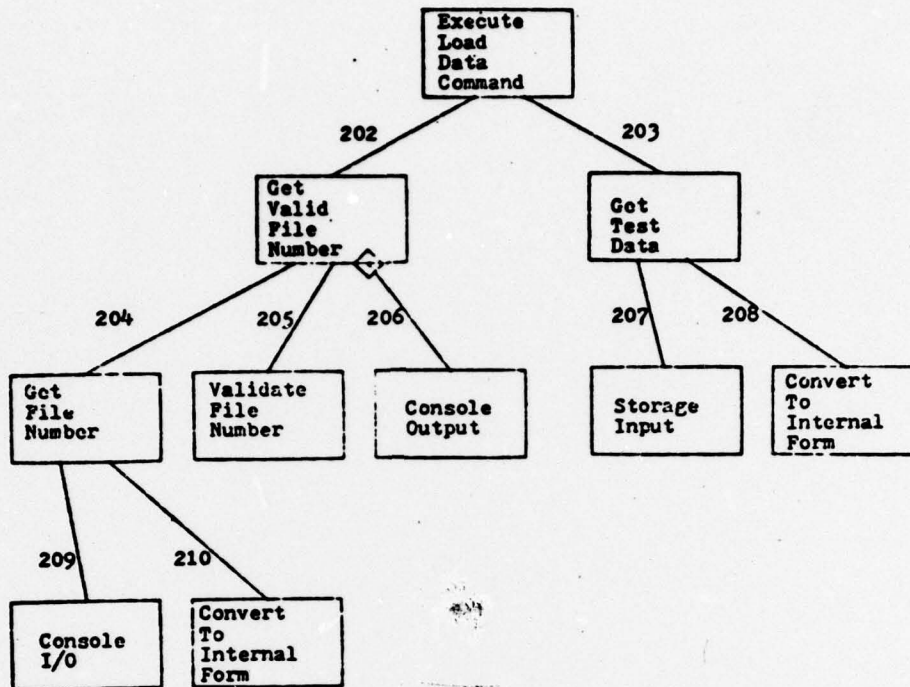


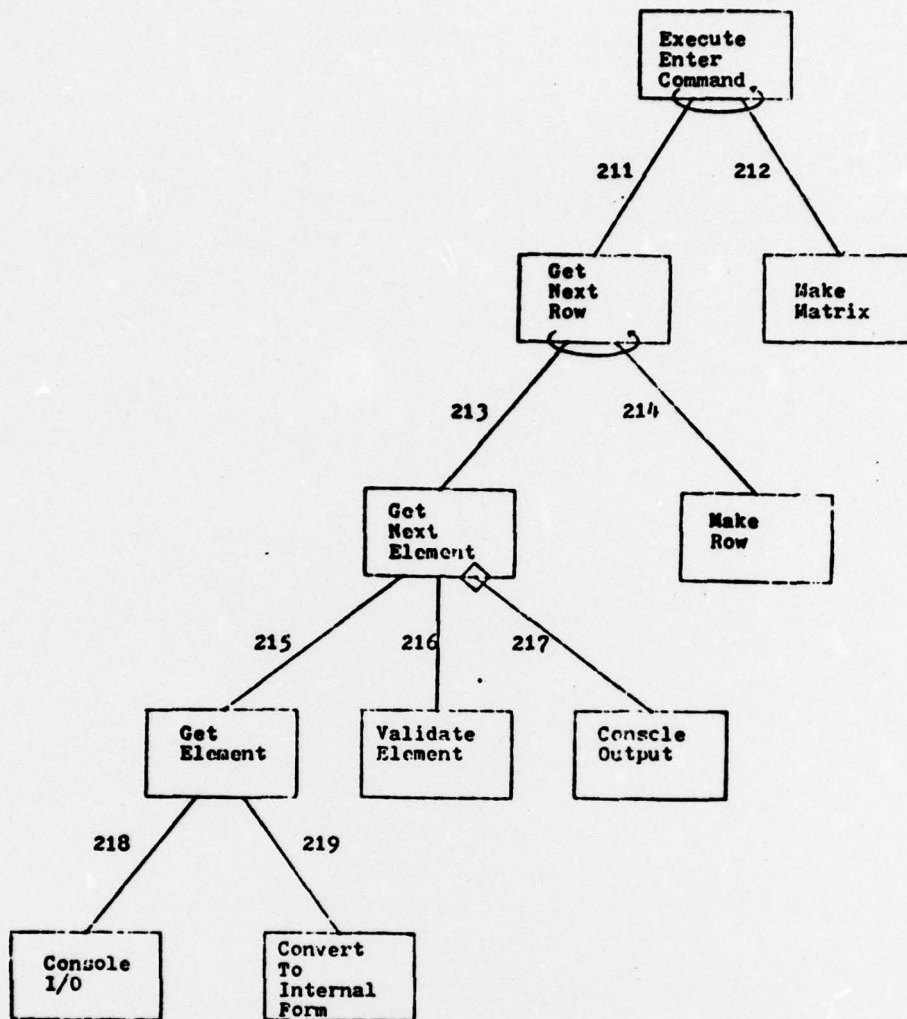
Figure 35. Execute Analysis Command (Transaction Center)



