NASA CR- 132405

FEASIBILITY STUDY OF AN INTEGRATED PROGRAM FOR AEROSPACE-VEHICLE DESIGN (IPAD) SYSTEM

by C. A. Garrocq, M. J. Hurley et al

VOLUME V

DESIGN OF THE IPAD SYSTEM

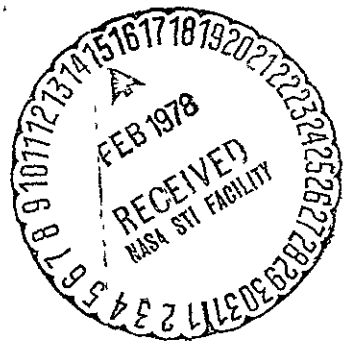
PART II - SYSTEM DESIGN

PART III - GENERAL PURPOSE UTILITIES

(PHASE I, TASK 2)

30 August 1973

Publicly Released
February 10, 1978



Prepared Under Contract No. NAS 1-11431 by

GENERAL DYNAMICS/CONVAIR AEROSPACE DIVISION
San Diego, California
for
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

FEASIBILITY STUDY OF AN
INTEGRATED PROGRAM FOR AEROSPACE-VEHICLE
DESIGN (IPAD) SYSTEM

VOLUME I - SUMMARY

VOLUME II - CHARACTERIZATION OF THE IPAD SYSTEM
(PHASE I, TASK 1)

VOLUME III - ENGINEERING CREATIVE/EVALUATION PROCESSES
(PHASE I, TASK 1)

VOLUME IV - DESIGN OF THE IPAD SYSTEM
PART I - IPAD SYSTEM DESIGN REQUIREMENTS
(PHASE I, TASK 2)

VOLUME V - DESIGN OF THE IPAD SYSTEM
PART II - SYSTEM DESIGN
PART III - GENERAL PURPOSE UTILITIES
(PHASE I, TASK 2)

VOLUME VI - IMPLEMENTATION SCHEDULE
DEVELOPMENT COSTS
OPERATIONAL COSTS
BENEFIT ASSESSMENT
IMPACT ON COMPANY ORGANIZATION
SPIN-OFF ASSESSMENT
(PHASE II, TASKS 3 to 8)

FOREWORD

This investigation was conducted for the NASA Langley Research Center by the Convair Aerospace Division of General Dynamics Corporation under Contract NAS 1-11431.

The NASA Technical Monitor was Dr. Robert E. Fulton, Head, IPAD Technology Section, Design Technology Branch, Structures and Dynamics Division, assisted by Dr. Jaroslaw Sobieszczanski and Mrs. Susan J. Voigt. The Convair Project Leader was Mr. C. A. Garrocq. This volume was prepared by M. J. Hurley, W. E. Jenkins, R. R. Diaddigo, R. H. Schappelle, D. W. Peterson, J. M. Maughmer, J. D. Neilson, G. M. Oing and J. M. Mallory, assisted by T. M. Wooster, J. T. Gordon, M. J. Cronk and R. A. Westerwick.

The Control Data Corporation participated in the performance of this study as a subcontractor to the General Dynamics Corporation. Also indirectly assisting were personnel from UNIVAC Division of Sperry Rand and International Business Machines.

The period of performance was from 15 March 1972 to 30 August 1973.

TABLE OF CONTENTS

PART II - SYSTEM DESIGN

TABLE OF CONTENTS, Contd

PART II - SYSTEM DESIGN

TABLE OF CONTENTS, Contd

PART II - SYSTEM DESIGN

TABLE OF CONTENTS

PART III - GENERAL PURPOSE UTILITIES

PART III - GENERAL PURPOSE UTILITIES

TABLE OF CONTENTS, Contd

PART III - GENERAL PURPOSE UTILITIES

PART III.- GENERAL PURPOSE UTILITIES

TABLE OF CONTENTS, Contd

PART III - GENERAL PURPOSE UTILITIES

LIST OF FIGURES

PART II - SYSTEM DESIGN

LIST OF FIGURES, Contd

PART II - SYSTEM DESIGN

LIST OF FIGURES

PART III - GENERAL PURPOSE UTILITIES

LIST OF FIGURES, Contd

PART III - GENERAL PURPOSE UTILITIES

LIST OF TABLES

PART II - SYSTEM DESIGN

LIST OF TABLES

PART III - GENERAL PURPOSE UTILITIES

An IPAD system is defined herein as consisting of four major components, as shown in Figure S-1: (1) A Management Engineering Capability represented by a battery of automated Operational Modules for various management/design/engineering disciplines, (2) an IPAD Framework Software which supports and augments the Engineering Capability, (3) an Operating System Software, which features a comprehensive Data Base Management System, and (4) a Computer Complex Hardware, on which all the Engineering, IPAD, and System software will be mounted and exercised. From this statement, it can be inferred that the Management/Engineering Capability can and should be tailored to the specific needs of the management/design/engineering team (i.e., the battery of Operational Modules for aircraft design would be different than that for missiles, or navy vessels, or terrestrial vehicles, or civil engineering projects, although many common elements could be identified). On the other hand, the IPAD Framework Software, the Operating System Software, and the Computer Complex Hardware could have essentially the same basic capabilities for all users, with freedom of choice in specific software, and type and quality of equipment desired within each computer complex.

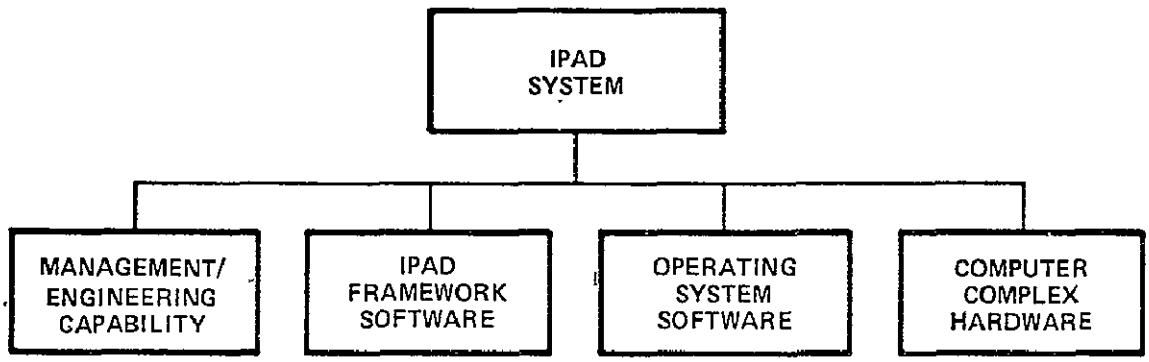


Figure S-1. Major IPAD System Components

The organization, engineering usage philosophy, and the accompanying IPAD design concept developed in this study provide the flexibility required to satisfy the project needs of any management/design/engineering team which will use and exploit the IPAD system's capability in any way it sees fit.

IPAD Computer Software

The various elements of computer software associated with IPAD are illustrated in Figure S-2.

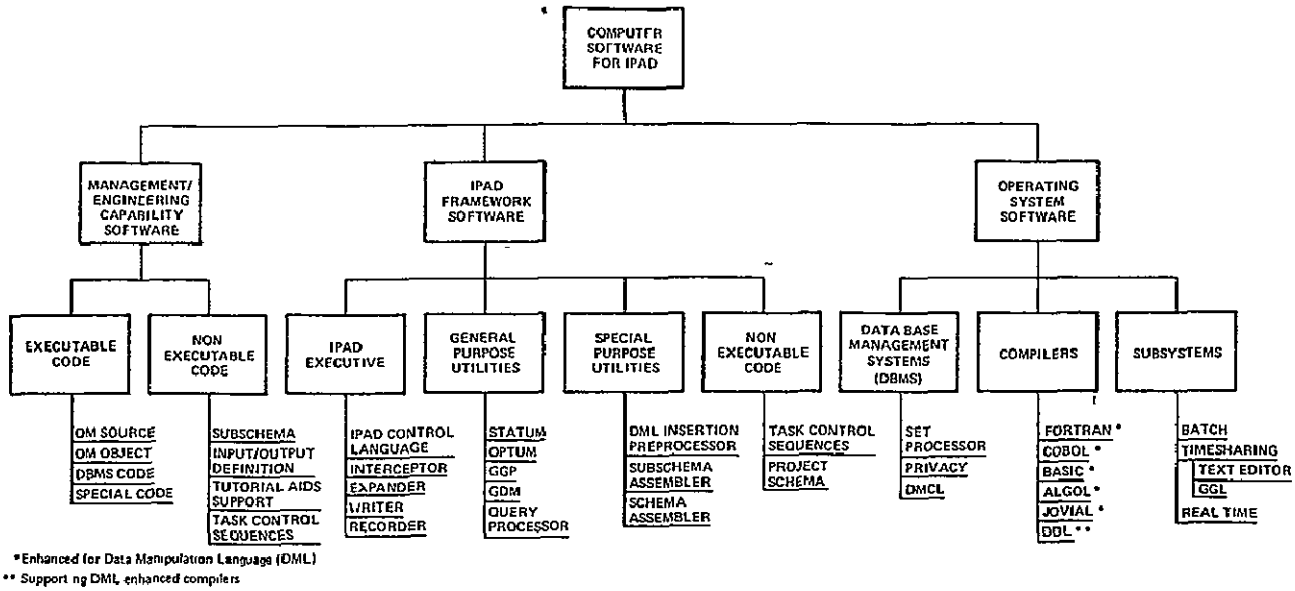


Figure S-2. Computer Software Associated with IPAD

The three major classes of software are:

1. Management/Engineering Capability Software. - The total automated capability of the management/engineering/science community is resident in a library of automated operational modules consisting of both a public domain library, accessible to all parties, and private libraries containing modules with limited or restricted availability due to the nature of its contents being private data, classified information, or the like. From the total gamut of available modules a project team will select those which are applicable to their specific project to assemble a project library of automated operational modules that will be installed on the IPAD Computer Complex. The contents of this library are dynamic in the sense that programs are added or removed from it as the need arises, and are resident on disk or tape depending on their usage rate. All project related activities such as management, marketing, economics, technical disciplines, and design/drafting will have their respective automated capabilities installed in the system. The position of this software in relation to other computer software required for IPAD is shown in the first two columns of Figure S-2.

2. The IPAD framework software. - From the user's point of view, IPAD is a framework which supports and augments the capabilities of his computerized management, design/drafting, and analytical tools. From this viewpoint, the framework is composed of a number of utilities and interfacing capabilities, as shown in Figure S-2. The elements of this software are:

   a. The IPAD EXECutive, which is the principal contact that the IPAD user has with the system. The EXEC is additionally supported by four utilities:

      - The Task Control Sequence Skeleton Writer, which is an interactive program to assist the user in writing a sequence of automated tasks.

- The Task Control Sequence Skeleton Expander, which is an interactive (or batch) program to assist the user in tailoring a task sequence to his specific application.
- The Interceptor, which enables the user to maintain a record of his transactions.
- The User's Task Trajectory Recorder, which performs automatic recording of the sequence of transactions in which the user was actually engaged.

b. General Purpose Utilities, to provide an augmentation of the user's Operational Modules. There are five general purpose utilities in the present design of IPAD, some of which are highly modular and can be developed in various release levels as discussed below.

- The statistical utility module (STATUM), which provides the user with a statistical package that can be used at an interactive terminal.
- The general purpose optimizer (OPTUM), which provides the user with an interactive collection of multivariable search techniques and tutorial aids for optimization and parametric studies, fulfills one of the basic needs of a project engineering team. Its modularity permits releases at various levels of capability.
- The Query Processor, which is an interactive COBOL program that enables the user to control and manipulate the contents and structure of his data sub-bases.
- The General Graphics Plotter, which addresses the need of producing the geometrical, graphical and pictorial displays required by interactive users in any design process, is a basic element of the new IPAD design environment. It is highly modular and considerable design and implementation is presently underway in industry at large. The impact of this current development activity could substantially reduce the costs estimated for this task.
- The General Design Module, which augments the design function through interactive-design/automated-drafting software and equipment, is the cornerstone of board design activities in the new IPAD environment. It is destined to be the largest, most system-demanding, and second most frequently used IPAD utility (after the Query Processor). It is perhaps the most modular utility and involves the development of large amounts of new code. Due to the criticality of response time for the design function, this module is to be supported mainly by a minicomputer with proper interfaces to the host operating system. The basic elements of this module are a comprehensive information storage and retrieval system, 3-D geometrical building blocks, and design and analysis program libraries.

c. The Special Purpose Utilities, to assist IPAD users in interfacing their computer programs with the Data Base Management System for which a great deal of information is required. Without the support of the utilities this interfacing would entail a prohibitive amount of time and labor. These utilities are considered indispensable for all release capabilities of IPAD. They are:

- The Data Manipulation Language Insertion Preprocessor, which is a batch utility to replace conventional FORTRAN input/output coding with logically equivalent Data Manipulation Language statements.
- The SUBSCHEMA Assembler, which is an interactive utility to extract data descriptors from the conventional input/output of a program and generate Data Description Language statements to interface with the data base.
- The SCHEMA Assembler, which is an interactive utility to integrate several computer programs into one execution sequence and resolve conflicts with common and duplicated data items, data base structure, and required transformations.

d. Non-executable Code, to define the extensive data base organization related to the selected aerospace-vehicle project activity, and to specify project-oriented task control sequences to be used during the design process.

3. Operating system software. - IPAD is designed to fully exploit the host computer's operating system software. In particular, the operating system must be upgraded to contain a capable timesharing (also called conversational) subsystem and a comprehensive Data Base Management System patterned after the language specifications of CODASYL's Data Base Task Group's recommendations. Two new languages must be developed: (1) a Data Description Language, to describe the data in the data base; and (2) a Data Manipulation Language, to cause the transfer of data between programs and the data base. The functions of the Data Base Management System are: (1) to control the input/output functions of the operating system to satisfy Data-Manipulation-Language requests issued by programs in execution; (2) to perform transformations to correlate SCHEMA and SUBSCHEMA data descriptions; and (3) to provide means of enforcing and maintaining the data integrity and logical structure detailed by the Data Base Administrator. The proposed implementation plan provides support to a FORTRAN Data Base Management System via COBOL and subroutine CALLs.

Objectives of this Volume

The major objectives of this volume are to present viable designs of various elements of the IPAD Framework Software, Data Base Management System, and required new languages in relation to the capabilities of Operating Systems Software.

These objectives were pursued by a thorough evaluation of the basic systems functions to be provided by each software element, its requirements defined in the conceptual design, the operating systems features affecting its design, and the engineering/design functions which it was intended to enhance. The various software elements are discussed in more detail in Parts II and III, and various Appendices that follow.

PHASE I - TASK 2

PART II - SYSTEM DESIGN

1 INTRODUCTORY NOTES

The best summary of the problem leading to the IPAD design concept is contained in the IPAD Statement of Work (Volume IV, Section 1 of Part I), paraphrased as follows:

> The design process involves an army of specialists with conflicting requirements to be resolved via design decisions, and involves many design iterations extending over a very long period of time. Each iteration and each design decision involves the transfer of mountains of related design data.
>
> The overall goal, then, is to level the mountains and collapse the iteration time to the point that aircraft may be properly designed before they are manufactured.

The Conceptual Design (see Volume IV, Section 2 of Part I) developed objectives of a software system conceived to achieve this goal. The basic features of the Conceptual Design are:

1. A central data base - the Multidisciplinary Data Bank (MDB).
2. Communication and tutorial facilities to provide systematic avenues for human-to-human interface in the design process.
3. Computational capabilities provided through a user-oriented command structure to facilitate the user-to-software interface.
4. An I/O formatting capability to interface the independently developed computational capabilities (viz., the OMs).
5. Interactive capabilities to give the user direct contact with the computer processes.

The intent of Part II is to present a viable system design consistent with the objectives put forth in the Conceptual Design. Essentially all the objectives of the Conceptual Design are met by improving the human interface and exploiting software developed (or being developed) independently of IPAD.

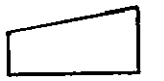
The work of the Data Base Task Group (DBTG) of the Conference on Data System Language (CODASYL) was indispensable in the development of the system design. The impact first becomes apparent in Section 3 in conjunction with Input/Output Formatting, then permeates succeeding sections. This in no way implies endorsement of the system

design by CODASYL; the extent to which CODASYL is involved is in framing a policy which put the DBTG report in the public domain.

The contribution of Control Data Corporation (CDC) as subcontractor is also acknowledged, principally for pointing out that the work of the DBTG was applicable to the IPAD conceptual design and for working documents lending visibility to their future plans with respect to the DBTG recommendations. It is emphasized here that the material made available by CDC was in the nature of preliminary working documents and were accompanied by the following disclaimer:

> DISCLAIMER: This document is a working paper only and does not necessarily represent any official intent on the part of Control Data Corporation.

The figures of this volume generally adhere to the following symbolism for the convenience of the reader:


Any interactive console. Any IPAD user at an interactive console.


Any file on disk.


Command or functional interface.


Data interface.


Interactive communications interface.


Any functional block being executed.


Decision functional block.


Card input. TCS being executed.


Any display information (CRT, printout, or display file).

The reader is referred to Appendix A for a concise glossary of acronyms and special terminology used throughout this report.

2 THE EXECUTIVE

The major function of the IPAD EXECutive is to drive a task through a sequence of job steps as directed by a Task Control Sequence (TCS). A job step consists of the execution of an Operational Module (OM), an IPAD utility, a host operating system utility, etc. In most cases the EXEC, once started on a particular task, completes it without user intervention, but in some cases the user will wish to interact with the task. The IPAD EXECutive (Figure 2-1), therefore, is operationally attached to one user and, in general, one TCS per job step. There may be multiple users at any instant of operation, but they will deal with their own individual copies of the EXEC. (The copy, in implementation, can either be physical copies or a single re-entrant version of the EXEC.)

Since it is likely that the IPAD EXEC will operate within an interactive environment, such timesharing facilities are assumed to be installation-dependent and not part of the EXEC function. Also, the EXEC is dependent upon the host facility for actual loading and execution of OMs/utilities required by the user. The EXEC deals with its data base storage and operation requirements via a data base management system.

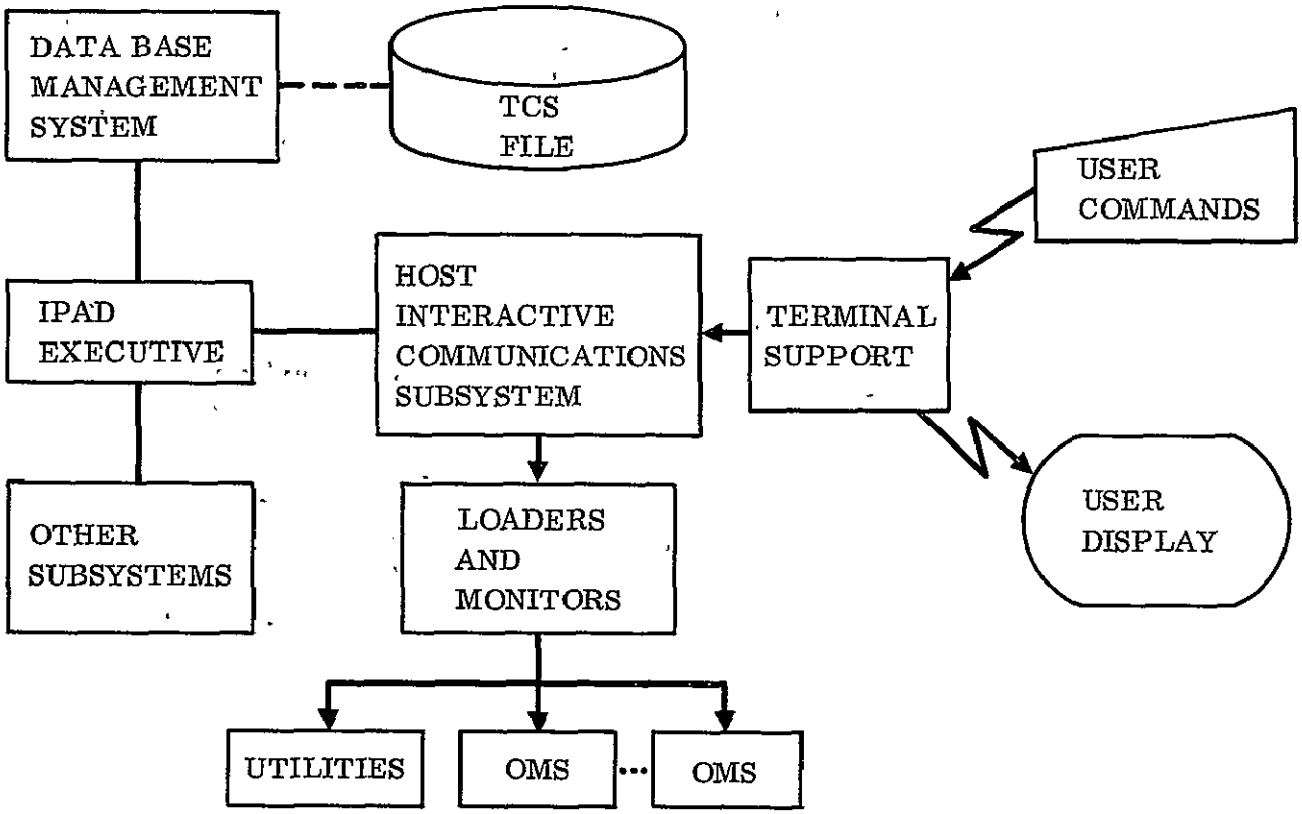


Figure 2-1. IPAD EXEC and Relationship with Other IPAD System Components

2.1 TCS Files

The overall EXEC design is based upon the use of TCS files. (Reasons for this choice are covered in Section 2.6.) A TCS may consist entirely of user-terminal input (simple form) or may contain labels and direct input flags (general form).

2.1.1 Simple form of a TCS file. - In its simplest form a TCS file consists of exactly the input (commands, data, control characters) that a user must type at his terminal to run a particular task. This form is already available on many systems and is commonly called a commands file; the name comes from the GENIE* system and is something of a misnomer because the file can contain much more than commands.

This form of TCS (or commands) file replaces all terminal input. Whenever a program requests input (e.g., because it has just executed a READ statement directed to the interactive terminal), or the operating system itself requests a command (e.g., because a program has terminated or aborted), the input is provided by reading from the TCS. If the end-of-information occurs on the TCS file while that file is still active, control reverts to the terminal.

2.1.2 General form of a TCS file. - In its general form a TCS file may also contain labels and direct input flags. Both are denoted by special characters to separate them from the simulated terminal input making up the bulk of the file. Labels are used to mark specific locations in the file - very much like statement numbers are used in a FORTRAN program - so that the EXEC can reposition the file to these locations. This allows conditional branching, looping, etc., within the TCS.

Direct input flags are used where input is to be taken from the interactive terminal instead of from the TCS. When a direct input flag is encountered on the TCS, the normal connection to the terminal (or to the input file in the batch mode) is re-established, and all further input is taken from there until either a specified number of lines or a specified termination code has been read. The type of termination used is determined from the direct input flag itself.

2.1.3 Examples of a TCS. - A TCS may be a mixture of the commands for the variety of subsystems that support an IPAD task. Figure 2-2 shows the TCS command structure as it might look using CDC's interactive communication subsystem (INTERCOM 4.1

*GENIE is a timesharing system developed on an SDS 940 at the University of California at Berkeley.

under SCOPE 3.4). On the left side of the figure is the actual TCS; on the right side is the action as a consequence of the TCS.

1)	ATTACH (OLDATA, PERMFILENAME, PW = XYZ123)		ATTACH INPUT FILE
2)	XEQ, FIRSTOM (OLDDATA, NEWFILE)		EXECUTE FIRST OM
3)	150		DATA READ BY FIRST OM —
4)	229568.41		2 LINES FROM TCS
		(PAUSE)	
5)	(DIRECT INPUT FLAG)		2 LINES FROM TERMINAL
6)	XEQ, SECNDOM.		EXECUTE SECOND OM
7)	88		DATA READ BY SECOND OM
		(PAUSE)	
8)	(DIRECT INPUT FLAG)		USER TYPES CATALOG COMMAND
9)	XEQ, LOAD=IPADEX, EXECUTE, EOJ		RELOAD IPAD EXEC & RESTART AT END-OF-JOB ENTRY POINT

Figure 2-2. TCS Example with User Interaction (CDC Format)

In this example, the user is to run two OMs using an existing permanent file. The output of the first OM is used by the second OM; at the completion of the second OM, the results are to be saved. The INTERCOM commands "ATTACH" and "XEQ" are self-evident in their operations of notifying the host computer's operating system to attach a permanent file and to execute particular programs as identified. At the third and fourth lines of the TCS two pieces of data occur that will be requested by the OM during execution. In this instance the data is read directly off the TCS file and supplied to the OM. At the fifth line, a different situation occurs: the user has indicated that at this point he wishes control returned to the terminal, where he may insert the required data values. When this is done, possibly with clarifying comments to say what is expected of him, he types in the lines of information; for example:

```
0., -1., -2., -3., -4., 0.
33
```

The process then continues (without further action by the user) executing the second OM.

At the end of the second OM's execution, the user has left open the option of what he wishes to do; again control is returned to the terminal for his direction. In

the example given, the user has decided that he wishes to save the output resulting from his sequence of operation. He therefore issues the command for cataloguing:

```
CATALOG(OUTFILE,OUTFILEPERMFILENAME,TK=PASWRD)
```

The return is then back to the IPAD EXEC for further user action.

2.2 TCS File Generation.

There are several ways a TCS file can be built. Undoubtedly the easiest is to employ a utility called the TCS interceptor, which is part of the EXEC. When this utility is brought into effect, by user command, everything the user types at his terminal will still have its normal effect, but in addition almost everything (except for commands directed to the interceptor itself) will also be written out on a file. Thus, a user can proceed step-by-step through a task and produce a TCS file that can be used to rerun the same task automatically later. Even if the user makes some mistakes in the initial run-through, he can go back and clean up the file afterwards with the system's interactive Text Editor. The Text Editor can also be used to modify a given TCS, by putting in direct input flags in place of one or more lines of input. More complicated TCSs, however, such as those involving labels and calls to loop-generating or condition-testing EXEC routines, will probably be written from scratch using the interactive Text Editor, or even from regular card input.

An additional feature is provided to help engineers run complex IPAD tasks: this is the Task Control Sequence Skeleton, or TCSS, and the TCSS Expander (see below). In its simplest form a TCSS amounts to a TCS with blanks to be filled in; in its general form it may involve macro definitions and calls, symbolic parameters, and conditional or multiple expansio. .

2.2.1 The TCSS Expander. - The TCSS Expander is intended to simplify and expedite the production of complex character-oriented files. Although it is designed particularly to aid IPAD users in setting up TCS files, it can be employed as a general-purpose macro expander to generate many types of character-oriented (as opposed to binary) files.

2.2.1.1 Structure of a skeleton file: The left-hand side of Figure 2-3 illustrates schematically the structure of a skeleton file. A skeleton file , which serves as input to the Expander to produce the expanded file, is composed of TCS image text, control statements, dummy arguments and a tutorial prologue. The definitions of these components are as follows:

1. TCS image text is the invariant portion of the TCS image data. In the Expander operation, it is simply copied character-for-character from the skeleton file to the expansion file, without modification.

2. Control statements are used to control the operation of the Expander. They create dummy arguments and parameters, initiate the conditional expansion or skipping of parts of the skeleton file, define nested macros, etc. Unlike the image text, control statements do not directly produce output on the expansion file.

3. Dummy arguments are used to mark the place on the skeleton file where actual arguments are to be substituted. They can be used alone, or embedded in the image text or control statements. The substitution process is the same in either case, because actual arguments are substituted for all dummy arguments in each line even before that line is scanned to determine whether it is to be written on the expansion file as image text or interpreted as a control statement.

4. Tutorial prologue is displayable data that explains to the users the function of the TCSS and any other additional information the author may consider important for the user's understanding.

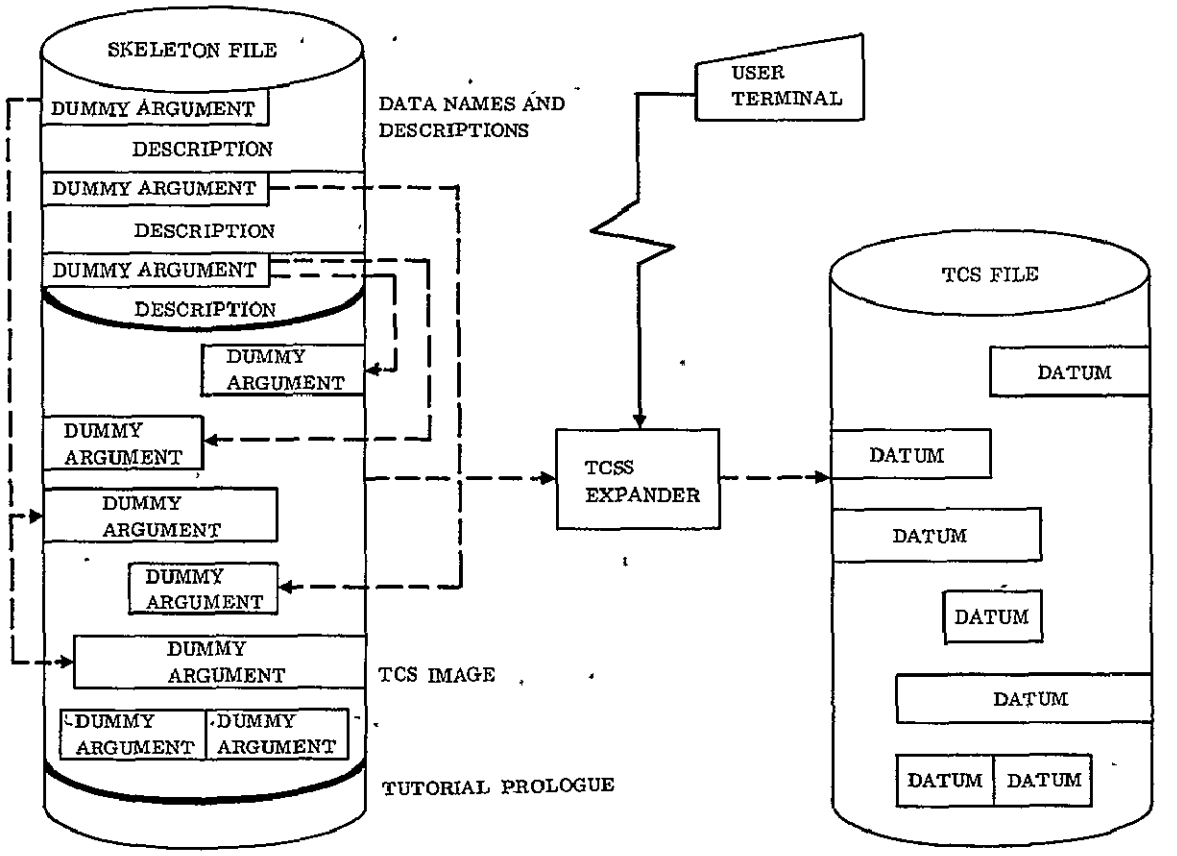


Figure 2-3. TCSS and the Expansion Process

2.2.1.2 Operation of the Expander: Figure 2-3 also depicts the operation of the TCSS Expander. A skeleton file will typically begin with a series of control statements that define all the dummy argument names for which the user is to supply input. Each dummy argument name is accompanied by a short explanatory writeup. When the Expander encounters a dummy argument statement it asks the user for the argument by name. If the user recognizes the name — and meaningful mnemonic names should always be used for dummy arguments — he enters the appropriate argument. If the user does not recognize the name or does not understand the type of data he is supposed to provide, he can ask for help, and the Expander will print out the writeup for that dummy argument and then ask for it again. Of course, whether or not the user is able to provide it on the second try, depends on how good a job the author of the skeleton file did on his writeups. Given good documentation, the Expander can lead an inexperienced user through highly complex file-setup procedures very easily.

Once actual arguments have been supplied for all dummy arguments, the Expander continues through the skeleton file and creates the expanded file. The original skeleton file is not modified in any way; it can be enabled for multi-read access and many users can expand it simultaneously, each producing a different expanded file tailored to their specific needs.

The Expander is not limited to merely reading in data supplied by the user and substituting it into predetermined slots in the skeleton. Some of its additional features are described in the following subsections.

2.2.1.3 Parameters: A parameter is a dummy argument whose actual argument is supplied by the control statement defining it, instead of by user input. Thus the value of a parameter can be changed only by changing the skeleton file, unless the actual argument in the definition of the parameter itself involves a dummy argument (which it legally can). A parameter argument is useful for a value that is likely to remain fixed for some time but may be changed occasionally, as when a new configuration is established; it can be changed simply by changing the parameter definition, thus avoiding a search through the whole skeleton file to find every occurrence of that parameter's value.

Parameters are also useful for internal symbols, counters, etc., in connection with some of the other features described below. One type of parameter allows redefinition, which makes it particularly good for counters since the new value can be defined in terms of the old value; e.g., COUNTER=COUNTER+1.

2.2.1.4 Expression evaluation: Expressions are arithmetic or logical formulas that result in numerical values. They can be used in both image text and control statements. Operands must be constants or dummy arguments whose actual arguments are constants. Operators will include as a minimum addition, subtraction, multiplication and

division; logical AND, OR and EXCLUSIVE OR; and some type-conversion functions. The usual FORTRAN precedence rules are envisioned including parenthesization.

Expressions are first evaluated during expansion then replaced by their own computed values. In effect, an expression is given two complete scans:

1. Actual arguments are substituted for all dummy arguments.
2. The expression is evaluated, and the result is converted to numeric form according to some specified format and substituted for the entire expression.

2.2.1.5 Conditional expansion: A conditional expansion control statement causes the Expander to process a specified section of the skeleton file if and only if a prescribed logical condition is true; if the condition is false, that section of the file is skipped without being processed. Typical conditions might be whether or not a particular argument is blank, a given expression evaluation is negative, or one dummy argument is equal to another. Logical connectives can be used to combine two or more conditions; e.g., "Expand the following section if and only if this argument is four or more characters long and that expression is greater than ten." Conditional expansion sections can be nested.

One obvious use for conditional expansion is in generating different expansion versions from the same skeleton file. For instance, a TCSS could be set up to include debugging statements in the TCS file if the user sets a dummy argument named DEBUG nonzero. Other uses include error checking (conditionally generating an error message if the user's input is out of range) and default values (conditional generation of a preset parameter statement if the user inputs a blank).

2.2.1.6 Multiple expansion: Multiple expansion provides a built-in looping capability. These control statements cause a given section of the skeleton to be expanded repetitively, either a specified number of times or until a specified condition is met. The output generated may be only a series of identical copies of the original, or it may be different for every repetition. Conditional expansion and redefinable parameters can change the results on each iteration. Multiple-expansion sections can also be nested.

2.2.1.7 Macros: The macro facility makes it possible to define a commonly used block of image text and/or control statements once, and then call it out for use any number of places within the skeleton file. In many respects a macro definition resembles a small skeleton file; however, the actual arguments for a macro's dummy arguments are supplied by the call statement, rather than by initial user input. Macros can make use of all control statements, including macro calls (nesting of calls) and macro definitions (nesting of definitions). A macro can even call itself (recursive

nesting), provided some means such as conditional expansion statement testing a counter is used to prevent a loop.

2.2.1.8 Concordance mode: This mode is used to obtain a concordance listing of the skeleton file; it has no effect on the expansion file produced. A concordance listing is an alphabetical listing of all the dummy argument names and macro names used in the skeleton, together with a list of all references to each name. A listing of this type is useful for debugging a new skeleton file, and invaluable for modifying an existing one.

2.2.2 The TCSS Writer. — The TCSS Writer is a utility that prepares the preliminary TCSS information for processing by the TCSS Expander. The relationship between the TCSS Writer and other components of the system is depicted in Figure 2-4.

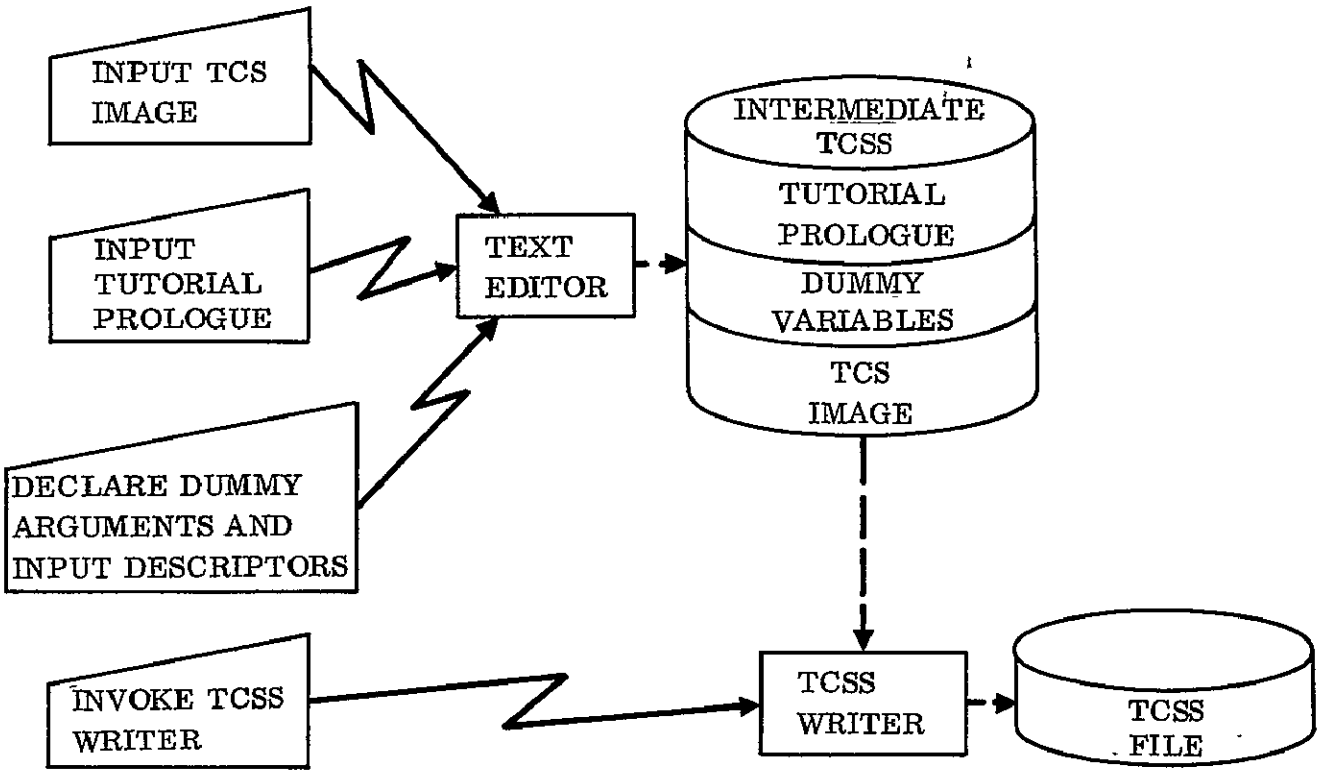


Figure 2-4. TCSS Writer Operation

A user employs the Text Editor to generate an intermediate TCSS file. In the process he incorporates into the intermediate file the following types of information (with identifiers for the TCSS Writer):

1. TCS image — The user specifies the body of text that constitutes the TCS image that will be processed by the Expander.

2. Dummy argument — For each dummy argument, the user specifies the character string within the TCS image that he wants treated as a dummy argument and a suitable chosen name for the dummy argument. Optionally the specified character string can constitute the name of the dummy argument. The user also provides the description for the dummy argument that is to be incorporated into the TCSS file.

3. Tutorial prologue — The user identifies and generates the character string that will constitute the tutorial prologue in the resultant TCS skeleton file. Care must be taken to include sufficient detail to prevent a user from initiating the expansion process prematurely. The user then invokes the TCSS Writer to obtain the TCSS file for the TCSS Expander.

In processing the intermediate TCSS file by the TCSS Writer:

1. When a tutorial prologue is encountered, it is transferred to the TCSS file.
2. When a dummy argument input is encountered:
   a. The name and description are transferred.
   b. The TCS image is scanned and the positions where the dummy character string occurs in the image text are recorded.
   c. Control statements and position information are then made part of dummy argument data and output to the TCSS File.
3. The TCS image is made part of the TCSS file. The TCSS Writer thus forms a bridge between:
   a. The user and his creation of TCSSs and
   b. The system form required for efficient processing.

2.2.3 TCS Interceptor. - The TCS Interceptor acquires the TCS employed by the user via the terminal and constructs a TCS file. This enables the user to maintain a record of his TCSs in sequence.

Figure 2-5 depicts the operational relationship between the TCS Interceptor and the IPAD EXEC:

1. In the first instance, the user must issue a command to the EXEC to activate the TCS Interceptor.

2. With the TCS Interceptor activated (lower part of the figure), the following activity occurs:

   a. The TCS is issued by the user to the EXEC.

   b. The EXEC carries out/controls the activity designated by the TCS.

   c. Since the TCS Interceptor is activated, the TCS image is passed through it to be recorded within a file via the IPAD data base management system.

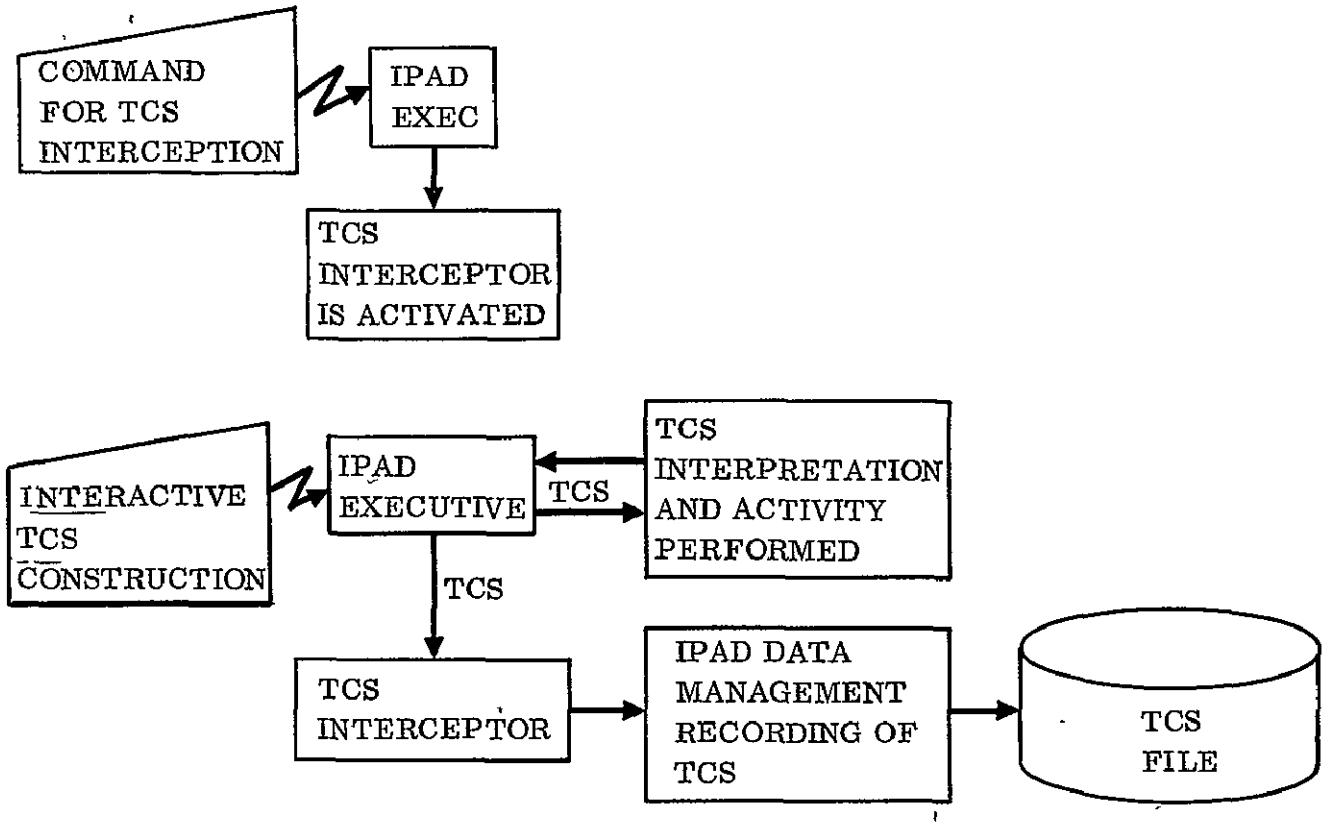


Figure 2-5. TCS Interceptor/EXEC Operation

The user, during his interactive session, can turn off the Interceptor, recording that portion of the TCS he desires. He now has a TCS image he can execute directly or modify as required.

2.3 User's Task Trajectory (UTT) Generation

In addition to the preceding subsidiary functions, which are explicitly invoked or used by the user for assistance or efficiency in performing his task, the IPAD system has a function associated with the EXEC that performs the automatic recording of the sequence of operations in which the user has actually engaged. Figure 2-6 shows the relationships between the User's Task Trajectory (UTT) generation and the IPAD EXEC operation:

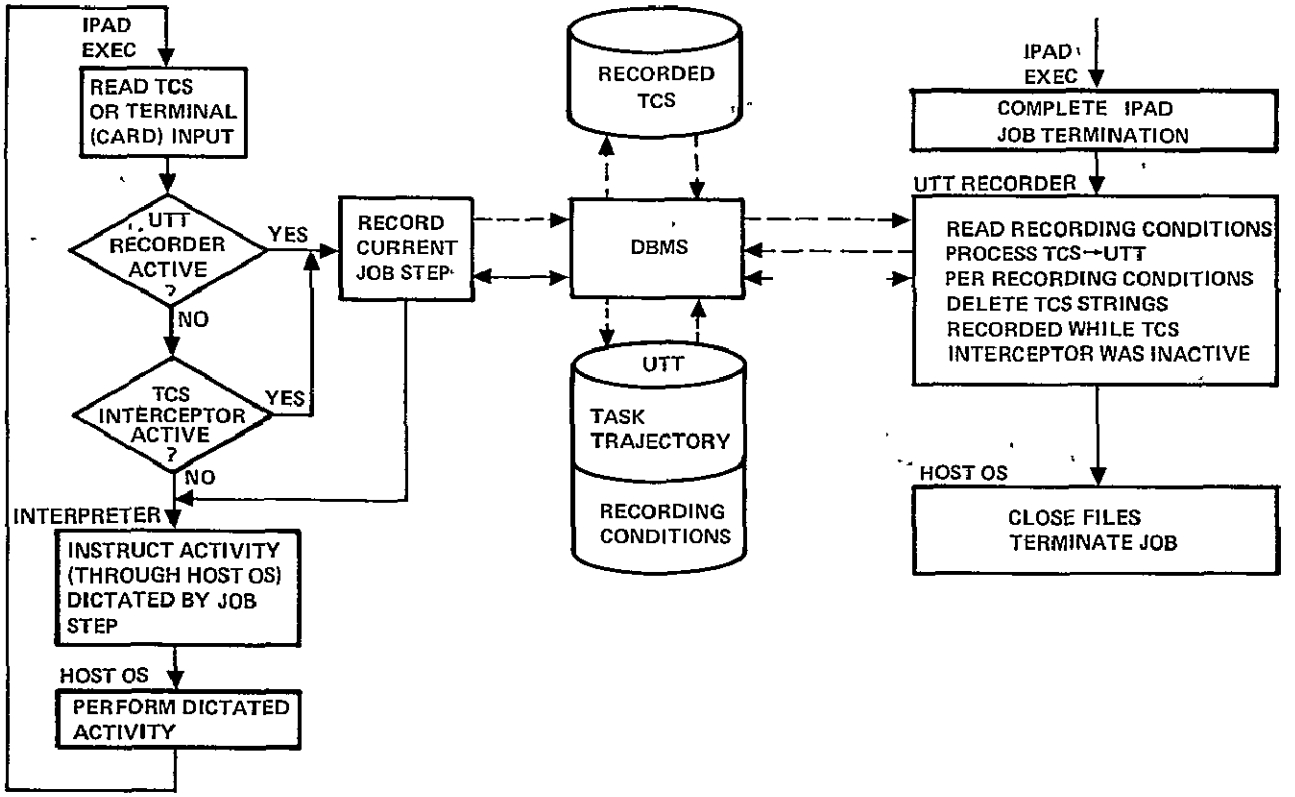


Figure 2-6. User Task Trajectory (UTT) Generation

1. The actual activation of the mechanism is determined, a priori, when a User File (UF) is initialized into IPAD and needs for the UTT have been determined by the project management. If this determination has not been made, the mechanism remains inactive.

2. In the operation of the EXEC, as each command is passed through the TCS Interpreter portion of EXEC:

   a. The TCS data is collected according to the recording conditions. In the recording conditions, the type of information to collect will be specified. Examples:

      - Complete recording of the TCSs will be made.
      - Only the OM-related portion of the TCS will be recorded; e.g., OM used, data version used, etc.

   b. Additional data is added, such as time of activity.

   c. The data is rewritten onto the User Task Trajectory (UTT) file.

Sections 4.1 and 4.2 provide further detail of the requirements and uses of the UTT.

2.4 EXEC Functions

2.4.1 EXEC functions, control and data flow. - Most of the command and data flow involved in running an IPAD task issues directly from the TCS file (illustrated in Subsection 2.4.3). In addition to these directly-TCS-driven functions, the EXEC itself performs the following functions:

1. Accepts and interprets user commands; e.g., requests to run a particular task or to start the TCS Interceptor.
2. Maintains a directory of IPAD task names and the corresponding TCS or TCSS file names and passwords. This makes it possible for a user to run a task file without having to know its passwords or even its name. The actual storage of the files is controlled by the data management system. The EXEC sets up the equivalence between the IPAD task name and the data base name, which may refer to files local to the user, common to a project, or common to all users of the IPAD system.
3. Activates and deactivates TCS files.
4. Directly (not via a TCS) controls the execution of certain utilities like the TCSS Expander.
5. It can (if directed) maintain trajectories of tasks which the users have run (UTTs).

Flow is also effected by the loop-controlling and condition-testing utility routines, which reposition the TCS. These utilities will interface with the data management system to obtain the data on which to base their decisions. For example, a typical request might be, "If the value of word 7 of record 1 of file INTRMED is less than 1.0005, reposition the TCS to label LASTCASE."

2.4.2 EXEC functions, system support. - The IPAD EXEC, per se, is essentially limited to the above functions. For the IPAD EXEC to operate within a host facility, the host facility operating system must provide the following features.

1. Time-sharing capability (such as CDC's INTERCOM 4.1) with which EXEC will interface to obtain host facilities.
2. Loaders for OMs and utilities.

2.4.3 Example of an IPAD task. - The following example (illustrated in Figures 2-7 and 2-8) does not necessarily purport to be a typical IPAD run; it is intended merely to illustrate most of the system features. In this example two OMs will be run in sequence with output from the first serving as input to the second.

2.4.3.1 First OM: This OM will be placed in execution and run without the intervention of the user (Figure 2-7):

1. The user logs in.
2. The time-sharing monitor accepts a user command, telling it to load the IPAD EXEC.
3. The IPAD EXEC accepts a user command, telling it to run a particular task.
4. The EXEC locates the corresponding file for the task (i.e., permanent file name) via the IPAD data management system, which is explicitly shown in the figures but is implicit in the following discussion for other operations involving the IPAD data file.
5. The EXEC discovers that the file is a TCSS, not a TCS. It calls the TCSS Expander utility.
6. The Expander provides the tutorial prologue to the user, which describes the TCSS function for verification. It then reads the data names of the TCSS and determines what inputs are needed to build a TCS (reference Figure 2-3).
7. The Expander asks for an input data item, by name.
8. The user types in a data item.
9. Steps 7 and 8 are repeated until the user encounters a name he does not recognize. He asks the Expander for help.
10. The Expander prints out a writeup, provided initially by the person who wrote the TCSS, explaining what is needed.
11. The user, having understood the writeup, types the data.
12. Steps 7 and 8 are repeated until all data items have been entered.
13. The Expander reads the rest of the TCSS and converts it to a TCS using the inputs supplied by the user.
14. The Expander returns to the EXEC.
15. The EXEC activates the TCS file.
16. The TCS attaches two data files.
17. The TCS loads an Operational Module (OM) and puts it into execution.

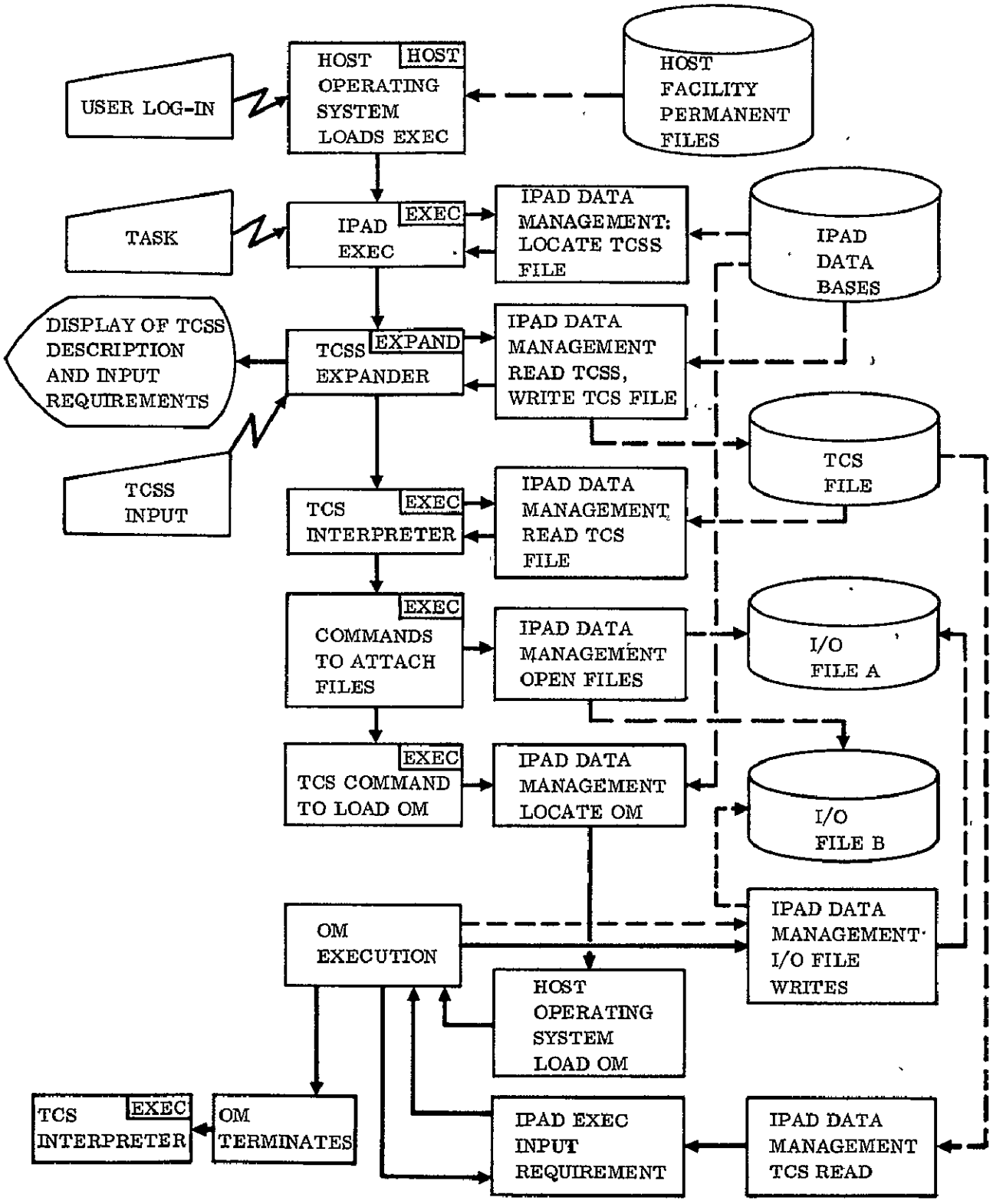


Figure 2-7. TCS Multi-OM Execution Example — First OM

18. The OM requests several lines of terminal input. The requests are intercepted and supplied with input from the TCS.

19. The OM terminates.

2.4.3.2 Second OM: The second OM requires direct input by the user during the course of operation (Figure 2-8).

20. The TCS loads the condition-testing utility and puts it into execution.

21. The utility determines that the specified condition has been met and accordingly spaces down the TCS until it comes to a specified label, bypassing several intermediate steps on the TCS.

22. The utility terminates.

23. The TCS, repositioned but still active, loads the second OM and puts it into execution.

24. The OM requests a few lines of input. These requests are intercepted and supplied with input from the TCS.

25. The OM requests more input. These requests are also intercepted but encounter a direct input flag on the TCS, and so are passed on to the terminal.

26. The user types in the required input.

27. The OM terminates.

28. The TCS returns to the IPAD EXEC.

29. The EXEC deactivates the TCS file.

30. The EXEC accepts a user command that tells EXEC the user has completed his job.

31. The EXEC returns to the timesharing monitor.

32. The user logs off.

2.5 IPAD EXEC — Implementation Considerations and Trades

Three families of computing systems, one of which has two distinct sets of software, were considered in the analysis of IPAD EXEC implementation, namely:

1. CDC Family, CDC CYBER 70 or 6000 series with SCOPE 3.4, INTERCOM 4.1, and GPGT/IGS.

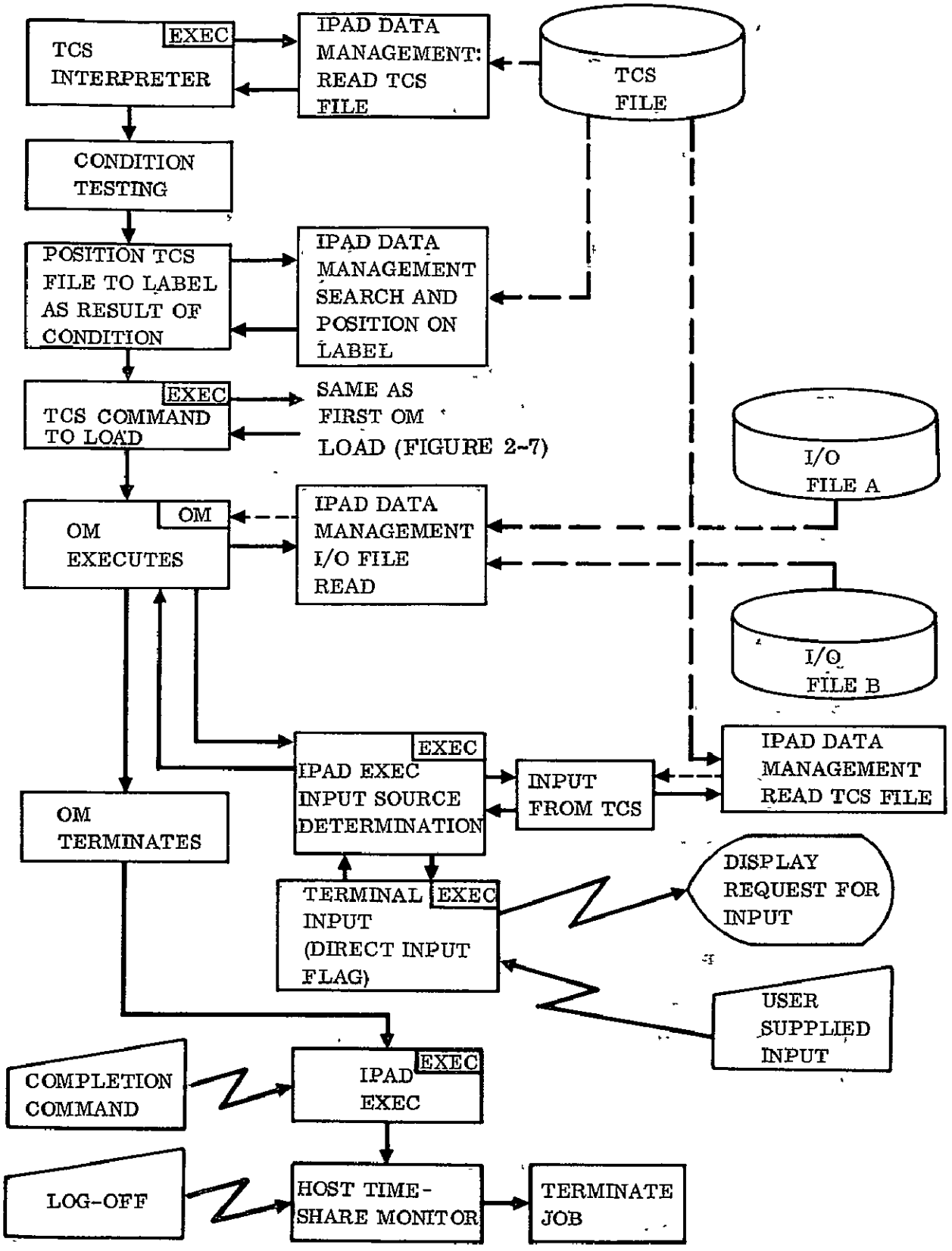


Figure 2-8. TCS Multi-OM Execution — Second OM

2. IBM Family

   a. IBM 360/67 with CP-67.

   b. IBM 370/145, 158, or 168 with VM/370.

3. UNIVAC Family, 1106, 1108, or 1110 with EXEC 8.

These families and the extent of required modifications are described in the following subsections for use with IPAD. Table 2-1 summarizes the capabilities of these respective systems.

TABLE 2-1. OPERATING SYSTEM CAPABILITIES OF INTEREST TO THE IPAD EXEC

Capability	CDC Cyber 70 or 6000 Series with SCOPE 3.4, INTERCOM 4.1 and ... IGS V.1	... GPGT/IGS (Expected Fall 1973)	UNIVAC 1106, 1108 or 1110 with EXEC 8	IBM 360/67 with CP-67, IBM 370/145, 158 or 168 with VM/370
Interactive graphics (level 3, 4, or 5 CRT terminal)	Yes (to level 5)	Yes (to level 5)	Yes (to level 5)	Yes (to level 5)
Conversational interactive time-sharing	Yes	Yes	Yes	Yes
Time-sharing capability from a graphics (level 3, 4, or 5) terminal	No	Yes	No	No
Batch job spinoff from a time-sharing job	Yes	Yes	Yes	Yes
Commands files	No	No	Yes	Yes
Subsidiary processes	No	No	Yes	Yes
Programmed request to load another program, without terminating the requesting program	Yes	Yes	Yes	Yes
Programmed request to load an overlay	Yes	Yes	Yes	Yes

2.5.1 IBM and UNIVAC. - EXEC 8 on the UNIVAC 1106, 1108, or 1110; CP-67 on the IBM 360/67; and VM/370 on the IBM 370/145, 158, or 168 are all able to support an IPAD EXEC without modification. (Some system modifications are needed for other reasons, but not for the EXEC.) All these systems provide for the creation of multiple subsidiary processes, where each process amounts to a complete virtual machine with its own address space, registers, etc., much like the machine each user gets when he logs in. These processes may be interconnected in various ways, or they may be almost completely independent, except for the restriction that when one terminates, any other process it has created must also terminate. Thus the IPAD EXEC can be deactivated

and even swapped out (but not terminated), until the job step does something that the EXEC has designated as an activation event (for instance; a request for the terminal I/O). At this point, the EXEC will be reactivated and given a flag telling what has happened in its subsidiary process. The EXEC can then read the TCS file to determine appropriate action.

IBM and UNIVAC share a significant common deficiency which is the inability to perform normal graphics timesharing tasks. The graphics systems are separate from the timesharing systems, which precludes interactive editing, compilation, loading, and execution from a graphics CRT. (Although this problem also currently exists in the CDC family, the GPGT/IGS package will correct this deficiency.)

To solve the problem on the IBM and UNIVAC, modified communication drivers must be added to the timesharing systems. Since the graphics packages contain similar drivers, modifications should not be too difficult.

2.5.2 The CDC CYBER 70 and 6000 Series. - This family of machines is a major constraining factor for the intended EXEC design because, as is, it is insufficient to fully support the EXEC as envisioned. INTERCOM must be modified to provide enough facilities to implement a complete EXEC system. The approach selected is to extend INTERCOM so it can accept command files directly as do the other systems (Table 2-1). Once the capability to activate and deactivate TCS files under program control is made available, creation of the EXEC will be a fairly straightforward programming task. (Much of it could probably even be written in FORTRAN.)

2.5.3 Implementation of modifications. - The following considerations show the advantages of having all the systems modified by the manufacturers:

1. Since, in most cases, the host operating system code is complex and sparsely documented, it can be most efficiently modified by the group that originally produced it.
2. Compatibility with current and planned versions of the system is assured.
3. There would be less reluctance by the manufacturer to support the modifications.

2.6 Alternate EXEC Design Approaches

Two alternate approaches were considered for the EXEC design. The first was to build the IPAD EXEC directly into the timesharing subsystem, and the second was to design all programs to always return to the EXEC.

2.6.1 Add EXEC to the time-sharing subsystem. - To build the IPAD EXEC directly into the timesharing subsystem, e.g., INTERCOM, would, in effect extend INTERCOM's command language to include such functions as "Run Task." The TCS would then consist of a table built into the EXEC instead of a file. This method has the advantage of high power and efficiency, but the large amount of recording needed increases cost and risk. Flexibility is also poor, because any modification to the EXEC is necessarily a modification to INTERCOM, which is in system code and not amenable to quick changes as is a TCS file. Finally, the sheer bulk of the code added to INTERCOM could impact non-IPAD users by cutting into the total available field length.

2.6.2 Return to separate EXEC. - To design all new programs and modify all OMs so that they always return to the EXEC upon completion was immediately abandoned because of several major deficiencies:

1. A great deal of reprogramming would be required (although none of it would be to the timesharing subsystem).
2. The IPAD system would be inflexible and difficult to modify.
3. The EXEC would not, in general, be able to recover control if a program terminated abnormally instead of normally.

Also, this method is essentially the hard-wired system previously ruled out in the IPAD conceptual design (see Volume IV, Part 1, Subsection 2.2.3.1).

In view of the deficiency of the two approaches discussed above, the use of TCS files emerges as the best approach. Sufficient power will be realized and efficiency will be high, unless the system is misused. Efficiency will drop catastrophically if a TCS is used to step through a large number of very small (atomic level) processes. Great flexibility is allowed, both in terms of what can be done via TCS files and in terms of the ease of building and modifying them; yet good protection against unauthorized modification is provided by the operating system's password mechanism.

There will be an increase in system overhead for non-IPAD users, but it should be quite small. To compensate for this factor, however, there are some advantages available to non-IPAD users: handling TCS files will be a function of the timesharing subsystem, not of IPAD itself; and while other users may have to call the files by a different name, these users may well find them useful. With some extra work, even the TCSS expansion facility could be made available to outside users.

3 INPUT/OUTPUT FORMATTING

A major objective of IPAD is to provide a framework within which the IPAD user can exercise individual OMs and sequences of OMs. The IPAD system is to include software to solve the interface problems between OMs. That is, the user need only concern himself with the engineering problem and its interface with other problems. The IPAD system relieves him of the need to:

1. Perform the clerical labor associated with interfacing OMs with each other and with project data.
2. Know and implement the details of data format and data storage.

The interface problem could be solved by any one of four basic approaches (or mixtures and variations of these approaches):

1. Data storage standards could be specified and all OMs written (or rewritten) to conform to these standards.
2. For each use of each OM, a special formatting program could be written to produce input as required by that OM.
3. The formatting program (above) could be generalized, providing one general purpose utility to produce input for any use of any OM.
4. A highly sophisticated formatter utility could be provided, all I/O requests to be routed through the utility; the utility to determine the OM's intent and satisfy that intent.

The first two approaches were rejected on the grounds that they conflict with the operating philosophy and intent of IPAD (see Volume IV Part I, Sections 1 and 2). The third approach, that of a general purpose Input/Output Formatter (IOF) was determined to be substantially simpler than the fourth approach and was adopted as a basis for a preliminary design. Section 3.1 and 3.2 present the guidelines and requirements of the IOF; Section 3.3 presents the IOF's preliminary design.

During a Preliminary Design Review - after the design concepts had been firmly established - it was uncovered that the developing concepts meshed with a development effort on the part of the business data processing community (represented by CODASYL), basically using the fourth approach. Section 3.4 summarizes this development; Section 3.5 shows the relationship between the two efforts. The remaining sections illustrate why the IOF utility was dropped in favor of CODASYL's Data Base Task Group's (DBTG's) recommendations.

3.1 Constraints and Guidelines for Developing the IOF

A major IPAD design objective is to enable the user to incorporate existing OMs into IPAD essentially without changes, or with minor ones. Another objective is to include human-factor features in the design of IPAD to facilitate user tasks. Both objectives impose a number of constraints on the design of the Input/Output Formatter (IOF) support function.

1. No procedural requirements shall be imposed upon the user:
   a. The user is to define his own I/O procedures by actually manipulating data.
   b. Once defined, the user can reiterate the procedure without redefining it.
   c. Corollary: random access/update of any data associated with the OM's I/O is to be permitted.
2. Incorporation of an OM into IPAD shall not require extensive modification of the OM. In particular, no changes to program logic shall be required:
   a. Input is to be provided in the form required by the existing OM.
   b. Output as written by the OM is to be processable.
3. OM input may be derived via the IOF from a variety of sources where the format and organization do not coincide with that of the OM, such as:
   a. The safeguarded, blessed project data base.
   b. Local User Files (UFs).
   c. Other OM output.
   d. User (keyboard) input.
4. OM output may be routed to a variety of destinations where the format and organization do not match that of the OM, such as:
   a. Data base update file (using an update procedure).
   b. User local files.
   c. Other OM input.
   d. Displays.
5. The user is to be provided with tutorial assistance to support:
   a. I/O requirements of OMs.
   b. Command options available.
   c. Error diagnostics and prompting.

6. The IOF must operate in both an interactive and batch mode.

3.2 IOF Requirements

The constraints and guidelines listed above and objectives relating to the user give rise to the following set of requirements:

1. Provide the user with random access/update capability. This requires random access files.
2. Prohibit OM program logic changes. This requires that the random access files be intermediate duplications of the OM files.
3. Provide tutorial assistance. This requires that certain data be generated by a person responsible for the OM, viz:
   a. Glossary of data names and definitions.
   b. Data names linked with engineering units.
   c. Coordinate systems for selected data names.
   d. User/program options, purpose and tutorial aids.
   e. I/O data impact of each option.
   f. Default/assumed values of inputs.
4. Incorporate OMs without restrictions on their use or interface. This requires that the person responsible for the OM provide a definition of the interface (I/O) requirements, independent of the use of the OM:
   a. Definition language must be consistent and applicable to all OMs.
   b. Language must specify
      - File attributes
      - Technique for identifying data "records"
      - Structure of data within "records"
        - Sequence relationship of variables
        - Characteristics of variables (type, etc.)
        - Formats (padding, etc.)
        - Dimensions.
5. The definition of the interface requirements must be translatable or compilable such that a compilation results in a directory or directories of files involved.
6. The functional requirements of an IOF utility must include:

a. Creating an IOF (intermediate) file given the compiled directory.

b. Updating records in an IOF file:

- Inserting values.
- Modifying values.

c. Extracting values from an IOF file.

d. Mapping an IOF file onto an OM (input) file.

e. Mapping an OM (output) file into an IOF file.

f. Desk-calculator level computations on data within an IOF file.

g. Extracting data from the IPAD data base, user local files, etc. and inserting this into an IOF file.

h. Creating or adding to the data-base-update file, user local file, etc.

i. Accessing tutorial data associated with an OM and displaying this to the user.

7. A user-oriented command language (and interpretive processes) to accomplish the above functions.

3.3 IOF Preliminary Design*

This section develops the concepts of:

1. The purpose of the IOF utility.
2. Information needed by the IOF utility and methods of producing that information.
3. OM setup, i.e. processing the support information of 2. above.
4. The functional flow of input/output formatting.

The basic functional requirements of the IOF are:

1. To construct input files for OMs and utilities.
2. To read output files generated by OMs and utilities.

The requirements are to be accomplished through the use of the following intermediate IOF files:

1. ISPONGE - a random access direct representation of an input file.
2. OSPONGE - a random access direct representation of that portion of an output file that the user wishes to access.

*It is suggested that the reader re-read Subsection 2.3.4 of Volume IV, Part I before reading this section.

These intermediate files are constructed by the IOF from Input DEFinitions (IDEFs) and Output DEFinitions (ODEFs) and from the user's response to options presented to him by the IOF. The user's response may be either actual terminal input or pre-recorded terminal input from a Task Control Sequence (TCS).

Data contents of input files are to be taken from:

1. User input.
2. The safeguarded, blessed project data base.
3. Local User Files (UFs).
4. OM output files.

3.3.1 Requirements of the IDEF and ODEF. - The I/ODEF is divided into two main parts:

1. User oriented data ignored by the IOF in I/O file processing, to be displayed at the user's option to assist the user.
2. IOF oriented data usually ignored by the user except where decisions are necessary. The IOF prompts the decision by informing the user of options available.

3.3.1.1 User oriented data: User oriented data is user reference material having no functional use to the IOF.

3.3.1.2 IOF oriented data: The basic requirement of IOF oriented data is to provide for each data name the corresponding location or locations within the file. The approach taken to meet this requirement is to adopt a proven, consistent method of defining data format and data variables without specifying procedures to produce the file.

The COBOL language provides a means of defining small substructures within the file (e.g. one logical record, its contents and format) and grouping these in outline or hierarchical form. An extension of this grouping defines the entire file as a tree structure of data groups. This extension is completed by specifying the following at each branch of the structure where it applies:

1. Name of the data or data group.
2. Index or repetition specifications for the substructure (further branches). This includes vector and matrix dimensioning.
3. Identification of invariant or padding data. These fields may be used as structures, or variable definitions when reading output files.
4. Values of data for constants, or for default or assumed values.

5. Option identification used to allow the user to delete a branch and any substructure (IDEF especially).

6. Conversion algorithm to be used in external/internal conversions.

7. Redefinition of a branch at the same level when branches are mutually exclusive.

8. Specification of fields which may be used as structure or data identification (ODEF especially).

Figures 3-1, 3-2, and 3-3 illustrate the transition from a human oriented overview of an input file to a formalized specification of its structure usable by the IOF.

Figure 3-1 illustrates a file composed of a particular sequence of major sections where some sections have a detailed subsequence.

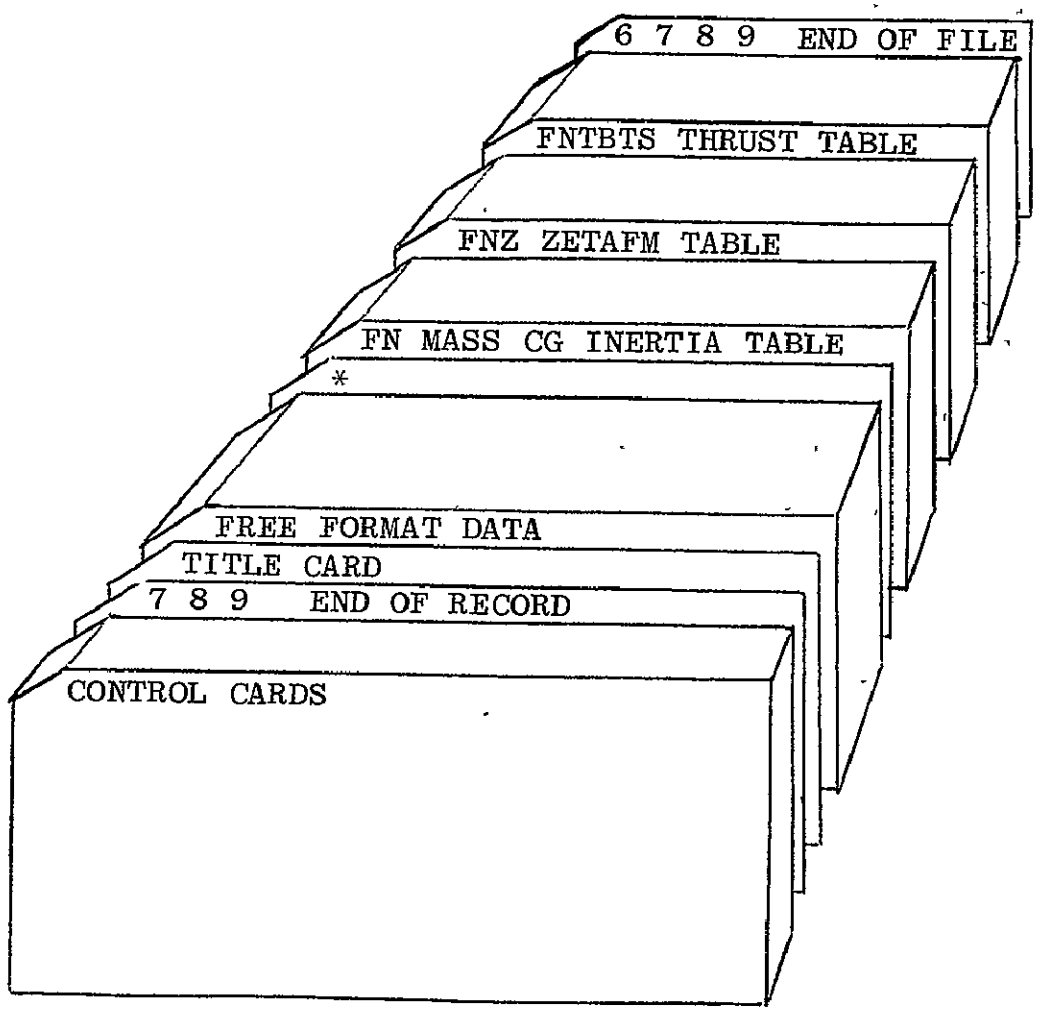


Figure 3-1. Typical Input Deck (Source)

Figure 3-2 is an example of a more abstract way of conveying the same information a tree structure composed of a particular sequence of major branches. The substructure of the last major section is shown here, as defined by program documentation (Reference 1).

Figure 3-3 shows that the same structure can be represented in a formalized tabulation, as needed in the IDEF. This figure does not attempt to show detail specifications required in the IDEF but only an overall structure.

From Reference 2, figures 3-4, 3-5, and 3-6 illustrate the transition from a sample output page (Figure 3-4) to a formalized tabular description of the page (Figure 3-6). (The figures do not give an accurate count of blank lines and spaces but are intended to indicate that such items are accounted for in the tabulation).

Figure 3-5 indicates the similarity between viewing the output page and computer scanning (reading from left to right):

1. The page is separated into two parts called HEADING and OUTPUT.
2. A particular character string within the HEADING (indicated as HEADING ID) is sufficient to identify the page format shown on Figure 3-4.
3. A particular character string (OUTPUT ID) at a specific place within the page corresponds to a substructure definition. This is not important in the example but allows for variations in the kind of data produced under one heading.
4. Once the page is identified by ID fields, a six-line template is provided to associated data names with values.

Figure 3-6, like Figure 3-3, is simply a formalized tabulation of the hierarchical (tree) structure in Figure 3-5.

3.3.2 Use of the IDEF and ODEF. - The IOF, in an interactive mode, converts an I/ODEF into an IOF file, a portion of which is a directory (see Figure 3-7). Thus the IDEF becomes a directory to a random access version of a sequential input file. The ODEF becomes a directory to a random access version of a sequential output file (see Figure 3-8). In processing an IDEF, the IOF gives the user the name of each program/user option when a branch definition specifies that options exist. The user chooses the option, to include the branch. In processing an ODEF, the IOF gives the user the option of skipping any data or groups of data in the output file which are of no interest to his particular task. The output directory will contain no reference to these, consequently they will be by-passed.

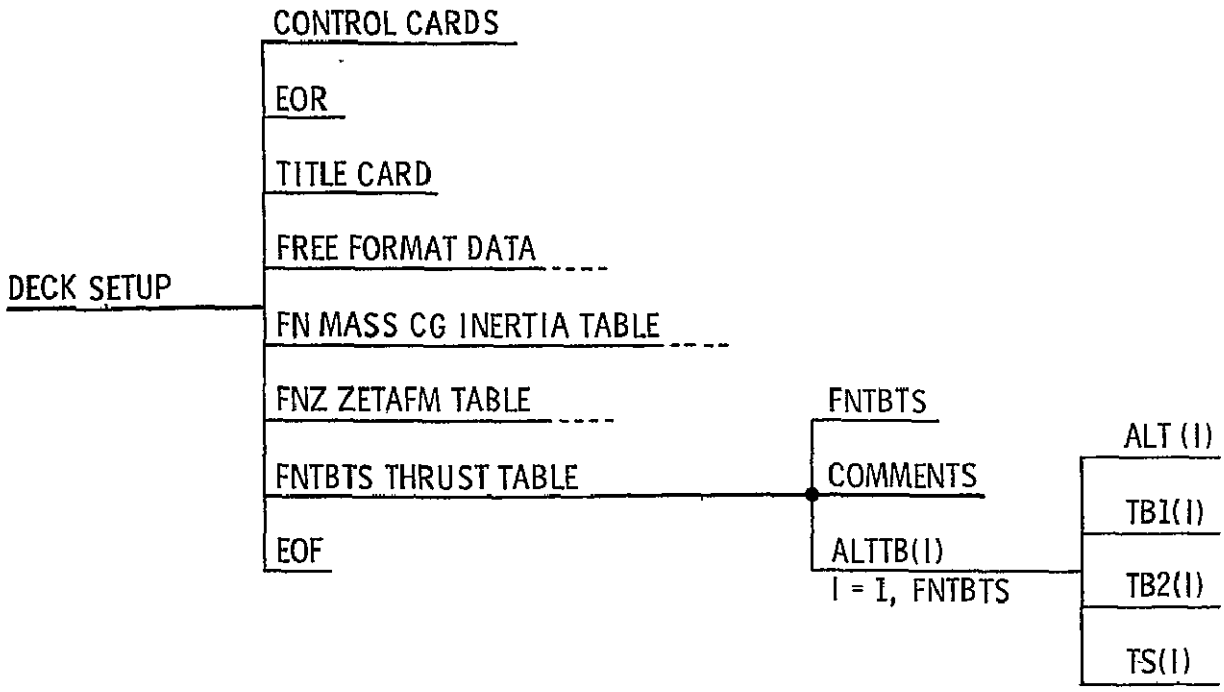


Figure 3-2. Input Source → Input Tree

```
01   DECK - SETUP.
     02   CONTROL-CARDS.
          03 . . .
     02   'EOR'.
     02   TITLE-CARD.
     02   FREE-FORMAT-DATA.
          03 . . .
     02   FN-MASS-CG-INERTIA-TABLE.
          03 . . .
     02   FNZ-ZETAFM-TABLE.
          03 . . .
     02   FNTBTS-THRUST-TABLE.
          03   FNTBTS.
          03   COMMENTS.
          03   ALTTB OCCURS FNTBTS TIMES.
               04   ALT.
               04   TB1.
               04   TB2.
               04   TS.
     02   'EOF'.
```

BASICALLY COBOL EXCEPT

- COBOL TREE DEFINES RECORDS, IDEF TREE DEFINES A FULL FILE
- ADDITIONAL CLAUSES (NOT SHOWN) TO DEFINE PROGRAM/USER OPTIONS
- DATA ID FIELDS WITHIN FORMAT MASKS

Figure 3-3. Input Tree → IDEF

OUTPUT FOR LINE* 2 PANELS

AT CORNER I, U-PRIME IS LOAD IN DIRECTION FROM I TO I+1, V-PRIME IS IN DIRECTION FROM I-1 TO I

STRESSES ARE SHEAR FLOWS / PANEL THICKNESS

PANEL	CORNERS	U-PRIME	V-PRIME	SIDE		LENGTH	SHEAR FLOW	STRESS
			LOADING CONDITION* 1					
1*								
	4	1.39160E+03	-1.19853E+03	4	16	10.000	3.48623E+02	2.63434E+03
	16	-1.49849E+03	2.09463E+03	16	15	20.706	-4.35779E+02	-3.54292E+03
	15	2.09459E+03	-7.52466E+03	15	3	10.000	3.48623E+02	2.63434E+03
	3	-6.01999E+03	1.39157E+03	3	4	25.882	-2.78899E+02	-2.26747E+03
2								
	5	2.10922E+03	-6.84149E+03	5	17	10.000	5.25925E+02	2.69982E+03
	17	-6.55137E+03	3.14992E+03	17	16	20.706	-6.57406E+02	-3.37477E+03
	16	3.14999E+03	-5.06074E+03	16	4	10.000	5.25925E+02	2.69982E+03
	4	-4.04826E+03	2.10926E+03	4	5	25.882	-4.20743E+02	-2.15936E+03
3								
	10	2.03183E+03	-3.90549E+03	10	22	10.000	5.25263E+02	2.14568E+03
	22	-4.88235E+03	3.22079E+03	22	21	20.706	-6.56579E+02	-2.68214E+03
	21	3.22072E+03	-8.71262E+03	21	9	10.000	5.25263E+02	2.14566E+03
	9	-6.97049E+03	2.03179E+03	9	10	25.882	-4.20210E+02	-1.71655E+03
4								
	6	-3.68773E+01	-5.01659E+03	6	18	10.000	1.60073E+01	9.16797E+01
	18	-6.27323E+03	1.96947E+02	18	17	20.706	-2.00091E+01	-1.14600E+02
	17	1.96947E+02	5.85896E+03	17	5	10.000	1.60073E+01	9.16797E+01
	5	4.68717E+03	-3.68773E+01	5	6	25.880	-1.28058E+01	-7.33438E+01
5								
	7	-1.42125E+03	1.32632E+03	7	19	10.000	-3.66633E+02	-2.36385E+03
	19	1.65824E+03	-2.24508E+03	19	18	20.706	4.58291E+02	2.954d1E+03
	18	-2.24503E+03	7.83103E+03	18	6	10.000	-3.66633E+02	-2.36385E+03
	6	6.26510E+03	-1.42122E+03	6	7	25.882	2.93306E+02	1.89108E+03
6								
	8	-1.48844E+03	6.03453E+03	8	20	10.000	-3.66728E+02	-2.64285E+03
	20	7.54281E+03	-2.17876E+03	20	19	20.706	4.58410E+02	3.55357E+03
	19	-2.17881E+03	1.94893E+03	19	7	10.000	-3.66726E+02	-2.64285E+03
	7	1.55887E+03	-1.48847E+03	7	8	25.882	2.93383E+02	2.27428E+03
7								
	9	3.17346E+01	4.60407E+03	9	21	10.000	1.54601E+01	8.86472E+01
	21	5.75509E+03	1.22863E+02	21	20	20.706	-1.93251E+01	-1.10809E+02
	20	1.22863E+02	-6.15520E+03	20	8	10.000	1.54601E+01	8.86472E+01
	8	-4.92416E+03	3.17346E+01	8	9	25.880	-1.23681E+01	-7.09178E+01

* IDENTIFICATION FIELDS FOR MASKING

Figure 3-4. Typical Output Listing (Source)

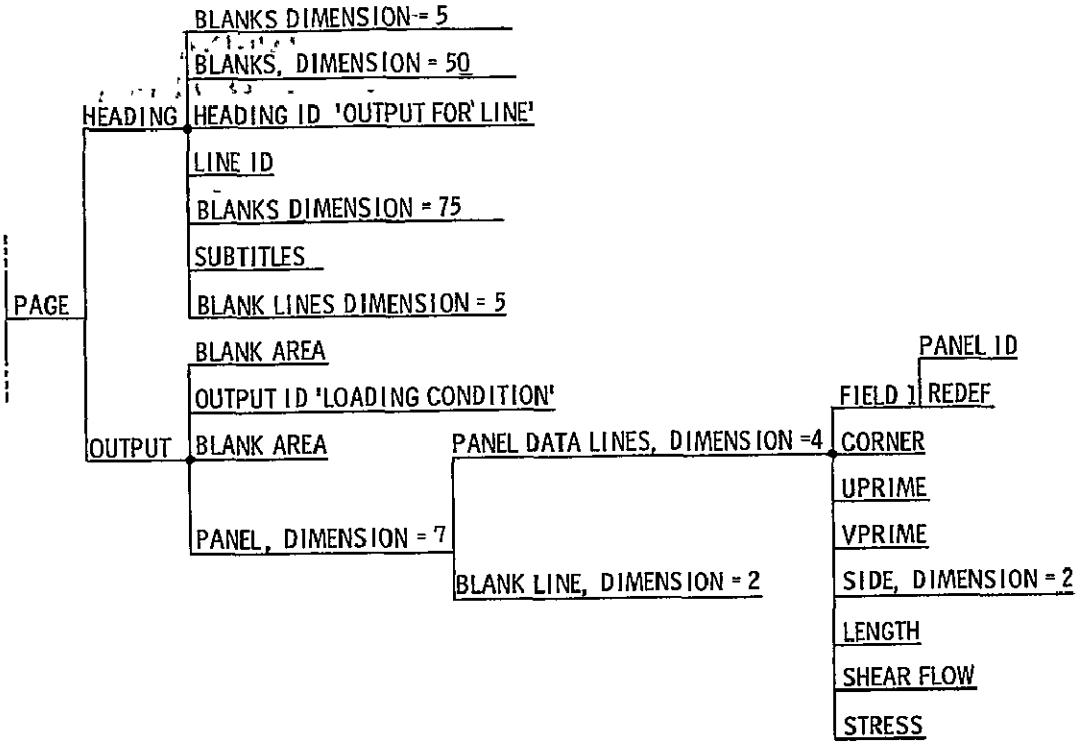


Figure 3-5. Output Source → Output Tree

```
06   PAGE
     07   HEADING
          08   BLANK-LINES OCCURS 5 TIMES.
          08   HEADING - ID.
               09  FILLER PICTURE X (60) VALUE SPACES.
               09  ID PICTURE X VALUE 'OUTPUT FOR LINE'.
               09  FILLER PICTURE X (58) VALUE SPACES.
          08   LINE-ID . . .
          08   BLANKS . . .
          08   SUBTITLES . . .
          08   BLANK-LINES-1 OCCURS 5 TIMES.
     07   OUTPUT
          08   BLANK-AREA . . .
          08   OUTPUT-ID . . .
          08   BLANK-AREA-1 . . .
          08   PANEL OCCURS 5 TIMES.
               09   PANEL - DATA - LINES OCCURS 4 TIMES.
                    10   FIELD 1.
                         11   PANEL - ID . . .
                         11   FILLER REDEFINES PANEL - ID VALUE SPACE . . .
                    10   CORNER . . .
                    10   UPRIME . . .
                    10   VPRIME . . .
                    10   SIDE OCCURS 2 TIMES . . .
                    10   LENGTH . . .
                    10   SHEAR FLOW . . .
                    10   STRESS . . .
```

Figure 3-6. Output Tree → ODEF

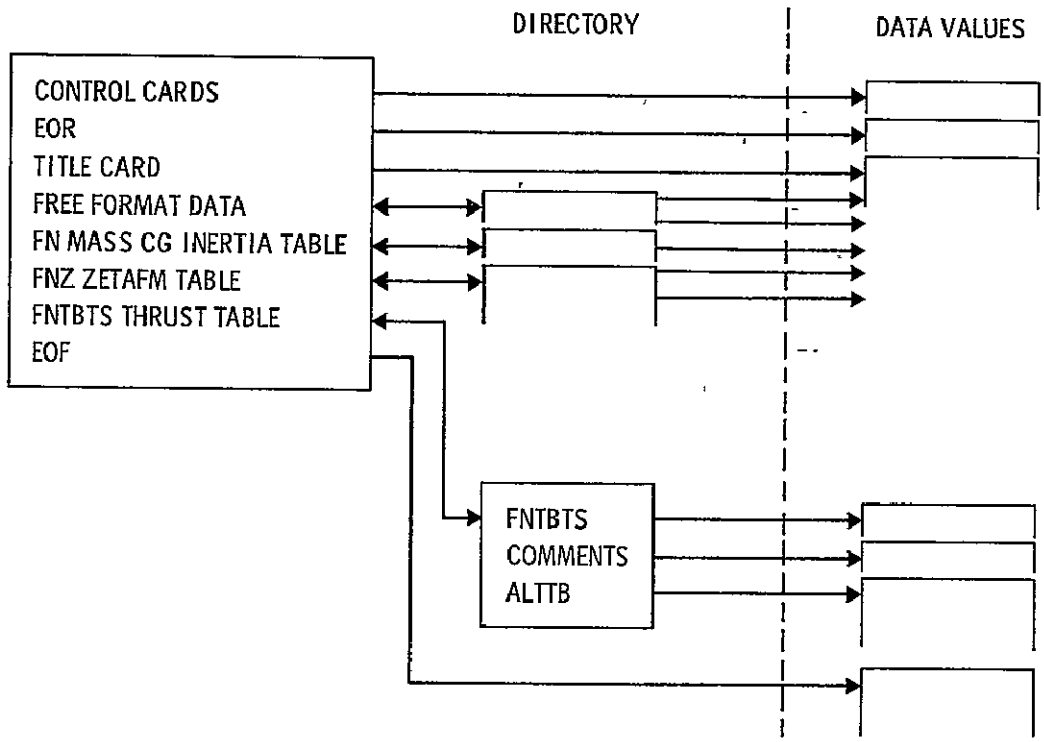


Figure 3-7. IDEF → ISPONGE

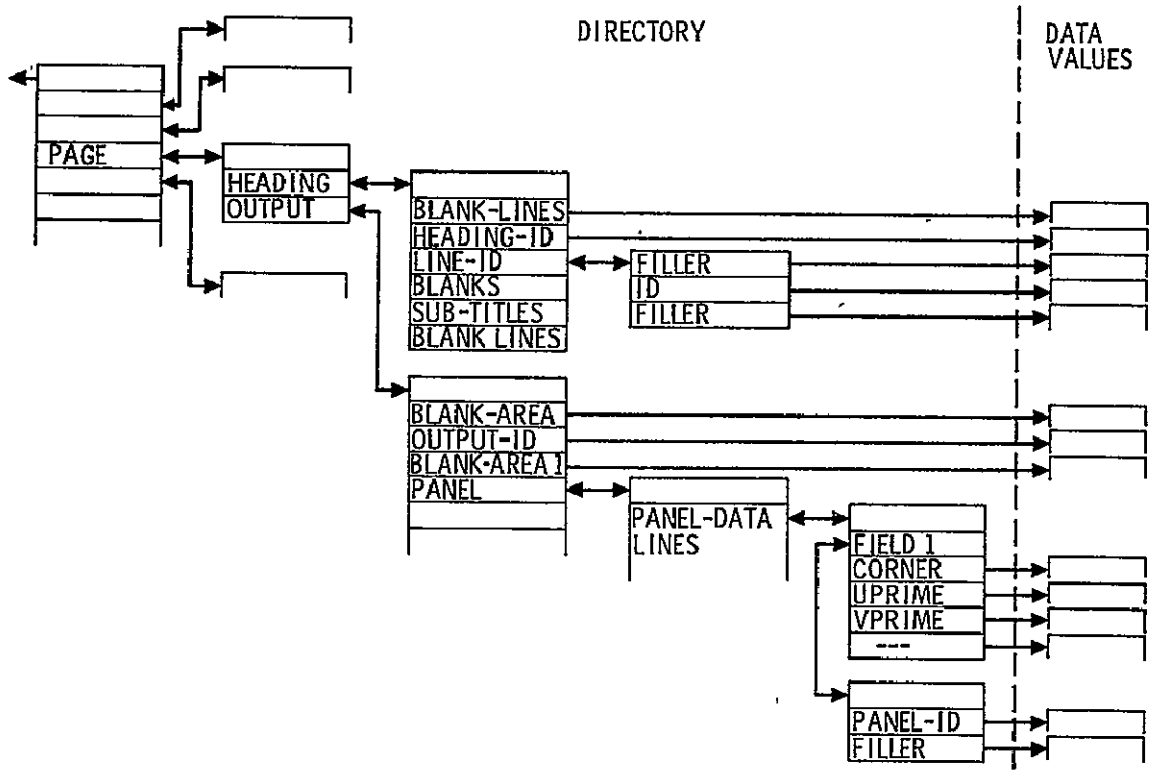


Figure 3-8. ODEF → OSPONGE

3.3.3 Definition of the ISPONGE (OSPONGE). - A SPONGE is an intermediate working-storage version of an input (output) file. Its data format is generally the same that of the input (output) file (which typically is coded) and has the following characteristics:

1. The directory of this file reflects a specific one of all the possible file structures given in the IDEF(ODEF).
2. The combination of directory and data content of the ISPONGE reflects the input required by the OM. Default values are inserted (but flagged OFF) at creation. The input file is in fact a sequential copy of the data content of the ISPONGE.
3. The combination of directory and data content of the OSPONGE reflects the user's selection of the output generated by the OM. The data content is in fact selected from the OM output according to the directory content.
4. Data values may be inserted or modified by the user in random order.
5. The ISPONGE is saved from run-to-run so the user can make modifications, generate a new input file and rerun the OM efficiently.

The ISPONGE (OSPONGE) is distinguished from the input (output) file primarily by characteristics 1 and 4. The directory allows user/IOF communication about the data contents on a data-name basis. The user need not be concerned with the location of the data within the file. The ISPONGE (OSPONGE) is a random access file; the input (output) file is typically sequential access.

The construction, updating and reading of the ISPONGE (OSPONGE) depend on a data management function equivalent to the CDC Graphics Data Handler* (Reference 3). It was decided to base the IOF design on software functionally similar to the Data Handler. Features of this software package are:

1. A list structuring technique known as PLEX.
2. Data access at the character level.
3. I/O support such that the IOF considers the entire ISPONGE (OSPONGE) to be incore data.
4. Efficient space management to give the effect of inplace updates.

* This is a software package installed to support the CDC Interactive Graphics System

3.3.4 Functional description of the IOF operation of constructing an input file. Constructing an input file is a four phase operation whereby the user defines the input file requirements and constructs the ISPONGE including default values; proceeds to fill the ISPONGE with data values selected from the MDB and other files; flushes the ISPONGE creating the input file and subsequently executes the OM; and finally redefines or modifies the data as might be appropriate to subsequent use of the saved ISPONGE.

Figure 3-9 illustrates the four distinct and sequential IOF phases associated with OM input:

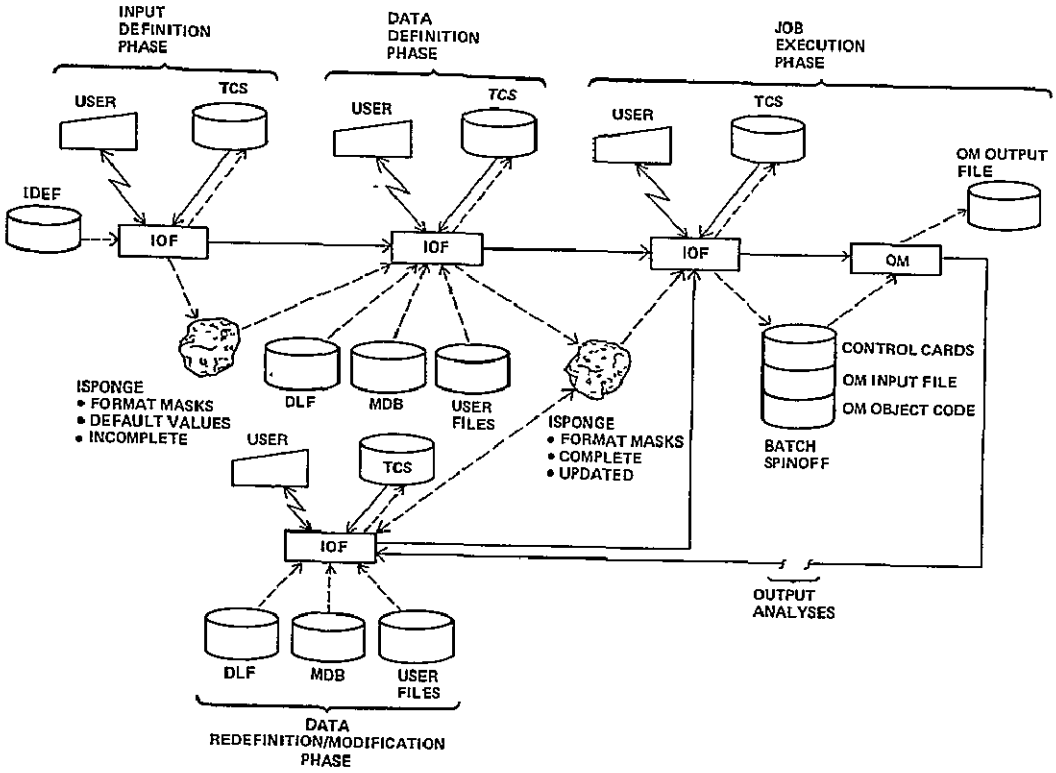


Figure 3-9. Input File Construction

1. Input Definition Phase: According to user selected options, the IOF extracts data specifications from the IDEF and constructs an ISPONGE (random access file) corresponding to an OM input file. Contents of the ISPONGE are format masks (e.g. blank cards), default values where appropriate, templates for repeating items (more blank cards) and at least a partial directory.

2. Data Definition Phase: The user identifies data sources and corresponding locations in the ISPONGE and the IOF responds by mapping and converting data values into the ISPONGE, maintaining the directory. Parameters are inserted into the ISPONGE by conversion of keyboard input.

3. Job Execution Phase: The IOF maps the contents of the ISPONGE onto a file acceptable to the operating system and the OM, and releases it as a batch (or interactive) job.

4. Data Redefinition/Modification Phase. Two situations occur:

    1. After analysis of output, the user accesses and updates the ISPONGE via keyboard input and repeats the Job Execution Phase (e.g. parameter modification).

    2. A previously completed task must be repeated due to update of the MDB. In this case the user essentially repeats part of the Data Definition Phase.

Details of these phases are presented below.

3.3.4.1 Input definition phase: Transform the IDEF into a directory for a random access representation of the input file.

Inputs required:

1. The IDEF contains the following structure specifications which are used to construct the directory and are contained in the directory:

    a. Names of data or groups of data (branches).

    b. Hierarchical level of the name.

    c. Indexing, dimensioning directives.

    d. Program/user options.

    e. Redefinition of mutually exclusive branches.