Interface

IN	OUT
202 ----	valid file number
203 valid file number	test data
204 ----	file number
205 file number	valid file number, <u>okflag</u>
206 error message	----
207 file number	input data
208 input data	test data
209 request message	input line
210 input line	file number

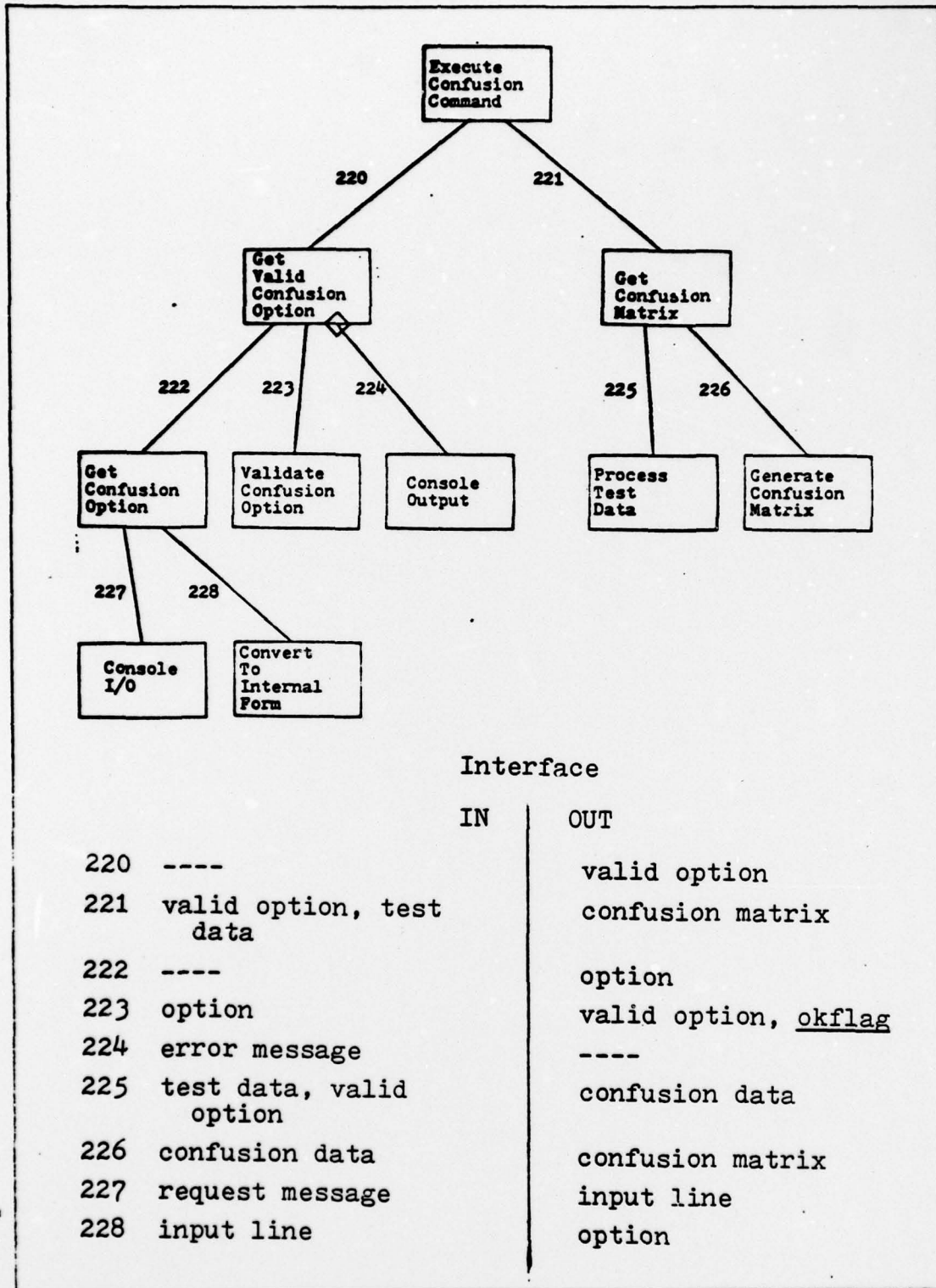
Figure 36. Execute Load Data Command



Interface

	IN	OUT
211	----	row
212	row	predictor matrix
213	----	valid element
214	valid element	row
215	----	element
216	element	valid element, <u>okflag</u>
217	error message	----
218	request message	input line
219	input line	element