The following is transferred from the IDEF to the data portion of the file.

    f. Format masks and constants.

    g. Conversion algorithm codes.

    h. Default values.

    i. Engineering units.

2. User Directives - interactive or commands file:

    a. User responds to IOF interrogation on options identified by the IDEF.

Output resulting:

1. An empty ISPONGE results. This is a random access representation of a sparse (i.e. no data except default values) card deck:

a. The columns to receive data values are defined.

b. Internal/external conversion codes are specified.

c. The space requirements for repeated cards or groups of cards is known or can be calculated at execution time.

Processing required:

1. Decompose IDEF statements.

2. Translate decomposed specifications to more processable form.

3. Construct a set of pointer tables to represent the hierarchical structure of the data:

   a. Each level of the hierarchy is a table associating group names with pointers to the next level of tables, or data names with data locations.

   b. Match IDEF option specifications to user responses. If user elects not to select an option at a given level, skip IDEF statements to the next branch at the same level.

   c. Each table of the hierarchy is a separate entity (record or bead) in a random access file containing pointers to:

      - Next higher level table.
      - Lower level table for each named group.
      - Data - if there is no substructure.

   d. Each entry of each table contains (in addition to the name and pointer):

      - Dimension parameters required for that level.
      - User options chosen at that level.

4. Transfer information from IDEF to data fields locatable by directory tables:

   a. Constants (to arrange space) and format masks.

   b. Internal/external conversion codes.

   c. Default values.

Subprocessing required:

1. IDEF file access.

2. Data management.

3.3.4.2 Data definition phase: Insert data values into the ISPONGE.

Inputs required:

1. User directives - interactive or commands file:
   a. Data values.
   b. Source identification.
      - MDB/user files.
      - Subsets, data names.
   c. Structure, substructure, and data names in ISPONGE.
2. MDB/user file fetches:
   a. Name/location correspondence.
   b. Dimensions.
   c. Classification (external/internal conversion code).

Output resulting:

1. Data contents of the ISPONGE.
2. Updated ISPONGE directory for name/location correspondence.

Processing required:

1. Display data group names of ISPONGE directory, user picks one.
2. Display source files available, user picks one at a time.
3. Map data items or data groups, at user specified level, from source to ISPONGE.
4. Conversions:
   a. Internal/external-coded format.
   b. Units.
   c. Interface with utilities for complex conversion requirements (e.g., coordinate system transformations).

Subprocessing required:

1. Data management.
2. File management.
3. Operating system I/O support.

3.3.4.3 Job execution phase: Generate an Input File. Map data contents of ISPONGE to sequential card image file.

Inputs required:

1. ISPONGE - Directory and data contents.

Output resulting:

1. Sequential, coded card images in format required by the OM.

Processing required:

1. Detect the condition that required data has not been supplied by user and present options to user:
   a. Use default value.
   b. Write the file with no value (use blank, as when OM supplies default).
   c. User supplies data (refer back to Subsection 3.3.4.2).
2. Directory processes each directory table in top to bottom sequence. When a table entry points to a second table, mark the position in the current table, process the lower level table, and resume with the current table.
3. Indexing (repetition) of the processing of lower level tables as specified in higher table entries.

Subprocessing required:

1. Data management.
2. File management.
3. Operating system I/O support.

3.3.4.4 Data redefinition/modification phase: This is the same process as the Data Definition Phase except that the ISPONGE is already filled. Frequently the user supplies data values (e.g., as in a parameter study) so access to data source files is often not required.

3.3.5 Functional description of the IOF operation for processing an output file. - Processing an output file is a four phase operation by which the user selects portions of an OM output and routes the data to IOF intermediate files or to an edited sequential copy of the output file.

Figure 3-10 illustrates the four phases of IOF operation in which the user routes OM output to appropriate destinations:

1. Output Definition Phase: By matching ODEF specifications with user intentions the IOF constructs an OSPONGE (random access file) containing templates or format masks to identify data which the user wishes to retrieve from an OM output file. This is a once per design task operation.

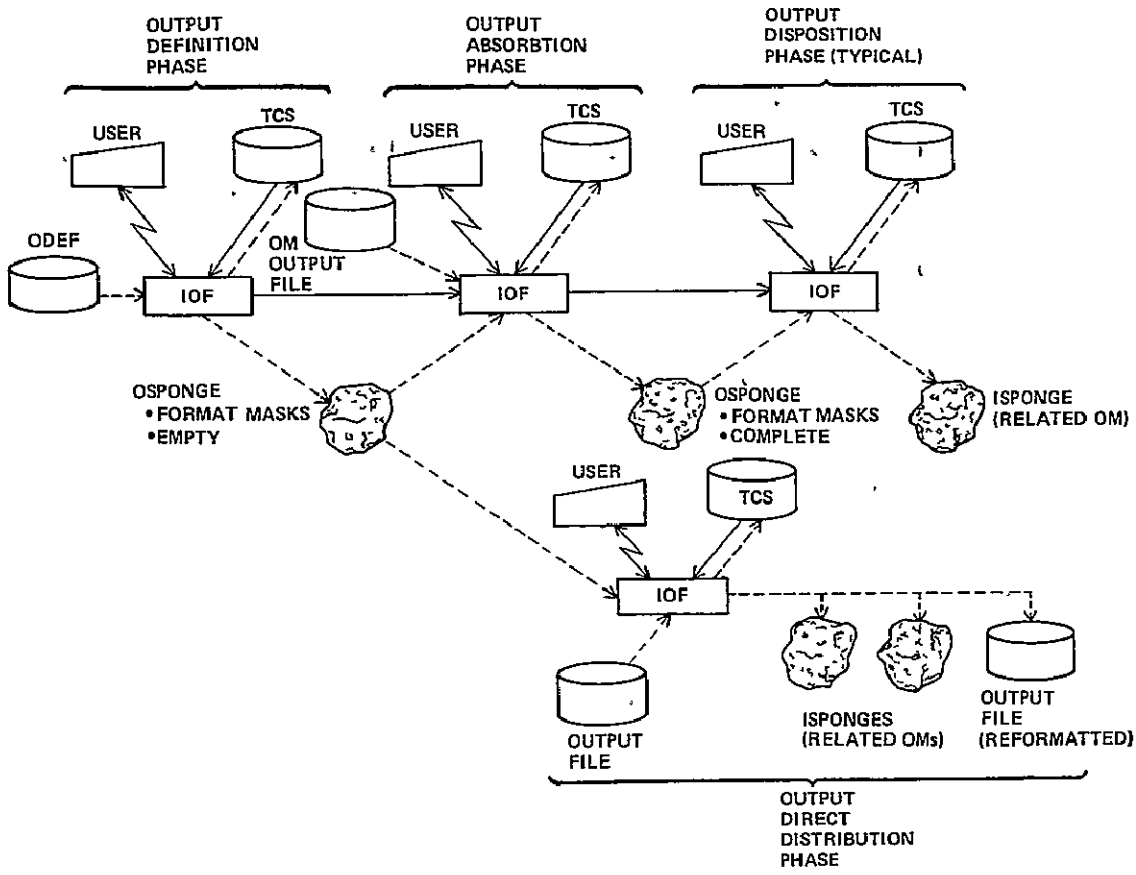


Figure 3-10. Output File Processing

2. Output Absorption Phase: The templates constructed during the definition phase are matched with data read from the OM output file. Identifiable data is copied into the random access file and a directory is established. This occurs once per OM output file, i.e., once per iteration of the OM in a design task.

3. Output Disposition Phase: This is envisioned as a non-contiguous process. That is, the user is concentrating on assembling the input for a related OM and the OSPONGE constructed by the absorption phase is a source for some of the data required.

4. Output Direct Distribution Phase: As an alternative to the output absorption and disposition phases, the user may have his task organized such that the destination of data is known when it is identified in reading the OM output. In this case there is no need for an intermediate copy in the OSPONGE.

Details of these phases are presented below.

3.3.5.1 Output definition phase: Transform the ODEF into a directory for a random access representation of the output file, OSPONGE. This is essentially the same

process described in Subsection 3.3.4.1. For input, the IOF reviews program options with the user and constructs the file required by the options; for output, the IOF reviews with the user the kinds of data which the output file may contain. The user specifies which options are useful to accomplish his current task and which are to be bypassed. The data portion of the OSPONGE at completion of this phase contains format masks and conversion specifications. Default values are not applicable to an output file.

3.3.5.2 Output direct distribution phase: Map data from the OM output file into ISPONGE(s) of related OMs and/or into an edited sequential OM output file.

Inputs required:

1. User directives - interactive or commands file (specifies disposition of data groups or data items).
2. OSPONGE - directory to format masks and data identifiers.
3. OM output file
   a. Typically, pages of output listings.
   b. Typically, data items not identified by the name given in ODEF.
   c. Typically, data items identified by a combination of an ID field (e.g. titles) and position within the page.

Output resulting (according to user directives):

1. Updated data content of related ISPONGE(s).
2. Updated directories of related ISPONGE(s).
3. Sequential copy of OM output (report reformatting) if desired.

Processing required:

1. Access OM output file.
2. Access OSPONGE containing format masks.
3. Transfer from output file to processor (to core) a unit of data specified by the top level OSPONGE hierarchy (e.g. one page).
4. Match top level ID fields (e.g. a portion of a title) to data contents.
5. If no match exists, skip the page, transfer next page to core.
6. If match does exist, match to next level ID fields and proceed.
7. When the page structure has been identified, directory contains data names per ODEF. Display hierarchy of names, user response identifies level or item to map.

8. Display hierarchy of available destination, file-data group, data subgroup, data name; user choice at each level specifies destination. Map data from core to destination file.

9. Conversions:

   a. Required by differing external/internal conversion specifications.

   b. Engineering units.

   c. Interface with utilities for complex conversion requirements (e.g. coordinate system transformations).

Subprocessing required:

1. Data management.

2. File management.

3. Operating system I/O support.

3.3.5.3 Output absorption phase: Map data from the OM output file into the corresponding OSPONGE. The OSPONGE is retained for later disposition in the Data Definition Phases for individual related OMs (or Data Redefinition/Modification Phases). Refer back to Figure 3-10.

This is the same process as that described in subsection 3.3.5.2 except for processing steps 7 through 9, which in this case are:

7. All data identified is mapped.
8. Destination is OSPONGE.
9. No conversion is required since the OPSONGE is a direct representation of the OM's output file.

3.3.5.4 Output disposition phase: This is actually one step in the Input Data Definition Phase of a related OM (refer back to subsection 3.3.4.2); the OSPONGE is one source of data to fill the related ISPONGE.

3.4 The Contributions of CODASYL's DBTG Recommendations

When the design requirements for the I/O Formatter Utility had been completed and preliminary documentation of the IOF conceptual design prepared for review (Section 3.3), a Preliminary Design Review (PDR) was held in conjunction with the Data Base requirements stressing the interface played by the IOF in data acquisition from

the Data Base. Although feasibility had been demonstrated utilizing available, current CDC system software with an eye towards roughly structuring the system and uncovering potential design-approach problems, the PDR uncovered several critical development and operational problems.

Concurrently with the PDR, a search was underway to uncover data systems that addressed several of the many design requirements of the Data Base (see Section 4). During a review of the recommendations from the Data Base Task Group (DBTG) of the Conference On DAta SYstem Languages (CODASYL), a recognizable parallel emerged between their report (Reference 4) and the requirements of both the Data Base and the IOF. Since it appeared that the CODASYL DBTG approach could do a substantial portion of the I/O Formatter's function and be highly transferable if the various computer manufacturers implement the CODASYL DBTG recommendations (as it is indicated they will), CDC's current intent to implement the DBTG recommendations and CDC's future plans were investigated in detail.

A brief history of CODASYL's DBTG follows. For a more complete discussion of CODASYL, the reader is referred to Appendix E.

CODASYL is an organization sponsored by computer users and manufacturers, motivated by their common interest in data systems. It was first chartered in mid-1959 to develop a COmmon Business Oriented Language (COBOL), then rescoped for various other purposes. In 1965 it formed the DBTG to study the problem which now appears fundamental to IPAD: coordinated control of data and interfacing of non-coordinated OMs with the data. It should be noted that CODASYL does not design software but develops and specifies standard languages through which a software user expresses his demands. Software vendors and/or computer manufacturers then commit themselves to support the intent of the CODASYL language specifications.

Figure 3-11 illustrates the I/O formatting (IOF) aspect of the DBTG recommendation. OM_1 produces data which must be subjected to a transformation T before OM_2 can use the data. The standard example of T in the business community is a SORT. Examples of more interest to IPAD include:

1. Unit conversion.
2. Coordinate transformation.
3. File reorganization (e.g. sequential to random).
4. Selection of particular variables.
5. Arbitrary reordering of variables.

Figure 3-11a is the traditional approach. Read a file, transform the data, write a file. Notice that:

Data flow is through some form of supporting software. OM_1, OM_2, and T do not access external devices directly.

2. Movement of data between internal storage and external devices is by far the most time consuming operation performed by a computer.

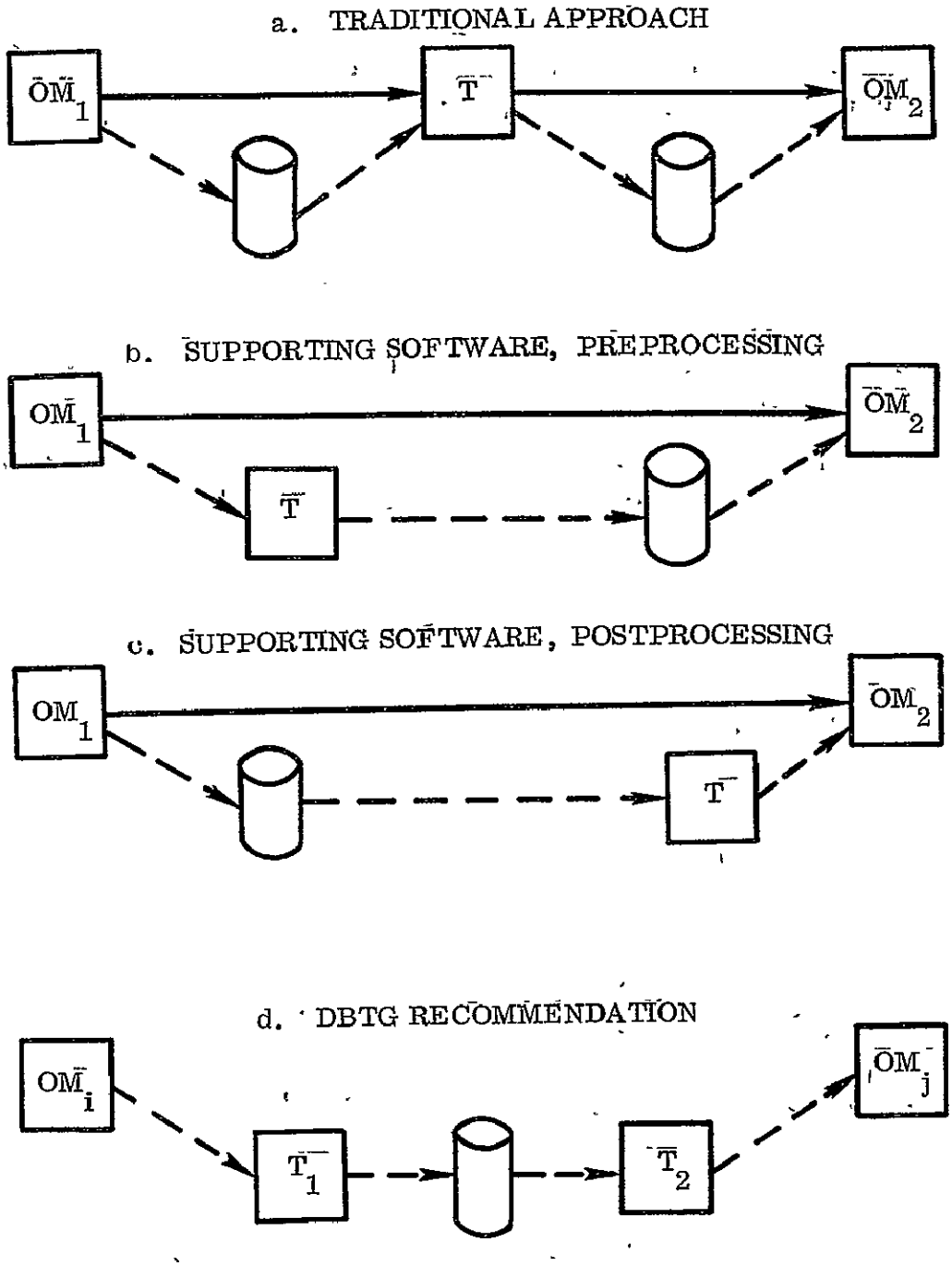


Figure 3-11. Data Transformation Implementations, A General View

Figures 3-11b and c indicate the flow when T is provided by the supporting software. The transformation may occur between generation by OM_1 and transfer to external storage (Figure 3-11c), or between external storage and before use by OM_2 (Figure 3-11b). Notice that:

1. Multiple use of the same logical data is provided without requiring extra storage space.
2. Throughput of OM_1 to OM_2 is increased since two transfers of data between external and internal storage are eliminated.

Figure 3-11d indicates the DBTG recommendation for the general problem. Notice that:

1. Output of any particular OM (OM_i) is transformed to an optimum external form.
2. The optimum external form is transformed to satisfy the input requirement of any other particular OM (OM_j).
3. The optimum external form is to be decided upon by a Data Base Administrator (DBA) applying his own criteria. The criteria would consider the following factors and trades:
   a. Maximize the number of unity transformations on any given data (i.e. no format change required).
   b. Providing redundant copies of frequently transformed data (to further increase unity Ts).
   c. Physical localization of logically related data.

The DBTG report (Reference 4) specifies three languages through which the supporting software is informed of:

1. What tranformations to make:
   a. A Data Description Language (DDL) to describe the data as it exists (or will exist) in external storage (SCHEMA DDL)*.
   b. A Data Description Language to describe the data required and/or generated by each OM in the system (SUBSCHEMA DDL).
2. When to make the transformations:
   a. A Data Manipulation Language (DML) to permit OM control of the supporting software.
      - Functionally replaces and includes conventional I/O imperatives.
      - Contains references to SUBSCHEMA DDL declaratives.

* Throughout this report, the special terminology of CODASYL's DBTG appears in caps (e.g., SCHEMA and DDL).

The DBTG envisioned an environment such as an IPAD project in which some large volume of logically related data is to be managed and controlled. The report (Reference 4) summarizes the Data Base Task Group's analysis of requirements as follows: "A point has now been reached where in designing systems capable of handling our current demands, it is essential to develop databases that are available to and suitable for processing by multiple applications and that can be interfaced by multiple languages." (Ibid, p 6).

The DBTG recognized the requirements of three classes of system users.

1. The Data Base Administrator (DBA), envisioned as a technically oriented (i.e. familiar with data processing techniques) person or persons charged with responsibility for the overall construction and maintenance of the project oriented database.
2. Programmers, envisioned as technically oriented and responsible for detailed procedural use and production of data within the database.
3. The non-programmer, characterized as the IPAD user, who needs a language "aimed at performing a specific set of database functions in a way which obviates conventional programming." (Ibid, p 7).

The DBTG recommendations provide the SCHEMA DDL by which the DBA specifies the logical structure of the database. The functional requirements of maintainance and physical structuring of the database are left to implementors.

In considering the needs of programmers versus non-programmers, the DBTG concluded that the needs of the programmer were more basic and decided to defer the specification of a self contained language (Query Processor Language in IPAD discussions, see Section 3.6), but to provide "a solid foundation for such self-contained capabilities." (Ibid, p 8).

The languages specified in the report are appropriate for a COBOL environment. The COBOL DML in particular is an enhancement of COBOL. CODASYL recognizes the need for further iterations in view of other languages and the need for experience to be gained by implementation. The following quote is taken from the foreword of the DBTG report (Reference 4).

> "With this report as the base, CODASYL has now established a new standing committee, the Data Description Language Committee, independent of and equal to the Planning, Systems and Programming Language Committees. It is envisioned that from the base already established by the DBTG report this new committee will finalize the specifications for a common data description language, independent of but common to many other higher level programming languages. Simultaneously, the Programming Language

Committee will develop COBOL extensions based on Section 4 (COBOL Sub-schema) and Section 5 (COBOL Data Manipulation Language). It is hoped that organizations responsible for other programming languages will develop the appropriate sub-schema and data manipulation language features for their language.

While this report is not classified as a final CODASYL specification of a common language, it does represent many years of work and the best thinking of many recognized experts. Therefore we believe it is practical and appropriate for implementations to be made based on these specifications so that a foundation of experience may be established from which to further evolve and refine the specifications of a common data description language. [End Quote.]

The following definitions are quoted from the Major Concepts section of the DBTG report. (Ibid p. 13, 14).

The DDLs are the languages used for describing a database, or that part of a database known to a program. These descriptions are in terms of the names and characteristics of the DATA-ITEMs, DATA-AGGREGATEs, RECORDs, AREA, and SETs included in the database, and the relationships that exist and must be maintained between occurrences of those elements in the database.

A DATA-ITEM is the smallest unit of named data. An occurrence of a DATA-ITEM is a representation of a value.

A DATA-AGGREGATE is a named collection of DATA-ITEMs within a RECORD. There are two types: vectors and repeating groups. A vector is a one-dimensional, ordered collection of DATA-ITEMs, all of which have identical characteristics. A repeating group is a collection of data that occurs an arbitrary number of times within a RECORD occurrence. The collection may consist of DATA-ITEMs, vectors, and repeating groups.

A RECORD is a named collection of zero, one, or more DATA-ITEMs or DATA-AGGREGATEs. There may be an arbitrary number of occurrences in the database of each RECORD type specified in the SCHEMA for that database. For example, there would be one occurrence of the RECORD type PAYROLL-RECORD for each employee. This distinction between the actual occurrence of a RECORD and the type of the RECORD is an important one.

A SET is a named collection of RECORD types. As much, it establishes the characteristics of an arbitrary number of occurrences of the named SET. Each SET type specified in the SCHEMA must have one RECORD type declared as its OWNER and one or

more RECORD types declared as its MEMBER RECORDs. Each occurrence of a SET must contain one occurrence of its OWNER RECORD and may contain an arbitrary number of occurrences of each of its MEMBER RECORD types.

An AREA is a named sub-division of the addressable storage space in the database and may contain occurrences of RECORDs and SETs or parts of SETs of various types. AREAs may be opened by a run-unit with usage modes which permit, or do not permit, concurrent run-units to open the same AREA. An AREA may be declared in the SCHEMA to be a TEMPORARY AREA. The effect of this is to provide a different occurrence of the TEMPORARY AREA to each run-unit opening it and at the termination of the run-unit, the storage space involved becomes available for re-use.

The concept of AREA allows the Data Base Administrator to subdivide a database rather than considering the database as a single unit. The use of AREA allows the Data Base Administrator or the Data Base Management System (DBMS) to control placement of an entire AREA to provide efficient storage and retrieval. The opening of AREAs by run-units also gives implementors an opportunity to optimize access to the data base since the run-unit has narrowed the range of interest in the data base to a relatively small number of subdivisions of the entire data base. AREAs are convenient units for recovery, as duplication or backup can be carried out selectively. AREAs also provide a convenient natural subdivision for allowing certain unused portions of the database to be saved in archival storage while the remainder of the database is actively accessed.

A database consists of all the RECORD occurrences, SET occurrences and AREAs which are controlled by a specific SCHEMA. If an installation has multiple databases, there must be a separate SCHEMA for each database. Furthermore, the content of different databases is assumed to be disjoint.

A SCHEMA consists of DDL entries and is a complete description of a database. It includes the names and descriptions of all of the AREAs, SET occurrences, RECORD occurrences and associated DATA-ITEMs and DATA-AGGREGATEs as they exist in the database.

A SUBSCHEMA also consists of DDL entries. It, however, need not describe the entire database but only those AREAs, SETs, RECORDs, DATA-ITEMs and DATA-AGGREGATEs which are known to one or more specific programs. Further, it describes them in the form in which they are known to those specific programs and it may also rename them.

The DATA MANIPULATION LANGUAGE (DML) is the language which the programmer uses to cause data to be transferred between his program and the database. The DML

is not a complete language by itself. It relies on a host language to provide a framework for it and to provide the procedural capabilties required to manipulate data. [End Quote.]

3.4.1 SCHEMA-SUBSCHEMA. - Figure 3-12 illustrates the relationship of the SCHEMA to a SUBSCHEMA:

1. An operating system level I/O support module looks at I/O devices and files.
2. The contents of the I/O files are described for any specific use through a SCHEMA, hence the collection of files is labeled DATA BASE.
3. A mapping of a portion of the SCHEMA through a SUBSCHEMA is indicated with the Data Base Management System (DBMS) viewing both descriptions as well as the mapping paths.
4. The individual OM sees only the SUBSCHEMA representation and has the illusion that the database as described by the OM's creator actually exists.

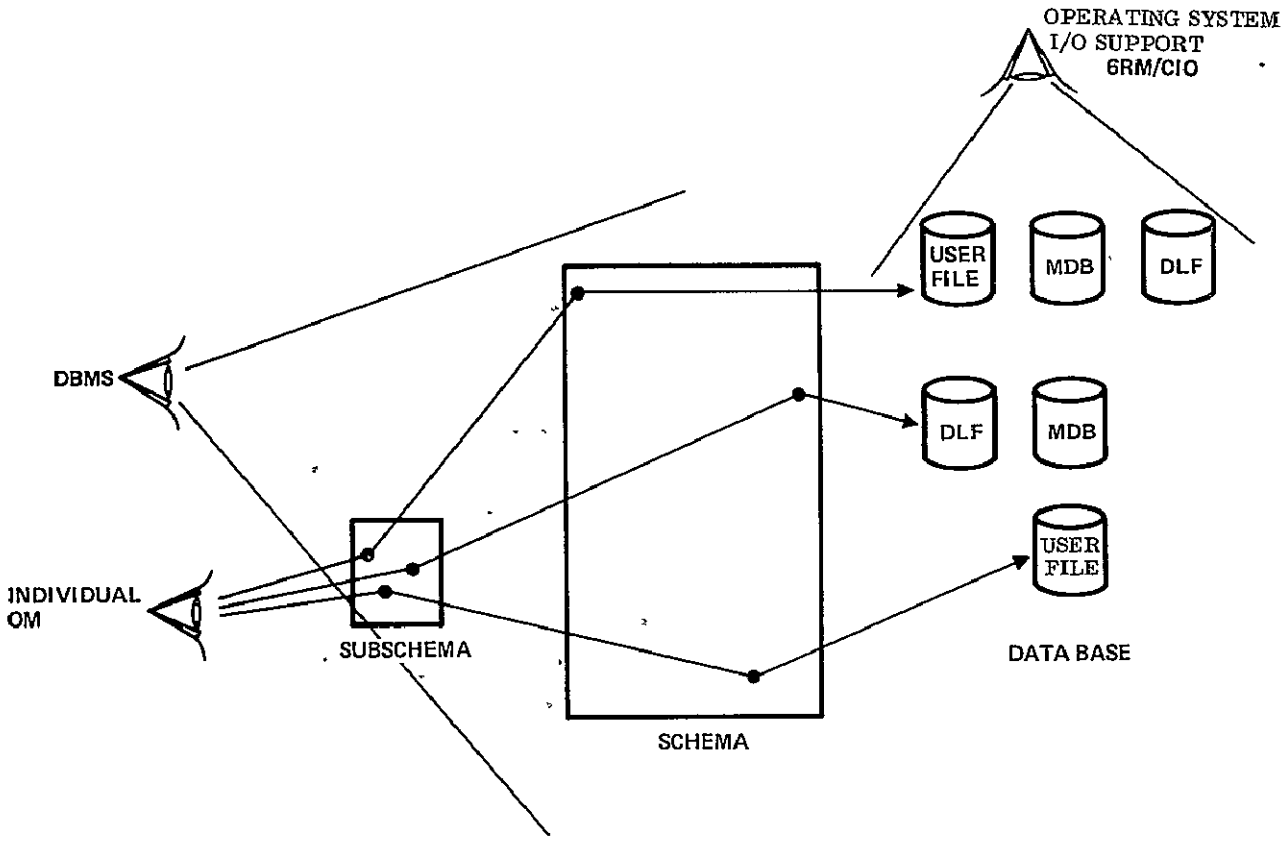


Figure 3-12. Data Base Structure as Viewed by Various Support System Software

The overview presented in this figure glossly simplifies the functional support details of DBMS, viz:

1. Mapping. As illustrated, the OM is provided with the illusion of physical entities envisioned by the programmer. However:
   a. An OM "file" may be mapped into many database files.
   b. All the OM "files" may be mapped into one database file.
   c. One OM "record" may be composed of many database RECORDs.
   d. One OM "record" may consist of selected parts of one database RECORD.
2. Data Item Transformations. Two additional kinds of conversion may be implied by corresponding data item descriptions:
   a. Data as seen by the OM may be the result of a functional calculation such as unit conversion; DDL may specify the function to be called by DBMS, but does not contain the function (or procedure) itself.
   b. Reformatting and/or reordering of items.
3. The SUBSCHEMA may describe the data as it actually exists (i.e. SUBSCHEMA DDL matches SCHEMA DDL). In this case no mapping or transformation procedure occurs.

3.4.2 DML. Figure 3-13 illustrates the features of DDL and DML:

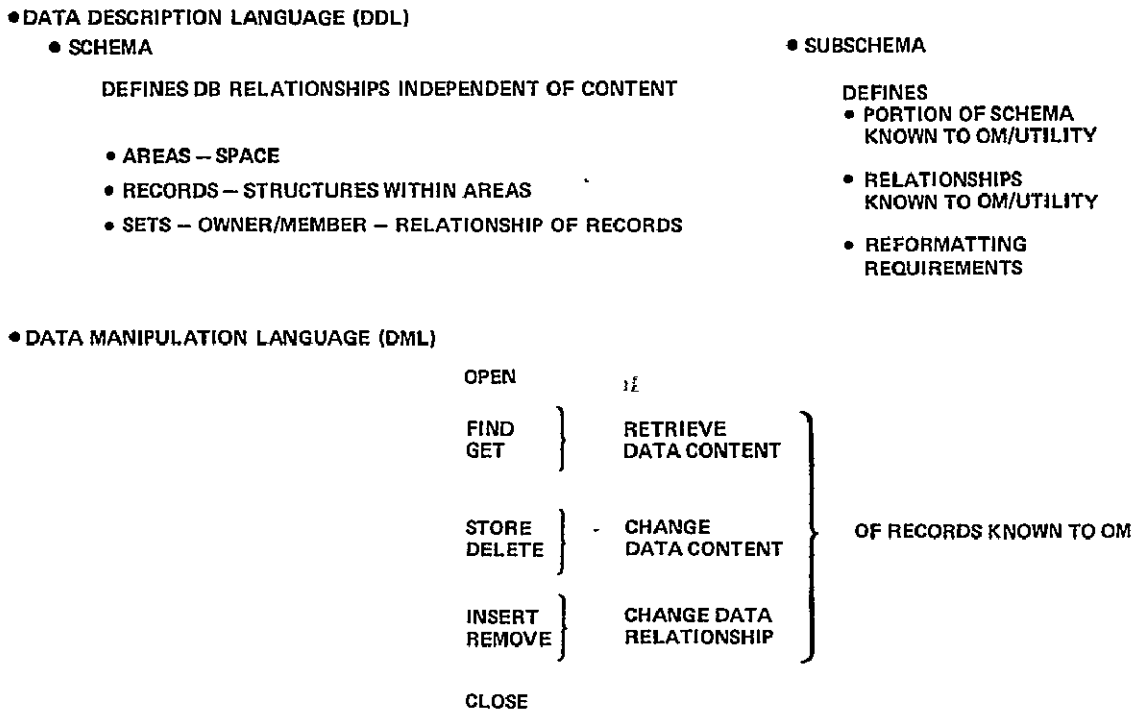


Figure 3-13. Features of DDL and DML

1. Through DML directives, the OM expresses intentions of accessing, creating and/or updating records as described in the SUBSCHEMA.

2. The DBMS has access to both the (object) SCHEMA and SUBSCHEMA and:

   a. Performs any transformation implied by the two descriptions.

   b. Translates the I/O intention into operating-system level I/O action.

3.4.3 CODASYL and the OM Interface Problem. - Interfacing OMs in a DBMS environment requires, as a preliminary condition, that data corresponding to the OM requirements exist in a database (i.e. the data must be described in SCHEMA DDL). A pair of OMs view the same data through SUBSCHEMAs.

OM incorporation consists of:

1. Analyzing the I/O requirements of the OM and expressing them in DDL for a SUBSCHEMA.

2. Analyzing the intent of the OM's I/O activity and converting to DML directives related to a SUBSCHEMA.

3. Specifying in the SUBSCHEMA DDL the relationship between SCHEMA and SUBSCHEMA.

Conceptually, all interface problems can be resolved (in a variety of ways) by appropriate DDL specification in conjunction with primitive Data Base Procedures.

The role of the IOF as such has been replaced by DBMS!!!

IPAD's IOF objectives still require a Query Processor (QP) function to give the user access to the data but this requirement is now distinct from the IOF problem. The DBTG report deferred development of a QP language but indicated that a QP function should access data in the same manner as the application programs (i.e., the OMs).

3.5 An Implementation of DBMS

Preliminary documentation from UNIVAC (References 5 and 6) and CDC (References 7 through 10), and discussions with a CDC-sponsored member of the CODASYL DBTG indicate that major computer manufacturers are committed to the development of software incorporating, and supporting the intent of the DDLs and DML of the DBTG report (Reference 4).

The IOF task was realigned to avoid developing an IPAD IOF utility in parallel with a similar effort on the part of the computer manufacturers. The revised approach

consisted of three tasks:

1. Plan to exploit the capabilities of a DBMS.
2. Analyze CDC's planned* first release capability of the DBMS as applied to IPAD requirements.
3. Recommend modifications to or extensions of the DBMS capabilities to support IPAD requirements.

3.5.1 CDC's DBMS. - Figure 3-14 represents the relationship of the database and software components as related to a possible CDC implementation of the CODASYL recommendations.

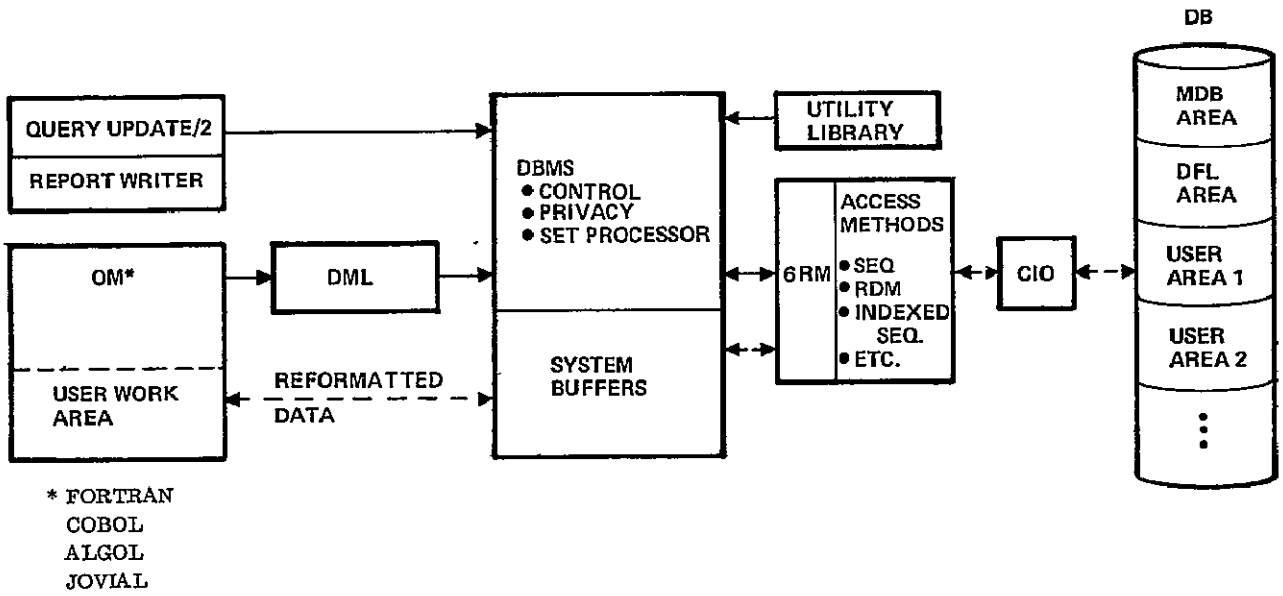


Figure 3-14. CDC's Implementation Plan for CODASYL's Data Base Management System

The database is composed of a number of files distributed over a number of I/O devices of various kinds. All software accesses the database through CIO (Circular I/O, a peripheral processor support system subprogram) which arranges transfer of data between external storage and core storage. This is the level of software which provides I/O device independence to higher levels. CIO is existing code.

* Any discussion concerning CDC's planned or intended implementation of DBMS must be stated in qualified terms: such implementation may or may not occur and the DBMS design and features to be included are subject to constant re-appraisal.

The next level of software, 6RM (the Integrated Record Manager), provides detail support for various data access methods. This provides the interface between a logical organization of data and its physical organization. This (6RM) is the level of software that permits higher levels to use standard access methods independent of their physical implementation. The 6RM module is existing code (Reference 11).

The DBMS level of software provides access to data independent of the access methods by which the data is logically organized. The DBMS Control modules establishes and maintains tables and pointers denoting logical relationships of data. The Set Processor modules (Reference 10) translate these into parameters required by 6RM in managing data organization (files). The privacy modules of DBMS insure data integrity with respect to secrecy specifications and with respect to concurrent users.

The utility library contains functions necessary for the maintenance of the database by the DBA, e.g. statistical gathering and analysis functions, dump and edit functions, etc.

Query Update/2 (QU/2) is CDC's implementation of a "self contained capability" mentioned in the DBTG report (Reference 4, page 7). QU/2 (References 8 and 9) provides access to the database without conventional programming. Its Report Writer feature provides the capability of listing portions of the data base in user-specified formats. Query Update/1 (Reference 12) is CDC's existing interim version of QU which uses a special DDL called QUIDDL (Query Update Interim Data Description Language). QUIDDL's features are documented in Reference 13.

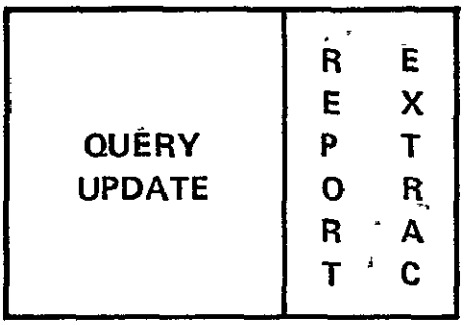
The total DBMS system is envisioned to support OMs, written in several languages, providing control of DBMS to the OMs through DML commands. DBMS delivers to and accepts data from a user's work area associated with the executing OM, performing any required transformations between the user's work area and its own buffer area. Data in the system buffer area is as delivered to or accepted from 6RM.

It should be noted that:

1. The syntax of the OM's source language shall be enhanced by DML and the source code of the OM modified to replace conventional I/O techniques with DML.
2. DML object code routes all I/O requests through DBMS modules which analyze SCHEMA/SUBSCHEMA specifications and:
   a. Reformat and restructure data as required.
   b. Make conventional I/O requests of the operating system software to achieve the specified result.

3. Query Update/2 (References 8 and 9) operates on a level parallel with the OM (i.e. making I/O request of the DBMS modules) to give the IPAD user interactive access to the database.

3.5.2 CDC's Query Update/2. Figure 3-15 highlights the features of QU/2:



QUERY UPDATE

- USES DDL DIRECTORIES
- INTERACTIVE AND BATCH WITH NO CHANGE OF CAPABILITY
- POWERFUL ARITHMETIC, BOOLEAN DIRECTIVES
- CATALOG PROCEDURES
- EASY TO USE
- RECOVERY + RESTORATION COMMANDS
- AUDIT TRAILS

REPORT EXTRACTOR

- CATALOG REPORTS
- ON-LINE PREVIEW
- TABULAR FORM
- CAPABILITY

Figure 3-15. CDC's Query Update Version 2.0

1. Database areas (files) are described by CODASYL standard DDL which allows QU/2 to access data produced by (or produce data for input to) OMs written in FORTRAN, COBOL, ALGOL or JOVIAL.

2. Extracts or updates existing data or creates new data according to user directives either interactively or in batch mode.

3. Provides a desk-calculator level of arithmetic directives and complex Boolean expressions for conditional directives.

4. Provides facilities for retaining (cataloging) a series of directives (a session) for subsequent reexecution (similar to IPAD's TCS).

5. Provides a verbose, COBOL-like directive language which is easy to read, but potentially cumbersome to an IPAD user.

6. Provides two special purpose commands:

   a. RECOVERY - to reset data sets to a previous condition. (e.g. condition lost due to system error).

   b. RESTORE - to undo the effects of the current session.

7. Provides report extractor directives which allow the user to specify the format and content of any output listings to be generated.

3.5.3 First Release Capability of CDC's DBMS. - No documentation of this system was available for review, however, extrapolating from the QU/2 and DDL-V1 preliminary documentation (References 7, 8 and 9), it would appear that:

1. The first release DBMS implements DDL and DML essentially verbatum from CODASYL's DBTG report with a COBOL syntax and (perhaps) some COBOL oriented restrictions.

2. The raw functional capability to support a FORTRAN environment will be provided but with no FORTRAN syntax for DDL, no DML enhanced FORTRAN compiler, and with DBMS functions possibly limited by COBOL restrictions on the SCHEMA DDL.

Unofficial reports indicate that CDC is studying FORTRAN syntax but has no current effort going to include FORTRAN support in DBMS.

3.5.4 Review of Design Constraints and Guidelines. - (Refer to Section 3.1).

1. Procedural requirements imposed on the user and the range of choices open to the user are fixed by the SCHEMA/SUBSCHEMA DDLs:

   a. QU/2 allows the user to manipulate data described in a SUBSCHEMA as he chooses, subject to limitations specified in the SCHEMA (e.g. PRIVACY limitations and/or concurrent user limitations).

b. QU/2 provides a user option to retain a series of directives (a "session") for reexecution (similar to IPAD's TCS).

c. All data accessible through the SUBSCHEMA is conceptually random access. The actual organization as described in the SCHEMA could degrade response time so as to prohibit random access. Such limitations derive from project administration rather than QU/2 or DBMS.

2. Modification of OMs is required to route I/O requests through the DBMS. These are not logic changes in that the DML should express the same intent as the conventional I/O requests. The extent (difficulty) of the changes depends on the development of the DML and/or a DML insertion preprocessor.

3. Input to an OM may be derived from a variety of sources in either of two ways, depending on decisions reflected in the DDLs:

   a. The user can direct QU/2 to extract data from any source provided in the SUBSCHEMA and collect it in a SUBSCHEMA defined destination for input to the OM.

   b. The SUBSCHEMA invoked by the OM's DML may refer directly to the various sources rather than one localized source.

4. Output of the OM may be distributed in the same manner as the input is collected.

5. Tutorial assistance can be designed and provided as specified by the installation, the project, a particular design task, or an individual OM. Two methods, using QU/2, are possible:

   a. Tutorial data can be provided in the data base to be displayed to the user according to user directives.

   b. Predefined tutorial "sessions" may be cataloged to achieve a MACRO/MICRO Menu effect.

3.5.5 Review of IOF Requirements. - (Refer to Section 3.2 on a paragraph basis).

1. Random access/update capability is provided with the DBMS approach for all data, subject to limitations imposed by database administration.

2. The DBMS approach circumvents the requirement for intermediate files as well as the requirement that all files be random access:

   a. QU/2 and the OMs can access the same data.

   b. The "file" as seen by the OM may map into several files in the database.

c. Decisions may be reflected in the SCHEMA DDL to localize those portions of the data that the user will access via QU/2 providing efficient random access to that data and no access to the rest.

3. Tutorial assistance data can be stored in the database, made available to QU/2 through a SUBSCHEMA and displayed by the user. Such data could include that listed in Section 3.2 and can be extended as desired by project administration. In addition, the user has facilities to generate tutorial notes as he pleases for his own future reference.

4. DDL provides for describing all I/O data of the OM independent of the use of the OM as required. In addition, it provides facilities far beyond this basic requirement which can:

   a. Specify interface techniques to relate the OM's requirements to other data descriptions.

   b. Share available data storage with other OMs.

   c. Insure integrity of shared data.

5. DDL compiles into an "object table" which serves the purpose of a "file directory" within DBMS.

6. Integration of interface requirements through DDL circumvents many of the functional requirements of the IOF utility approach. The remaining functional requirements are met by QU/2:

   a. Intermediate IOF files are not necessary.

   b. QU/2 updates RECORDs that the OMs access.

   c. QU/2 extracts data from RECORDs accessible to OMs.

   d., e. Mapping is accomplished implicitly (in accordance with DDL specifications) during execution of DML directives in the OMs or explicitly as the user directs QU/2.

   f. QU/2 provides desk-calculator level arithmetic directives.

   g., h. The DBMS approach makes these function equivalent to mapping as described above.

   i. Tutorial data is accessible by the same QU/2 directives as I/O data.

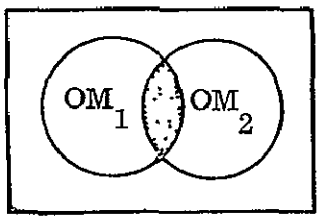
7. QU/2, as previously discussed, provides a user's command language.

3.5.6 Review of the IOF Conceptual Design. - (Refer to Section 3.3). Under a CODASYL DBMS, it is possible to express the solution of OM-OM interface problems in the SCHEMA/SUBSCHEMA DDLs. In this case the DBMS modules and primitive

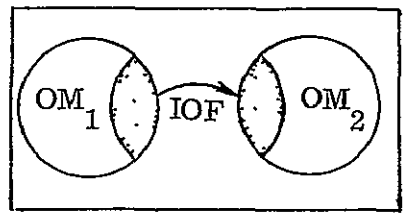
Data Base Procedures would effect the solution to the interface problem during execution of the OMs. No data manipulation between execution of linked OMs would be required and the OMs would access and update the database directly. This approach was functionally depicted in Figure 3-11.

At the opposite extreme, the DDLs could express interface requirements between themselves and the database only, with no solution to the OM-OM interface problem. This would result in redundant copies of all common data involved and would require additional mapping procedures between execution of any successive pair of OMs. Functionally, this approach would be the same as that depicted in Figures 3-9 and 3-10. Because this approach is optional with the user (although less efficient computer-wise), this section will relate the concepts of the IOF utility (Section 3.3) to the equivalent CODASYL capabilities.

These two solutions to the OM-OM interface problem are contrasted in the Venn diagram below as it relates to data common between OMs.



CODASYL DBMS



IOF UTILITY

3.5.6.1 I/O FILES and I/O SPONGES: In a DBMS environment, the emphasis shifts from construction and disposition of files to construction and disposition of RECORDs. A file, as visualized in Section 3.3, is a particular relationship of RECORDs. Since DBMS deals with the OM on a RECORD by RECORD basis, there is no logical requirement for any particular physical organization of the RECORDs. In the CODASYL approach the OMs are presented with the illusion of files through the functions of:

1. Constructing input RECORDs, as described in the DDL, in response to DML commands executed by the OM.
2. Disposition of output RECORDs, as described in DDL, in response to DML commands executed by the OM.

The concept of ISPONGE and OSPONGE, as randomly accessible representations of data requirements, applies to all data to be processed by DBMS. However, there is no need to consider intermediate representations nor arbitrary file structures.

Through DML commands the OMs can access the ISPONGEs and OSPONGEs. Furthermore, all the data associated with an OM (or a particular linkage of OMs) can be considered as one I/OSPONGE from which DBMS selects data to construct input RECORDs and into which it distributes data given in output RECORDs.

3.5.6.2 The IDEF and ODEF: The two parts of the I/ODEF described in Section 3.3 correspond (within CODASYL's DBMS) to:

1. Tutorial data residing in the database, described in the DDL for the SCHEMA and accessible by QU/2 through a SUBSCHEMA. This data describes the I/O requirements of the OM from a user's viewpoint (man readable).

2. SUBSCHEMA DDL describing the I/O requirements of the OM to the DBMS (compiler and man readable). This is partial SUBSCHEMA DDL. It describes RECORD structures required by the OM. The SUBSCHEMA is to be completed for use by the OM by relating these structures to SCHEMA DDL. Provisions of DDL include the list of Subsection 3.3.1.2 except that RECORD structures are defined rather than file structures.

3.5.6.3 Use of the IDEF and ODEF: The tutorial section of the I/ODEFs have exactly the same purpose as in the IOF conceptual design. This is user oriented information to be displayed as the user requests. However, the use of DDL (item 2 above) for this purpose is substantially different:

1. The DDL in the I/ODEF converts to a SUBSCHEMA rather than a directory or directories. The SUBSCHEMA:

   a. Describes the data names, RECORD structures and RECORD relationships required by the OM.

   b. Relates these descriptions to a SCHEMA, a similar description of the data as it exists or will exist.

2. In the IOF conceptual design, processing of the I/ODEF included editing and deleting data descriptions on the basis of the user's intent regarding program options. In the CODASYL approach, the SUBSCHEMA can provide the full range of OM capabilities rather than only one consistent choice of options.

3.5.6.4 CODASYL's DBMS functional relationship to IOF preliminary design: The IPAD objective that the user shall be in control and the user defines his procedure by accomplishing a task, requires that the CODASYL approach provide for the functional steps presented in the IOF preliminary design:

1. Input (output) definition.

2. Data (input data) definition.

3. OM execution - generation of output data.

4. Output processing:

   a. Direct distribution.
   b. Item by item distribution.

5. Data (input) redefinition/modification for re-execution of the OM.

Input (Output) Definition. This operation converts the I/ODEF (DDL portion) into a SUBSCHEMA. For this discussion it is assumed that logically equivalent, i.e. mappable, data descriptions exist in the SCHEMA. Preparation of the SCHEMA is discussed in Section 7.

The I/ODEF DDL contains:

1. An AREA description for logically distinct data (e.g., an AREA for an input file and a different AREA for an output file).

2. Descriptions (identification) of RECORDs and possibly SETs of RECORDs within the AREAs.

3. Descriptions of the structure of RECORDs in terms of DATA NAMEs.

4. Attributes of DATA NAMEs.

The user must provide renaming and mapping specifications to relate the above to the SCHEMA. The DDL compiler produces an object SUBSCHEMA through which DBMS can map data occurrences from OM internal storage to the database and vice versa.

Data Definition. For this phase, the user will interact with QU/2 which will have access to a SUBSCHEMA (which is equivalent to a copy of the SCHEMA).

Through QU/2 directives he will:

1. Define variables in QU/2 temporary storage.

2. Access blessed data AREAs (MDB) and/or other OM output.

3. Extract data values into temporary storage.

4. Access OM input AREAs.

5. Insert values from temporary storage.

6. Insert values via keyboard.

OM Execution. The user interacts with the IPAD EXEC and TCSS Expander to accomplish OM execution. The OM interacts with DBMS through DML commands to retrieve its input data and generate its output data.

Output Processing. No special step is necessary for the usual situation in which the OM output is to be mapped to other OM input. The output is available to Data Definition and/or Data Redefinition steps without intermediate processing:

1. Direct distribution can be achieved either by DBMS through DDL specifications or through a Query Update Session.
2. Single DATA-ITEM disposition is a step in Data Definition or Redefinition Phase for related OMs.

Data Redefinition/Modification. This is a subphase of the Data Definition phase. Using the same QU/2 setup, the user will usually:

1. Access the OM input AREA.
2. Modify data values as required.

3.6 IPAD's Query Processor

The DBTG report recognized the need for data base operations by two classes of (DBMS) users, the technically oriented programmer and the non-programmer. The needs of the programmer were seen as the more pressing, consequently the needs of the non-programmer were deferred. In discussing this decision, it was noted that the needs of the non-programmer were essentially undefined; that a range of possibilities existed from simple predefined operations to a full "self contained capability" (Reference 4, page 8). The needs of the IPAD user as relates to the IOF capability require the latter. The name Query Processor (QP) was chosen to signify a "self contained [interactive] capability" following the lead of UNIVAC (Reference 6) and NSRDC's COMRADE (Reference 14).

The scope of CDC's QU/2 envisions conversational inquiry and updating of information. The need for this capability is seen to arise from the large gap between available special purpose interactive software (for application oriented users) and the general purpose interactive software (oriented toward knowledgeable programmers). The extent of capabilities provided through the QU/2 language place QU/2 in the QP family. The primary difference between QU/2 and IPAD's QP is the difference of language (see Section 5.5). Like the DBTG specifications, the language specifications of QU/2 are closely related to COBOL. The typical IPAD user could not be induced to express his needs in this language.

The concept of a prefabricated TCS (see Subsection 2.3.3 of Volume IV, Part I) and tutorial assistance in expanding TCSSs (Subsection 2.2.1 of this Part) can alleviate the problem of an appropriate QP Language (QPL) to some degree, but not altogether. A feature of QP is the ability to identify and save the contents of a QP Session (QPS) and the ability to execute a previously saved (or prefabricated) QPS. Consequently, the QPS is an extension of the TCS concept, with the QPS to be accessed and executed by QP rather than by the IPAD EXEC. Just as a TCS may be derived by substituting task dependent parameters in a TCS Skeleton (TCSS of Subsection 2.2.1), a QPS may be derived by substituting task dependent information in a QPS Skeleton (QPSS). The TCSS Expander capability associated with the EXEC will expand either skeleton file. In further discussions, QPSs are considered as subsets of TCSs and QPSSs as subsets of TCSSs.

3.7 Extensions to the DBTG Recommendations

The IPAD design must account for three problem areas which were beyond the scope of the DBTG work:

1. Engineering oriented language development. This requirement is the subject of Section 5.
2. Special Purpose Utilities (SPUs) to assist in constructing the DBMS interface (DDL/DML) for:
   a. Incorporating existing OMs into IPAD (a programming function).
   b. Integrating a group of incorporated OMs into a task-oriented entity (a design/engineering function).
3. The special data handling requirements of interactive graphics applications. This problem is discussed in the following subsections.

3.7.1 Data handling for interactive graphics. - Early experience with interactive graphics applications have demonstrated that the standard data access methods are inadequate to cope with the problems of interactive computing. Efficient use of conventional techniques requires orderly procedures devised by skillful programmers who are aware of the advantages and limitations of the access methods they use. In the interactive graphics environment, a terminal user controls the sequence of data access, the volume of data handled, and the handling procedure itself. This situation resulted in the development of special software capabilities which must be maintained in the transition to a general purpose DBMS. The special capabilities consist of:

1. Support of arbitrary data structures. This is provided for in the DBTG recommendations as will be illustrated.

2. Functional techniques consistent with acceptable response time. This is a DBMS implementation consideration and the DBTG did not attempt to specify implementation techniques.

The following subsections identify two existing implementations of the required capability. Advantages associated with providing the capability within DBMS instead of retaining existing implementations follow.

3.7.2 CDC's Data Handler. - CDC implemented the Data Handler (Reference 3) to meet the mass storage quick-access demands of interactive graphics. The Data Handler is a set of FORTRAN callable COMPASS subroutines that optimize access to mass storage and perform incore list processing. The programmer can create and manipulate data structures much more efficiently than with conventional techniques because the Data Handler provides (as data) the addressing parameter of each substructure. With this parameter, the programmer can devise very efficient search strategies. Figure 3-16 illustrates possible data structures; each arrow indicates a direct record-to-record search strategy. (The record concept is signified by the word "bead" in this context.) Also, the programmer can make effective use of secondary storage space by defining the substructure of data words.

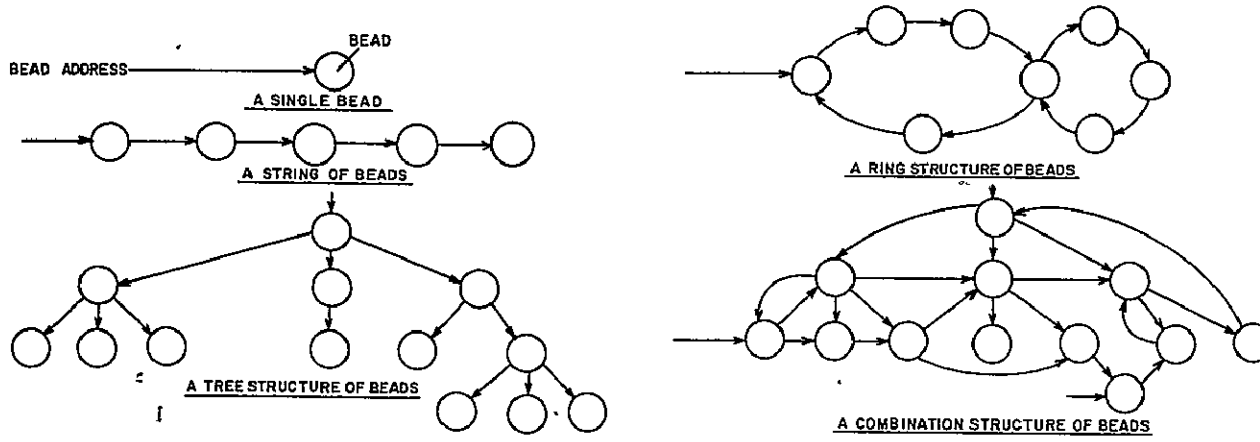


Figure 3-16. Typical Data Structures

Behind the scenes, the Data Handler improves response time and storage requirements through several techniques:

1. Copies of a number of "most used" data structures are maintained in core.

2. Structures read but not modified are not rewritten.

3. Dead space accummulated by deleting data is immediately available for reuse.

3.7.3 NSRDC's Interactive Data Manager (IDM). - The NSRDC Interactive Data Manager (Reference 15) is an improved version of CDC's Scope 3.3 Data Handler (Reference 3) and is written in FORTRAN except for a few COMPASS primitives. It additionally features:

1. Easy-to-use (for programmer) mechanism for handling pointer (addressing) data.

2. Separation of programming functions of data description from data manipulation.

3. Increased portability.

4. Reduction of core requirements and core-to-core transfers of data.

5. Elimination of program abort conditions.

6. Increase in efficiency due to improved data management algorithms.

7. Standard file formats to increase portability.

It however had several significant disadvantages which detract from its applicability as a general purpose IPAD software module, namely:

1. It presumes CDC's 6000-Series 60-bit words.

2. It accomplishes functions with FORTRAN that can be accomplished much more efficiently in assembly language (e.g. bit manipulation).

3. It duplicates certain operating system function (e.g. file management) thus circumventing host computer software which either is or is trending towards highly efficient, reentrant code.

4. The structure, type and contents of an IDM file are not available; much of this is embedded within the using program. This is contrary to the IPAD requirement (see Volume IV, Part I, Section 2) that every file:

   a. Shall contain its own (arbitrary) file structure.

   b. Shall contain its own (arbitrary) file contents/directory.

   c. Have provisions for a definition and units/coordinate system for every file variable (as applicable).

Considerable rework and extension to the IDM would be required to provide removal of these restrictions.

3.7.4 The DBMS approach. - The languages specified in the DBTG report provide all the advantages of structuring and efficient retrieval provided by the CDC Data Handler. Further, by making the OM independent of the external form of the data, it overcomes the disadvantages of the Interactive Data Manager:

1. Word structure specification is a DDL feature separate from executable object code.
2. All functions of packing, mapping, etc. are supplied by supporting (system level) software.
3. No duplication of capability or object code is needed. The OM/GPU programmer concentrates on the problems of using data. Specialists at the computer hardware level provide highly efficient software support.
4. The file structure of any data processed by DBMS is specified external to the program, consequently it can be processed by any other program.

Figures 3-17, 3-18, and 3-19 (taken from the DBTG report, Reference 4) illustrate the range of SET relationships which may be provided for in DDL declarations. In particular, Figure 3-19 indicates that arbitrary networks of RECORDs may be defined, providing all the logical flexibility required in the interactive graphics environment.

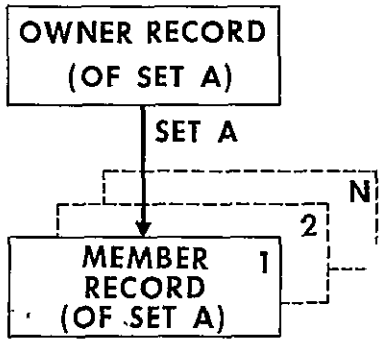


Figure 3-17. SET Representation of a Sequential Structure

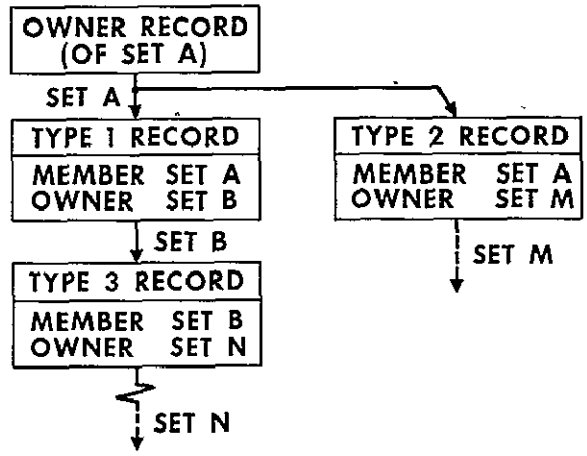


Figure 3-18. SET Representation of a Tree Structure

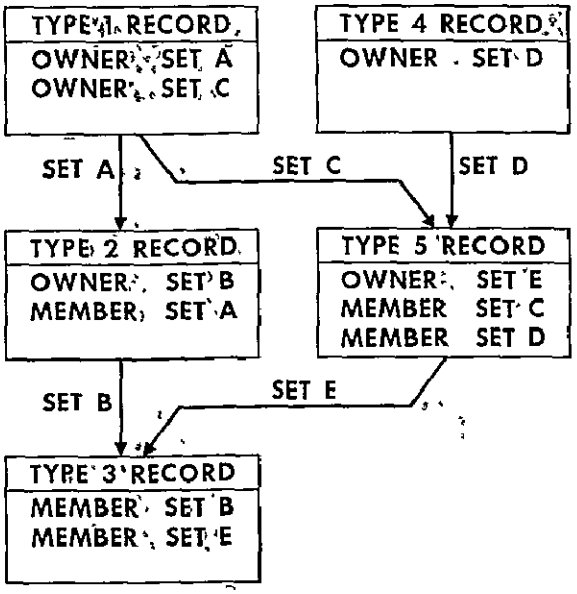


Figure 3-19. SET Representation of a Network Structure

3.8 Conclusions

Guidelines for an IOF utility were drawn from the objectives of the conceptual design. The requirements for such a utility were analyzed and a functional design was developed. A review of this work indicated that adopting this approach would entail large development tasks for:

1. An I/ODEF data description language.
2. A "compiler" for the I/ODEF language.
3. An IOF command language.
4. Interactive interpretation and support of the command language.

In addition to the development problems, the redundancy of data required by I/OPSONGES and corresponding sequential files was recognized as a distinct disadvantage (e.g., see mass storage sizing discussion in Volume IV, Section 5.4 of Part I).

The recommendations of the DBTG were identified as applicable to IPAD by CDC personnel who also revealed preliminary plans to implement a DMS as re-

commended by the DBTG. The IOF function is preempted by some of the objectives of such a DBMS. The introduction to the DBTG report (Reference 4) envisions a central data base:

1. To be suitable for processing by multiple applications and that can be interfaced by multiple languages.
2. To allow data to be structured in the manner most suitable to each application, without requiring redundancy.
3. Allow programs to be as independent of the data as current techniques will permit.

The functions of the IOF with respect to the IPAD user are not provided but the need for these functions are recognized and provision is made for future development. Curiously, CDC's implementation plans indicate that these (QP) functions will be implemented (in the form of QU/2) prior to the full DBMS implementation. Currently QU/2 has 6RM calls to access a data base. Ultimately QU and the OMs will use a similar mechanism (DML) to submit data base requests.

Further analysis of the DBTG recommendations indicate that NASA should sponsor extensions of the DBTG work to incorporate an industry-standard DBMS in the final IPAD design. The extensions are in the areas of:

1. Language development (see Section 5).
2. Special Purpose Utilities (see Section 7).
3. Special data handling needs of interactive graphics (see Section 3.7).

4 DATA BASE AND DATA BASE MANAGEMENT

IPAD data base and data base management are based upon the IPAD fundamental operating philosophy and principles discussed in Volume IV, Part I, Section 2. IPAD is basically designed as a project organized system structure to integrate the various disciplines of aircraft design within a single computer system and a common repository of project data. The principal focus of attention in bringing these disciplines together is to centralize the design data among them, reduce duplication when practical, and provide for effective dissemination and control of the data so that the entire procedure progresses in an orderly fashion with a degree of visibility and confidence to all concerned.

Section 4.1 presents the top level description and interrelationship of the data bases. Section 4.2 presents an overview of the organization of the data bases and data base management within IPAD from a particular implementation viewpoint. Section 4.3 presents a summary of personnel requirements for data base management. Section 4.4 presents a brief summary of the IPAD data bases. Appendix F represents a detailed set of requirements and possible design implementation.

4.1 Introduction

There exists a sufficient variation in the design approach and functional group organizations among the aerospace companies that imposing rigid standards on data definition is impractical (Volume IV, Part I, Subsection 2.2.3.5). Therefore, the system structure of IPAD must account for and permit the users to operate in a most effective manner considering their skills, design problems, and project organization. The resultant IPAD data base design requirements provide a minimum and effectual project organization for disciplinary interaction within IPAD. Within IPAD's boundaries, both the project management and the users can adjust and organize processes and data according to their specific requirements.

This is accomplished by providing:

1. IPAD utilities which can be sequenced by appropriate command files to tailor IPAD processing activity to the needs of a single discipline.
2. An IPAD general data base organization (and management) that provides the foundation on which the requirements and organization of a project can be constructed. These facilities will be available to all users.

In order to determine the most effective data base organization and management for IPAD, it is necessary to look at the typical personnel organization for an

aircraft design project.

4.1.1 Project Organization. - Figure 4-1 shows a typical aircraft design organization. Within the project there is a top level coordination group that exercises the overall control for project decision making and evaluating the design. This group decides what support activities, areas for design, etc. it needs. It employs and coordinates the various engineering design disciplines to carry out its design requirements and decisions.

Disciplinary groups in turn organize themselves to carry out their tasks and cooperate with one another where their design areas interface thereby overlapping and exchanging the appropriate design data. Users within the various disciplines perform their own (less formal) cooperation and exchanges of data.

Figure 4-2 presents a more formalized personnel arrangement that takes into account the project requirements and the formal requirements dictated by IPAD because of the nature of its computer-system dependence. However, the project objectives are accomplished by still adhering to the project organization and responsibilities.

In Figure 4-2, the Engineering Review Board (ERB) function remains the same as the original top level control. Between the ERB and IPAD is the Engineering Review Board Coordinator (ERBC) whose function is to (via IPAD):

1. Monitor the progress of various activities.
2. Interpret directives and data from the ERB and disseminate these to the disciplinary groups.
3. Evaluate results of design activity preparatory to review by the ERB (check on results, computer derived presentation formats, etc.).

A new personnel responsibility in the form of the Data Bank Administrator (DBA) is added to control the dissemination of design data between disciplines, the baselines for subdesign activities, etc. in accordance with directives from the ERB. This function has control over approved design data and is able to show current design status at any requested time. Part of this function's responsibility is to insure that the various users do indeed have the current design information available to them.

The functions of the members of the disciplinary groups remain the same within IPAD. However, their capabilities and control over their areas are enhanced via

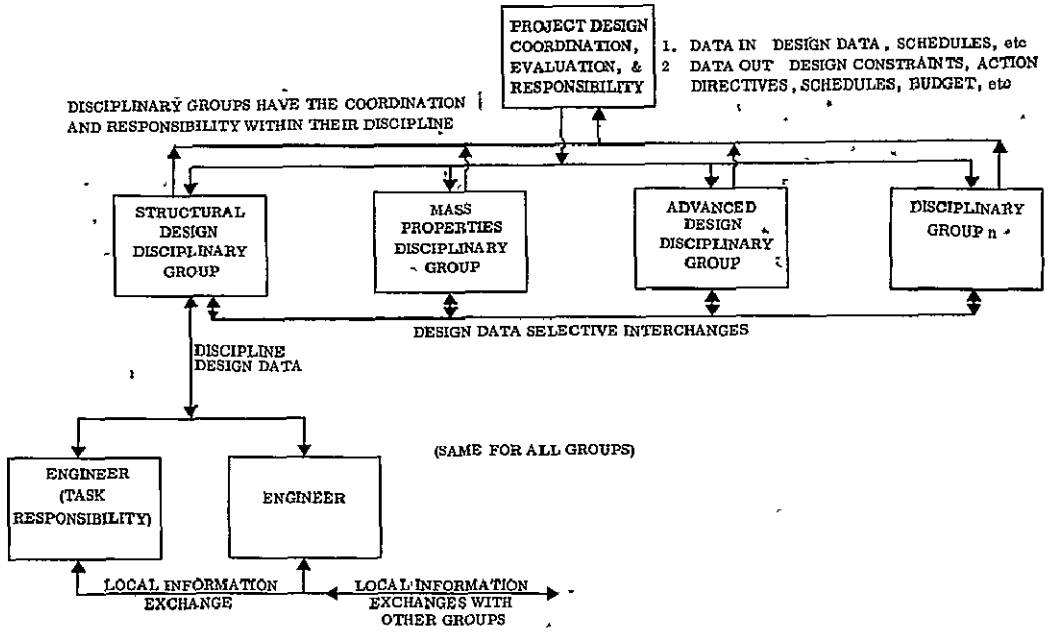


Figure 4-1. Typical Project Organization for Aircraft Design

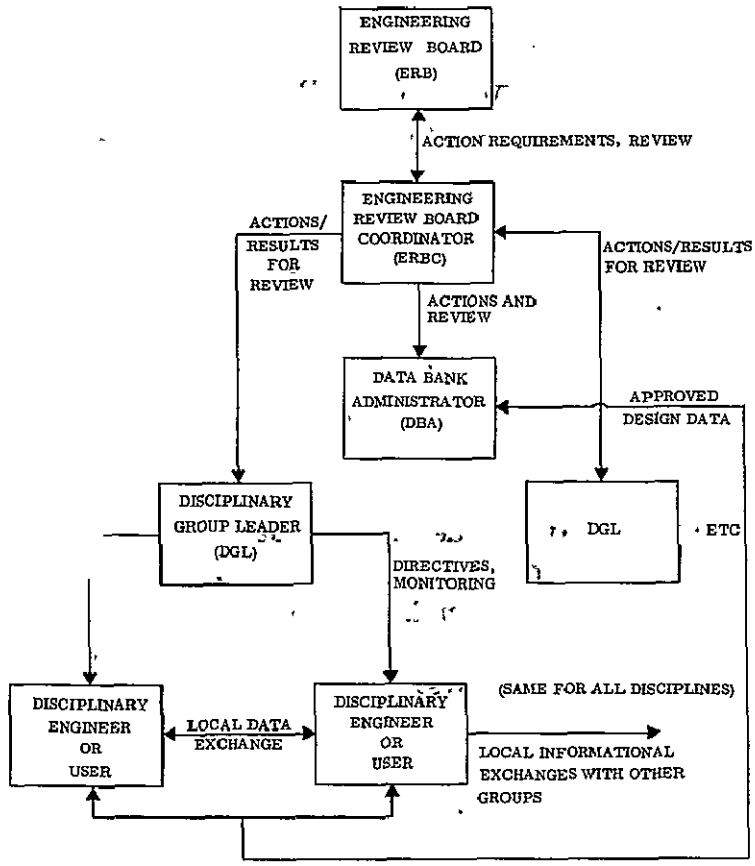


Figure 4-2. Project Arrangement for Aircraft Design Within IPAD

IPAD. The Disciplinary Group Leaders (DGL), for example, can monitor the accomplishment and method of accomplishment within their areas by the more objective means of interrogating engineering task histories and status information within IPAD.

The individual users can obtain the information they require more effectively and automatically by dealing through a common system and procedures.

Finally, Figure 4-3 transposes this organizational structure into the basic data bases within IPAD. The nature and function of these data bases are outlined in subsequent paragraphs. A summary of the definitions of the various data files are given in Subsection 4.1.2.2.

4.1.2 Terminology. - The following list and Figure 4-3 define the data base terms used throughout this section.

4.1.2.1 General IPAD data base terminology:

Data Base. Data base is used as a generic term to apply to any organized collection of data within IPAD. The term is used when no specific data base (subbase) is intended in the discussion.

Data Bank. The term data bank is project oriented and used to identify collections of specific data with the following attributes:

1. Their contents are applicable to all users of IPAD.

or

1'. Their contents are applicable to all members of a project,

and

2. They are required to be directly accessible by the user (or the IPAD support system) during the performance of his activity without any explicit directive on his part to the IPAD system.

Libraries. The term libraries is used to refer to collections of data/code that have the following attributes:

1. They consist of information applicable to a group of IPAD users (project members, or functional group members),

and

2. Explicit directions are required on the part of the user stating that a particular library is to be used.

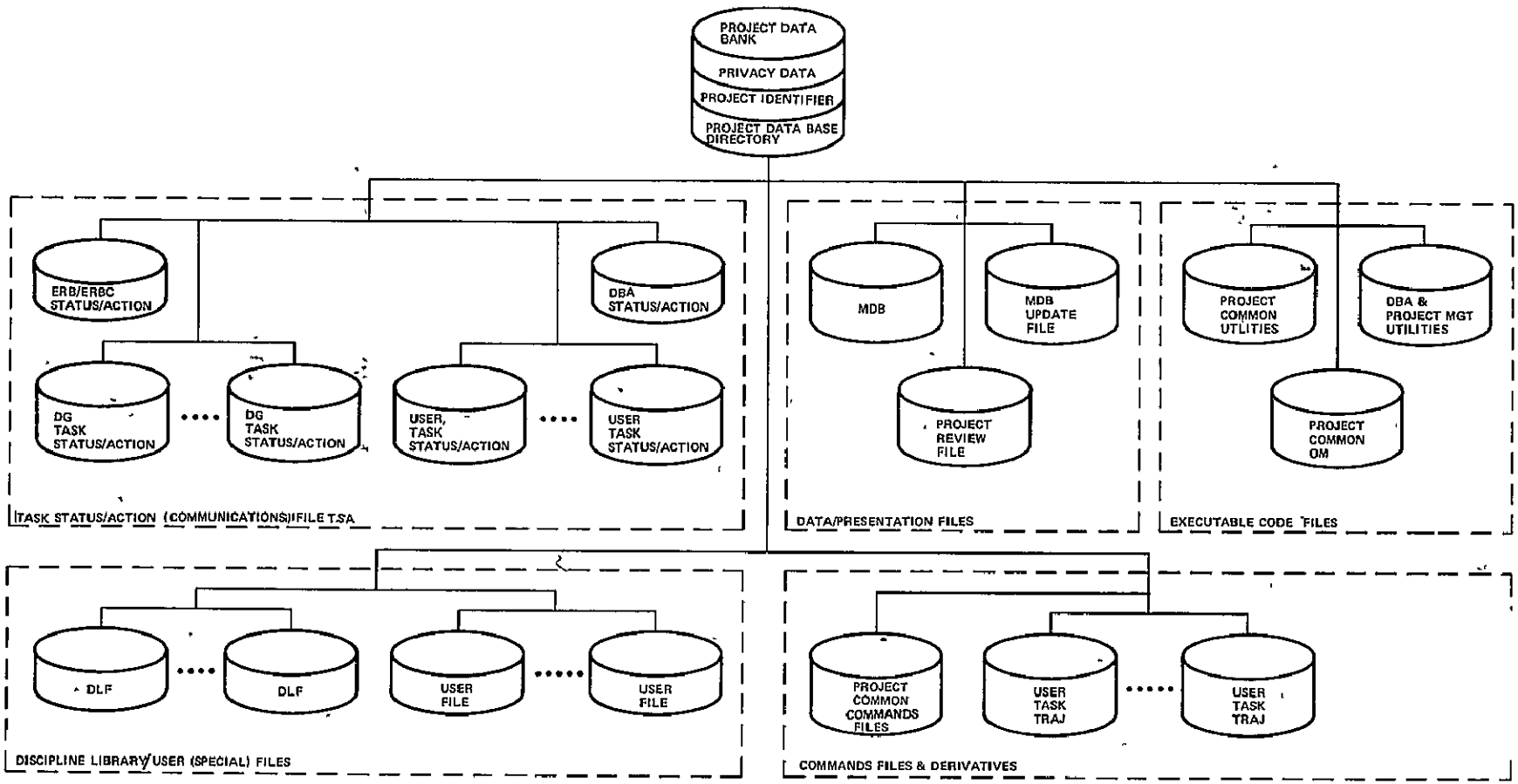


Figure 4-3. Data Bases Associated with the Project, an Overview

The term is also applied to common sets of data/code that can be made available to all users of IPAD and is normally maintained in a directly accessible form for the user. These particular libraries, their structure and contents are strictly under system control.

File. The term file is used as a qualifier on a data base to identify those data bases or portions of data bases which are treated by the IPAD support system as the highest grouping of information referable to by a user that provides him access to all information within the file.

Data Sets. The term data set is used to refer to named collections of information within banks, libraries or files directly referable by the users. Data set names may be recursive.

Data Items. The term data item is used to refer to the name representing valued data. The next lower division in the data contains the numerical values.

Data Base Identifier. A name associated with a data base which a user employs to locate information within IPAD.

4.1.2.2 Specific IPAD data base terminology: The following terms refer to the IPAD data base organization presented in Figure 4-3:

1. Data/Presentation Files:
   a. Multidisciplinary Data Bank (MDB) — Project collection of all specially approved design data that project members must produce or use.
   b. MDB Update File (MDBU) — Project data base file containing data designated for incorporation into the MDB subject to review and approval by the DBA.
   c. Project Review File (PRF) — Project level data base file that contains data and appropriate display commands that permits the ERB/ERBC to review the design data or generate presentation or report material.
2. Task Status/Action (Communications) File (TSA):
   a. Status/Action Sets — Collection of data sets within a project level file whereby messages, directives, and data base identifiers are transmitted between various members of a project. Specific data sets are assigned to the DBA, ERB/ERBC, each Design Group (DG) and each user.

3. Discipline Library/User (special) Files:

   a. Disciplinary Library File (DLF) — Collections of utilities, operational modules (OMs), design data, etc. that are used primarily within a single functional group or discipline.

   b. User File (UF) — Collection of data/programs used exclusively by an individual member of a project on any specific task.

4. Commands Files and Derivatives:

   a. User Task Trajectory (UTT) — Each UTT is a data set that summarizes actual activity on a task in terms of commands and data bases used.

   b. Command Data Sets — The command (e.g. TCS, TCSS) data sets are incorporated into a variety of data base files. They contain commands that can be invoked and used for specific purposes.

5. Executable Code Files:

   a. Project Common Utilities File — This file contains project common utilities, viz. those applicable to all users within a project. Utilities common to a single discipline (or user) may also be stored as data sets within other files such as DLFs.

   b. Project Common OM Files — This file contains OMs applicable to all users within a project. OMs common to a single discipline (or user) may also be stored as data sets within other files such as the DLFs.

   c. DBA and Project Management Utilities — This file contains the general utilities/commands that perform the general data base operations (see Section 4.3).

6. IPAD Support System Data Bases: The IPAD Support System Data Bases encompass those elements of an IPAD data base that are independent of projects and are maintained for either the convenience of all users of IPAD as for general management of the IPAD system.

   The data bases within the support system are discussed in detail in Appendix F.

4.1.3 Project level data bank, a functional description. - The kernel of data base organization and management is to integrate together the various members of the design team. There are two paths of project level exchange which organize a project's activity in an orderly fashion:

1. Design Data Exchange — Design data exchange is accomplished by establishing a central residency for approved design data and baselines accessible by all

authorized users. This facility, with its appropriate support processing and control is the Multidisciplinary Data Bank (MDB).

2. Action Directives and Status Information — Action directives are required to permit various members of a project to transmit information and the requests for information to other members. Status information is required to permit tracking and analysis of the various activities. This requirement is basically fulfilled via the establishment and usage of the Task Status/Action File (TSA), which contains the necessary information associated with each project member.

The following subsections discuss these data bases and their subsidiary data banks and their usages.

4.1.3.1 Multidisciplinary Data Bank (MDB): The MDB is the key data base, whereby the control of the design process data is maintained.

As an aircraft's design progresses through its various design phases — either serially or in parallel — design data controlling various aspects is generated. This data must be disseminated within the design project for utilization by or for controlling of design activities. The Multidisciplinary Data Bank (MDB) is the mechanism by which project level control is maintained over the design process. Each project has its own MDB.

The MDB is the repository of all project approved design data. In general, as the design activity progresses, design data will be altered and replaced. The choice of whether superseded data should be deleted or retained (appropriately labeled) is determined by the project management. It is conceivable that on some projects, the choice could result in only current data while in others, the choice could result in maintaining all data ever entered into the MDB. The meaningfulness and propriety of retaining all information as well as its implied complexity of usage is the responsibility of a project. Individual users operating disciplinary OMs within IPAD extract the current design data from the project MDB. Conversely, when design data that is required for other disciplines is produced and approved, the MDB must be updated and expanded.

The contents and labeling of contents within the MDB is strictly a project oriented function. IPAD provides the facilities for the construction of the MDB with appropriate labeling of data at the project's discretion . . . that is, IPAD does not force categorization of data into specific groupings. The individual project makes decisions — a priori or during design development — as to how the data is to be classified for access by the various users who have need of that data. As examples, the same type of data in one project may be classified in the same disciplines regardless of the

project. On the other hand, other types of equivalent data may be classified as belonging to separate disciplines in different projects. Finally, the same data may be classified as belonging to several different disciplines within the same project.

Figure 4-4 depicts a typical data arrangement within a project's MDB.

Category and subcategory identities and directories are used to organize the design data into the appropriate classifications for the project. The individual data items have classification data items associated with them that further identify the nature of the valued items and version classification.

The MDB is further organized to permit more than one design to be considered concurrent for an aircraft. Each alternative design is treated as a separate logical entity. From the viewpoint of a disciplinary user working within the MDB, the MDB appears as though it contained exclusively one design. As shown in Figure 4-4, the MDB has further data items to identify alternate designs. If common design data exists between designs, it will be referenced by all the alternative designs that require it. Control over modification to the common design data resides with the Data Bank Administrator (DBA) who must decide whether such modification is permitted.

In order to obtain required information from the MDB, the user executes a sequence of commands (which may have been prefabricated) to accomplish the data extraction. The commands sequence begins with the accessing of the data bank, passing the user through authorization checks and selecting the appropriate design alternative to place the user in the proper context. The user in the first step of retrieval symbolically drives down the appropriate category branches until he is within the data subset he requires. His next step in the retrieval sequence is to obtain the actual data he requires. The user may obtain the latest data or he may employ a procedure that permits him to obtain some previous data which has subsequently been updated. The procedure is represented schematically in Figure 4-5.

The decisions as to what data resides within the MDB and how the information is to be referenced lie solely with the project management. IPAD only provides the system structure to permit this. Therefore, MDBs among various projects may vary in the requirements for specifying what data is to be recorded. Some data types, declared to be part of the MDB for one project, may be considered strictly to be local files in another project. The responsibility for the organization and control of the contents of the MDB belongs to the DBA. Proposed modification to the MDB are inserted into the MDB Data Update file. In accordance with decisions by the engineering review board, the modifications are made at the direction of the DBA. No other user has modification access to the MDB. Figure 4-6 shows an abstraction of this process.

C-2

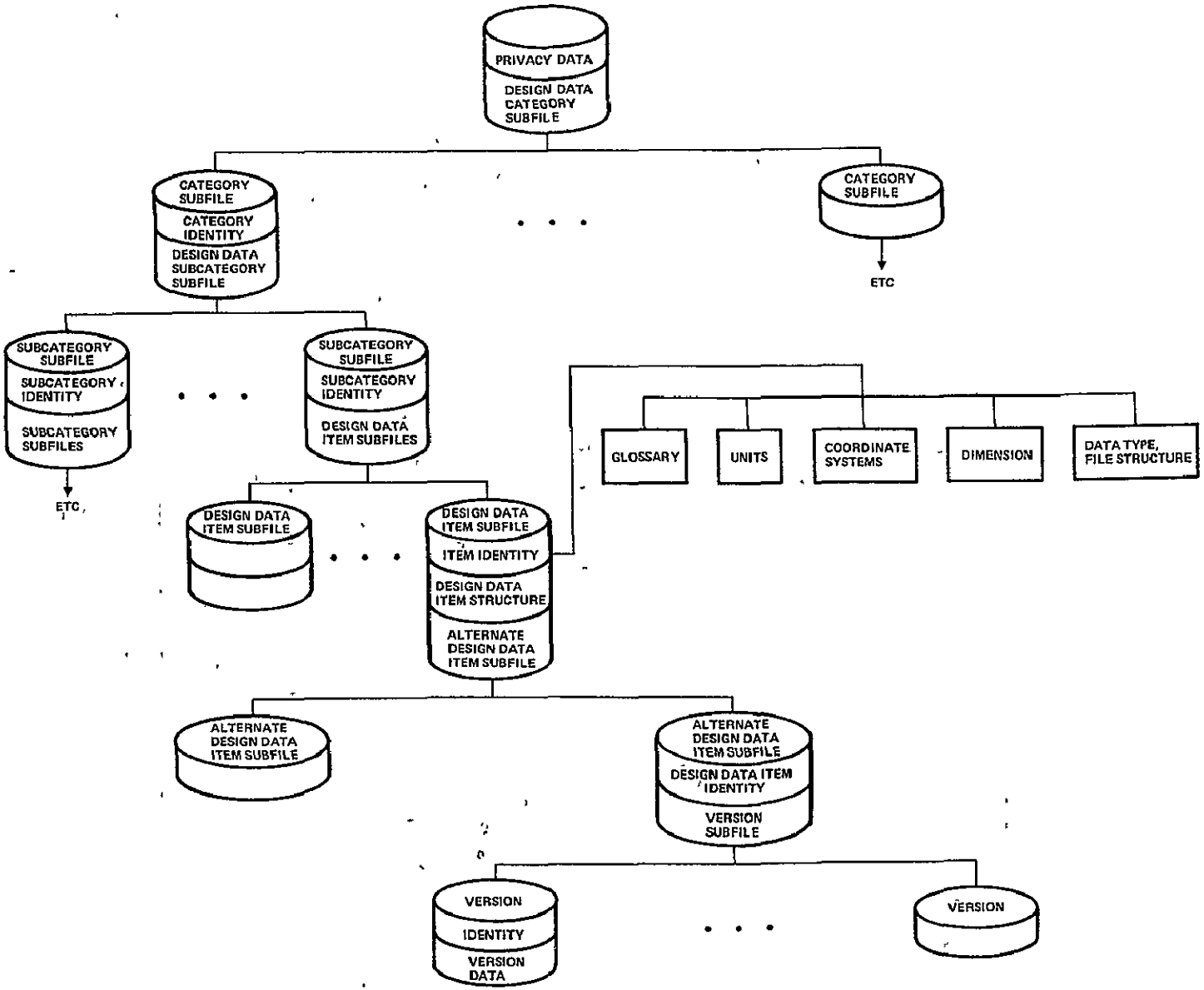


Figure 4-4. MDB General Data Organization

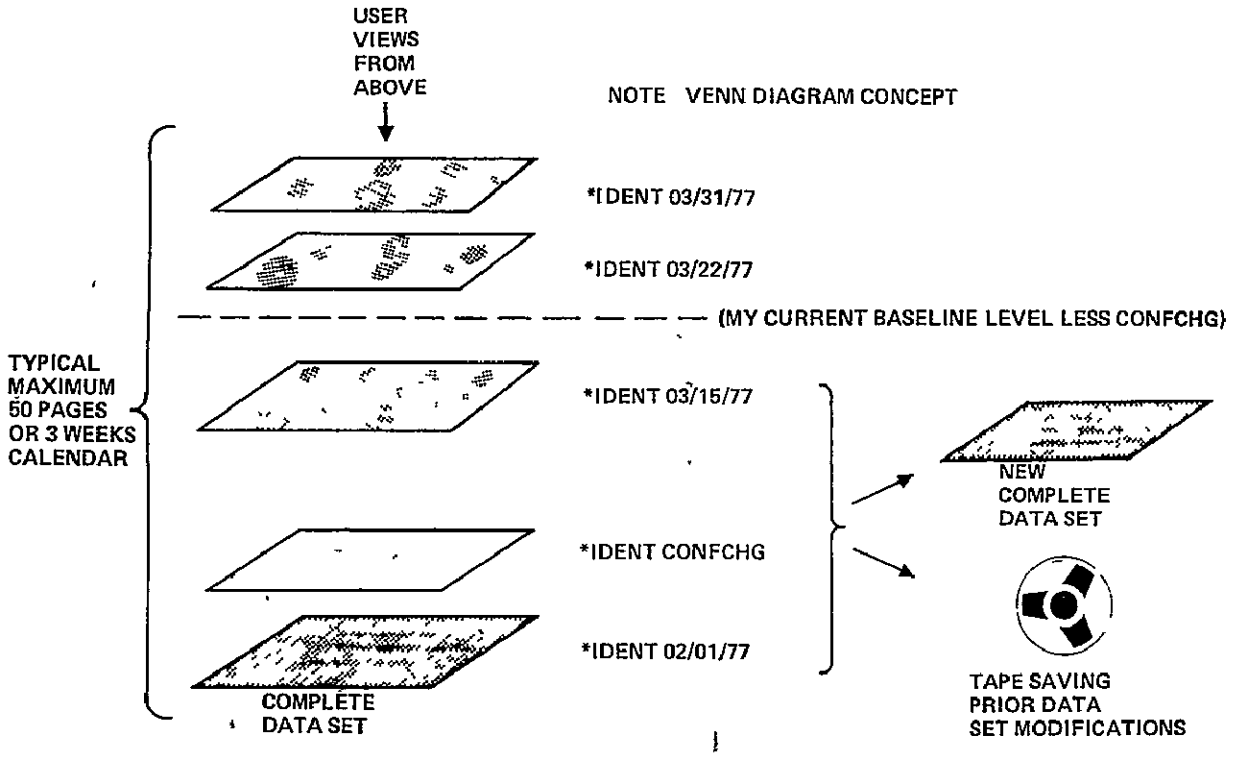


Figure 4-5. Accessing and Maintaining the MDB

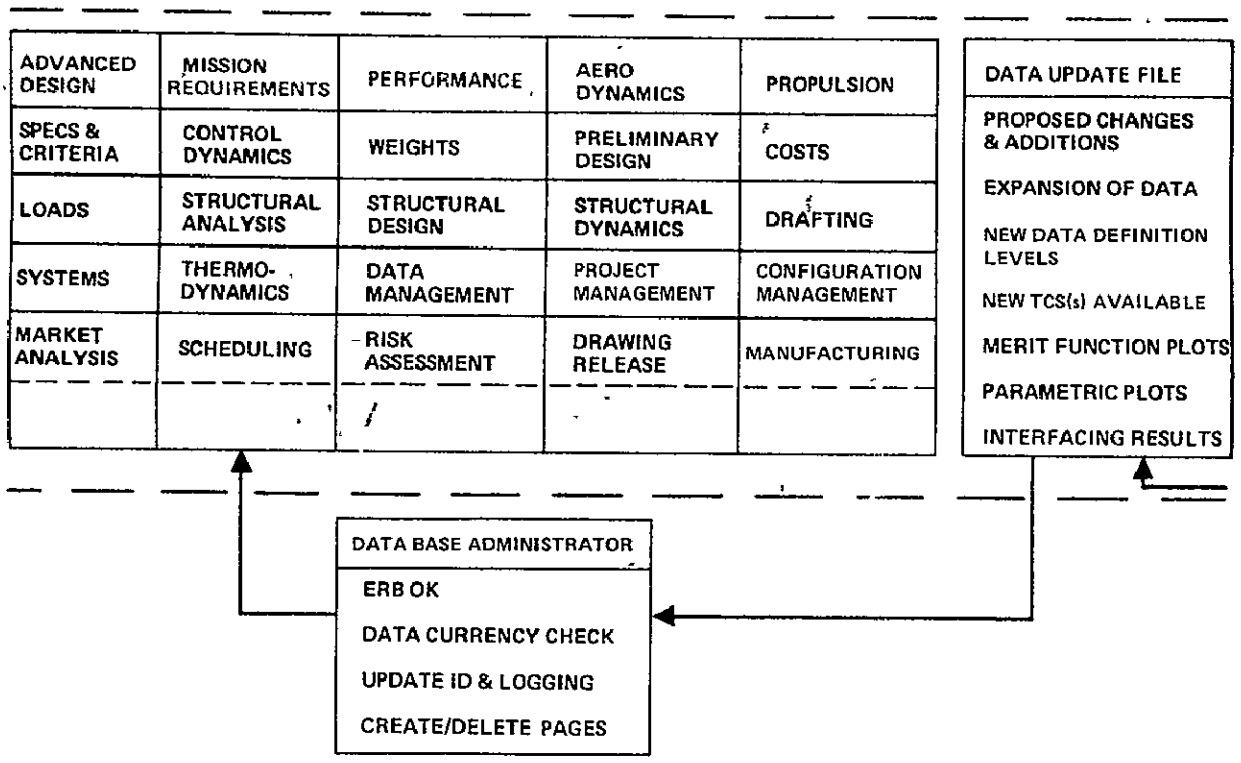


Figure 4-6. Updating the MDB

The MDB interface in the overall design data production process is depicted in Figure 4-7 which also shows the total data base requirements that may be involved in design data production.

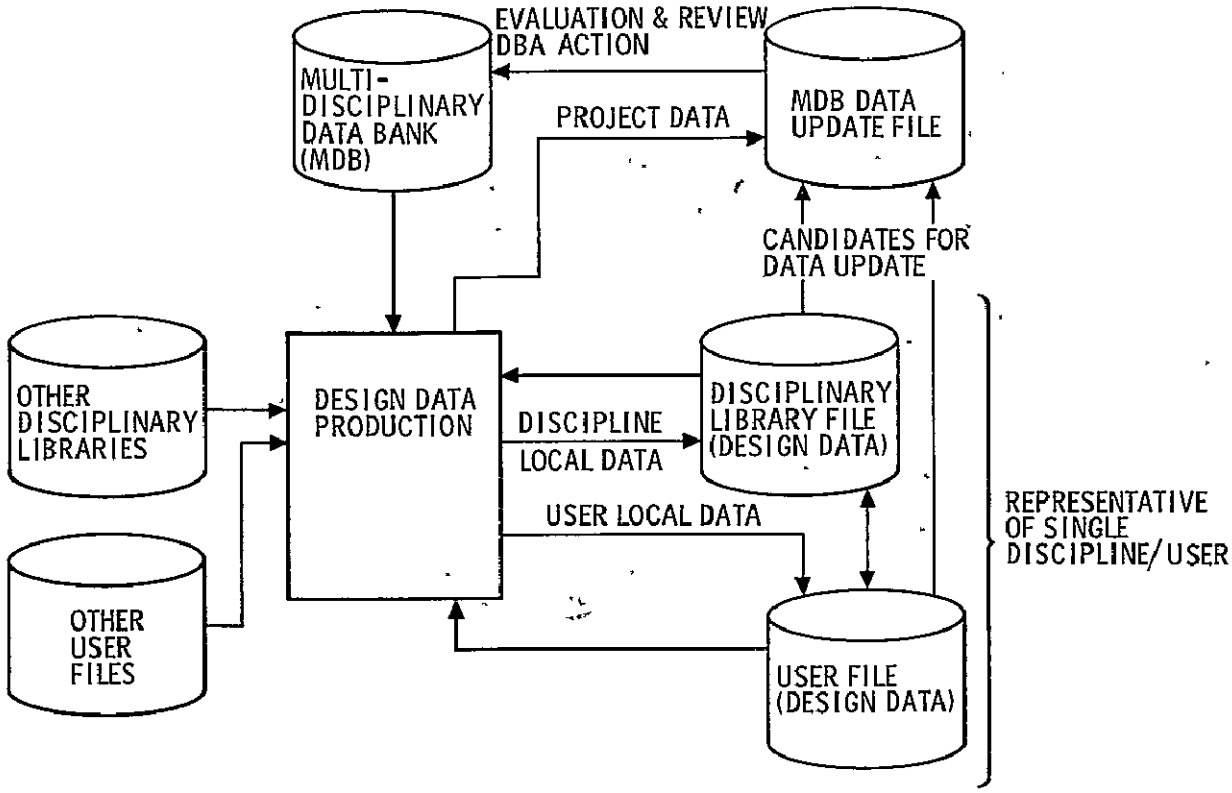


Figure 4-7. Design Data Production Process

The figure represents the inherent procedure (involving both man-man and man-machine communications) but does not illustrate the actual steps an individual user may go through. For example, he might already know that the data he requires is not and will not be part of the MDB and he can jump directly to the operation of providing it from his own local data base or those of other users.

What data is or is not to be part of the MDB and the procedure for updating it are unique to a project. It is the responsibility of the user and his supervision to insure that data generated by an individual user which is decreed by the project to be destined for the MDB does indeed end up in the MDB. The Data Bank Administrator (DBA) can only report, at any one time, what exists within the MDB and any possible updates that have not been incorporated.

4.1.3.2 MDB Update File (MDBU): The MDBU contains the design data that is a candidate for incorporation into the MDB. The general form of the MDBU is a presentation file which is depicted in Figure 4-8. The specific typical form is shown in

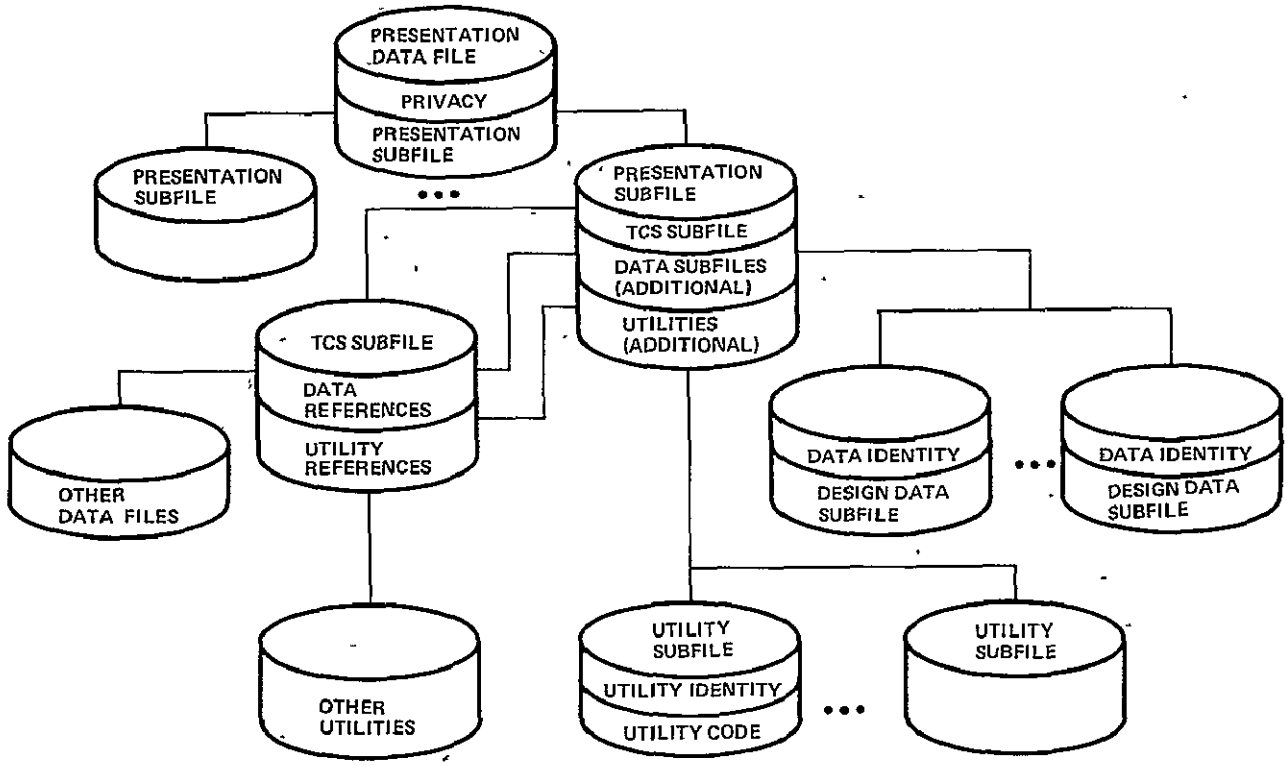


Figure 4-8. Data Presentation File, General Form

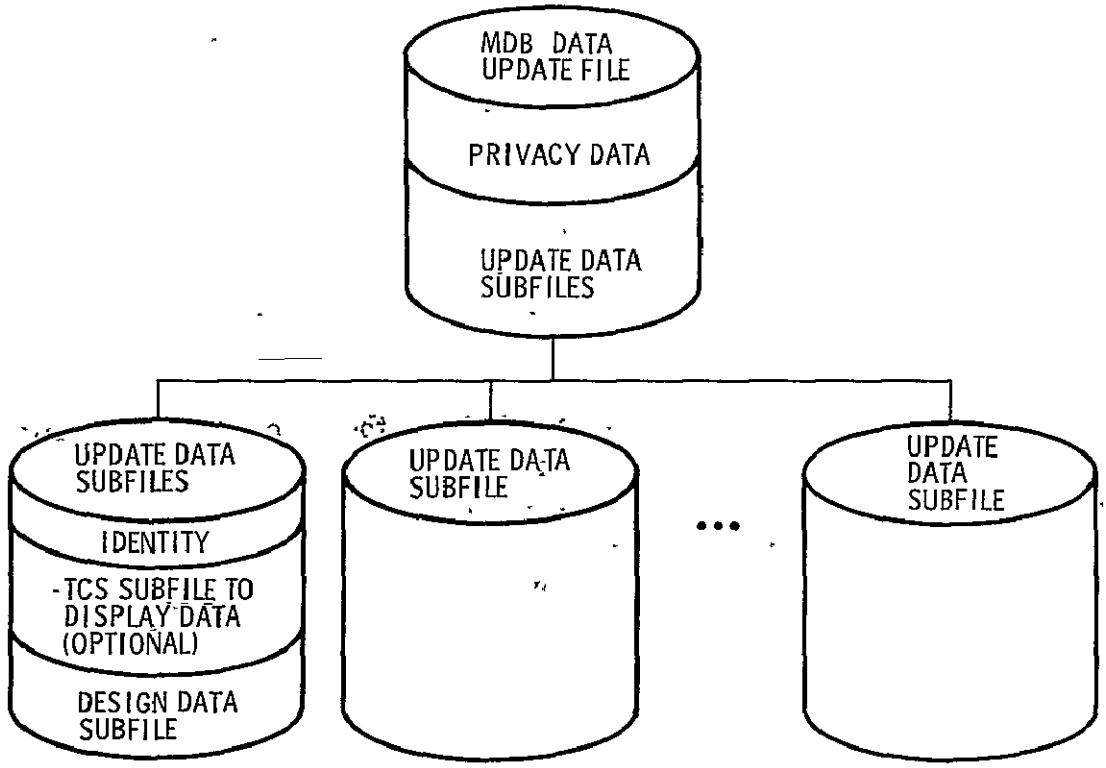


Figure 4-9. MDB Data Update File

Figure 4-9. This form of the file permits the DBA to use the TCS (QPS) and utilities sections, generated by the creator of a subfile entry, to review the data before incorporating or rejecting the data.