Figure 37. Execute Enter Command



Interface

	IN	OUT
220	----	valid option
221	valid option, test data	confusion matrix
222	----	option
223	option	valid option, <u>okflag</u>
224	error message	----
225	test data, valid option	confusion data
226	confusion data	confusion matrix
227	request message	input line
228	input line	option

Figure 38. Execute Confusion Command

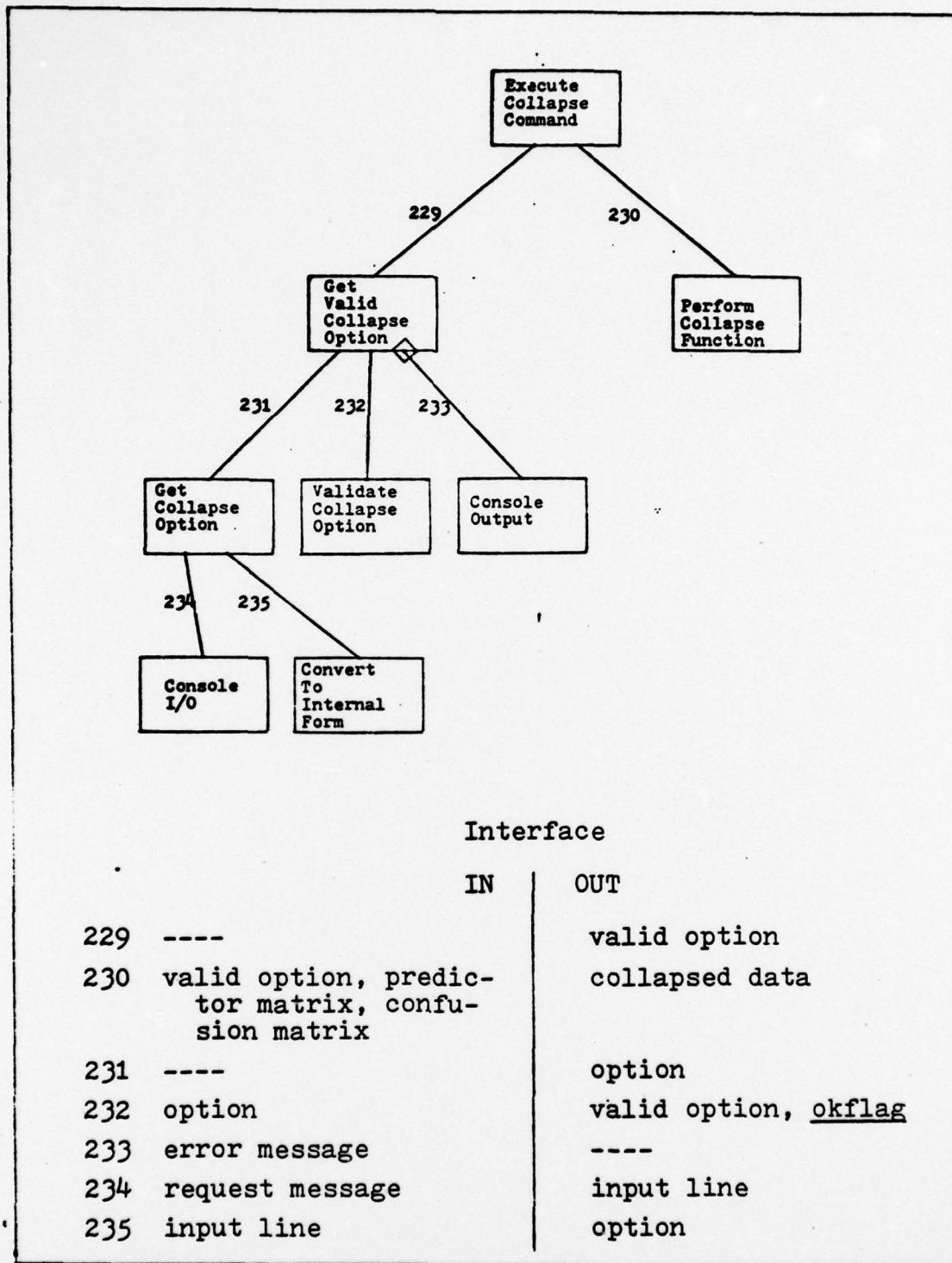
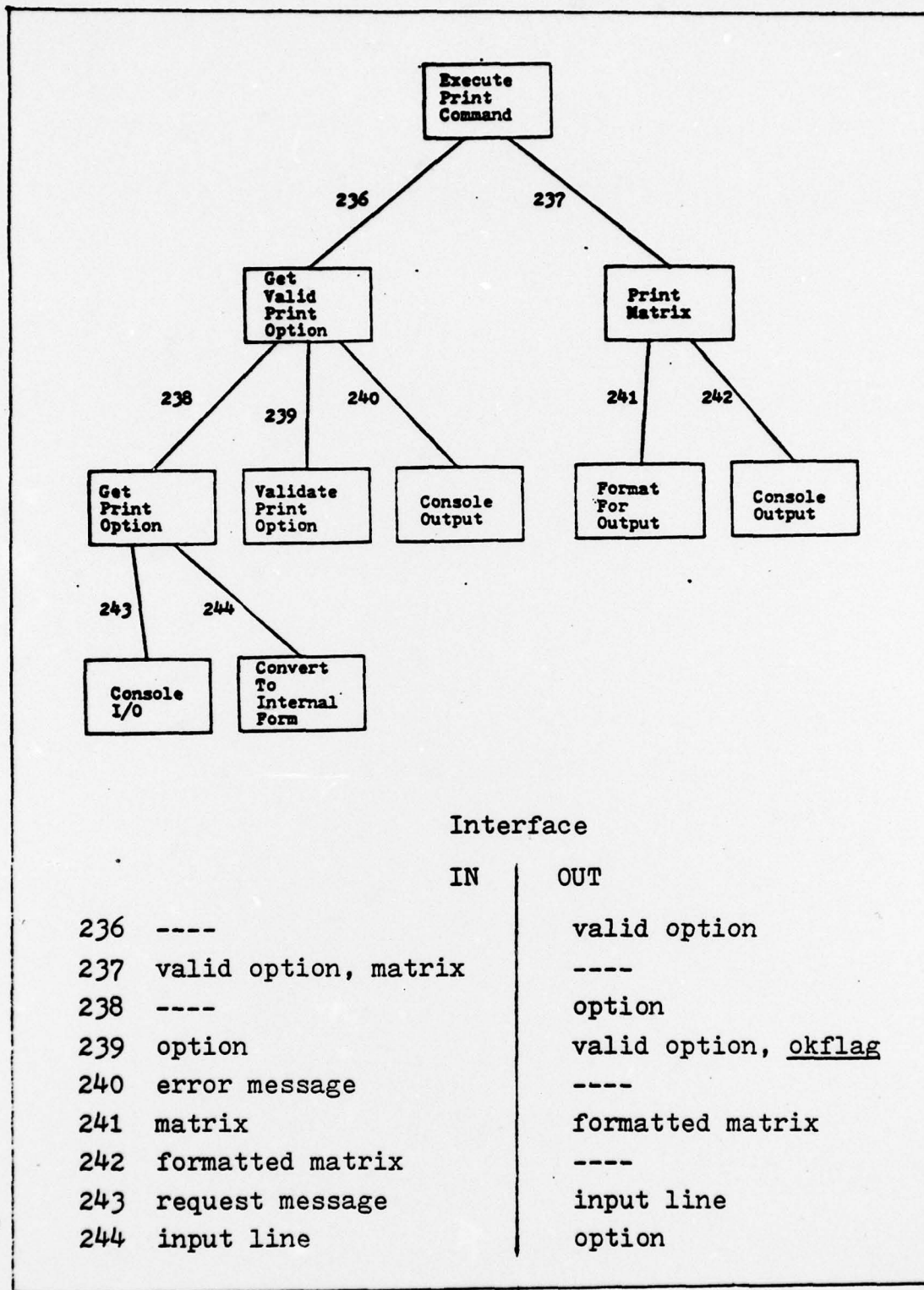


Figure 39. Execute Collapse Command



Interface

	IN	OUT
236	----	valid option
237	valid option, matrix	----
238	----	option
239	option	valid option, <u>okflag</u>
240	error message	----
241	matrix	formatted matrix
242	formatted matrix	----
243	request message	input line
244	input line	option