4.1.3.3 Project Review File (PRF): The PRF contains data that is to be presented to the ERB for review or for presentation or report generation. It also has the form of a presentation file as shown in Figure 4-8. The specific typical form is shown in Figure 4-10. This arrangement permits the data to be presented as controlled by the TCSs (QPSs) and utilities generated by the creator of the file.

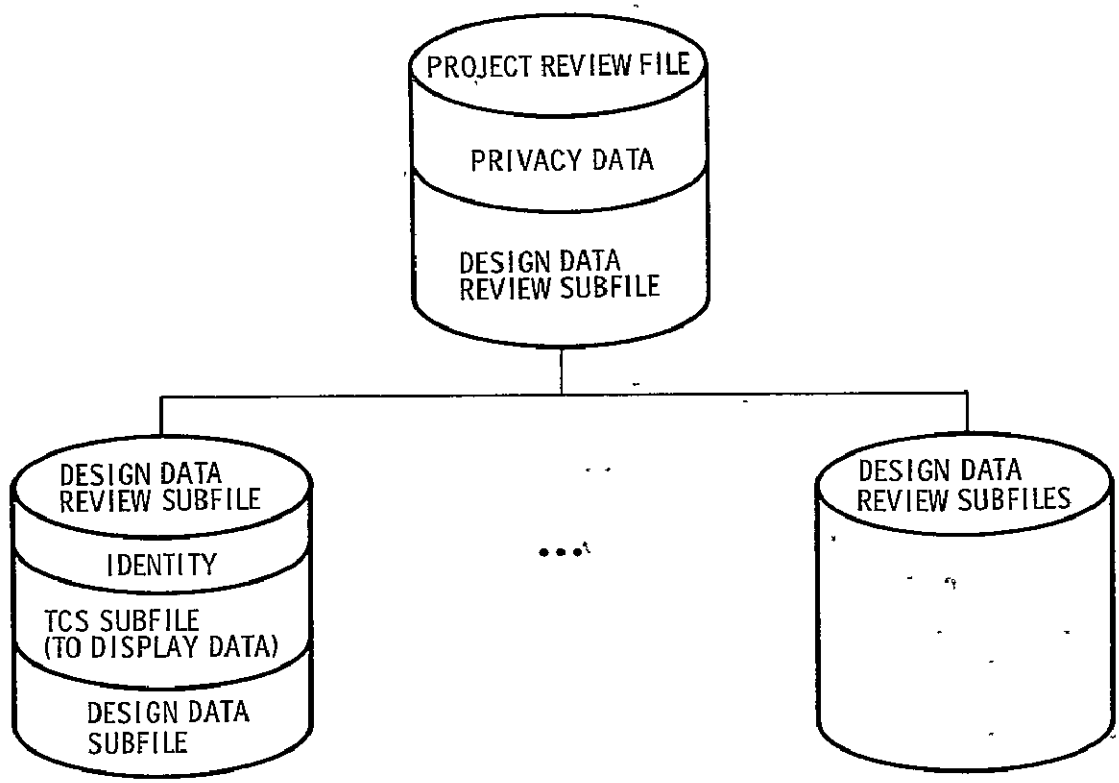


Figure 4-10. Project Review File, A Data Presentation File

4.1.3.4 Task Status/Action file (TSA): The Task Status/Action file (Figure 4-11) consists of a special collection of data files to facilitate the communication of action requirements between members of a project. These are the primary data bases used to maintain communications and control among various project members.

Five distinct types of users of these files are identifiable:

1. Engineering Review Board (ERB) — the ERB has the overall responsibility for design evaluation and direction of the design project.

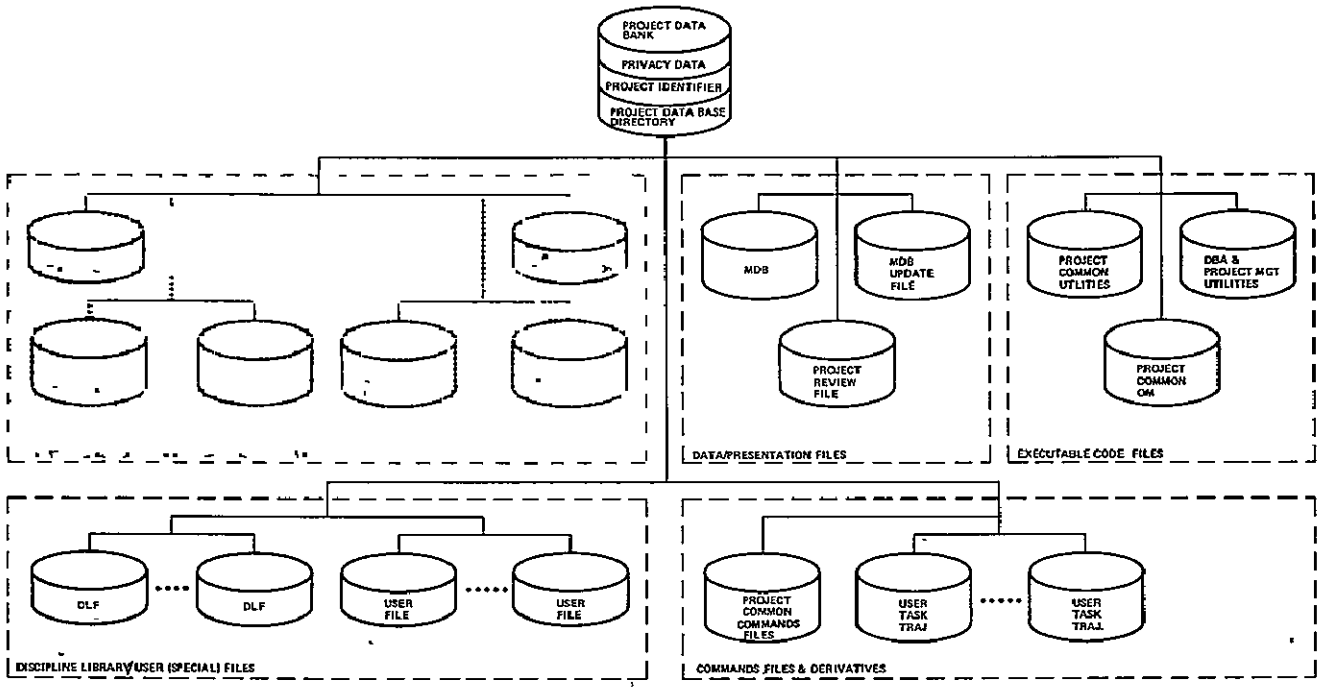


Figure 4-11. Data Base Organization, Communications Files

2. Engineering Review Board Coordinator (ERBC) — The ERBC has the responsibility for translating the requests of the ERB into the individual actions required on the part of members of the project and the responsibility for evaluating the material to be presented to the ERB.

3. Data Bank Administrator (DBA) — The DBA has the responsibility for maintaining the currency and accuracy of the MDB.

4. Disciplinary Users — The disciplinary users have the responsibility for carrying out the design process within their disciplines as designated by the ERB. The users also have the responsibility of preparing and reporting the results of their activity.

5. Disciplinary Supervisor — The Disciplinary Supervisor has responsibility for coordinating and monitoring the work within his discipline.

Three distinct types of status/action files are identifiable:

1. ERB Status/Action — Repository of actions required by or status of the ERB and results for ERB review.

2. DBA Status/Action — Basically a record of requests for update-of-the MDB and record of action taken by the DBA on requests.

3. Task Status Action — Directives to and status of various disciplinary users concerning action required on their part. Both the discipline and the individual users have status associated with them.

In general, entries into the various action files are maintained as project history. Some of this information can be selectively deleted by users.

Various types of IPAD users within a project can communicate via the Task Status/Action files for various purposes. Figure 4-12 shows the typical conditions for communication. The communications required to direct the top level assignment of the design activity are carried out between the ERBC and the individual disciplinary groups. Messages are exchanged in terms of actions required (assignments made) and responses to the requested activity.

The communications between a DG and its individual engineering users assigned to carry out the details of the design consist in general of simple messages delineating the tasks and responses specifying the status of the task. The engineering user who actually produces the design data reports the existence and location of the data for review and incorporation into the MDB via the Task Status/Action files to the ERBC and DBA. Additional types of messages, such as request for location of particular design data or a request to report on activities can be communicated via the files.

All TSAs have the same general data base storage requirements; Figure 4-13 shows the general arrangement. Each set of messages associated with the particular user are grouped together with sufficient information to identify completely the significance of the messages both for the user and their relationships to other users on the project.

Each TSA has its own privacy data to control operations on it. Each message has several parts associated with it for convenience of the user, such as message ID and sender/receiver identities. Access statistics are provided for each message to indicate whether the owner of the file has had the message displayed for himself (to make him aware of requirements).

The history portion permits the owner to maintain a past record of activities for his area.

Data file references are provided so that a user, where necessary, may know what designation he will have to use in order to obtain the data referenced by the messages.

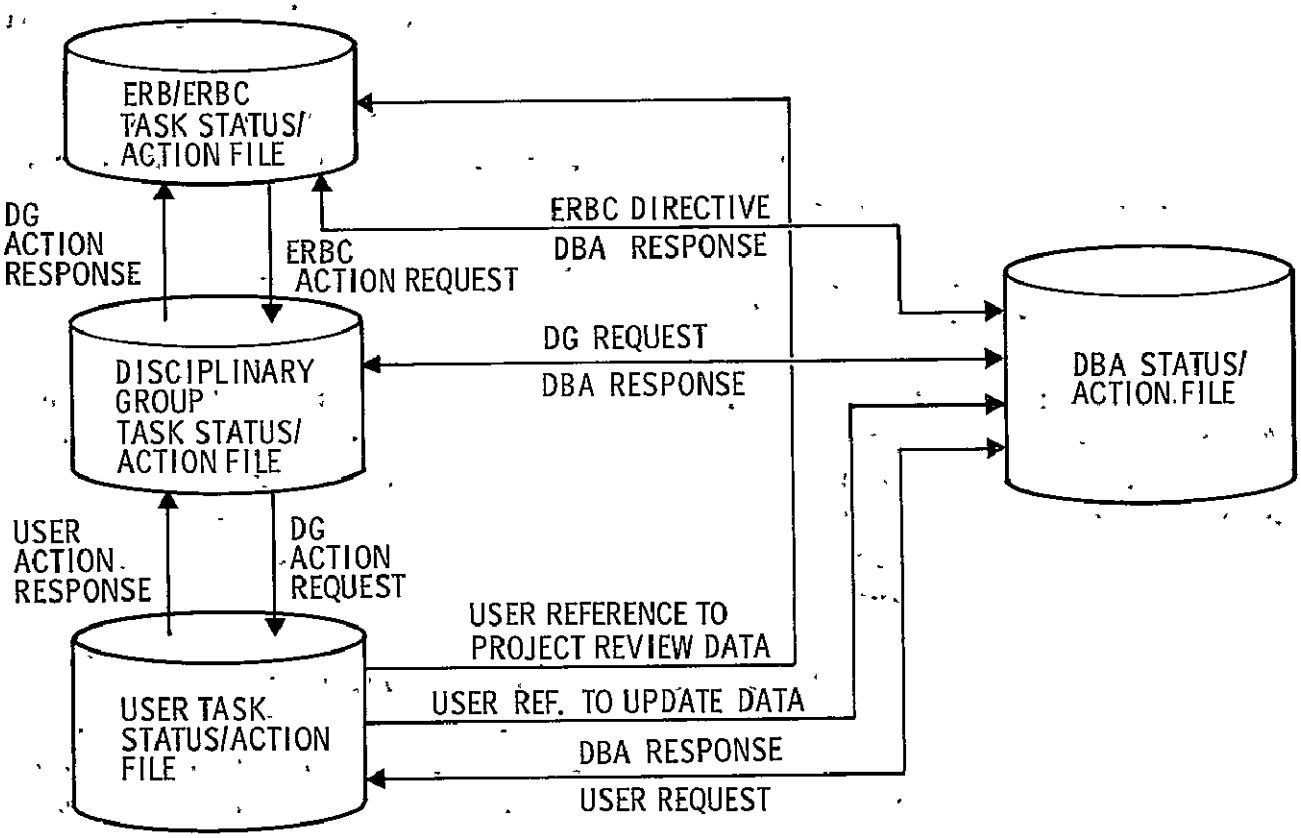


Figure 4-12. Communications File, General Relationships

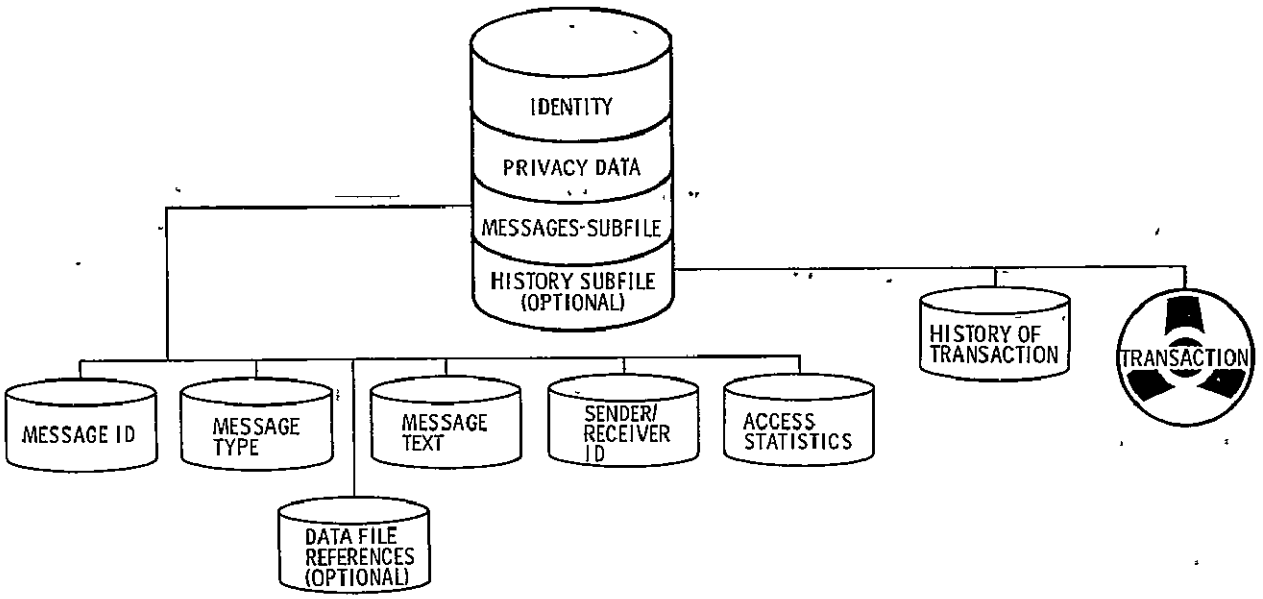


Figure 4-13. Communications File, General Form

Figures 4-14 and 4-15 show how the general organization is interpreted for typical DBA and ERBC operation, respectively. The data file references, for example, in the DBA's TSA permit him to locate and review particular data within the MDB Update File that is a candidate for incorporation in the MDB.

Figures 4-16, 4-17, and 4-18 show, for the three major types of TSAs, what a display of a message file might look like. In each message complete information is displayed to permit further action if required.

As an example, in the ERB's TSA (Figure 4-16) message reference number 01 informs the ERB that the aerodynamics group has produced the requested study recommendations associated with wing size. The actual data for review and incorporation in the MDB is located at an entry within the Project Review file which is identified by the name WING SIZE. The access status field shows that the ERBC has not yet viewed the message.

4.1.4 Disciplinary-level data bases, a functional description. - Below the level of project data base management the usage of the system becomes more variable. The level of organization is within the functional or disciplinary groups. The functional groups themselves establish all additional data requirements and procedures for their discipline. In general, the functional groups' individual requirements are classifiable into two categories (Figure 4-19).

1. Disciplinary Level — This level of information is common to all users within a functional group and is, so to speak, the property of that group. The controls and usage of the information is determined by that group.
2. Individual User Level — This level is provided for individual users for their own files and programs. The usage and controls belong exclusively to the user as long as he is interfacing properly with the project.

4.1.4.1 Disciplinary Library File (DLF): A Disciplinary Library File is concerned with operational support requirements to a single discipline within an IPAD project. The DLF is constructed during the course of operations within the discipline. Data within the DLF is available to all members within the discipline and the DBA constructs the appropriate data structures to permit the basic types of data to be accessed through direct reference by the user.

Although the system provides no utilities to construct cross-referencing linkages between the DLFs, data within the DLF can be made available to other disciplines/users. This can be accomplished by transmitting to the interested user, verbally or via the Task Status/Action file, the identity of the DLF and the desired data. The

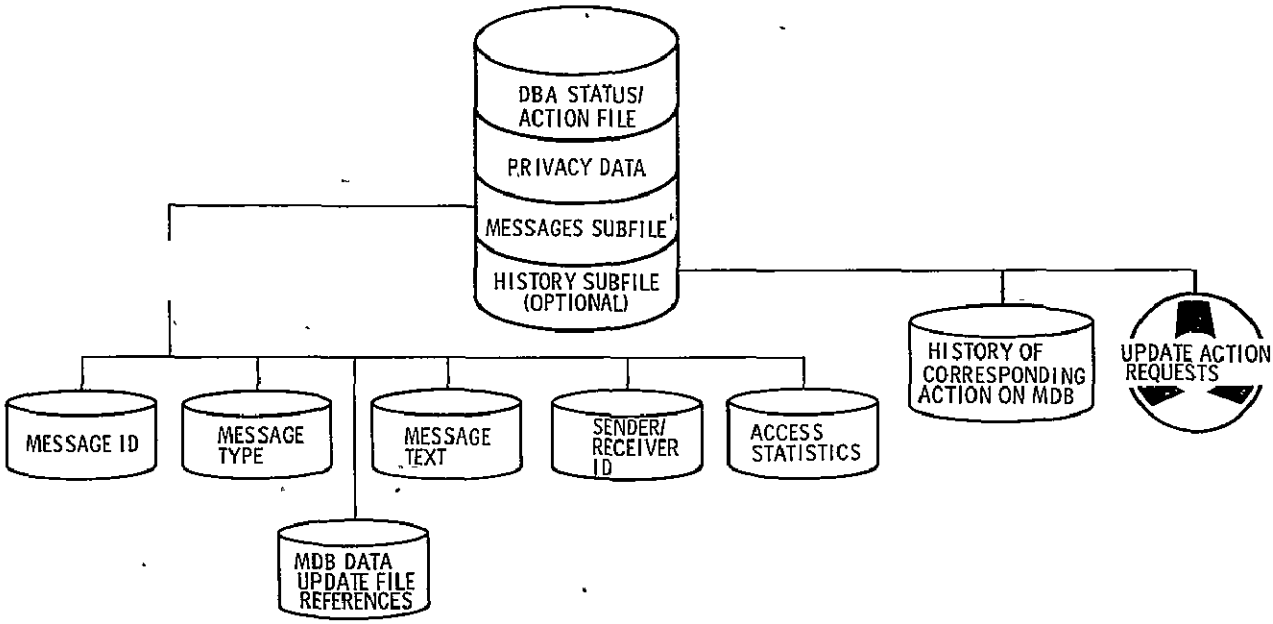


Figure 4-14. DBA Status/Action File

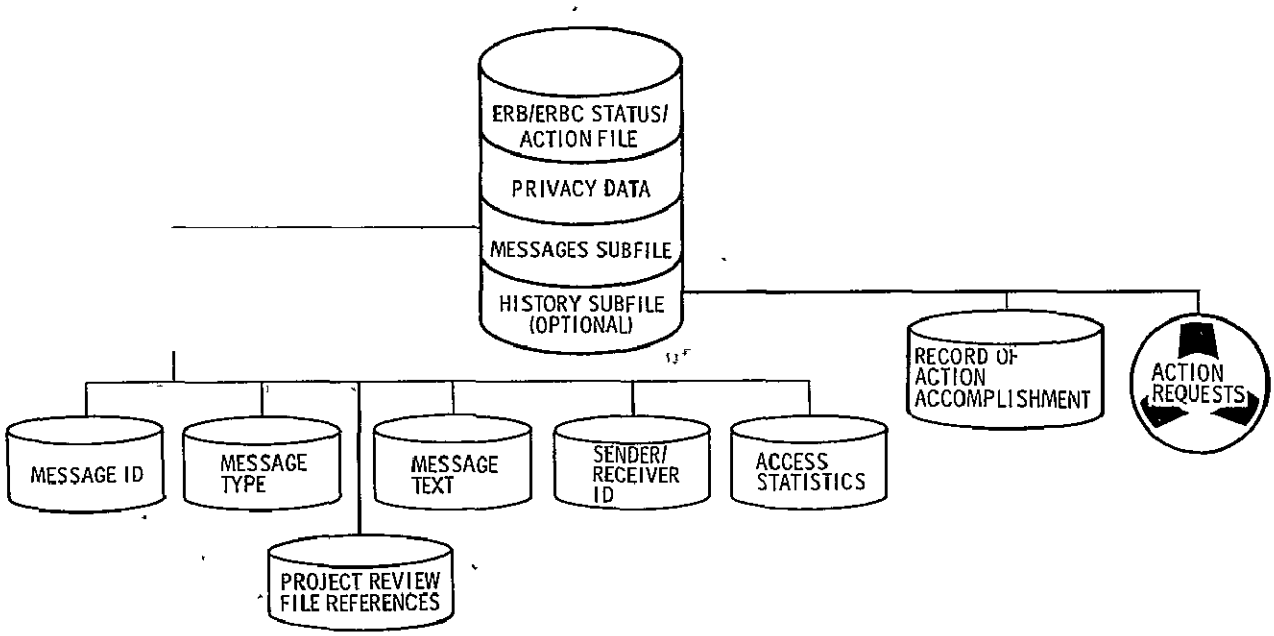


Figure 4-15. ERB/ERBC Status/Action File

MESS. REF.	MESS. ORIGIN	MESSAGE DESTINATION	MESSAGE	PROJECT REVIEW FILE REFERENCES	ACCESS STATUS
01	AERO	ERB	RESULTS OF WING SIZING STUDY WITH RECOMMENDATIONS	WING SIZE	0
02	ERB	STRUCT	LANDING GEAR STRUT LOADING PROBLEM. URGENT	LG LOAD	1
03	AERO	ERB	RESULTS OF CANARD LOCATION SUB-OPTIMIZATION	CANARD	0
04	COST	ERB	PROGRAM COST PROJECTION 5% OVERRUN		1
05	ERB	PERF	CRITICAL PATH ANALYSIS: 10 PROBLEMS, -4 WEEKS SLACK, URGENT		1
06	PERT	ERB	MANPOWER LOADING ANALYSIS: 7 SKILL, 11 LEVEL PROBLEMS		0
07	RAT	ERB	RISK ASSESSMENT: 4 CRITICAL AREAS, DECISION URGENT		0
08	SHELLY	ERB	NEW IDEA USE OF ALLOY 718 TO RELIEVE NOSE GORE TEMPERATURE PROBLEM		0
09	TAT	ERB	THREAT ASSESSMENT: (CLASSIFIED TITLE) INABILITY TO MEET		0
10	SMITH	ERB	MESSAGE: WILL BE OUT OF TOWN, CAN'T MAKE TUESDAY SESSION		0
11	PERF	ERB	REQUEST NEED DATE SLIPPAGE: CRUISE PERFORMANCE TO 4:18:77		1

Figure 4-16. ERB's TSA, Typical Display

user can then direct the IPAD system to access the appropriate DLF. He can then execute (or set up for execution) the TCSs (QPSs) required to extract the information required. The same procedures apply if the DLF is of value to users working on other projects within IPAD.

Figure 4-20 shows typical contents of a DLF. Since a DLF is normally restricted to a limited set of users, it is not necessary to make permanent residency assignments for it within a host computer system such as is necessary for central project data bases like the MDB. In the actual operation with a DLF, depending on magnitude of the

MESS. REF.	MESS. ORIGIN	MESSAGE DESTINATION	MESSAGE	MDB REQUEST REF.	ACCESS STATUS
17	MASS. PROP.	DBA	WING WEIGHTS RESULTING FROM SIZING STUDY 03/11/77 — 181	WING SIZE	0
18	FLT. CONT.	DBA	ERROR CORRECTION. HINGE LIMIT LOADS FOR CANARD	—	3
19	PROP.	DBA	WEIGHTS UPDATE· PROPULSION GROUP 04/03/77, URGENT	—	0
20	AERO	DBA	AERODYNAMIC TRIM DATA FOR SUBMODEL 3-311	TRIM 3-311	2
21	DBA	PERF.	PERFORMANCE UPDATE: DASH 03/29/77 * * * REJECTED ON CREDIBILITY * * * ACTION REQUIRED	DASH	1

Figure 4-17. DBA's TSA, Typical Display

MESS. REF.	MESS. ORIGIN	MESSAGE DESTINATION	MESSAGE	ACCESS STATUS
01	DBA	PERF	DASH· UPDATE COMPLETE & SUBMITTED 03/29/77 * * * REJECTED ON CREDIBILITY * * * ACTION BY 04/05/77	0
02	PERF	ERB	LOITER: UPDATE IN WORK, EXPECTED COMPLETION 04/07/77 * * * NEED DATE SLIP * * * NEW NEED DATE 04/18/77— 15:30:00	0
03	ERB	PERF	REQUEST: EFFECT OF WING SIZING STUDY ON DASH PERFORMANCE — USE UPDATE IDENT WINGSIZE WHEN AVAILABLE. NEED DATE 04/15/77 — 15:30:00	1
04	PERF	ERB	CRUISE. IN HOLD AWAITING PROPULSION GROUP WEIGHTS UPDATE	0
05	RAT	PERF	REQUEST: PROVIDE SENSITIVITY DATA ON EFFECT OF PROPULSION VARIABLE STRING ON DASH PERFORMANCE. NEED SUGGESTED COMPLETION DATE	0
06	MIKE	AL	MESSAGE: HAVE GONE TO DENTIST. BE BACK AT 4	0

Figure 4-18. Performance's TSA, Typical Display

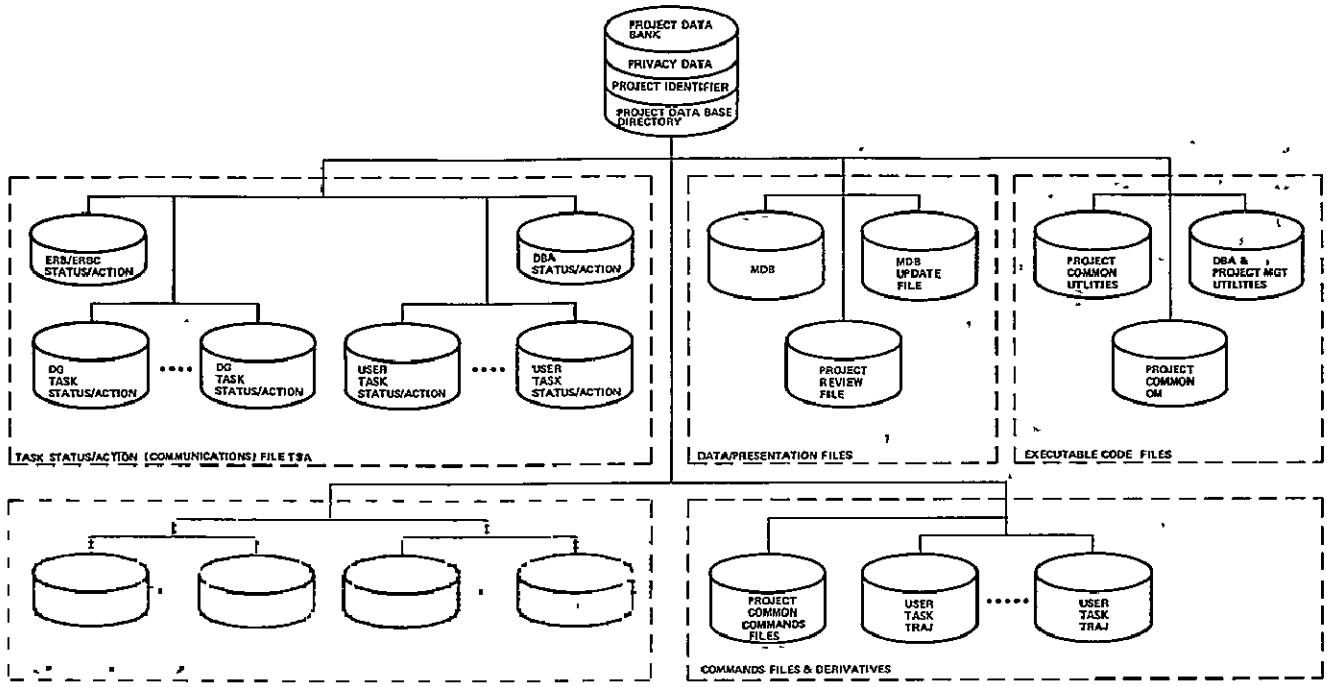


Figure 4-19. Discipline Library/User Files

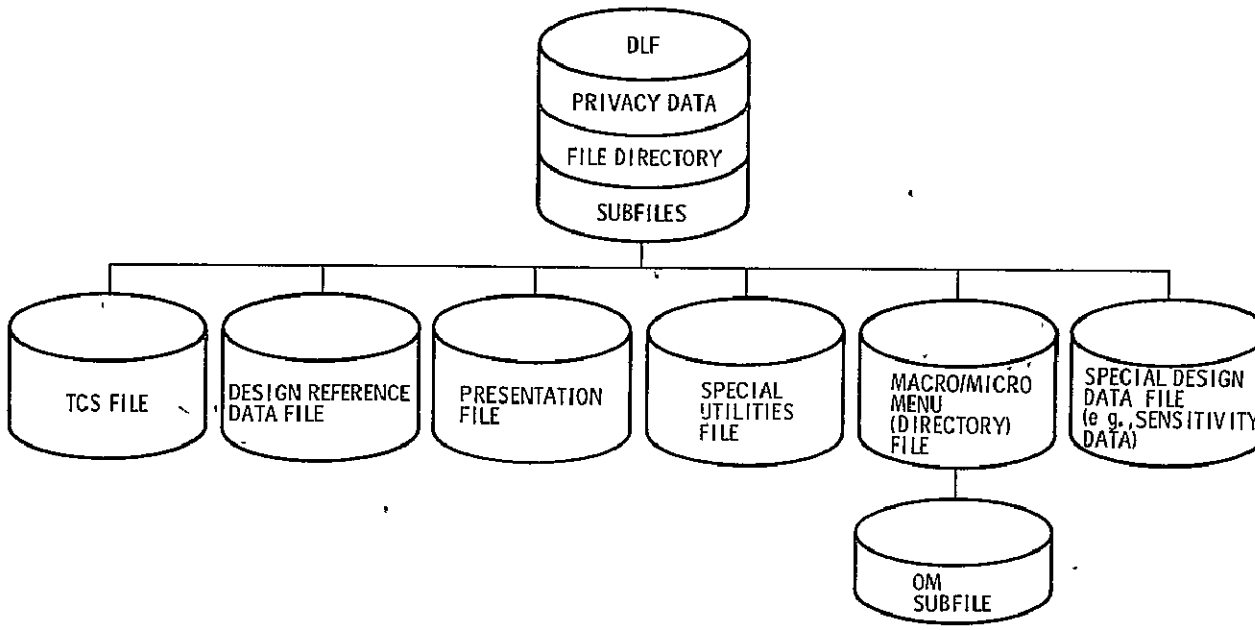


Figure 4-20. Disciplinary Library File (DLF)

groups' activities, portions of the file may all reside within the IPAD system and be directly referenceable.

Micro and Macro Menus are constructed exclusively by and for the convenience of the disciplinary group. The Macro Menu identifies to the user the portion of the aircraft design to which the OMs within his discipline will apply. The user employs the Macro Menu to localize the field of search to a small group of OMs. The result of selection within a Macro Menu may lead to other Macro Menus. At some level in the access process, the user will make a selection from a list (the Micro Menu) that directly references a particular OM.

Figure 4-21 shows a Macro Menu as it might appear in an interactive mode and its relation in usage to the Status/Action data and the resultant Micro Menu based on a selection from the Macro Menu. At the lower right is the corresponding OM's required input categories based on a selection from the Micro Menu. The structure and contents of the various menus are constructed by the disciplinary group.

The Operational Modules (OMs) within the DLFs are OMs that have been linked to the particular project's discipline. Other general purpose OMs will exist in the system (within the IPAD utility library) that are accessible by any user of IPAD (e.g. NASTRAN). The OMs within the DLFs have several parts:

1. Input Definition (IDEF) files — These are the information provided to users representing the input required for OM operation which the user must supply.
2. Output Definition (ODEF) files — These are the information provided to the users as to the output of the OM during execution.
3. OM file — The executable OM code.

Figure 4-22 shows the data relationships between data bases within the IPAD project and the execution of an OM. The user obtains the input from a number of sources both within his group and from other project sources. The disposition of the data likewise will go to various destinations.

Figure 4-23 portrays the overall relationships with the project data bases of Macro/Micro Menus and the corresponding OMs.

4.1.4.2 User Files (UF): The User File (UF) is exclusively associated with an individual user and is the repository of information exclusively under his control. The contents and management of the UF are his responsibility (some project or system level controls are however provided). The utilities provided by IPAD permit him

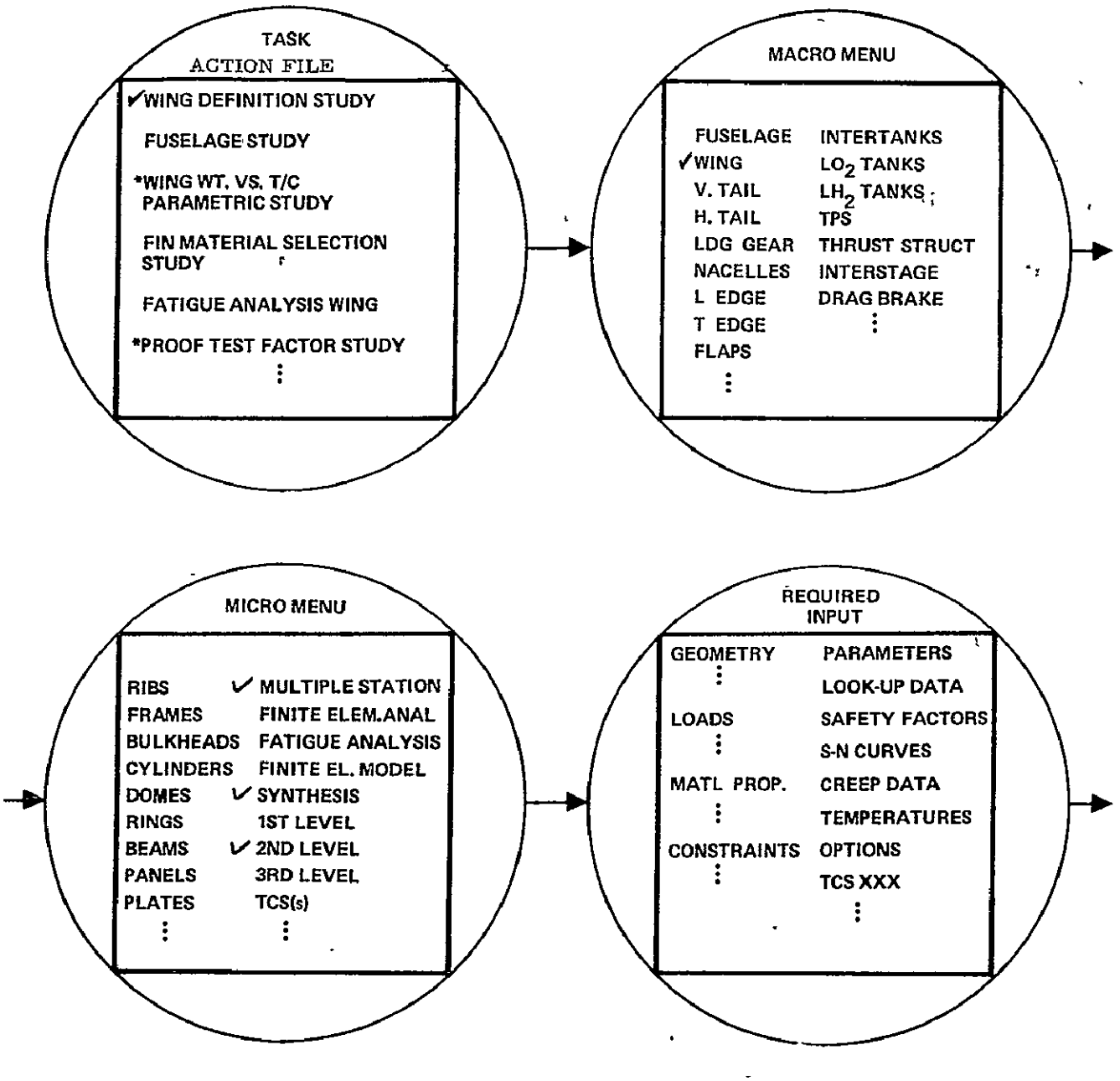


Figure 4-21. Sequence of Interactive Operation Illustrating Menu Usage, Structures Discipline

to keep track of the data and dispose of it as he chooses. In short, the UF is treated no differently than I/O files within the host computer's operating system.

Figure 4-24 shows typical contents of a UF. The user, in constructing such a file

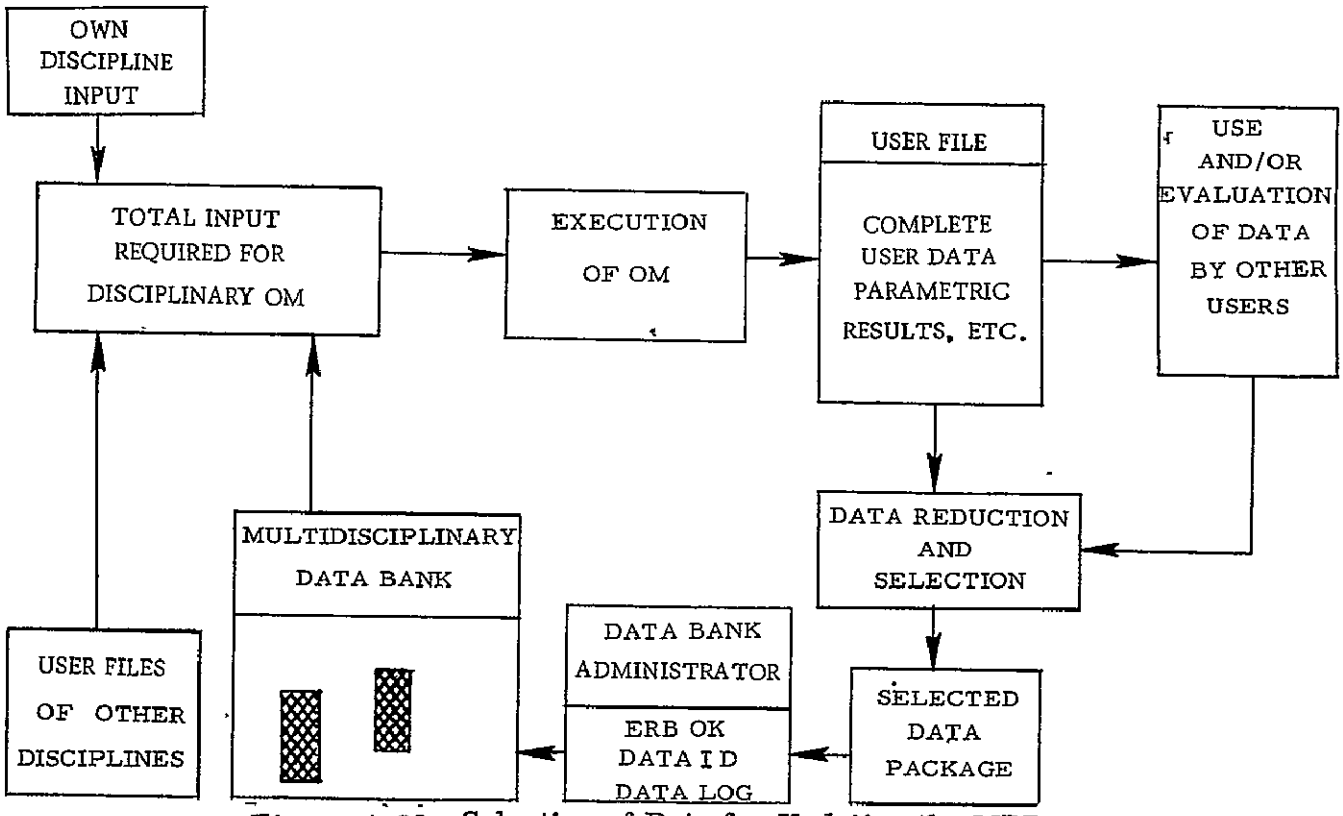


Figure 4-22. Selection of Data for Updating the MDB

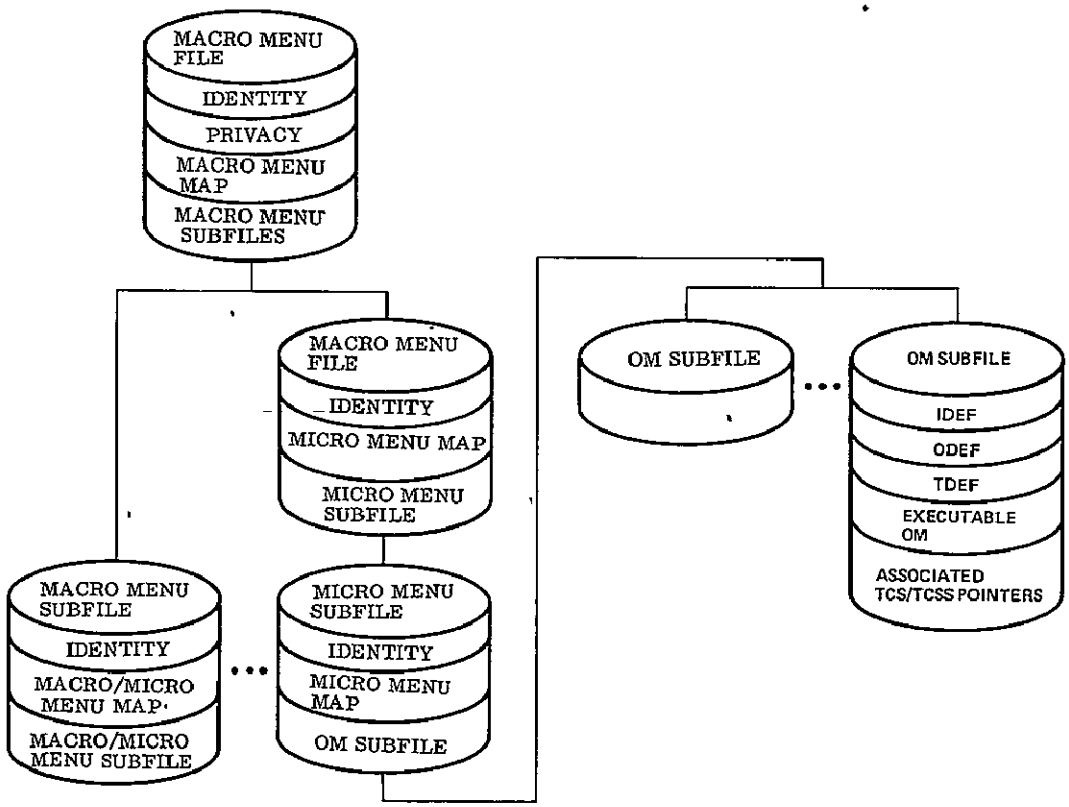


Figure 4-23. OM Files, A General Arrangement

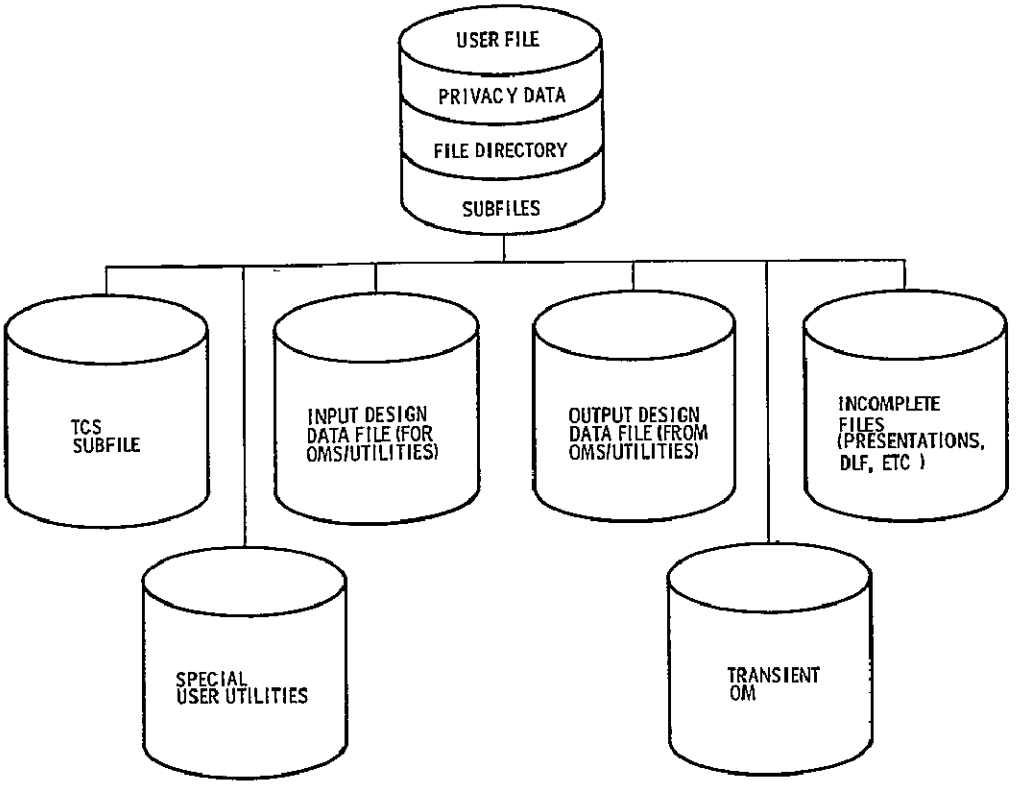


Figure 4-24. User File, General Arrangement

may name the various subfiles within it and provide himself with a more direct access capability to the information. In using this file, the user can employ if he chooses the various data base management functions that are used on the higher levels.

4.1.5 Command files, a functional description. - Command files (Figure 4-25) are, in general, file types used to store TCSs (QPSs) or their derivatives. Their basic function is for user support with prefabricated strings of operations that can be executed with a single command.

4.1.5.1 TCS/TCSS (QPS/QPSS) files: A commands file and its data arrangement is depicted in Figure 4-26. Although describable as a file it can be made a subfile to other files such as a DLF or UF or be represented as separate files for various purposes. Its components are depicted in the figure. Basically they consist of the individual commands (atomic level) and the data bases against which the commands are to be applied. Additionally, the data bases for operation can be of specified types with the actual data base substituted at time of usage. In general TCSs command general utilities of IPAD and the code is known for the TCS. In instances where special utilities are used, such as for a particular disciplinary group, provision is made for inclusion of corresponding utility code. To assist the users of a TCS/TCSS file, provision is likewise made for textual description of function and instructions for

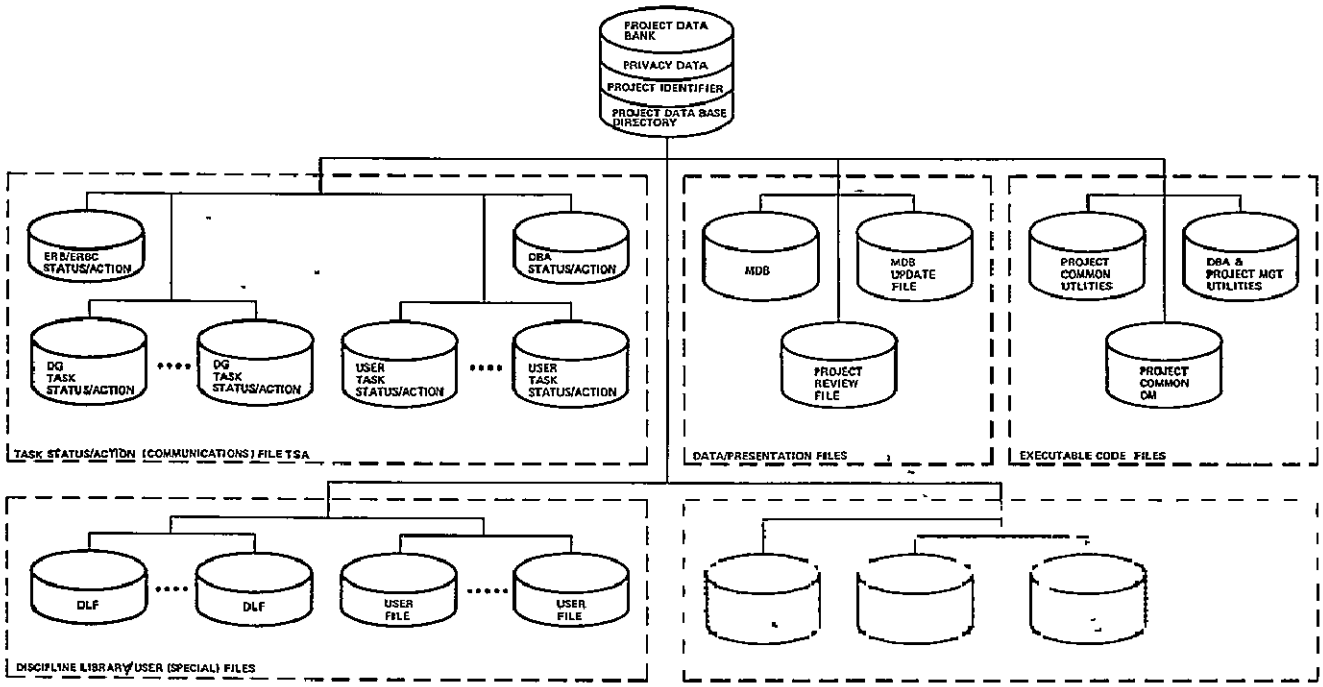


Figure 4-25. Data Base Organization, Commands Files (and Derivatives)

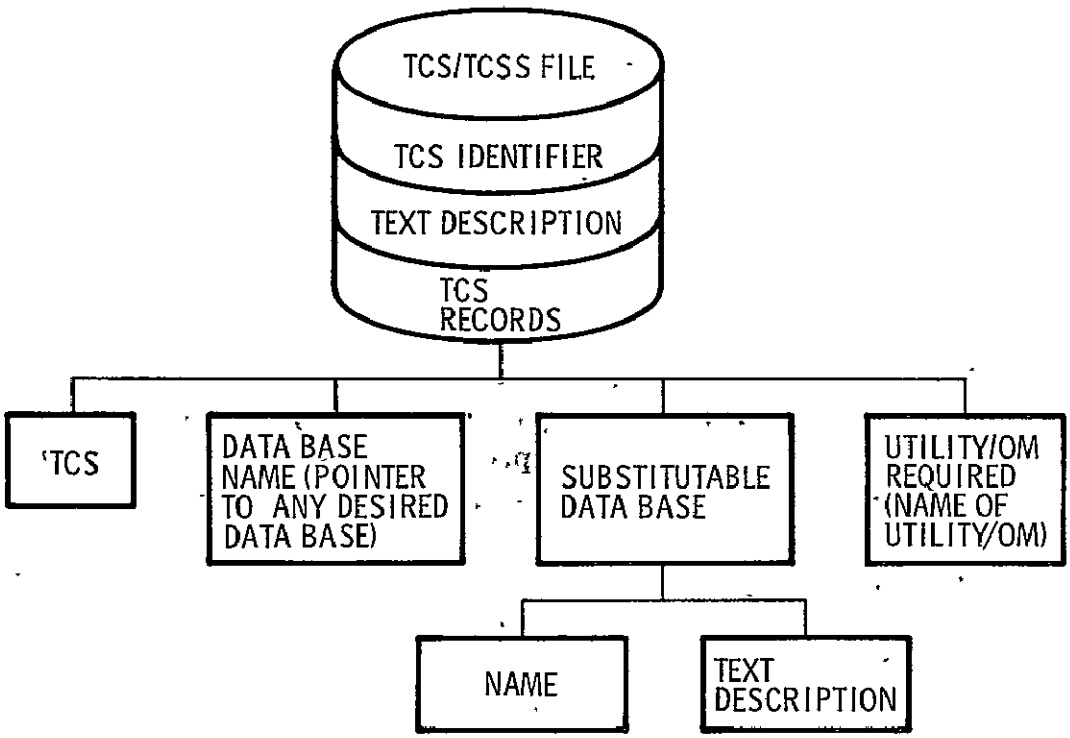


Figure 4-26. TCS/TCSS Files, General Form

using the TCS.

The QPS/QPSS files have the general requirements of the TCS/TCSS files. They however are recorded or prefabricated QP sessions and utilize the QP directives rather than the IPAD developed TCS. Further, they have no requirements for corresponding utility code since all directives of QP are satisfied by QP itself.

General QPS/QPSS can be constructed by various users, in particular the DBA, and transferred and modified into various files (DLFs and UFs) for specific purposes to satisfy the spectrum of usage.

4.1.5.2 User's Task Trajectory (UTT): Each user task as represented by a UF has automatically associated with it an IPAD task trajectory data set. This data set is automatically updated by the IPAD executive function. Its purpose is to permit the interrogation of the data set by responsible personnel who may wish to:

1. Monitor the user's activity to determine whether the task is being accomplished in the prescribed manner (proper data inputs are being used, appropriate design data being designated for incorporation in OM, computer time being used to accomplish task, etc.).
2. Duplicate a procedure of design development.
3. Determine what is required to do a job on the basis of actual operation.

The individual user has no authority to modify or delete items within the task trajectory. This is done only in response to authorized personnel action (an action which is itself recorded). The task trajectory also supports the user's request for current job status in the course of his job (a recovery consideration). Figure 4-27 shows the typical UTT file structure for a project. The individual UTTs are established by the DBA to exist concurrently with a UF.

4.1.6 Remaining project data bases. - Within IPAD the DBA can assemble other types of files for the convenience of the project. These choices of groupings are largely dependent on project requirements. Some of these types are as indicated in previous sections and could be made into subfiles of others (basically DLFs and UFs).

Figure 4-28 shows some of the types of files that can be generated for a project.

1. Project Common Utilities — always accessible for project users.
2. Common OM files — contains OMs applicable to many users, their access via the DLFs is achieved through references in the DLF's Macro/Micro menus.

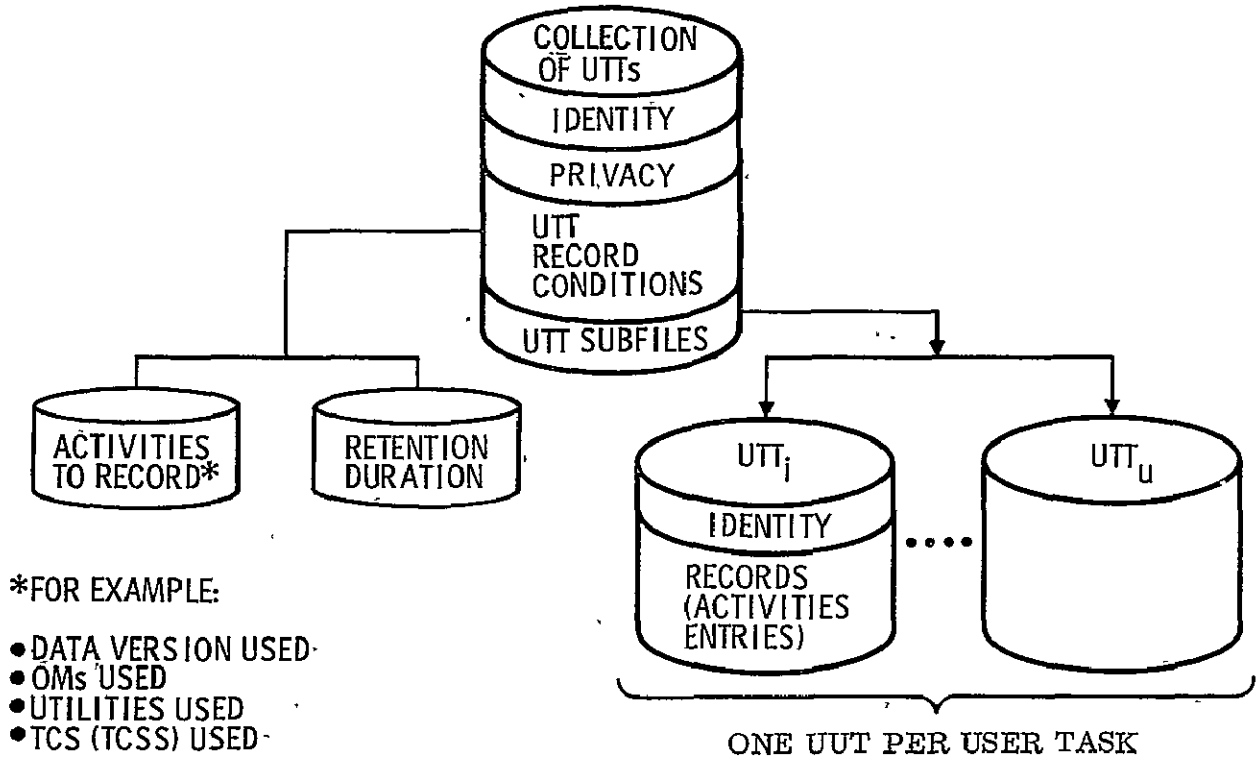


Figure 4-27. User Task Trajectory (UTT) File Structure

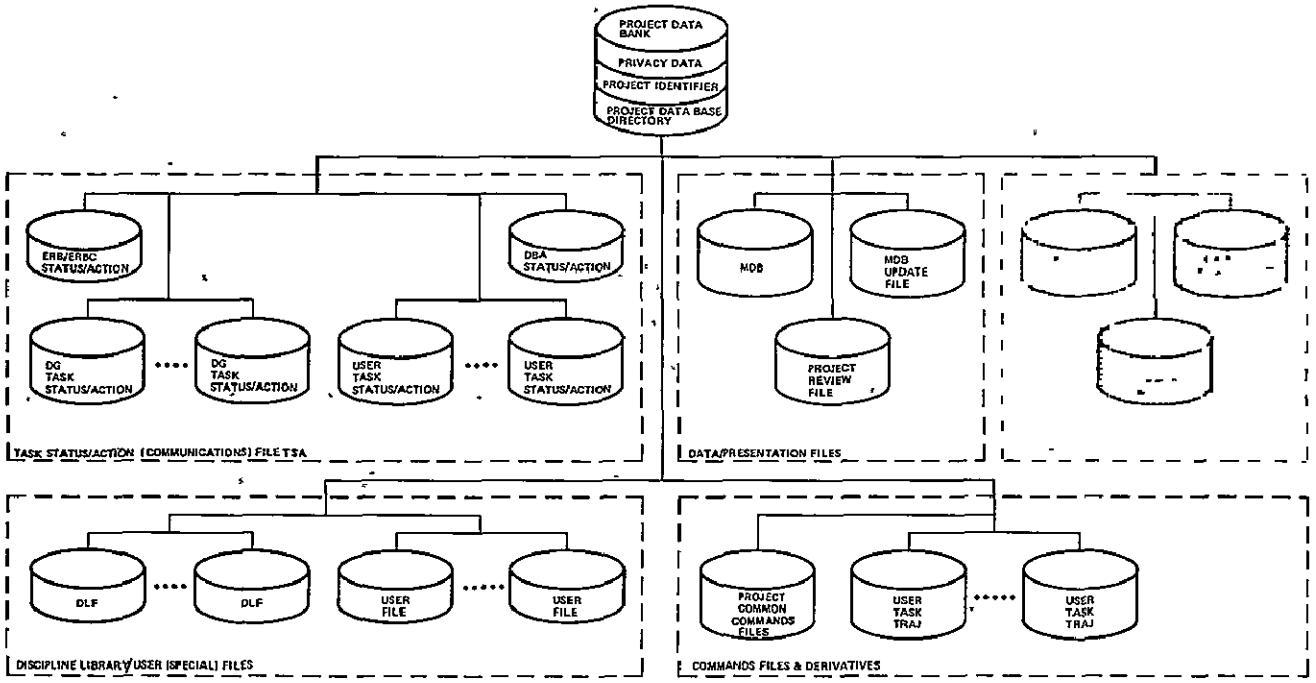


Figure 4-28. Executable Code Files

3. DBA and Project Management Utilities — constructed out of combinations of TCSs and utilities for the convenience of the DBA and project management.

The general forms of some of these file types have been previously described (e.g. OMs and TCSs).

Figure 4-29 shows the general form of the utilities when treated as a separate file. Other types of files are possible, and are dependent on the particular implementation of IPAD.

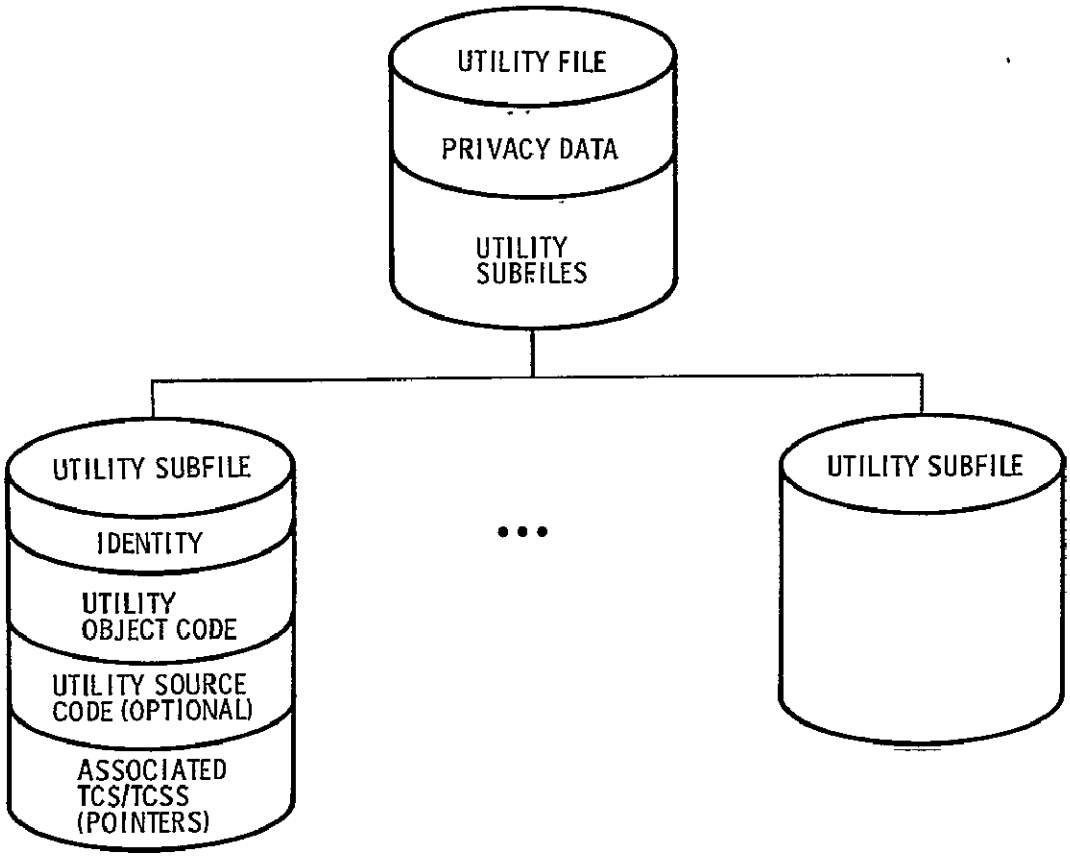


Figure 4-29. Utility File, General Form

4.1.7 IPAD support system data bases. — In operation, an IPAD facility must support several projects. IPAD users associated with a single project are operationally unaware that any other projects reside within the IPAD data bases.

The project data bases are isolated and protected from one another. There is no internal mechanism for referencing and accessing the contents of one project's data bases within the data bases of another project. If a user within one project wishes to use information that has been developed within another project, the user must employ

a procedure that involves the copying of the information from the data bases of the one project and then updating his project's data bases. The restriction is adopted because attempts to link together project data bases can lead to undesirable constraints on various project's DBAs in the organization of their projects for their exclusive project needs.

There is, however, a requirement for information that is useful to all projects within an IPAD facility. These data bases are designed, built, and maintained as part of the IPAD support system so that they can be referenced and directly accessed by a user in conjunction with his own project's data bases. For the users of IPAD, the builders and maintainers of IPAD provide the following types of libraries directly accessible by the various users.

4.1.7.1 IPAD general utility library: The Utility Library is the repository of data and software that are required in general to support any user of IPAD. Characteristics of any utility within the library are:

1. Executable only at the user's command.
2. Referenceable by a TCS.
3. General purpose and not directly dependent on any project design data.

The general structure of the library was shown in Figure 4-29.

4.1.7.2 IPAD general OM library: The OM Library contains OMs for common usage of IPAD projects (such as NASTRAN). Its general structure was shown as an OM file in Figure 4-23; it is bound to various projects by their DBAs via Macro/Micro menus.

4.2 IPAD Data Base and Data Base Management

After the general requirements of the IPAD data bases had been specified and further detailed, they were reviewed for implementation on existing facilities. Investigation of CODASYL's Data Base Task Group proposal, suggesting an implementation of a Data Base Management System (DBMS) and a Query Processor (QP) - to provide the users interactive interface to DBMS - showed that such a system, in general, satisfied all IPAD data base and data base management requirements. The remainder of the analyses of IPAD's data bases involved translating these requirements into operations performed within QP/DBMS.

Usage of the IPAD data bases for OM execution - i.e., a typical user's task - is discussed later in this volume in Section 1 of Part III. This section presents the IPAD

data bases in terms of their overall organization and usage via:

1. The Data Description Language (DDL) which provides the user of IPAD with a direct means of specifying his desired data base structure without requiring any additional coding or tailoring of the IPAD system for his particular usage.
2. The Data Base Management System (DBMS) provides the necessary code to support the DDL specifications and to perform the user's desired data base operations.
3. The user's direct interface with DBMS which is via a Query Processor (QP) that permits him to issue a sequence of commands to perform the desired data base operations.

These host operating system facilities were discussed in detail in Section 3.

Although the overview is presented in this section, the requirements details for each data base type and how these are satisfied by the appropriate combination of DDL, DBMS, and QP is presented in Appendix F. Specifications within Appendix F are expressed in terms of QP/DBMS/DDL. Although there are alternative methods of implementing a data base procedure using QP with DBMS, or DBMS alone, the specifications are limited to QP rather than proposing additional utilities in place of existing facilities. The actual specifications (Appendix F) are intended to present the overall structure and usage of QP, DDL, and DBMS as applied to IPAD. A Data Base Administrator (DBA) who follows these specifications will be able to satisfy his requirements for data base management within IPAD. However, since the system has sufficient generality and capability, the DBA is free to exercise considerable judgement and variance in usage of the specifications. To a user, there are three major components which he designs and utilizes:

1. Query Processor Session (QPS) - This is a sequence of QP directives designed to accomplish a designated data base activity.
2. SCHEMA - This is a DDL description of the total data base which the user is addressing.
3. SUBSCHEMA - This is a DDL description that interfaces with QP and defines and limits a particular QPS and the user to the immediate portion of the data base of concern for data base processing.

The operational relationship among these components are as follows:

1. The SCHEMA is designed first to describe the total residency requirements of a particular data base (e.g., IPAD aircraft project(s), IPAD Support System).

2. Data base operations required are designed as QPSs. To isolate the total data base only to that portion needed for the QPS, the SUBSCHEMA is designed for the QPS. A single SUBSCHEMA may suffice for several QPSs.

The remainder of this section is organized to present:

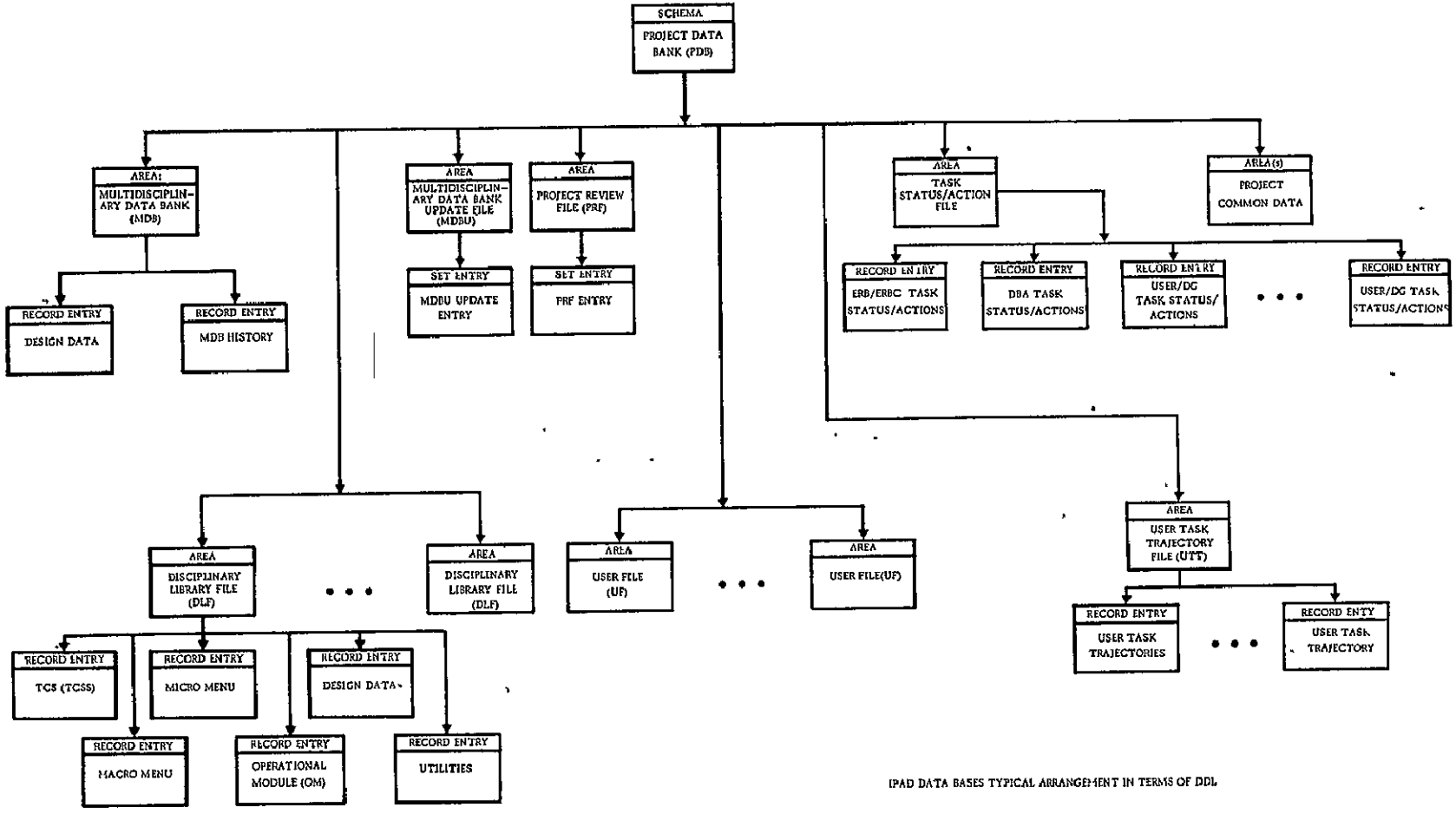
1. An overview of total organization of IPAD data bases and data base management in terms of DDL and QP/DBMS operations (Subsection 4.2.1).
2. A discussion of usages of QP and IPAD data bases as a user would operate with them (Subsection 4.2.2).
3. An illustration of representative data base operations via QP/DBMS (Subsection 4.2.3).

4.2.1 Data organization and management via DDL/DBMS/QP. - The facilities of DDL permit a direct translation of IPAD data bases as required by IPAD operational philosophy into the appropriate DDL specifications.

Figure 4-30 illustrates a typical total organization of an IPAD project data base via DDL depicting the more significant aspects of a Project SCHEMA. AREAs within the SCHEMA DDL are used to specify logical collections of data to be managed as distinctive files. RECORD entries within AREAs are used to describe the actual structure of data within the data bases. Data subentries are used to further detail the structure of data within a RECORD. SET entries are used to describe relationships among various RECORDs within the data base. The RECORDs comprising a SET may reside in various AREAs. The major subdivisions (AREAs and groups of AREAs) of IPAD are charted in the figure with their corresponding DDL assignments.

The expansion of the User File (UF) AREA(s) and Project Common Data AREA(s) are not shown in the figure. The expansion of the Disciplinary Library File (DLF) AREA(s) as shown on the figure also typifies the possibilities for expansion of the UF and Project Common Data.

Figure 4-31 illustrates a more detailed structuring of design data records in DDL terms. The DATA subentry specification of a RECORD is used to subdivide the design data structure into the appropriate category and subcategory subdivisions. The DATA subentries are utilized to detail the identity of design data and its relationship with a design data RECORD down to any level required by the DBA. The figure is restricted to one level of subdivision. Further levels of subdivision are achieved by repeating the category structure for each subcategory so desired.



IPAD DATA BASES TYPICAL ARRANGEMENT IN TERMS OF DDL

Figure 4-30. IPAD Data Bases, Typical Arrangement in Terms of DDL

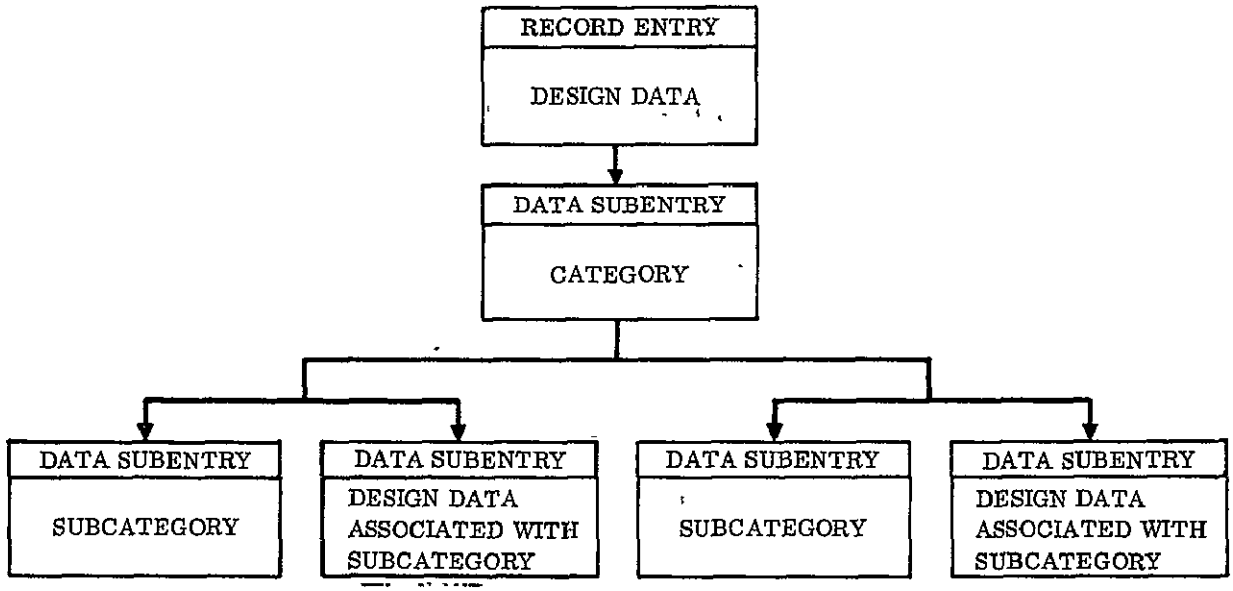


Figure 4-31. Design Data: DDL Expansion

Figure 4-32 shows the expansion of PRF/MDBU entires. The SET entry DDL specification is used since the structure of these files draw upon data sources from many files for their functions. The OWNER RECORD identifies the SET for operation. The MEMBER RECORDs illustrate the types of data that are brought together under the SET designation for the usage of the file.

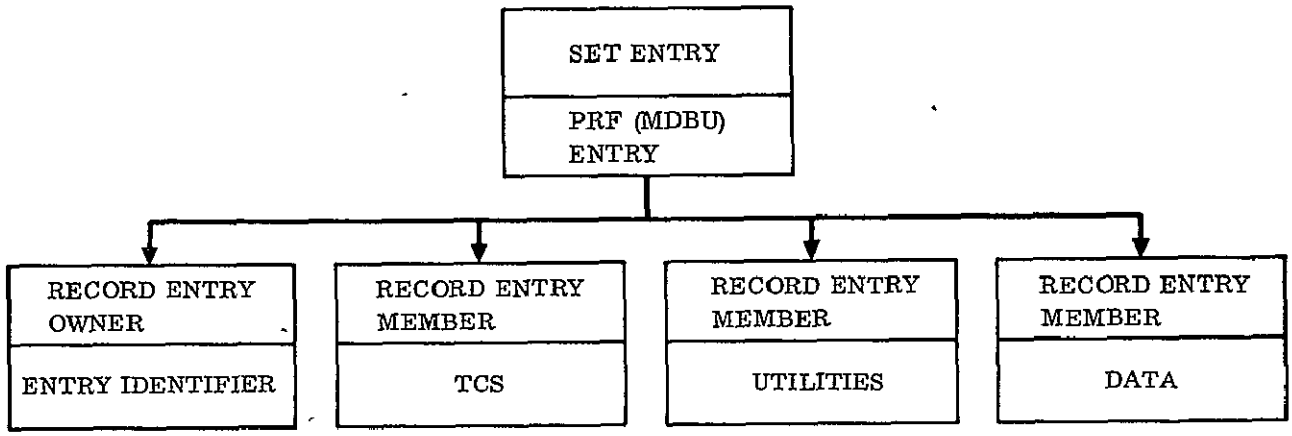


Figure 4-32. PRF (MDBU) Entry: DDL Expansion

Figure 4-33 shows the data bases in relation to the data base management functions. Direct data base operations of IPAD (as opposed to those performed in the operation of OMs, utilities, and the IPAD EXEC) are described in terms of a sequence of QP directives also designated as Query Processor Session (QPS). These directives are input to the Query Processor (QP) which communicates with the DBMS. The DBMS performs the required operations on the IPAD data bases described according to QP specified SCHEMA, SUBSCHEMA and QP directives.

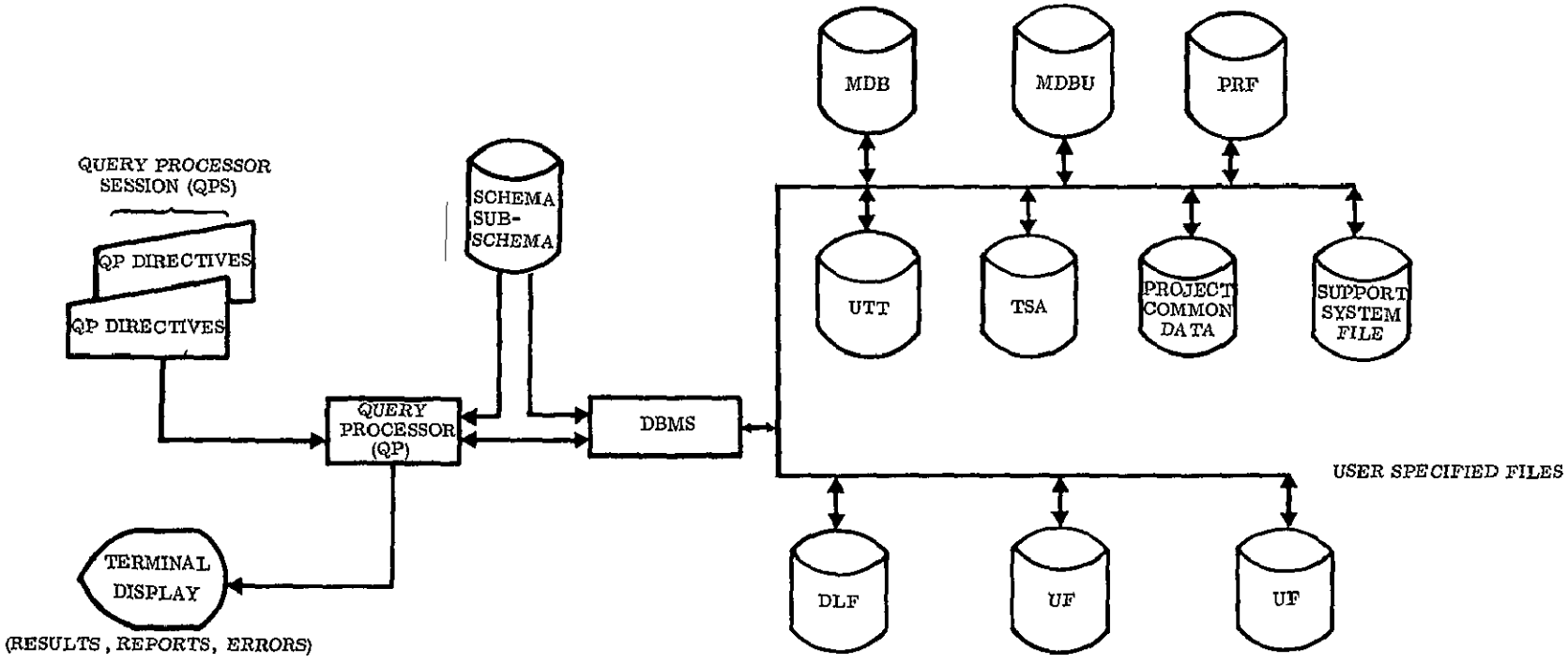


Figure 4-33. IPAD Data Bases Illustrating Typical QP Procedures

At the right side of the figure are the IPAD data bases represented as files but described as AREAs within a Project SCHEMA. The upper group of files are those that a user can always expect to be available for processing. The lower group are files the user must specify for his particular application or session.

QP communicates with the user via terminal display or printout.

4.2.2 Representative data base user operations. - Figure 4-34 traces through a typical sequence of activities illustrating user/data base interface:

1. Assignment of an area of responsibility pertinent to the design problem.
2. Performance of assigned tasks.
3. Preparation of task results for evaluation and review.
4. Incorporation of results into the total design data.
5. Parallel presentation of design data for further analyses.

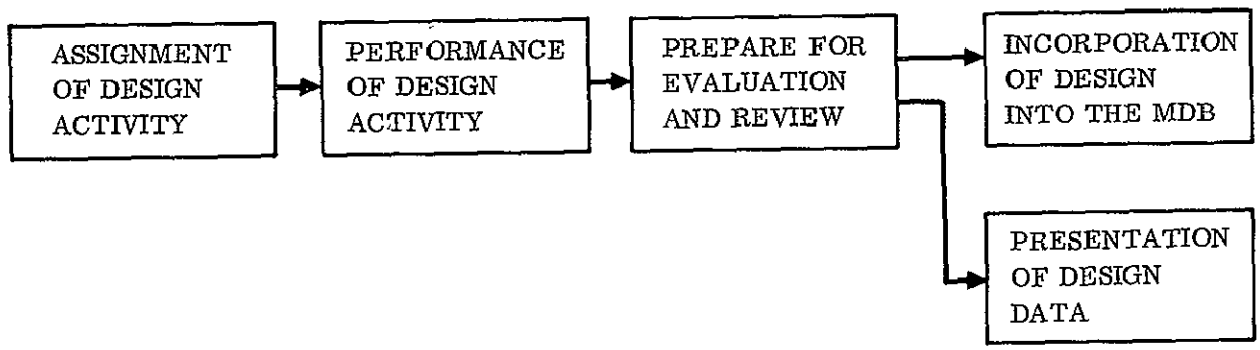


Figure 4-34. Design Data Production Cycle

4.2.2.1 Assignment of design activity (Figure 4-35): This involves a hierarchy of messages in TSAs corresponding to the chain of command for project. The message requirements are specified to the software in an object SUBSCHEMA. The source SUBSCHEMAs also serve as tutorial aids whereby users can determine the requirements. This dual role of the SUBSCHEMA is common to all operations, consequently will not be pointed out in subsequent discussions.

The ERBC assigns an area of design activity to a particular Disciplinary Group (DG). He does this via a QPS that permits him to enter the specific assignment into the TSA of that DG.

The DG supervisor executes a QPS which interrogates his TSA to determine whether any activity has been assigned since his latest interrogation. When he encounters a message specifying an assignment, he decides individual task assignments and distributes messages to the TSA for individual Disciplinary Engineers (DEs).

Each DE executes a QPS which interrogates his own TSA in order to receive his assignment.

4.2.2.2 Performance of assigned tasks (Figure 4-36): For each task, a user configures a software entity (see Section 1 of Part III for an example) including OMs/GPUs to provide the required capability and a UF appropriate to the I/O requirements of the OMs/GPUs. The user and the DBA interact via their TSAs to interface this entity with the total IPAD system. The user, via QP, initializes the UF from the approved design data, then controls the operation of the OMs/GPUs which access the UF.

During the course of his operations, the steps he performs are automatically summarized and placed on the UTT file. The DG supervisor can employ a QPS that permits him to display the contents of the UTT to evaluate the progress and methodology of any user. The user himself may also refer to his individual UTT to review his subtask status.

4.2.2.3 Preparation of design data for evaluation and review (Figure 4-37): When the user completes his assigned design task he will submit his results for evaluation and review, and subsequent inforporation into the MDB. He utilizes a QPS to bring together the necessary design data and processing commands to generate an entry for the MDBU. Correspondingly he uses a QPS to make entries into the DBA's TSA to notify the DBA of the existence of the new entry in the MDBU.

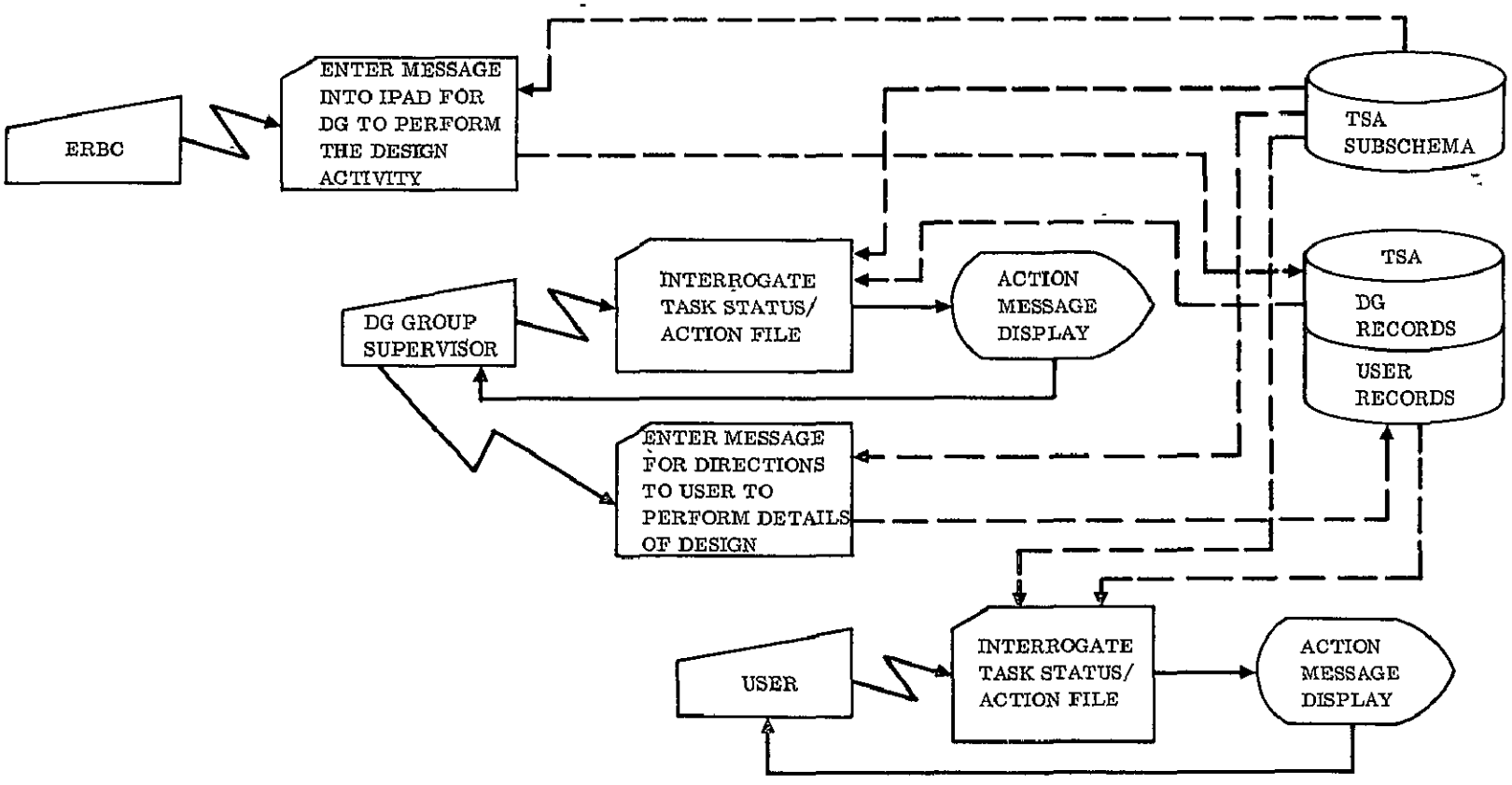


Figure 4-35. Assignment of Design Activity via QP

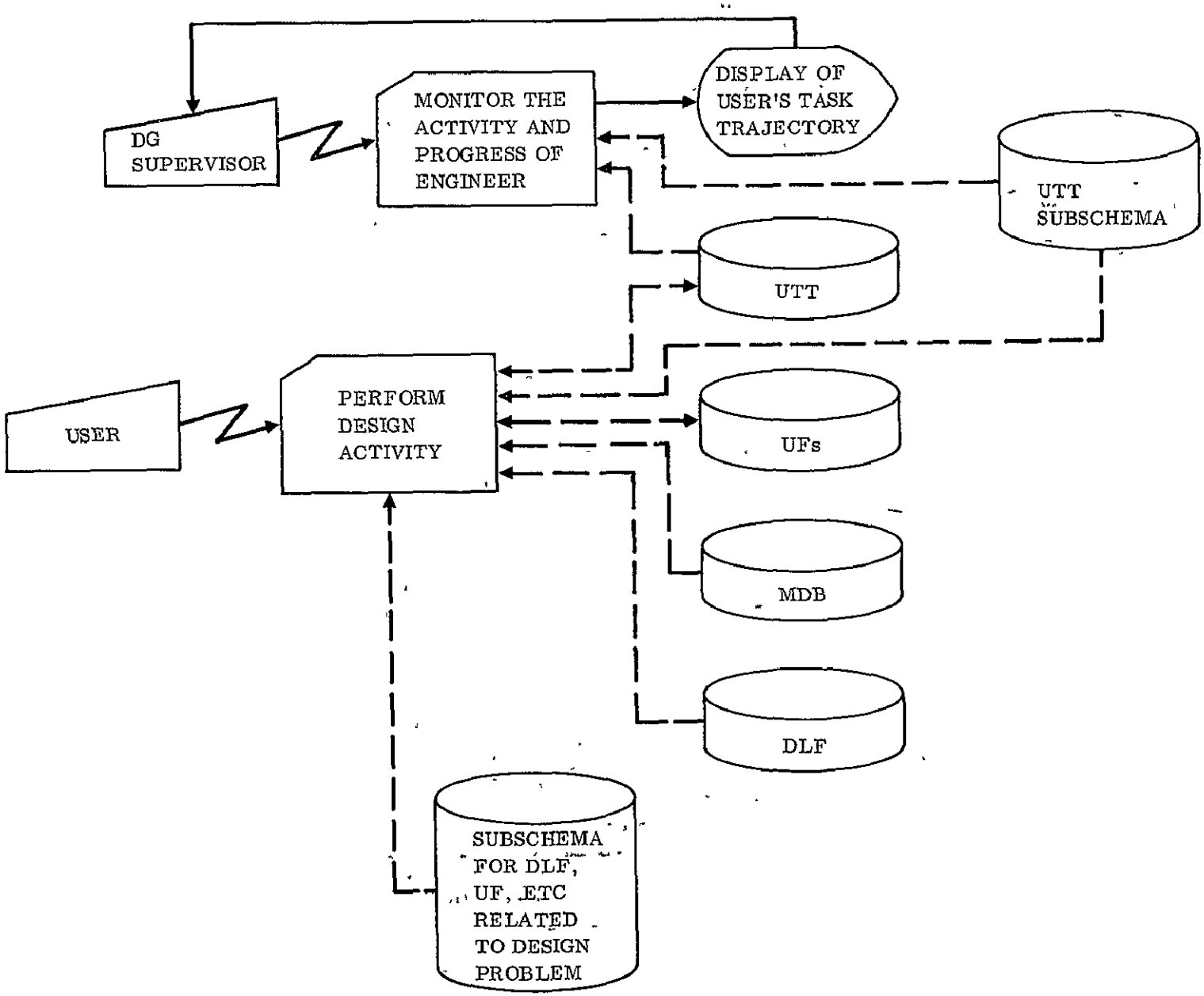


Figure 4-36. Performance of the Design Activity

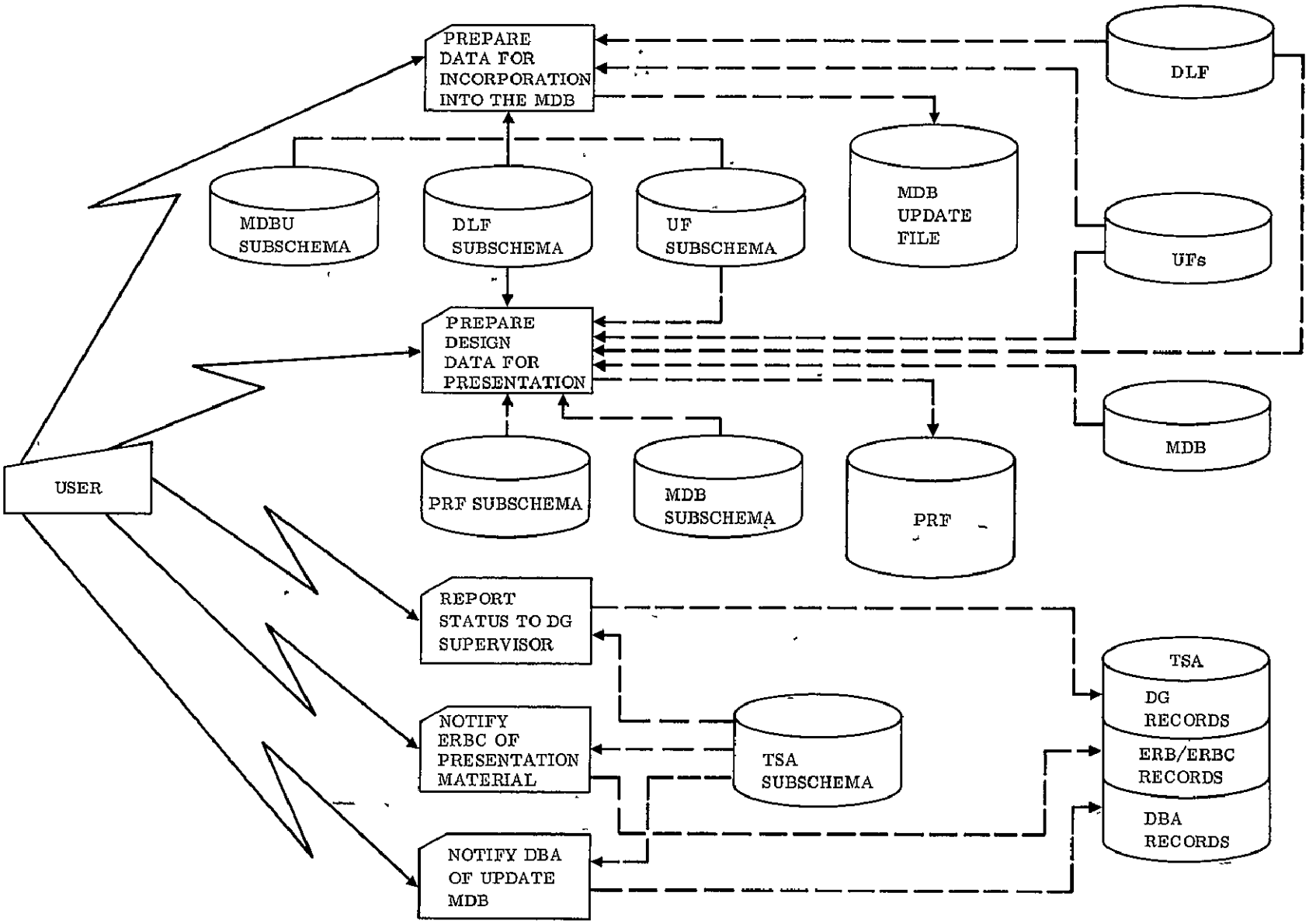


Figure 4-37. Preparation of Design Data for Evaluation and Review

Likewise, if required, he utilizes a QPS to assemble together an entry for the PRF to submit for presentation to the ERB/ERBC. Correspondingly, he uses a QPS to notify the ERBC of the existence of the new entry in the PRF. He also uses the QPS to report his status to the DG supervisor via an entry into the DG's TSA.

At this point, the various messages and data now exist within the data bases for the remainder of the project to use them.

4.2.2.4 Incorporation of design data into the MDB (Figure 4-38): The DBA can now find during the interrogation of his TSA (via a QPS) that the user has produced a candidate set of design data for incorporation into the MDB. He then employs a QPS to locate and display the design data (in the MDBU) via operations provided by the user within the MDBU. If the design data passes review for incorporation into the MDB, he then invokes another QPS that permits him to make the design data portion of the MDBU part of the MDB.

4.2.2.5 Presentation of design data (Figure 4-39): The ERBC can also determine, during the course of interrogating his TSA (via a QPS), that design data (corresponding to an original assignment) now exists within the PRF in a presentation form. The ERBC then employs a QPS that makes use of the user-provided commands to display this data. Upon approval of the presentation, the ERBC schedules the presentation for review by the entire ERB.