Figure 40. Execute Print Command

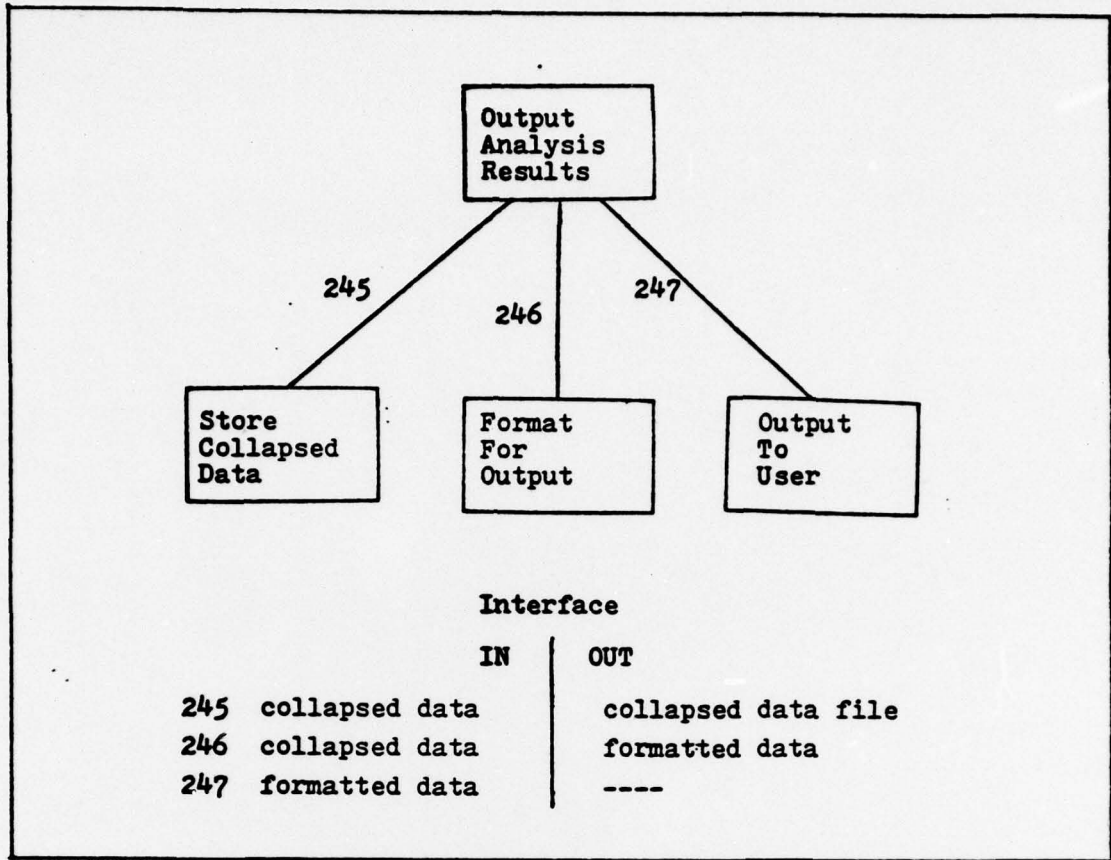


Figure 41. Output Analysis Results


VITA

Stefan G. Wenska was born on 23 October 1948 in Honolulu, Hawaii. He graduated from the United States Air Force Academy in June 1970. He went to pilot training at Columbus AFB, Mississippi. After receiving his wings, he flew the C-130 aircraft at Pope AFB, North Carolina until June 1976. During this tour, he had several two-month TDY's to Taiwan, Thailand, Germany, and England. In June 1976, he entered the Air Force Institute of Technology.

Permanent Address: 45-115E Waikalua Rd.
Kaneohe, Hawaii 96744

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GCS/EE/77-11	2. GOVT ACCESSION NO. ✓	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Preliminary Design For Multimode Matrix Perception Experiment Software		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
7. AUTHOR(s) Stefan G. Wenska Captain USAF		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology ✓ (AFIT/EN) Wright-Patterson AFB, Ohio 45433		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Flight Dynamics Laboratory (AFFDL/FGR) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE December 1977
		13. NUMBER OF PAGES 154
		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  JERRAL F. GUESS, Captain, USAF Director of Information		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Design Software Maintainability Structured Design		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The Multi-Mode Matrix Display Program is testing the acceptability of using light-emitting diode displays in USAF aircraft. The preliminary design for the software was done by following a method which enhances software maintainability. The method uses abstract data types, data flow diagrams, and structured design techniques to produce a complete design for the system. The software design is presented using structure charts together with functional descriptions of all modules and definitions of the interfaces between modules.		

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

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)