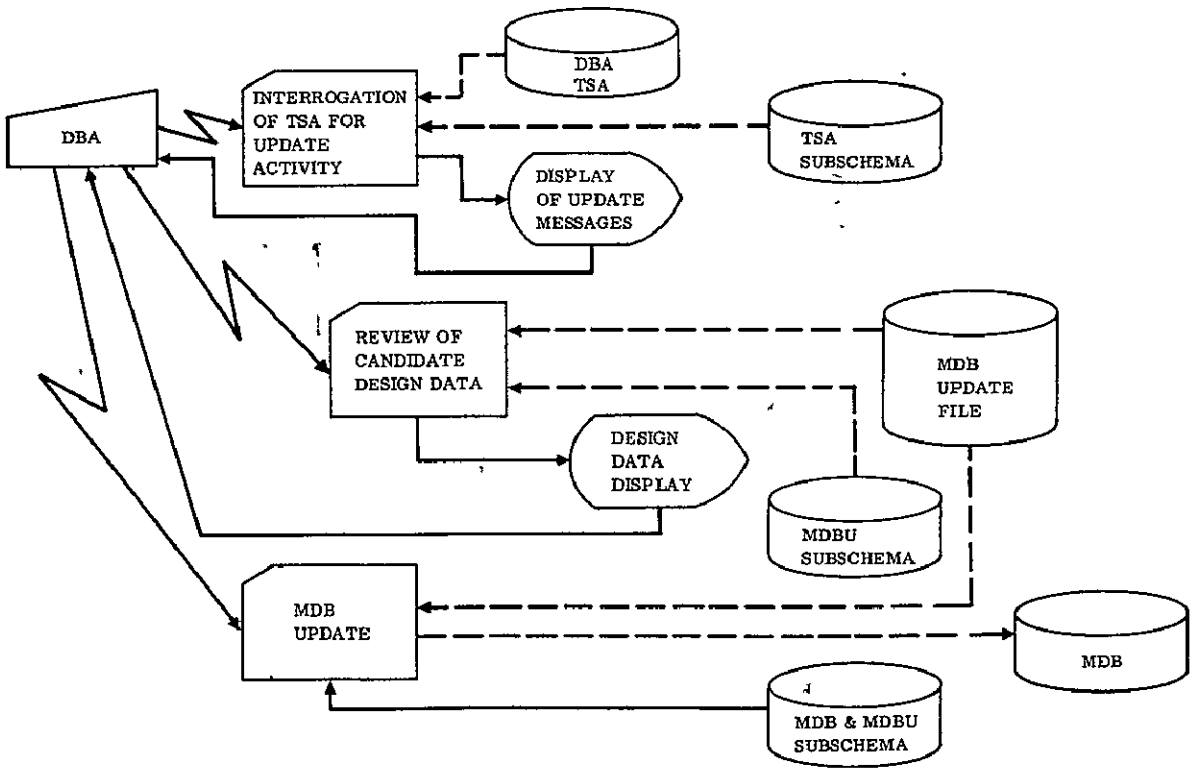


Figure 4-38. Incorporation of Design Data into the MDB

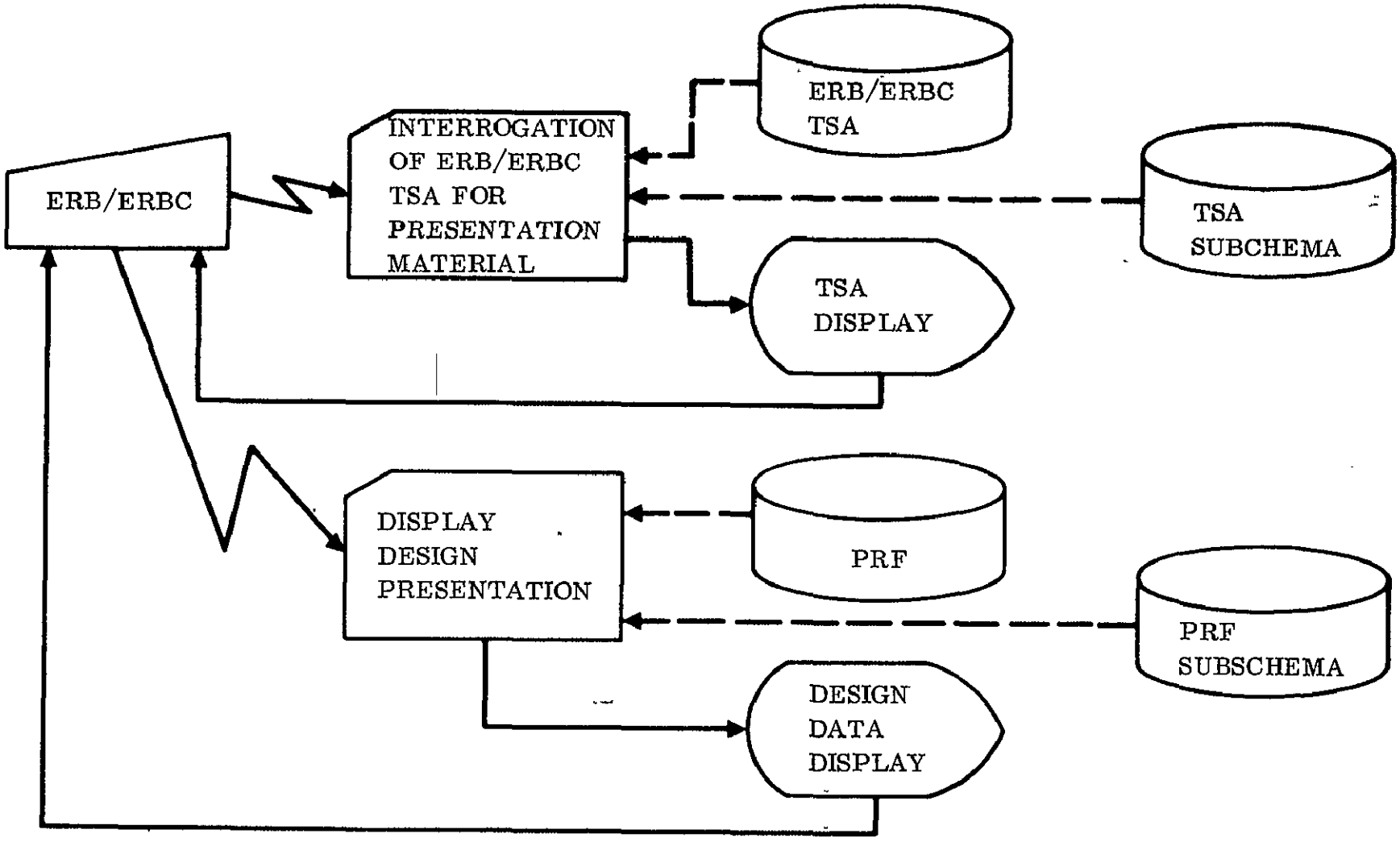


Figure 4-39. Presentation of Design Data

4.2.3 Representative data base operations via QP/DBMS. - In the prior discussion, the user accomplishes his task by specifying and invoking a QPS for the task. The QPS itself is actually a sequence of QP directives that interact with QP and DBMS. The user's operations therefore have to be translated into the appropriate system operations. The unique processes involved in the user's operations are therefore presented in system terms to show the relationships between the user operation and system support.

The subsections below present the details of the QPSS, covered in the previous figures, in terms of their individual QP components and the DBMS support.

4.2.3.1 Message entrance (Figure 4-40): In all operations with the QP, the first step involves the specification (by a QP directive) of the particular AREA(s), SCHEMA, and SUBSCHEMA(s) with which the QPS will operate. In all cases the SCHEMA used is the Project SCHEMA. Each QP/DBMS operation has the Project SCHEMA and the required SUBSCHEMA for operation.

The QPS steps for this operation are:

1. Specification of the TSA as the subject of operation. - This directive results in the opening of the TSA AREA for operation and the attachment of the Project SCHEMA and TSA SUBSCHEMA to the QP/DBMS.
2. Request that QP prepare an occurrence for a message RECORD. QP/DBMS uses the DDL specifications to prepare a RECORD occurrence (a template).
3. Create message. - QP using the SUBSCHEMA formats the information supplied by the user as the RECORD description requires.
4. When the user ends the operation, DBMS attaches the information to the TSA.

4.2.3.2 Interrogation of a TSA (Figure 4-41): This sequence of QP directives will search a specified TSA and display RECORDs appropriate to a particular user.

1. Again a QP directive sets up the appropriate facilities for operation;
2. A QP directive is used to specify the RECORD selection conditions for the TSA (e.g., user identification).
3. When a QP display directive is given, the QP/DBMS begins to access the proper TSA. For each RECORD occurrence:
   a. Conditions for selection are checked.
   b. If the condition is not met, the next RECORD occurrence is processed.

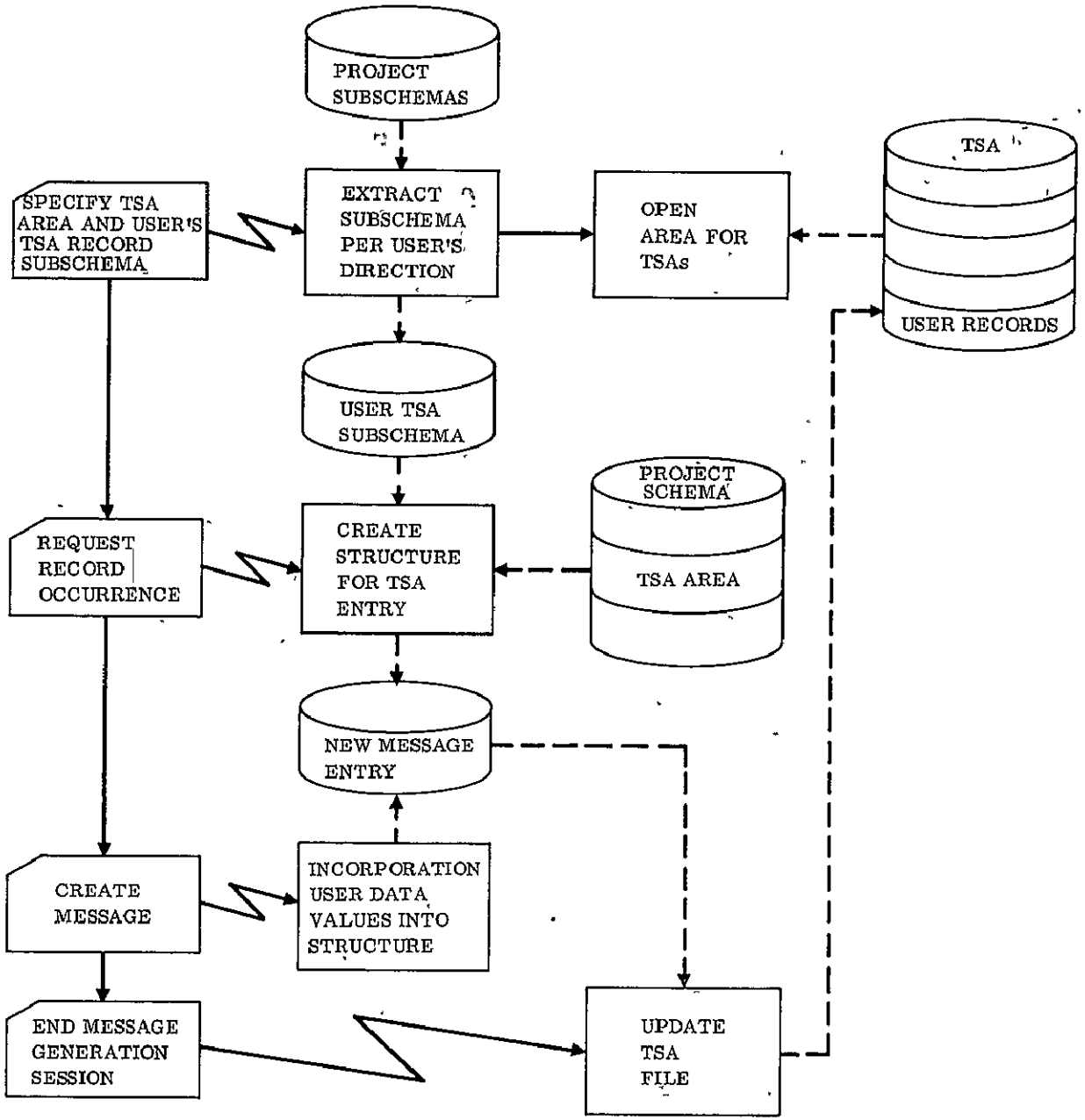


Figure 4-40. Message Entrance, Details of

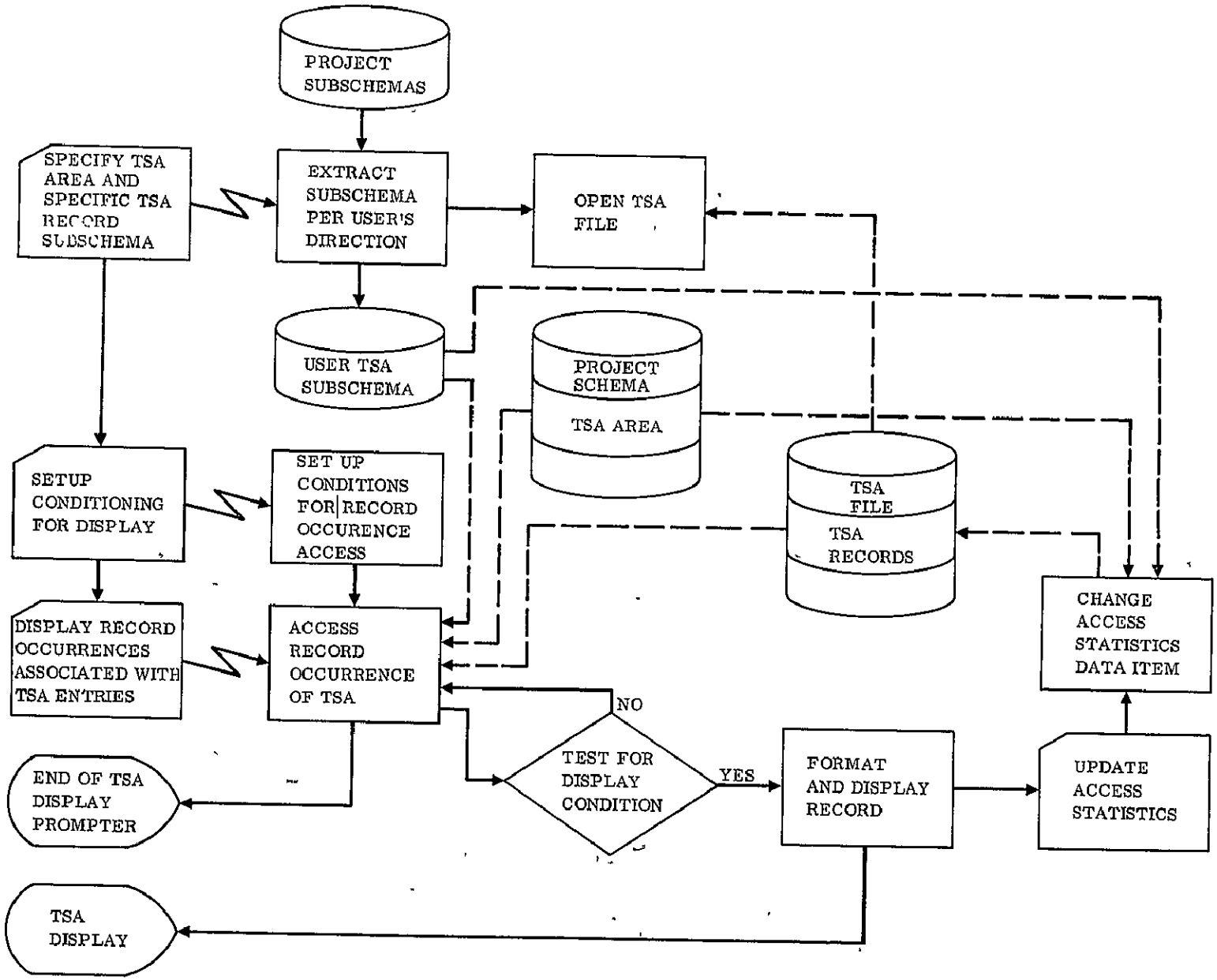


Figure 4-41. Interrogation of a TSA

c. If the condition is met, the data is displayed for the user.

d. A subsidiary DBMS function updates the access statistics for the TSA.

At the end of the complete TSA, the user is notified of the end of information.

4.2.3.3 Monitoring a UTT (Figure 4-42): In this operation the QP sequence will result in the display of a UTT covering only the individual user's portion of the UTT for a particular task.

1. The QPS first performs a QP directive to set up the facilities for the UTT processing.
2. The next directives permit the statement of RECORD selection conditions which are employed in the next step (e.g., task identification).
3. The next directive requests the display of the UTT. In this directive the QP/DBMS accesses each RECORD occurrence. Using the conditions set up by the previous directive, it is determined whether this RECORD is to be displayed.
4. At completion of the UTT data, the viewer is notified of the completion via the display.

4.2.3.4 Preparation of data for insertion into the MDBU (Figure 4-43): In this sequence of QP operations, data will be prepared for incorporation into the MDBU using SUB-SCHEMAs that cover the MDBU and the data source files.

1. The QPS again employs a directive to set up facilities for operation, that is to access data sources.
2. QP directives are now employed to create design data RECORD occurrences for the MDBU. This results (in this example) in the creation of a local file containing data that the user wants assembled for incorporation into the MDB.
3. Other QP directives are used to assemble any TCS, QPS, or utilities RECORDs for inclusion in the MDBU.
4. The assembled data is now ready for incorporation into the MDBU. Another QP directive prepares the facilities for this activity.
5 A QP directive is now given that incorporates the data into the MDBU.

The QPS can be repeated using various file combinations to satisfy the user's requirements.

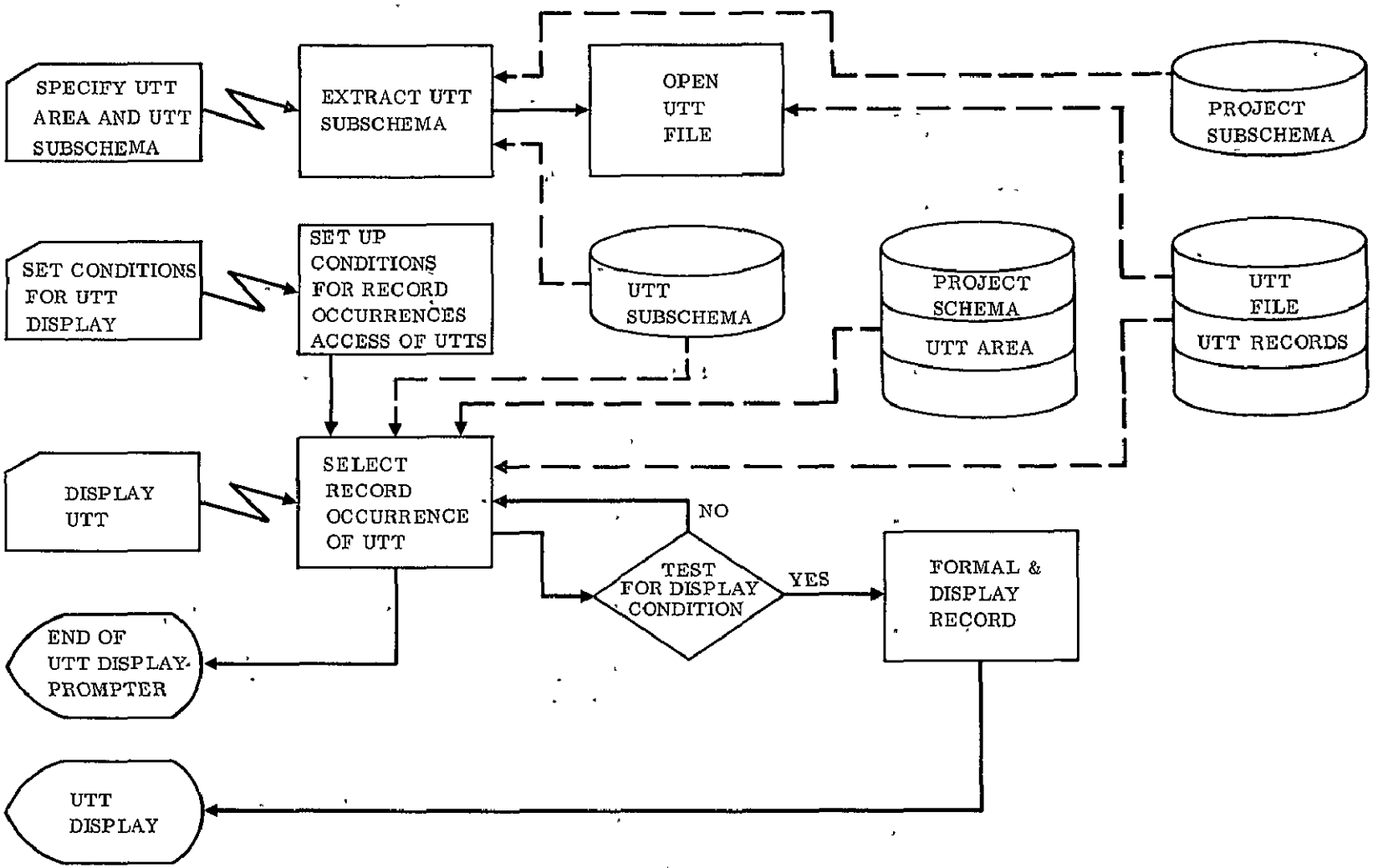


Figure 4-42. Monitoring a UTT

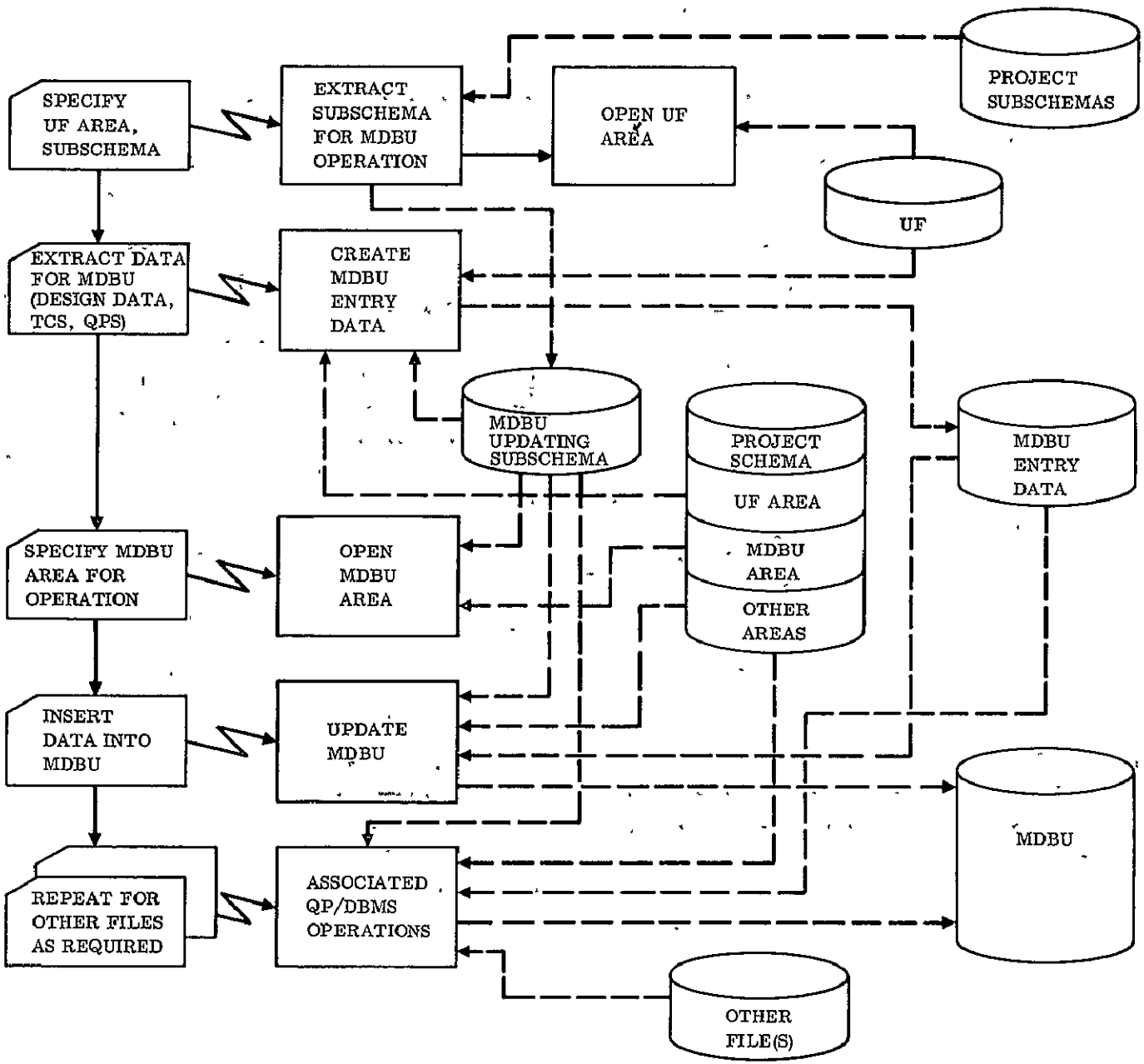


Figure 4-43. Preparation of Data for Insertion into the MDBU

4.2.3.5 Updating the MDB (Figure 4-44): In this operation, both the MDBU and MDB files will be used to incorporate data into the MDB:

1. A QP directive is issued to prepare the facilities. The SUBSCHEMA encompasses both files.
2. A QP directive then results in the extraction of the data from the MDBU.
3. The next QP directive results in the opening of MDB AREA for activity.
4. The final directive results in the placement of the design data as RECORD occurrences within the MDB.

4 2.3.6 Displaying PRF design data (Figure 4-45): This QPS will result in the presentation display of design data for the ERBC/ERB:

1. The first QP directive sets up the facilities for QP/DBMS operation.
2. The next QP directive selects the PRF for display. This is accomplished by specifying the commands (TCS or QPS) portion of the PRF entry under processing and passing these off to the IPAD EXEC for execution (in cases of TCS) or treated as an extension of QPS for display. In case of QPS extension, the next directive results in execution for display.

4.3 Role of the Data Base Administrator (DBA)

The role of the DBA in the IPAD environment is somewhat different from that envisioned by the DBTG. The DBTG envisioned a data base accessed directly by the individual OMs and envisioned a DBA whose prime responsibility was to mediate the conflicting requirements of the OMs. The IPAD MDB is a centrally controlled collection of data as described in Section 4.2, accessed almost exclusively by QP and the IPAD EXEC (e.g., fetching utilities). The requirements of independent OMs are mediated by the individual user when configuring a UF through the SCHEMA Assembler utility (see Section 7).

In IPAD, the DBA is cognizant of the task oriented activity but his primary concerns are the construction of the central data base, maintenance of its quality and dissemination of its contents. Incidentally, he approves and permits the attachment of configured UFs and arranges corresponding UTTs.

An overview of data flow is from the controlled MDB to a UF, from a UF to the MDBU, and from the MDBU into the controlled MDB. DBA activities include promoting and facilitating this flow but also include monitoring the flow and closely controlling updates to the MDB.

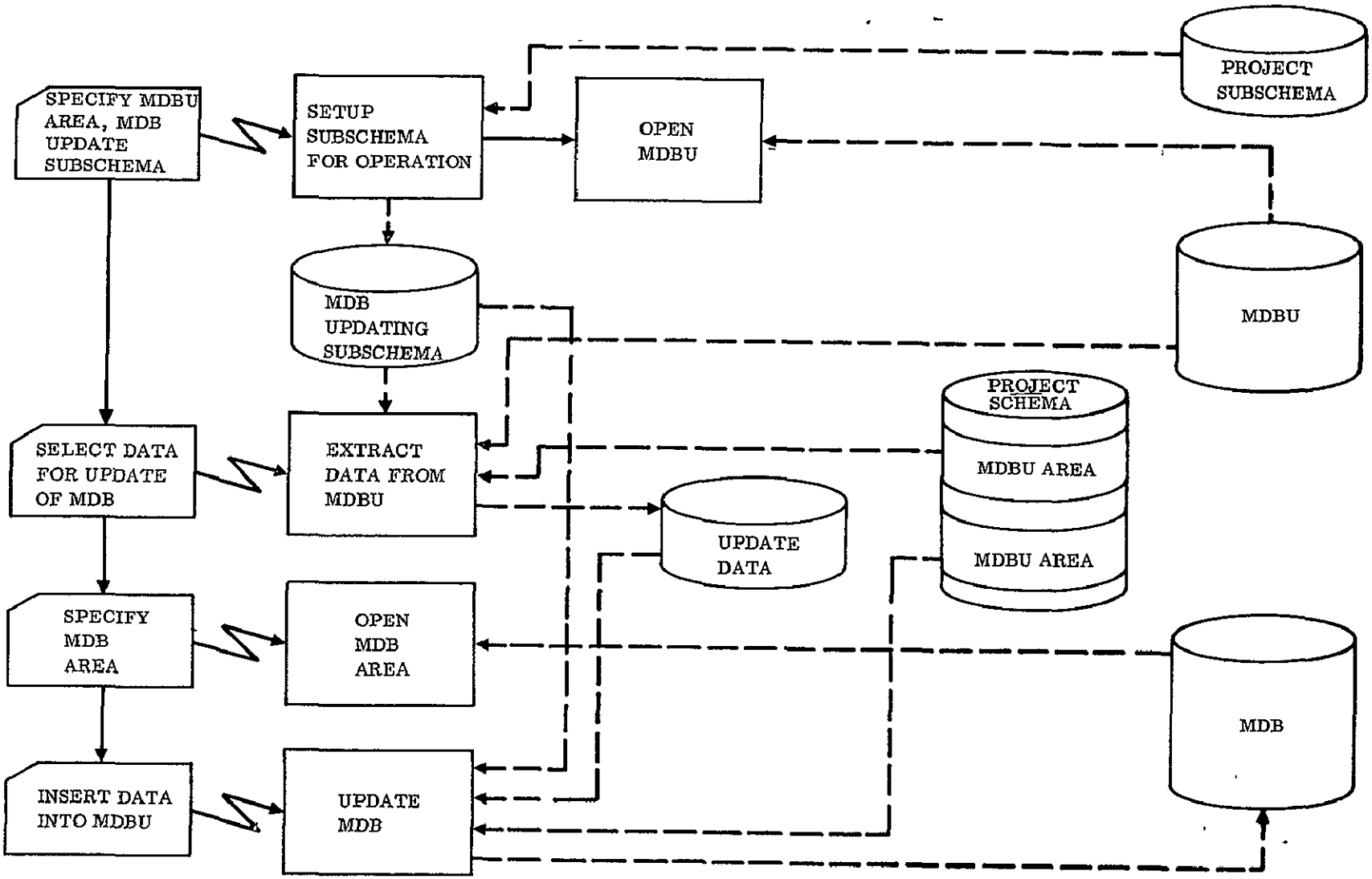


Figure 4-44. Updating the MDB

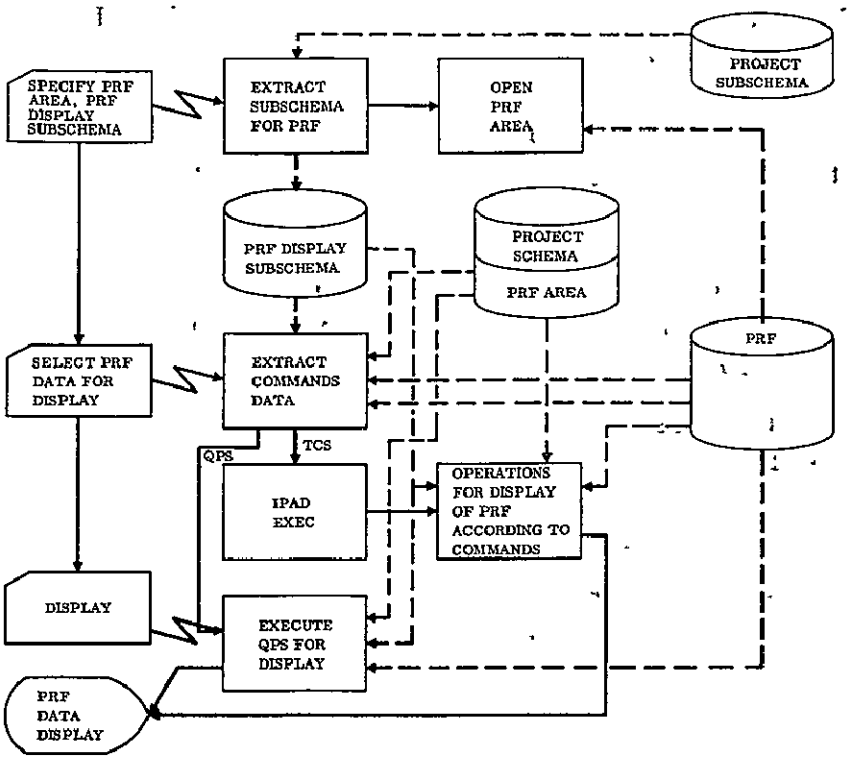


Figure 4-45. Displaying PRF Design Data

From a system operational viewpoint, IPAD appears to the user as a collection of various arrangements of data into files. The direct processing of these files is supported by Query Processor (QP) functions. The responsibility for constructing and providing the facilities for the various users of the IPAD data bases resides with the Data Base Administrator (DBA). In his capacity of responsibility for the IPAD data bases, the DBA has two primary functions:

1. DDL/QPS design and usage - In this capacity, the DBA is responsible for the design and maintenance of the SCHEMA that describe the assignment of data types to their respective files and the representation of the data within these files. Correspondingly, the DBA designs and maintains the Query Processor Sessions (QPSs) and associated SUBSCHEMA for his function and for other members of the project to perform operations on or with the IPAD data bases.

2. Multidisciplinary Data Bank (MDB) design and usage - The DBA also has the responsibility for the generation and maintenance of a key project data base, the MDB.

The remainder of this section details the role of the DBA in the execution of these two primary functions, the facilities to assist the DBA, and the skills he (or his group) must possess.

4.3.1 DDL/QPS design and usage. - Within this function, the DBA has a number of subfunctions which are detailed separately.

4.3.1.1 Project SCHEMA definition: The activity of Project SCHEMA definition involves the DBA in developing the total residency requirements of an IPAD project. The DBA must:

1. Decide what major data base file organization his project requires (MDB, DLFs, TSAs, etc.) and then design the SCHEMA portion that will allocate these files into appropriate AREAs
2. Decide what contents each of the files are to possess and what their most efficient representation under various usages must be. He will then describe in DDL the appropriate RECORD and SET descriptions to satisfy these requirements and assign them to the corresponding AREAs.
3. Decide the privacy controls that the data bases are to possess for the variety of usages for the data and incorporate these on the appropriate level (AREA, RECORDs, SETs) to control specific operations (INSERT, STORE, UPDATE, RETRIEVE, etc.)

In the process of performing the above, the DBA on the basis of expected configuration of usage, makes decision as to the physical residency requirements of the data. For example, he may require that the MDB always be physically available on permanent storage within the host installation but that designated User Files (UFs) may occupy other devices such as magnetic tape or disc packs and need not be physically attached, or existent, within the host installation until specified by a user.

4.3.1.2 Project SCHEMA operational development: After the DBA has created the Project SCHEMA that describes the totality of his project data base requirements, or at least that portion which requires the facilities of QP/DBMS, he has the function of maintaining it for the project. In the performance of this activity the DBA:

1. Assigns the Project SCHEMA Source to a permanent file with its own privacy controls so that the project has a permanent record of the DDL description of its data bases. This SCHEMA can be displayed via the facilities of QP.
2. Uses the DDL Compiler to compile the Project SCHEMA Source into the object form which is required by DBMS in performing its functions. The object form is compiled onto a permanent file with privacy controls set so that essentially only the DBA has authority to modify it. The object form is available and required to support all users of the IPAD project data bases.

4.3.1.3 Project SCHEMA maintenance: Once the Project SCHEMA has been developed, compiled and been in usage for a project, there will arise situations which

will necessitate changes to the SCHEMA (e.g., a new disciplinary group requiring new DLF, UF, and TSA definitions or the assignment or organization of a new category of design data within the MDB). To maintain the Project SCHEMA, the DBA:

1. Establishes, if he desires, DDL descriptions for the SCHEMA itself so that it may be manipulated via QP/DBMS for his convenience. Otherwise, he can establish the SCHEMA Source file so that another general purpose tool such as a Text Editor can be employed.
2. Updates the Project SCHEMA by adding, deleting, or modifying the appropriate portions of the DDL. Normally, there is a wide range of changes within a SCHEMA that the DBA can perform without concern as to invalidating the occurrences of information already within the data bases. He must, however, exercise caution when modifying existing RECORD descriptions. His limitations in this sense are to avoid the deletion or physical reconfiguration of information. Extensions or leveling are possible. If such restructuring is required then the DBA must employ an intermediate step of reading under the old DDL specification and rewriting under the new DDL specification. This procedure itself will accomplish the physical restructuring. The old structure and its information can then be deleted. As long as the data-base-identifiers are maintained, the SUBSCHEMAs that operate against these data bases are still viable. The new Project SCHEMA then can directly replace the original.

To assist the DBA in proper organization or reorganization of data bases, the DBMS provides statistics on frequencies of file usage in terms of transactions made (accesses, insertions, deletion, etc.). Utilizing these statistics the DBA can perform such activities as reordering of files and reassignment of RECORDs to residency in faster access devices, etc., to improve the overall efficiency of the data base operation performance.

4.3.1.4 SUBSCHEMA provisions: The user's view of the data base for specific data base operations is provided via the SUBSCHEMA where only that portion of the data base needed in the form required by a particular process need be described. The management problems of the SUBSCHEMA are similar to that of the Project SCHEMA. The DBA:

1. Designates, with appropriate privacy controls, a permanent file within the host installation (or project's data base) to contain the SUBSCHEMA.
2. Designs, or assists in the design of, various source SUBSCHEMA for the operations required.
3. Compiles the source SUBSCHEMA (via the DDL compiler) and places the resultant object SUBSCHEMA onto a permanent file for usage by the QP/DBMS.

4. Modifies, adds, or deletes existing SUBSCHEMAs as required. This is most conveniently done if the SUBSCHEMAs are described via DDL to reside within a project's data base.

The SUBSCHEMAs whose functions the DBA himself will design are those concerned with more basic and common processes, particularly those associated with MDB operations. For individuals (with their own specific needs), the DBA will provide allocation and appropriate privacy controls so that they may, when required, develop their own SUBSCHEMAs and have them managed and used by the QP/DBMS.

4.3.1.5 Query Processor Session (QPS) design and usage: The DBA also has the responsibility for developing the proper sequence of QP directives for performing desired operations on the data bases. In this capacity the DBA:

1. Designs, for basic operations on IPAD data bases, the appropriate QPS that will satisfy these operations. The QPS is given a particular identifier and then catalogued within an IPAD data bases for subsequent usage.
2. Designs, as AREAs or RECORDs within other AREAs, DDL descriptions for the QPSs to reside within IPAD to satisfy the above.
3. For ease of operation, the DBA can design a QPS that will display available QPSs or groups of QPSs much in the same manner as Macro/Micro Menus that permit him or other users to locate and direct a particular QPS for execution.
4. Designs, if desired, tutorials to go along with the QPS.
5. Assists other users in designing their own individual QPS. In conjunction with this, the DBA can also incorporate the DDL descriptions for the QPS into various files for convenience of the user.

Optionally, the DBA can perform the same type of activity for Query Processor Session Skeletons (QPSSs) rather than QPSs if the operations lend themselves to QPSSs.

Prefabricated TCSs, QPSs, etc. are incorporated and managed in the same way as any other data; that is, they are represented and described by DDL entries in the SCHEMA and various SUBSCHEMAs. Consequently they can be operated upon by QP (as well as the TCS Writer and Expander) and modified versions of any particular string may be stored by any user.

4.3.2 MDB design and usage. - The other primary responsibility of the DBA is the MDB. In this capacity the DBA must:

1. Decide, with the approval of the ERB, what data the MDB is to contain and how it is to be organized to contain the data.

2. Design the storage configuration for the data, build the required DDL specifications, and make the specifications available to the user who must employ or produce the data.

3. Examine and determine the validity of any design data submitted via the MDBU for incorporation into the MDB.

4. Be responsible for the updating (addition, deletion, and modification) of the contents of the MDB.

5. Be able to report the current contents and history of activity within the MDB.

4.3.3 General IPAD facilities for DBA assistance. - Under the implementation philosophy, each DBA essentially produces a unique SCHEMA (of DDL specifications), SUBSCHEMAs, and QPSs that are tailored to the requirements of the particular project to which he is responsible. In particular, although the DBA develops his data base in the general organization of Section 4.1, he has considerable flexibility in details. Much of the data types and procedures for utilizing them are expected to be sufficiently common among projects to permit the developers and maintainers of IPAD to create support facilities via data bases to assist the DBA. Such facilities include:

1. Prototype DDL specifications. This data is retrievable by the DBA. It contains source SCHEMA prototypes for AREAs such as DLFs, TSA, etc. The DBA selects those needed for his files, updates them to place them in context for his project and from them, in general, assembles his Project SCHEMA. He supplies any additional DDL specifications not contained in the prototypes or different versions.

2. Prototype QPSSs. This type of data performs for the QPS design function the same type of operations that the above does for the DDL specification.

3. Tutorials. This type of data accompanies either prototype and provides the DBA with explanations of the prototypes and instructions for developing his Project SCHEMA.

All of the above are incorporated into data bases controlled by SCHEMA, SUBSCHEMA, and QPSs that permit the QP/DBMS operations to be performed by IPAD system personnel and DBAs.

4.3.4 DBA skill level. - The DBA is the key individual responsible for the control of data bases within his project. In order for the DBA to perform his function, he or his group must possess the following capabilities:

1. Be knowledgeable about the project and design data associated with the project to the degree that appropriate DDL descriptions can be developed.

2. Be knowledgeable about the usage of design data and organization of usage to the degree that efficient physical placement of data within the MDB can be achieved.

3. Be knowledgeable about project organization and data requirements of individual types of users so that data groupings, AREA assignments, and physical organization will satisfy the individual members of the project.

4. Be knowledgeable about host machine configuration and device characteristics to enable him to make the most effective assignments to devices for the required physical organization and, conversely, to permit him to specify LOCATION MODE and SET relationships within the DDL.

5. Be knowledgeable about both the external usages and internal workings of QP and DBMS to permit:

   a. Efficient QPS and DDL development.

   b. Efficient QP and DBMS operation.

4.4 Summary

The central factor in the choices of establishing the data base requirements of IPAD is the recognition of the fact that the total set of data base requirements can be represented as several basic data types; the specific data bases as required and seen by the user are configurations and specific representations of these data types. An IPAD project data base then becomes a collection of files organized from several basic RECORD types. Likewise, the operations that are to be performed on the data bases are reducible to generalized basic data base operations. The general data types and their organization for processing are satisfied by the DDL which, in turn, is supported by the DBMS. The procedures for manipulating the data bases are satisfied by the QP.

Consequently, the usage of QP/DBMS presents a satisfactory implementation philosophy with two major advantages:

1. Transferability - The QP/DBMS is a general system which is to be developed in accordance with an industry-wide standard specifications of the DDLs and DML languages (Reference 4). These concepts thus present IPAD with a high degree of transferability.

2. Adaptability - The DBMS code can handle any physical record or file descriptions by appropriate DDL specifications. It is, therefore, not appropriate for IPAD to specify data structures and built special utilities to process

them. DBAs have considerable flexibility in their management of data bases both in defining their contents and in defining procedures to utilize them.

The concept of implementation via this approach leads to more dynamic and useful operations with the IPAD data bases. For a project, the DBA functional group, which is knowledgeable both in the data requirements of design and the characteristics of the host facility, can now design and develop the optimum data base configuration for the project. This is done, not via additional coding but by language specifications and command sequencing.

5 LANGUAGE DEVELOPMENT

The design approach presented in Sections 2, 3 and 4 emphasizes exploitation of software provided by computer system manufacturers. The supporting software is general in that:

1. It is not developed specifically for IPAD, and
2. With each supporting subsystem, languages exist to provide the capabilities of the subsystem to some general class of users.

The development required in support of IPAD is not a development of a new software system but rather a development of languages (and functional support to these languages) to exploit capabilities of manufacturer supplied software. The objectives of this language development are:

1. To provide the full range of capabilities of:
   a. The operating system,
   b. The timesharing subsystem,
   c. The Data Base Management System (DBMS), and
   d. The interactive graphics subsystem,

   all through a concise lexicon.
2. To produce language standards (eventually to become industry standards) for increased portability.

The following subsections discuss five language development tasks with respect to the four items in 1 above:

1. IPAD Control Language (ICL) to interface between the user and the operating system/timesharing subsystem.
2. Data Description Languages (DDLs) to define data structures and relationships that exist in the data base and those required by OMs/utilities thus making interface support via DBMS possible.
3. Data Manipulation Language (DML) which provides the procedural interface between specific OMs/utilities and the database via DBMS.

4. Query Processor Language (QPL) which provides the interactive procedural interface between any IPAD user and the database via the Query Processor (QP) operating through DBMS.

5. General Graphics Library (GGL) which provides interface between OMs/utilities and the interactive graphics subsystem.

5.1 IPAD Control Language (ICL)

For a given implementation, the ICL includes the commands associated with the supporting operating system/timesharing subsystem as well as those associated with the EXEC and its subsidiary functions.

5.1.1 ICL associated with supporting software. - This class of command languages has been given the generic name Operating System Control Language (OSCL) by ANSI, and the X3/SPARC/OSCL committee has recently been reactivated to investigate standardization (see Vol. IV, Section 6.2 of Part I and Appendices C and D for details. Until th OSCL committee recommends some standards, OSCL can be characterized as follows:

1. Commands and functional support are provided by the manufacturer, hence will be significantly different for each system.

2. To the typical (noncomputer oriented) IPAD user each OSCL is a cumbersome bother.

However, the language development objectives can still be attained and the full OSCL capabilities included in the IPAD user's repertoire of commands through the use of prefabricated TCSSs, as explained in Section 2.2. One of the objectives of a prefabricated TCSS is to relieve the user of labor and attention to details of OSCL syntax. Consequently the syntax of dummy arguments and the command language associated with the TCSS expander replace the OSCL language (from an IPAD user's point of view).

The development of rules for prefabricating TCSSs and the development of the Expander commands should be cognizant of the OSCL committee's recommendations. Also, the requirements derived for the Expander commands and TCSS rules should be provided to the committee since one of its study tasks is to categorize and define elements of functional control of computation systems (Appendix C).

5.1.2 IPAD EXEC commands. - Since the EXEC and its subsidiary functions are themselves a commandable software subsystem, IPAD has its own requirements for a set of unique commands different from the supporting software. The actual repertoire of IPAD EXEC commands which must be developed for implementation depend

on the final build-to specifications of IPAD design. However, the following types of commands are expected to be evolved.

5.1.2.1 TCS file control commands: These commands permit the user to direct the EXEC in the handling of TCS files. They include such items as:

1. Specification of TCS files to the EXEC.
2. Control commands to permit switching between a TCS file and terminal input.
3. Commands to direct the TCS Interceptor to begin and/or cancel TCS recording.

5.1.2.2 TCSS expansion commands: These commands are used to control TCSS expansion and involve one set that consists of control commands to:

1. Identify dummy arguments and make valued substitution within the TCS image.
2. Conditional commands that permit adjusting the logic of the resultant TCS based on user's input.
3. Requests for explanation of dummy argument meaning and methods of specifying information for the TCSS expansion process.

5.1.2.3 TCS execution commands: These commands control the TCS during execution They include such features as:

1. Conditional commands that modify logic of TCS execution.
2. Control commands that permit the EXEC to call upon and switch between various subsystems, such as QP.

5.1.3 ICL constraints. - In the development of IPAD, unique command constraints are necessary on language syntax.

5.1.3.1 Uniqueness: Uniqueness is required to permit the command to be readily distinguishable from commands that are associated with the various manufacturers (hence non-unique) IPAD support system languages (the OSCL). This is required to (1) prevent the need for additional commands and logic in the IPAD EXEC to distinguish the overlap, and (2) prevent confusion on the part of the user when his tasks necessitates the mixture of languages controlling the various subsystems.

5.1.3.2 Meaningfulness. Meaningfulness of the command is required to permit the user to more easily learn the language in terms of the functions he wishes to employ.

5.1.3.3 Prompting mode: The prompting mode of the ICL permits the IPAD system to assist the user in command specification. Once the operator portion of the command is recognized, prompting messages are supplied to the user explaining and requesting additional information for detailing the specifics of the operands required.

5.2 Data Description Languages (DDLs)

The Data Description Languages permit a high degree of independence in three functional specialties associated with the data base, by providing means to interface:

1. The Data Base Administrator (DBA) function.
2. The DBMS functions to control system level I/O operations.
3. The programming functions, creation and conversion of OMs/utilities.

The DBTG report (Reference 4) contains detailed specifications for two DDL types:

1. SCHEMA DDL used to describe data as it exists (or will exist) in the database.
2. SUBSCHEMA DDL used to describe some subset of the total database as an individual COBOL program requires.

The concept of the two independent descriptions in compiled (highly processable) form permits software (DBMS) to accomplish two of the most tedious tasks facing an IPAD user:

1. Collecting data pertinent to his design task and reorganizing it into a form acceptable to an OM/utility.
2. Transforming the output of an OM/utility into a form acceptable to the user for review or to another OM/utility.

Further, the DDLs describe logical structures and relationships rather than physical entities, which permits DBMS to relieve both the DBA and the general IPAD user of bookkeeping relative to the location of data.

As is evident, the support that DBMS could provide would be limited by the quality and applicability of the DDLs. All requirements for management of a database and interfacing different programs with the database must be expressed in the DDLs. Consequently the DDLs must adequately express the management concepts of an engineering-oriented DBA and engineering-oriented programmers who create and/or convert OMs for operation with IPAD.

The DBTG specified very powerful, high quality languages for a COBOL environment; the architecture of the implied DBMS was expected to interface the same database with OMs written in a variety of languages. It was further expected that the pioneer SCHEMA DDL would be developed to accommodate many languages and many SUBSCHEMA DDLs would be developed, each to interface with its own host language.

5.2.1 SCHEMA DDL. - The SCHEMA is a description of all data known to DBMS (i.e. of the database) at any particular time. Hence the SCHEMA DDL is the language by which the DBA effects a compromise between all the structural requirements of the individual IPAD users and expresses his own management requirements. The specifications of the DBTG are remarkably comprehensive, containing many features (e.g. implicit sort specifications) which may never be required in an IPAD implementation. However, the syntax of the SCHEMA DDL is extremely cumbersome in an engineering environment and the composition rules abound with restriction that seem arbitrary and artificial with respect to FORTRAN coding techniques.

5.2.2 SUBSCHEMA DDLs. - Basically the SUBSCHEMA DDLs are languages which describe a portion of the database as required by a particular OM/utility. The essential difference between SCHEMA and SUBSCHEMA DDL is that the SUBSCHEMA DDL must be compatible with the source language of the particular program involved since that program must invoke the SUBSCHEMA (and its implied procedures) and make functional requests of DBMS referencing SUBSCHEMA data specifications. The complete relationship of the SUBSCHEMA to SCHEMA is detailed on page 18 of Reference 4. The development of SUBSCHEMA DDLs for scientific program languages must consider the full implications of this relationship as well as:

1. The Data Manipulation Language (DML) enhancement for the host language (see Section 5.3)
2. The additional I/O support that may be provided outside of DBMS; that is, the traditional FORTRAN I/O software may supplement the features of DBMS.

5.3 Data Manipulation Language (DML).

The Data Manipulation Language is the vehicle which provides the functional capabilities of DBMS to the applications programmers. The DBTG report outlines the concepts governing the development of a DML and provides a complete specification for the COBOL DML. The following is quoted from pages 14-17 of the DBTG report (Reference 4):

> The DML is the language which the programmer uses to cause data to be transferred between his program and the database.

The DML is not a complete language by itself. It relies on a host language to provide a framework for it and to provide the procedural capabilities required to manipulate data.

The relationship between DDL and DML is the relationship between declarations and procedure. The declarations impose a discipline over the executable code and are to a large extent substitutes for procedures written in the DML and the host language; that is, they are implicit procedures which may be invoked by the execution of DML commands.

A users application program is written in a mixture of host language statements and DML commands. The DML provides the ability to interact with the database in that it is the language interface with the DBMS. All calls to and from the database to retrieve data, to add new data, to modify existing data or data relationships, and to delete existing data or data relationships are written in DML.

As a result of the successful execution of a call for data included in the database, the data requested is delivered to the UWA [User's Working Area] of the calling program and may then be referenced and manipulated using the facilities of the host language. To add new data or return modified data to the database, the host language is used to initialize the appropriate values in the UWA and the DML is used to call on the DBMS's services.

This is a departure from standard FORTRAN techniques where the host language never explicitly references the UWA described here. This difference may require additional DML development or additional software support from the standard I/O support routines. The quote continues:

The host language, then, is the language used to manipulate data in primary storage. The host language processes or provides the framework in which the DML functions, and the DML is the interface language with the database.

DML commands and host language statements are intimately mixed in an applications program. Indeed, the distinction between them is conceptual. The two languages may be mixed freely and there are no special "enter" or "exit" requirements from one language to the other. Thus, from the programmers point of view, he is using a single language - a language which has the combined capabilities of the host language and DML.

This provides the overview of a DML. The details of DBMS capabilities to be provided by DML are specified through the COBOL DML specifications of Reference 4.

5.4 Engineering Oriented Syntax For DDL and DML

The foreword to the DBTG report (Reference 4) envisions that report as a base upon which a SCHEMA DDL can be finalized "independent of, but common to, many higher level languages."

A proposal from the University of Edinburgh (Reference 16) specifies a mechanism for "A common SUBSCHEMA framework for all host languages." That is, it proposes a mechanism for identifying COBOL-peculiar features of the DBTG specifications and for inserting and identifying features peculiar to other languages.

The proposal for a common DDL reasons that since many data management problems are independent of programming languages, some measure of efficiency and compatibility will be achieved by providing a common DDL. The merit of this approach will not be disputed here, but the human engineering objectives of IPAD require that another approach be developed.

The existence and popularity of FORTRAN in the IPAD community and COBOL in the business community points out a valid need for different languages in the different environments. The need is not for a common (i.e. to engineering and business programmers) language that adequately expresses problems to a computer (such languages existed prior to FORTRAN/COBOL). The need is for languages compatible with the training and thought processes of the individuals involved. A DBMS that supports interfacing of an integrated database by multiple languages must also support the needs of humans that use those languages.

The need for languages tailored to the requirements of programmers and IPAD system maintenance personnel does not indicate a modification of the functions of DBMS nor a modification of the architecture of the DDLs. What is needed for a viable IPAD is only a different syntax to be processed by the DDL compilers; nearly the full range of capability offered by DDL is required for IPAD.

DBMS does not work with source DDL but with an object table which results from processing the DDL by the DDL compiler. Obviously a DDL can be specified to express the same ideas as the COBOL DDL but in an engineering syntax. A corresponding DDL compiler would produce an object table usable by DBMS.

In the pseudo-English tradition of COBOL, each of the DDLs recommended by the DBTG consists of a language of declarations, verbs, objects, subjects, clauses

and phrases. The explanation for each element begins with a definition of its function (what fact is conveys to DBMS, or what processing it requires of DBMS and what initiates the processing).

The University of Edinburgh specification for a FORTRAN DDL (Reference 17) proposes that all DDLs adhere as closely as possible to the syntax recommended by the DBTG, introducing more pseudo-English as required.

To begin the development of an engineering syntax for DDL, the definitions of the functions of the elements of the DBTG language should be listed and a panel of prospective implementers should decide on a syntax to express those functions in pseudo-math notation, introducing more functions and corresponding pseudo-math as experience inspires.

The approach to an engineering syntax for a FORTRAN DML should parallel that for the FORTRAN DDL. The functional capabilities of DBMS are defined through the functional description of the COBOL DML. The DBMS provides these functions in response to execution time requests from the application program and parameters supplied by that program. To re-emphasize the point, DBMS does not interface with the program on a source language basis but through its object code.

A panel of prospective implementers could review the definitions of DBMS functions and decide upon new FORTRAN statements to be transformed into DBMS requests by the FORTRAN compiler.

However, another approach is possible and this approach is almost traditional in enhancing FORTRAN capabilities, viz. to make no change to the compiler but provide DBMS functions accessable through standard FORTRAN CALL statements.

Which approach is to be followed should be left to the language developers (see Section 5.7).

5.5 Query Processor Language (QPL)

QPL is the language which provides the nonprogrammer (i.e. the IPAD user) interactive access to the database. The DBTG deferred development of this language (Page 8 of Reference 4):

> In facing this situation the Data Base Task Group came to the conclusion that an approach which permitted both programmer and nonprogrammer interface with the same database was essential - but that the programmer interface was more basic [essential] and should therefore be tackled first.

However, the implicit architecture of a DBMS contained in the report provides a solid foundation for a language to provide interactive capabilities such as (quoted from page 7 of Reference 4):

- interrogation - this is inclusive of data selection, sorting and report [e.g. ERB presentation] formatting.
- update - this is inclusive of selection of data to be updated and the process of changing the value content of the selected data.
- creation - this is the building of the initial instance of the database or a portion of it. [Individual users initialize UFs and the DBA establishes initial versions of design data.]
- restructuring - this involves changing the description of an existing database and adjusting the database to the new description.

The IPAD design extends this list as follows:

1. The full range of DML capabilities is to be provided interactively via QP (i.e. contained within the QPL).
2. The ability to save commands (directives) with or without execution from one session to another is to be provided (i.e. QPS generation).
3. The ability to perform all or any part of a saved QPS must be provided.
4. The IPAD users will require a desk calculator capability in QP and essentially the full range of the FORTRAN mathematical operator and subroutine library in support of the update function (See Section 3.2).

5.6 General Graphics Library (GGL)

The General Graphics Library represents a collection of subroutines that will support the entire spectrum of usage of interactive graphics terminals for IPAD. The necessity for a GGL arises from two basic considerations present in the solution process of problems in the IPAD environment:

1. The nature of the problem solved is graphical.
2. The solution process is best conducted in an interactive mode.

These two factors suggest a heavy dependence on the efficient use of a wide variety of interactive graphics terminals. It is these terminals that are supported by GGL.

5.6.1 Current basic software packages. - Each major developer or supplier of interactive graphics systems has usually developed his own basic graphics support software. This set of subroutines is usually coded in the computer manufacturer's assembly language and designed to fully exploit the capabilities of the particular hardware for which it was written. Even though all the support packages are built to be FORTRAN callable, transferability to other display hardware and host computing systems is very difficult.

Available interactive graphics software is briefly discussed in the subsections which follow.

5.6.1.1 IBM software: The basic IBM interactive graphics support library is currently Graphic Subroutine Package (GSP). GSP (Reference 18) is a set of FORTRAN callable assembly language routines that facilitates the creation of displays on IBM 2250 Display Units (refreshed CRT) attached to an IBM System/360 (or 370) Computing System. Historically, IBM started with the Alpine System, followed by GPAK, and then GSP. For remote locations there was even an IBM 1130 GSP which was not the same as the basic IBM 360 GSP.

5.6.1.2 CDC software: The current CDC interactive graphics support package is Interactive Graphics System (IGS). IGS (Reference 3) is a set of FORTRAN callable assembly language routines that are used to create displays on the CDC 274 Display Console (a refreshed CRT). An interactive graphics program using IGS executes in a host machine, usually a CDC CYBER/70 or 6000 Series computer, and uses the CDC 1700 minicomputer to control some of the basic functions of the CDC 274 graphics hardware. In the development of IGS, CDC started with systems on the CDC 250 terminals (associated with their microfilm recorder), the CDC 1700 stand-alone system, and the CDC 3000 series computer system - all of which were different. In 1968, CDC standardized their product line by bringing together all of the aforementioned systems in line with the (then forthcoming) 6000 IGS support package.

5.6.1.3 UNIVAC software: UNIVAC's basic interactive graphics support library is UNivac Interactive GRAphics Support Package (UNIGRASP). UNIGRASP (Reference 19) is a set of FORTRAN callable assembly language routines which a programmer can use on a UNIVAC 1100 Series computer to display images at the UNIVAC 1557/1558 Graphics Display Subsystem (a refreshed CRT). UNIGRASP was developed to be closely compatible with IBM's GSP.

A second interactive graphics support package which is also supplied by UNIVAC is the Graphic Programming Library (GPL). GPL (Reference 20) provides the software to easily manipulate data structures which describe display images for the display

device itself. GPL can functionally accomplish all of the capabilities present in UNIGRASP and in a significantly improved fashion.

5.6.1.4 TEKTRONIX software: The basic interactive graphics support package for applications that use one of the TEKTRONIX 4000 Series direct view storage tube terminals (DVSTs) is PLOT-10. PLOT-10 provides the functions required in alphanumeric or graphical display generation and interactive processing. It can be used with any computer system supporting ANSI FORTRAN IV standards and alphanumeric terminals with the full ASCII (128 characters) character set. PLOT-10 currently has two distinct parts.

The Terminal Control System consists of a set of FORTRAN IV subroutines to perform the primitive functions required to support interactive graphics applications (Reference 21): output a character to the terminal, receive as input a character from the terminal, position the beam, draw a line, display text, etc. Note that these routines are not only FORTRAN callable but are also written in FORTRAN. The only elements to be provided by an installation wishing to use PLOT-10/Terminal Control System are the routines to handle the input (TIMPUT) and output (TOUTPT) of a single character between the host computer and the terminal.

Advanced Graphing (Reference 22, also part of PLOT-10) represents a set of FORTRAN IV routines which call the TCS subroutines to fulfill the graphing needs for a wide range of problems encountered in engineering or business: drawing a linear grid, labeling an axis, displaying time series plots, etc. This higher level capability - drawing upon the Terminal Control System relieves the implementer of the drudgery of developing a capability for these much needed functions.

5.6.1.5 Other software: The above brief treatment is not meant to be exhaustive. There are other terminal manufacturers, such as COMPUTEK and VECTOR GENERAL (among many others) who have interactive graphics software packages to support the usage of their respective terminals. Moreover there is a whole class of specialized software packages which efficiently address specific problem areas such as Numerical Control, Circuit Mask Layout, Printed Circuit Board, etc.

Table 5-1 lists some of the other existing software support packages developed for specific purposes.

5.6.2 Elementary considerations in the design of a GGL. - The generation of a true general graphics library requires an understanding of the implications of some of the obvious elements of a GGL. Once this is accepted, the more complex aspects of GGL can be appreciated.

5.6.2.1 Interface with operating systems: GGL is a software support package and as such does not exist in a vacuum. It must be conversant with some aspects of the

TABLE 5-1

SOME GRAPHIC SUPPORT PACKAGES IN USE TODAY

Language	Initial Purpose	Interactive?	Passive?	Developed At	Host Computer(s)
DRAFT	Drafting	X	X	U of Wisc.	CDC 6000 Series
GRAPH-PAC	General	X		NSRDC	CDC 6000 Series
FLING	General Standardization	X		Lockheed, Ga.	IBM 360/370 UNIVAC 418 UNIVAC 1100 Series
EUCLID ICAD-M, IC ICAM-2+,3 DISECT	Drafting N/C PCB Wire Wrapping	X X X X		Integrated Computer Systems & Systems, Sciences and Software(S^3)	REDCOR 70 IBM 1130 CDC 6000 Series (other minis)
MASK	Circuit Mask layout	X	X	S^3	CDC 6000 Series
VIP	Non-analytic drawing	X		UC at Berkley, U of Toronto	CDC 6000 Series
FLEX	General	X		U of Utah	PDP-9
CAFE	Non-procedural animation	X (A/N only)	X	?	360/67 CMS
MASK MAKER PROGRAM	IC photomask layout	X		MIT	TX-2
GLANCE	General	X		Bell Labs	GE 635
GRAPHSYS (AED)	General	X		MIT	IBM 360 UNIVAC 1100 Series CDC 6000 Series

TABLE 5-1. SOME GRAPHIC SUPPORT PACKAGES IN USE TODAY (continued)

Language	Initial Purpose	Interactive?	Passive?	Developed At	Host Computer(s)
CALCOMP	General		X	CALCOMP	(many)
S-C4020	General		X	Stromberg Carlson	(many)
APL Graphics	General Mathematical	X		UC at Irvine	Sigma 7 TEKTRONIX 4013 (any computer with APL implemented.)
ITIS, 2	General Transferability	X		Lockheed, Ga. (under contract to NAVSEC	IBM 360/370 CDC 6000 Series
UGLI	General	X		Lockheed, Ga.	?

operating system with which it is being used. To this end, GGL software should not duplicate the supporting operating system/timesharing subsystem software. In fact, GGL would be well advised to build upon IPAD Control Language (ICL) objectives rather than the individual operating system.

5.6 2.2 Terminal characteristics: The totality of interactive graphics terminals comes in a variety of shapes. There are round, square, and rectangular display screens. Moreover the topological addressing schemes have differences not only in magnitude but in sign. A simple linear transformation is usually not sufficiently effective due to differences in character-ratio sizes. For instance, on a CDC 274 (round screen), a character takes 24 rasters out of maximum 4096 raster range (0.6 percent) while on an IBM (square screen) a character takes 14 rasters out of a maximum 1024 raster range (1.4 percent).

5.6.2.3 Terminal flexibility: One way to design a standardized software support package would be to utilize only those common hardware features available on all (or nearly all) modern graphics display devices. This is the approach taken by Lockheed-Georgia in developing the Interactive Terminal Interface System (ITIS, see Table 5-1). The price is terminal flexibility. The design of GGL must not excessively inhibit

the capabilities of one terminal or class of terminals for the goal of standardization. The intersection of capabilities must be identified but alternate solutions for recognized deficiencies must be prepared.

5.6.2.4 Functional classification of general support software: When each of the various manufacturers' software support packages are investigated, there are several common categories of functions to be performed. A particularly straightforward classification of these functions can be found in the paper that discusses FLING - a FORTRAN Language for Interactive Graphics (Reference 23). In that presentation, the authors identify four distinct categories. The following outline presents these categories with a few examples of representative functions within each:

1. Initialization and termination, e.g.:
    a. Attaching to an interactive terminal.
    b. Initializing communication areas.
    c. Releasing an interactive terminal.
2. Image (display byte-stream) generation, e.g., for:
    a. Points.
    b. Lines.
    c. Text.
    d. Circles.
3. Buffer or display manipulation, e.g :
    a. Making item (display byte-stream) visible.
    b. Erasing item from display.
    c. Moving item.
    d. Changing interrupt sensitivity.
4. Attention or interrupt processing, e.g.:
    a. Defining interrupt hierarchy.
    b. Retrieving interrupt information.
    c. Displaying/retrieving tracking symbol.

The design of GGL must surely address each of these areas and thereby advance the initial FLING objectives to a more sophisticated implementation wherein greater standardization is realized without an overwhelming sacrifice in flexibility.

5.6.3 Complex considerations in the design of a GGL - The complete realization of a sophisticated GGL that meets all of the IPAD requirements demands consideration of other complicating factors. In fact, a design of GGL based purely on the factors discussed in the previous subsection would be limited in scope and utility despite the fact that it would undoubtedly advance the state of the art.

5.6.3.1 Different terminal hardware types: There are two basic varieties of hardware being used for interactive graphics - the DVST and refreshed CRTs. *

The hardware deficiencies of the DVST (relative to the refreshed CRT) are centered around the integrity of the display itself, i.e., the essential immutability of items displayed on the screen. For instance, display entities are not light-pen pickable (there is no light pen on a DVST), items cannot be blanked, items cannot be selectively erased, and items cannot be dynamically moved on the screen.

It should be noted that there is usually a topological input device associated with a DVST, sometimes it is an analog tablet or thumb wheel driven cross-hair cursor. With this device and alphanumeric or functional keyboard interrupt processing, it is possible to artificially accomplish many of the aforementioned deficiencies.

In particular, the selective erasing of an item must be done by blanking the entire screen and redisplaying the unaffected items. The method is unmistakably crude but certainly the results are equivalent. It is natural to assume that an application with a lot of this type of activity would hopefully gravitate away from a DVST and toward the usage of a refreshed CRT. The added complication of being able to recreate any part of a display compounds the difficulty in designing a terminal-independent GGL.

5.6.3.2 Division of interactive tasks: The majority of existing interactive graphics software support packages spend a significant amount of time either communicating with or otherwise interrupting the host computer system. The elimination of unnecessary and repetitive interruptions of the host's timesharing subsystem can lessen the impact of interactive applications on the throughput of the total computer system itself.

The inclusion and usage of a minicomputer or programmable display controller which can process locally many of the mundane interactive operations performed at a terminal seems to be the solution of the future to the growing overload problem on the main computer. A futuristic GGL must consider the partitioning of functions between satellite and host, and be able to take advantage of the presence of an intermediate processor to streamline interactive communications. (See Vol. IV, Sec. 5.3 of Part I.)

One current support software package that provides the flexibility of a programmable display controller is UNIVAC's Graphic Programming Library (GPL). (GPL is available for UNIVAC's 1100 Series computers with UNIVAC's 1557/1558 Graphics Display Subsystems.) With GPL, a programmer can instruct the 1557 (display

*See Subsection 4.3.1, Part I, Vol. IV.

controller) via an Interactive Control Table (ICT) to locally process certain interrupts and take appropriate action without ever notifying the 1100 Series mainframe (Reference 20). The satellite (1557) has its own storage containing the data structure of the image- in effect duplicating part of the data structure existing on the mainframe - which allows such functions as windowing, translation, or selective erasing to be done without resourse to the host.

As more powerful minicomputers and display controllers are produced, the mainframe will continue to relinquish more of the mundane requirements for servicing interactive users. The design of a GGL must recognize this trend and actively support such systems.

5.6.3.3 Data accessing and manipulation: The design of a GGL must be aware of its own data requirements and how it coexists with other supporting data systems. The implementation of CODASYL's DBMS on IPAD's target host computing systems will undoubtedly provide for all the data needs of any design of a GGL. If however future interactive systems continue to include minicomputers or display controllers, the probability of maintaining a duplicate of a host's DBMS on the satellite system is small. And yet, if the satellite does not have a local data subbase with which to operate, the number of returns to the host rapidly increases. Thus the design of a GGL must not depend exclusively on a complex DBMS to control its data requirements but instead should consider the effects of trading its data base management activity against the advantages of satellite computers.

5.6.4 Justification for a GGL. - As recently as five to ten years ago there were only several off-the-shelf interactive graphics systems available. There was no identifiable reason to standardize the hardware and/or the software support system. Actually this might have been a good thing, the technology was still developing and premature standardization could have resulted in a product of limited usefulness and have had a stifling effect on new innovations.

The situation today has drastically changed. With new terminals and minicomputers appearing almost monthly, the time is ripe for a step toward standardization and away from wasteful duplication. The advent of inexpensive minicomputers has presented new possibilities that a general graphics software support package should not overlook. Moreover as more sophisticated interactive terminals (both refreshed and DVST CRTs) are developed, the initial design of GGL should be a priori structured to readily utilize these new advancements, at least to the extent possible.

The competitive atmosphere produced by the emergence of the variety of terminals and minicomputer systems has made it financially attractive to be flexible and independent of any one manufacturer's dominance of an installation's total interactive graphics system capability. The added diversity of institutions to be serviced by IPAD adds extra incentive to the realization of GGL.

However other approaches to the problems of standardization and flexibility have been proposed. In particular, there is currently work going on at Naval Ship Research and Development Center (NSRDC) in Carderock, Md., that considers the problem in a slightly different fashion. The concept is "consolidated graphics processing." Mel Haas of NSRDC (Reference 24) notes that currently each graphics I/O display has its own supporting software package and none of these packages are fully compatible. The proposed technique for consolidated graphics processing is to divorce the specification of a graphic display from the hardware commands used to draw those displays. Currently, each support software package produces a data stream of hardware commands specific to the device. This approach defines an intermediate data stream - the Consolidated Graphics Data Stream (CGDS) - of picture oriented commands. CGDS would be produced by substitute programs with the identical call sequences of the standard support software and new programs specific to each device would translate the CGDS to the required hardware commands. (Note: A minicomputer would be a logical candidate for the CGDS translation process.)

The primary benefit of such a system has to be the flexibility to use various graphics I/O devices now offered to the application programs. Moreover, the graphics I/O devices would now be divorced from their supporting software, thus permitting an application written for the CDC 274 to operate as well on an IBM or Vector General replacement. This concept, if it can be realized, means that terminal independent programs written with diverse software support systems can exist. The approach also has one unique advantage. Existing interactive (or passive) graphics OMs are easily transferred between differing computer installations. No reprogramming with a GGL is required. (Note that an existing interactive graphics OM which runs at a given facility will still run under IPAD without reprogramming.) In fact as the investment in interactive graphics applications increases, this approach will become even more attractive.

The current situation however finds many more batch OMs than interactive ones. In addition, with the CGDS concept, another level of overhead is introduced in an area where response and efficiency are most critical. Further, it does not provide for the additional capability sought and provided within the GGL framework (whether or not initially available to IPAD).

It is reasoned that the better long-term approach is to provide a GGL, specifically for IPAD but with the intent of providing for subsequent industry-wide developments.

5.6.5 Summary. - Certain characteristics of a GGL now appear obvious. GGL software should be FORTRAN callable with the ANSI standards for FORTRAN being observed. Moreover, once many of the display subroutine arguments and conventions are adopted, it should be relatively straightforward to repackage all of the major manufacturers' support packages - GSP, IGS, UNIGRASP, PLOT-10, et al - to perform the display functions specified in GGL.

The design of a GGL must include consideration of the two basic CRT types - DVST and refreshed. To service one at the expense of the other would drastically impair the overall efficiency of interactive graphics within an IPAD system.

Finally, the GGL design must acknowledge and utilize the potential of a satellite computer system (when present) to relieve the mainframe of as many interactive terminal servicing requirements as possible. This requires that the development of GGL provide for a separation of function so that selected FORTRAN code employing GGL calls may be implemented on either the host or the mini. Further, some data subbase management will be required on the mini and could be a part of GGL (a la UNIVAC's GPL).

5.7 Conclusions

In order to implement a viable IPAD, languages must be developed and implemented to provide IPAD community-oriented interface with DBMS and interactive graphics support software. Transferability considerations require that the same languages be implemented by all pertinent manufacturers.

Ideally, the languages should be developed by potential IPAD system developers and users, approved by ANSI (see Appendix D) and consistently implemented by the hardware vendors.

As a practical approach it is envisioned that NASA will sponsor the required language development in order to insure meeting IPAD schedules. Aside from the influence of the schedule, the languages should evolve from a CODASYL - like committee (see Appendix E) under the cognizance of both CODASYL and ANSI to insure acceptance as a standard. Implementation is to be insured by the potential IPAD market and through representation of the vendors on the committee.

6 SYSTEM RECOVERY

The IPAD system consists of a collection of computer facilities provided to the IPAD user (Figure 6-1):

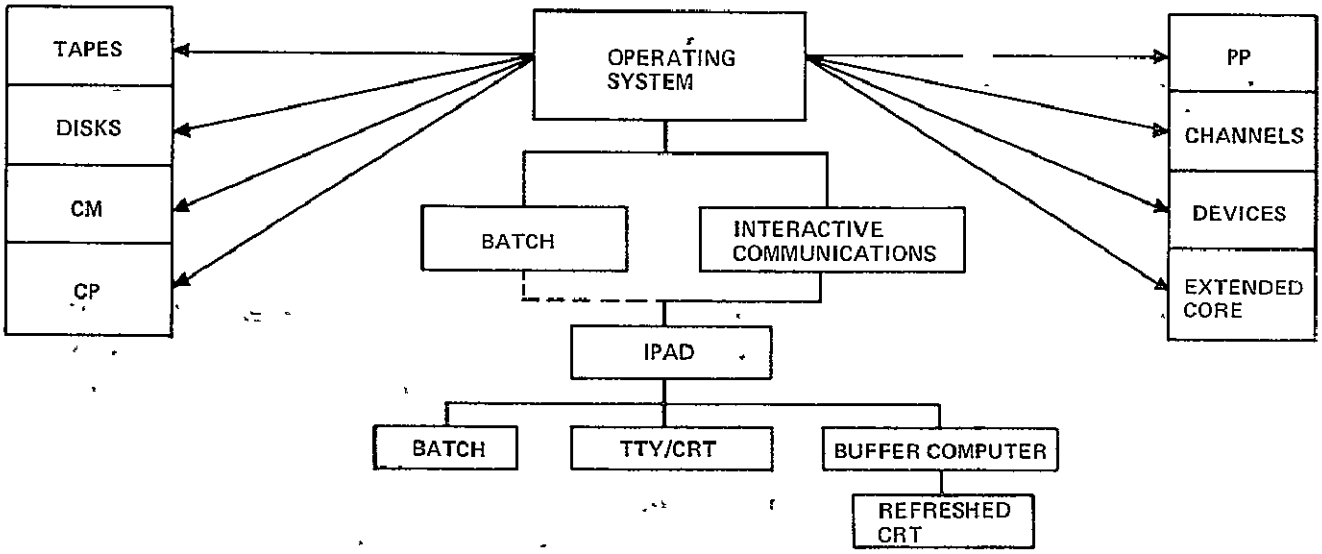


Figure 6-1. IPAD System Overview

1. Computer hardware:

   a. Central processor (arithmetic unit).

   b. Peripheral processors, e.g.

      - Integral buffered processors.
      - Channel couplers.
      - Remote minicomputers.

   c. Memory units, e.g.

      - Main memory (usually core).
      - Remote (extended) memory (usually core).
      - Disks.
      - Drums.

   d. I/O devices

      - Card readers.
      - Card punches.
      - Line printers.

- Tape units:
  - Magnetic.
  - Punch-paper.
- Online recorders (usually microfilm).
- Remote terminals.
- Operators consoles.

e. Communications equipment:

- Modems.
- Multiplexers.
- Channel controllers.
- Telephones.

2. Computer operations/maintenance personnel.

3. Operating system (e.g. CDC's SCOPE 3.4) with various subsystems:

   a. Data Base Management Subsystem (DBMS).
   b. Timesharing (e.g. CDC's INTERCOM 4.1).
   c. Query Processor (QP).
   d. Compilers (e.g. FORTRAN, COBOL, BASIC).

4. IPAD system:

   a. Executive functions (e.g. EXEC, EXPANDER).
   b. Utility functions:

      - Special Purpose Utilities (SPUs, e.g., SCHEMA Assembler).
      - General Purpose Utilities (GPUs).

   c. OMs.

From the user's point of view, a failure at any level in the collection constitutes a failure of IPAD although only category 4 is IPAD specific software.

All computing systems considered adequate for IPAD provide some degree of recovery, either within the hardware or the operating system software. The extent of that recovery differs among systems. This section reports on an investigation of the recovery capabilities offered by the three target computing systems:

1. The CDC Cyber 70 (or 6000) series with SCOPE 3.4, INTERCOM 4.1 and QP/DBMS.

2. The UNIVAC 1106, 1108 or 1110 with EXEC 8, CTS and QP/DBMS.

3. The IBM 370/158, 168 with CMS and QP/DBMS.

6.1 Recovery Within IPAD-Specific Software

This level of software consists of the IPAD EXEC and related software, Special Purpose Utilities (SPUs), the IPAD General Purpose Utilities (GPUs) and the OMs that have been incorporated into an IPAD implementation. IPAD's recovery features depend heavily on capabilities provided by supporting levels, hence they need not be very sophisticated or elaborate.

Specifically, all data associated with a users task (and processed by DBMS) will reside in permanent storage. A failure of any kind will not require repetition of an entire task but rather restart of the OM/GPU being executed at the time of failure. (This is due to recovery capabilities provided by supporting levels.) The IPAD EXEC provides the capability of stepping through a TCS without executing OMs/GPUs that had been completed prior to the interruption. GPUs are to be designed as a number of subtasks with the user in control of the sequence of execution. This provides a second level of task stepping; the user can resume work very near the interruption point.

Nominally, no standards are imposed on the individual OMs, consequently some will and some will not provide recovery capability.

At any restart, there is a requirement to restore the data base to the condition that existed at the beginning of the interrupted process, that is to undo the effects of a partially completed operation. This capability is provided by the next supporting level (following section) but is a user option in establishing an IPAD task, and must be accomplished by the user at restart.

6.2 Recovery Within the QP/DBMS Subsystem

The DBTG report left DBMS subsystem recovery to the realm of individual implementers with the comment (Reference 4 page 21):

> ...the specifications for a complete DBMS should include descriptions and language specifications for:
>
> · the data base Recovery System including activity logging, checkpoint and rollback.

Implementation; as typified by preliminary documentation of CDC's DBMS and QP (i.e. QU/2, see Reference 8 and 9) provides the option of logging before and/or after RECORD images as well as transactions of all modifications of a given AREA.

The user can access the log and Recover/Restore to a specific milestone via QP directives. Given a status of the AREA where the effect of a sequence of transactions has been lost, Recovery consists of repeating transactions to a milestone, i.e. the transaction corresponding to the last completed subtask. Restoring consists of executing the complement of the recorded transactions in reverse sequence.

In addition to functional continuity over system failures, the experienced IPAD user will employ activity logging to support the exploratory nature of his work. Through activity logging he can investigate the effects of a decision, restore the UF to undo those effects and try an approach based upon a different decision.

There are costs associated with activity logging in terms of response time per transaction and extra storage requirements for redundant data. The experienced user is permitted (and required) to make a tradeoff decision for each AREA established in the process of assembling a UF.

The advantages of activity logging are best shown in contrast to conventional file handling techniques outside the IPAD framework:

1. Within IPAD, the user specifies his need for activity logging during UF assembly. Thereafter DBMS performs the activity logging whether the activity is requested by a QPS, an OM, a GPU or other system utilities.

   Outside IPAD, data is normally "saved" a file at a time:

   a. If the user is in close control of the process, he periodically interrupts himself to accomplish the "save".

   b. For non-interactive OMs the "saving" must be provided by the programmer; the user has no facility to accomplish it for himself.

2. Within IPAD (using activity logging) the response time and space requirement costs are directly proportional to the transactions, i.e. to the actual need for redundant data.

   Outside IPAD, the "saved" data is typically a case of overkill, providing an excess of redundant data at excessive cost.

3. Within IPAD (using activity logging) the user has access to the log, can determine exactly what has been done and what has not been done, and can recover/restore to the single transaction level.

Outside IPAD, the user is confronted with file oriented data management and utilities. Files get lost in system failures, the exact status of saved files cannot be determined easily, etc.

The IPAD framework permits the use of conventional files where:

1. The format, content and preparation of the files has been previously standardized so the user need not directly access the files.
2. The user employs a manufacturer supplied utility such as a Text Editor which does not interact with DBMS (at least in present implementations).
3. The user prefers to use conventional files with his own OMs and utilities.
4. The user or DBA is mapping information into or out of the database via QP.

The unnecessary use of conventional files is not encouraged since it requires that the user interact with software which supports IPAD rather than with IPAD itself. Specifically, the system recovery and data management functions dealing with conventional files are not enhanced by either IPAD or DBMS.

6.3 Recovery Within the Timesharing Subsystem

The failures of the timesharing subsystem from which the subsystem attempts to recover are hardware failures in the communications devices. They include failures of modems (at either end), terminals, or phone lines; the most common failure is spurious phone-line disconnects. Most communications failures affect only a single user, but the loss of a multiplexer (MUX) or line adapter can affect a whole group of terminals.

A communications failure results in the termination of any program in progress at the instant of failure. None of the three target computing systems being considered for IPAD offers any automatic facility to restart a program where a communications failure occurred. All, however, provide methods for recovering files. The user is concerned with a UF region of the data base which is composed of one or more permanent files as viewed by the timesharing subsystem. All target systems retain permanent files associated with interrupted tasks.

6.4 Recovery Within the Operating System

Operating systems of the class considered suitable for IPAD contain features designed to maintain stability and reliability in spite of a multitude of programming

errors and software/hardware failures. The basic mechanism is that the hardware detects errors or failures and invokes appropriate software procedures to prevent or recover from harmful effects. Action taken by the procedures depend on the seriousness and nature of the error and on control options. The actions range from most to least serious including:

1. Immediate cessation of all processing.
2. Orderly termination of one or more tasks.
3. Diagnosis of the (permanent) error for the associated task.
4. Repetition of the operation producing a (marginal) error until a correct result is obtained.

The systems also provide utility programs to assist operations and maintenance personnel in preventive maintenance, creating backup copies of data, restoring data from backups, etc.

Various degrees of support for the checkpoint concept is provided but in practice this is not an effective automatic recovery feature relative to system failures.

6.4.1 System Failures. - Operating system failures result from hardware (other than communications) or software failure at the central site. They generally (but not always) effect every user on the system at the same time. (The loss of a tape unit or a private disk pack might effect only a single user.)

All three target systems attempt to recover from system failures in much the same way as they recover from communications failures, i.e., saving permanent files and restoring interrupted tasks. The degree of success in recovery however may be considerably less than with communication failures depending upon the initial cause of the failure. If an output operation is in progress when the system fails the information being transferred will generally be lost. If it is part of conventional file, that file can contain nonsense after recovery; if it is part of a directory, one or more files may be lost altogether. A related problem is loss of synchronization; if a new file is written and then the system fails before the directory can be updated the file may be inaccessible. On the other hand, if the directory is updated first it may point to a non-existent file. The systems attempt to detect any such effected files during recovery, but they are not always successful.

If an operating system failure interrupts an output operation under DBMS, the consequences are not so ominous because the user can inspect data via QP and repair the damage at the RECORD or even DATA-ITEM level. If activity logging is employed, the recovery should be complete and essentially painless.

One type of failure for which recovery provisions should be essentially perfect is loss of power. There is enough energy stored in the power supply's filter capacitors to give the computing system several milliseconds to store away a record of its condition when the failure occurred. In practice, though, a sudden power failure often causes other (often very severe) hardware failures such as memory failures or disk head-track collisions.

6.5 Recovery Within the Computer Operations Group

The computer operations group is responsible for developing procedures to minimize the effects of human errors and supplement the automatic procedures of the software to avoid heavy dependence upon automatic recovery. One of the most common methods is to save all disk-resident files periodically (usually every night) by dumping them onto magnetic tape. A variation is to dump the entire disk only infrequently (perhaps once a week) and dump only new or modified files (i.e., all files that have been opened for writing) every night; this lessens the amount of dumping but makes restoring the disk more difficult in the event of a disk system failure. Which method is used depends upon the frequency of these types of system failures. The archiving feature provided by disk dumping is also a useful protection against user errors: it is fairly common for a user to purge a file and then discover later that he still needs it.

6.6 Recovery Features of Typical Hardware

In large scale modern computer systems, the processing hardware performs two functions which were program responsibilities in antiquity:

1. The detection of a multitude of hardware/software errors.
2. Invoking software procedures to take appropriate action whenever an error is detected.

These are not post-failure recovery functions but are the basis of all recovery. The error handling procedures:

1. Prevent propagation of the effects of errors thus reducing the requirements for post-failure analysis.
2. Correct errors thus preventing failures.
3. Recognize imminent failures (e.g. loss of system residence disk), diagnose the cause for the maintenance personnel, and complete critical operations in progress.

6.7 Conclusions

IPAD-specific software does not include an explicit recovery module. Recovery procedures depend heavily on recovery capabilities provided by several levels of supporting software.

Each level of the supporting software provides some degree of recovery capability to reduce the impact of failures at that level and to enhance the recovery capability provided by levels of software that support it.

Since the IPAD user works with an integrated UF (consisting of permanent files), the QP/DBMS subsystem provides a user-oriented capability to recover/restore to the level of single transactions. This is in contrast to the file oriented recovery capability provided by the timesharing and operating system levels of software. To the IPAD user, the file oriented approach is no help at all. Consequently, the use of conventional files - viz files not processed by DBMS - is discouraged.

For all work attempted within the UF via DBMS, recovery problems are reduced by an order of magnitude and recovery can be accomplished by an unassisted user.

7 SPECIAL PURPOSE UTILITIES (SPUs)

Exploitation of DBMS entails a special human-factors problem to be handled by Special Purpose Utilities (SPUs). This section describes the human-factors problem and provides a functional description of the solution. The overall process also involves the General Purpose Utilities (GPUs) and special utilities associated with the IPAD EXEC. While this section is primarily concerned with SPUs (those whose only purpose is to shield users from the details of DBMS interface), the application of other utilities will also be described.

7.1 Human Factors Considerations: The Requirements for SPUs

The purpose of this group of utility programs is to assist IPAD users and programmers in providing for the requirements of DBMS. DBMS provides capabilities to greatly assist the users and programmers in interfacing with a common data base, but in preparation for this activity a great deal of detailed information is required by DBMS. This information must be supplied and expressed in new languages (DDLs and DML) which would entail a prohibitive amount of time and labor without utility program support. For example:

1. Each potential IPAD installation has a number of currently useable OMs to be converted and utilized within IPAD. This effort alone - without utility program assistance - would degrade the immediate usefulness of IPAD.

2. Learning the new languages is definitely not a trivial task, even for computer oriented personnel (i.e., programmers). An implementation that required the typical IPAD user to learn these languages would not be feasible.

3. For each user-task, the user (who is not required to learn the DDLs) must function as a DBA of sorts, mediating the conflicting requirements of the OMs, and configuring a UF which is a mini data base in itself.

Quite clearly, utility program assistance is required. The utilities provided especially for this requirement are:

1. The DML Insertion Preprocessor.

2. The SUBSCHEMA Assembler.

3. The SCHEMA Assembler.

The remaining sections describe these SPUs.

7.2 Incorporation of Existing OMs Into IPAD

Figure 7-1 illustrates the process of converting existing OMs to operate in the IPAD/DBMS/QP framework. The objective of this process is to perform the conversion task and provide any IPAD user with all necessary material to make use of the OM without further assistance from the originator or responsible programmer.

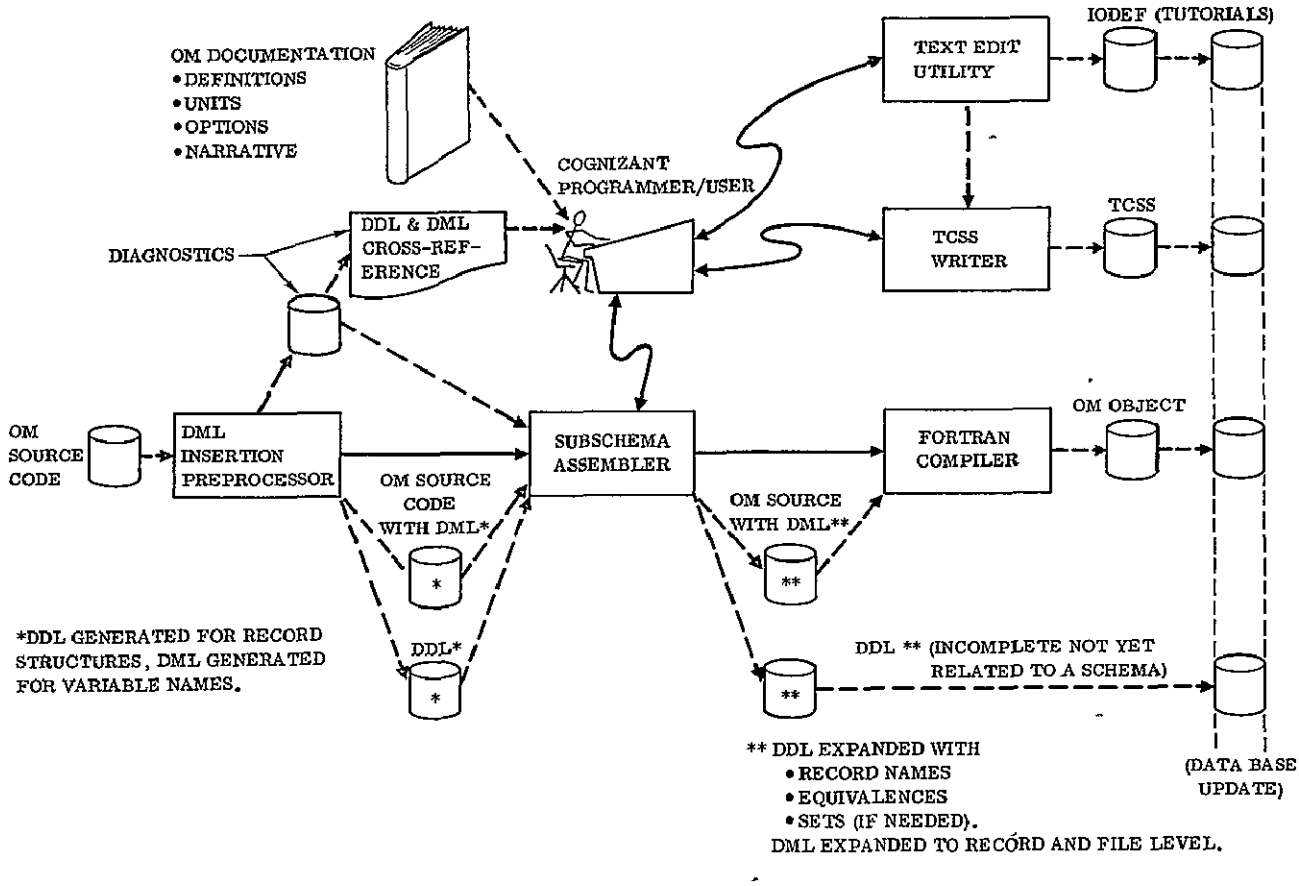


Figure 7-1. Initial OM Incorporation into IPAD

OMs to be incorporated in initial implementations are restricted to the FORTRAN source language (see Section 8). The incorporation process is essentially the same, however, with additional modules to process other OMs when language development permits

The inputs to the incorporation process are:

1. Existing FORTRAN source code of the OM.
2. Programmer's analysis resulting in:
   a. Source code modifications/additions.
   b. Text for tutorial IODEF.
   c. Text for Task Control Sequence Skeleton (TCSS), equivalent to a sample job setup.

The outputs are:

1. Updated version of OM source code.
2. Updated version of OM object code.
3. DDL expressing the I/O requirements of the OM.
4. Tutorial portions of the IODEF (e.g., variable glossary).
5. TCSS to be used in preparing to execute the OM.

The intermediate results produced are:

1. Diagnostics - to call the programmer's attention to problem areas and to summarize the conversion problem.
2. Updated version of OM source code containing FORTRAN DML and representing the first cut at conversion.
3. A first cut at SUBSCHEMA DDL required to interface with DBMS.

Through the SPUs, the programmer/user performs:

1. DML insertion - replaces conventional FORTRAN I/O operations with equivalent DML.
2. DDL generation - extracts data structure specifications from conventional I/O coding and constructs DDL for the SUBSCHEMA.
3. SUBSCHEMA assembly - orders and completes the DDL for a partial SUBSCHEMA (relationship to SCHEMA yet undefined) and insert DDL and DML at the file level.

Through GPUs, and other supporting utilities, the programmer/user performs:

4. FORTRAN compilation - produces a new version of the OM object code to interface with DBMS.

5. Text editing - creates data contents for the tutorial portion of the IODEF to be associated with the OM, and text for a TCSS to be associated with the OM.

6. TCSS writer - edits the TCSS produced via a text editor to finish preparations for the TCSS Expander.

7. Data base update - requests the DBA to insert all outputs (see Figure 7-1) of the conversion process into the data base, which is available for all IPAD users.

7.2.1 DML Insertion Preprocessor. - The DML Insertion Preprocessor (Figure 7-1) is required to assist the programmers in converting conventional FORTRAN programs to interface with DBMS. The objective is to replace conventional FORTRAN I/O coding with logically equivalent DML.

Actual implementation of this capability may be combined with the SUBSCHEMA Assembler (Section 7.2.2) and/or may decompose into two or more separate utility programs reflecting different requirements for interaction with the programmer. Figure 7-1 depicts such an implementation. The mechanical aspects of DML insertion and DDL generation are depicted as occurring within a single batch mode DML Insertion Preprocessor execution. A second pass is depicted during which the programmer interactively "cleans up" and finishes the conversion.

Regardless of implementations, the capability required of a DML Insertion Preprocessor is to replace the procedural aspects of FORTRAN I/O coding with DML, which is the procedural interface between the OM and DBMS.

The requirement for this capability should influence decisions in the DDL/DML language development; the detailed requirements of the capability, in turn, depend heavily on the DDL/DML language development.

An overview of the requirements:

1. Scan the FORTRAN source code of the OM to:
   a. Locate conventional I/O coding to be replaced.
   b. Locate ambiguities which cannot be resolved by the DML Insertion Preprocessor.
2. Permit the programmer to modify the FORTRAN source code as required.
3. Automatically replace conventional coding with equivalent DML:
   a. Locate occurrences of database RECORDs to be input.

b. Cause input RECORDs to be brought into the User's Work Area (UWA).

c. Distribute input RECORD contents from the UWA into internal storage locations.

d. Generate occurrences of database RECORDs to be output.

e. Assemble the contents of output RECORDs from internal storage locations into the UWA.

f. Cause output RECORDs to be transfered into the database.

The COBOL DML requires only that DBMS deliver or accept a contiguous block of data per RECORD. Conventional FORTRAN READ/WRITE statements result in compiler generated code to scatter-READ or gather-WRITE. Replacement of READ/WRITE may require explicit new code to scatter/gather (items c/e above) depending on conventions adopted for DML syntax and compiler extensions.

In summary of the above requirements, the conversion required relative to DML is translation from one set of imperatives to another. A FORTRAN READ statement for example results in three DML commands (per the COBOL DML, see Reference 4):

1. FIND - search the database and establish the existence of the desired RECORD.

2. GET - transfer data values from the RECORD occurrence into the UWA.

3. MOVE - transfer data values from the UWA into internal array, vector, or scalar locations related to mathematical procedures.

The function to be executed by the DML Insertion Preprocessor is to replace the READ with the DML sequence and distribute parameters - associated with the READ - into appropriate operands over the sequence of DML commands. The development of this functional capability is envisioned as a small task relative to the development of the capability to produce appropriate SUBSCHEMA DDL.

The scope of this function falls far short of complete conversion. In any conversion effort the human must inevitably finish the job. The overall process described here provides for the programmer to interactively modify the results achieved by an automatic implementation of mechanical conversion rules.

7.2.2 SUBSCHEMA Assembler. - The objective of the interactive SUBSCHEMA Assembler is to derive SUBSCHEMA DDL from the conventional I/O coding of an OM.

In contrast to the batch DML Insertion capability (Subsection 7.2.1) which is concerned with inserting the logically equivalent procedural interface, this capability is concerned with extracting data descriptions and producing DDL, which is the required declarative interface with DBMS.

As discussed in Subsection 7.2.1, implementation may combine DML insertion with DDL extraction such that the interactive SUBSCHEMA Assembler utility is the "clean up" phase of both functions. The capability to be provided is little more than that of (possibly) correcting and finalizing the source code for the OM and generating the associated DDL.

Regardless of the functional packaging, an analytic capability must be provided to produce a declarative interface with DBMS from analysis of conventional FORTRAN I/O coding. The requirement for this capability should profoundly influence the DDL/DML language development tasks (as they relate to FORTRAN) and the need for new software functions will be almost entirely dictated by the language development decisions. Conceivably, the new functional requirement could be as little as extraction of the FORTRAN FORMAT statements and the LISTs from all associated READ/WRITE statements. FORTRAN compilers and I/O support software have long provided the analytic capability required so that this capability could be transfered to a single utility functioning as the DDL compiler.

7.2.3 Other supporting utilities. - The relationship of other utilities to the DBMS interface problem is described in the following subsections.

7.2.3.1 The FORTRAN compiler: This is the usual manufacturer-supplied utility which converts FORTRAN source code into executable object code. Implementation of IPAD will require the manufacturer to upgrade existing FORTRAN compilers to process the DML enhancements to FORTRAN. Compilation of DML statements provided within the enhancement results in object code to interface with DBMS:

1. Provisions to invoke one or more SUBSCHEMAs at execution time.
2. References to names within the SUBSCHEMA invoked (to permit independent construction and compilation of the SUBSCHEMAs);
3. Calls for DBMS functions.
4. Provision for status communication (e.g., error diagnostic codes).

7.2.3.2 Text editor utility: The text editor is a manufacturer supplied general purpose utility (see Section 3 of Part III) employed by the programmer to:

1. Construct any appropriate tutorial aids (see Section 8 of Part III).

2. Construct a TCSS to include:

   a. Tutorial preface for user interface with TCSS Expander (See Subsection 2.2.1).

   b. Typical OSCL image distinguishing:

      - Invariant portion of the OSCL.
      - Task dependent dummy parameters.

7.2.3.3 TCSS writer: The TCSS Writer is an IPAD utility associated with the EXEC function (See Subsection 2.2.2). The programmer employs this utility to prepare the TCSS as constructed by the text editor for processing by the TCSS Expander. The objective is to locate and correct any syntax errors in the OSCL image and to construct cross-reference pointers between a list of the dummy arguments and their occurrence within the TCS image (See applicable discussion in Subsection 2.2.1.1).

7.2.3.4 Data base update: The data base update process is envisioned as one or more QP Sessions (QPSs) as outlined in Section 4.2. A conversion has been accomplished on an OM and all required data to incorporate it into IPAD has been produced. The remaining task is to install the data within the database and make it available to IPAD users. The exact procedure depends on project administration decisions resulting in DBA activities and controls. Typically, a QPS places data in an update file, along with identification and prefabricated display commands. A message is placed in the DBA's TSA requesting him to review the data and incorporate it in the data base.

The project data base (see Section 4.1) provides a Disciplinary Library File (DLF) to contain utilities, OMs, design data, etc., used primarily within a single functional group or discipline (but accessible by others). The SCHEMA DDL description of the DLF provides for new occurrences of OMs consisting of:

1. QPSs for tutorial aids - which include descriptions of tutorial data.
2. SUBSCHEMA.
3. TCSSs.
4. OM object code and, possibly, OM source code.

7.3 SCHEMA Assembler

The SCHEMA Assembler is an interactive utility with which the user, in effect, integrates a software entity to accomplish a study or design task. Each user has a recurring need for this utility, in that he will need to integrate an appropriate entity

for each task assigned to him. (For a description of this process, the reader is referred to Section 1.3 of Part III). The extent of support to be provided is indicated by noting that an inexperienced, problem-oriented IPAD user is to function in the capacity similar to that of the DBA as it relates to his UFs. The prime responsibility of the DBA is to mediate conflicting requirements of programs sharing data and to develop a SCHEMA which is a compromise. The DBA is envisioned (Section 4.3) as highly skilled and knowledgeable of DBMS and the DDLs. This utility is required to reconcile the difference in skills between the DBA and the typical IPAD user.

Evolution of the user's task is typically:

1. Definition of the task through interaction with the Task Status/ Action file (TSA).
2. Selection of a group of OMs which provide the required capability.
3. Permit examination and re-examination of tutorial data provided with each OM.
4. Permit examination and interpretation of SUBSCHEMA specifications provided with each OM.
5. Through analysis of the SUBSCHEMAs, assist the decision making process. For example, the user may decide (through intuition and/or understanding of the tutorial aids) what data is logically common to the OMs. After this decision, the following types of decisions must be reflected in the SCHEMA DDL without requiring that the user compose SCHEMA DDL directly:
   a. Are redundant copies of common data needed?
   b. If common data is known to various OMs by various names, what is its name to be used in the SCHEMA?
   c. If various OMs describe common data as structurally different, which description minimizes the transformations required of DBMS?
   d. What transformations, other than structural, are to be implicitly performed by DBMS?
   e. What transformations are to be explicitly performed via QP?
6. Produce SCHEMA DDL for his UF which is basically a translation of the assembled SUBSCHEMA DDL, modified by the user's integration decisions.
7. Produce the completed SUBSCHEMA for each OM by inserting appropriate references to the UF into the prototype SUBSCHEMAs provided.

Prefabricated TCSs may be available which would permit the user to repeat the processing which integrated a similar entity previously. In creating a substantially new entity, the utility is required to provide the following support (Figure 7-2 depicts the functional overview):

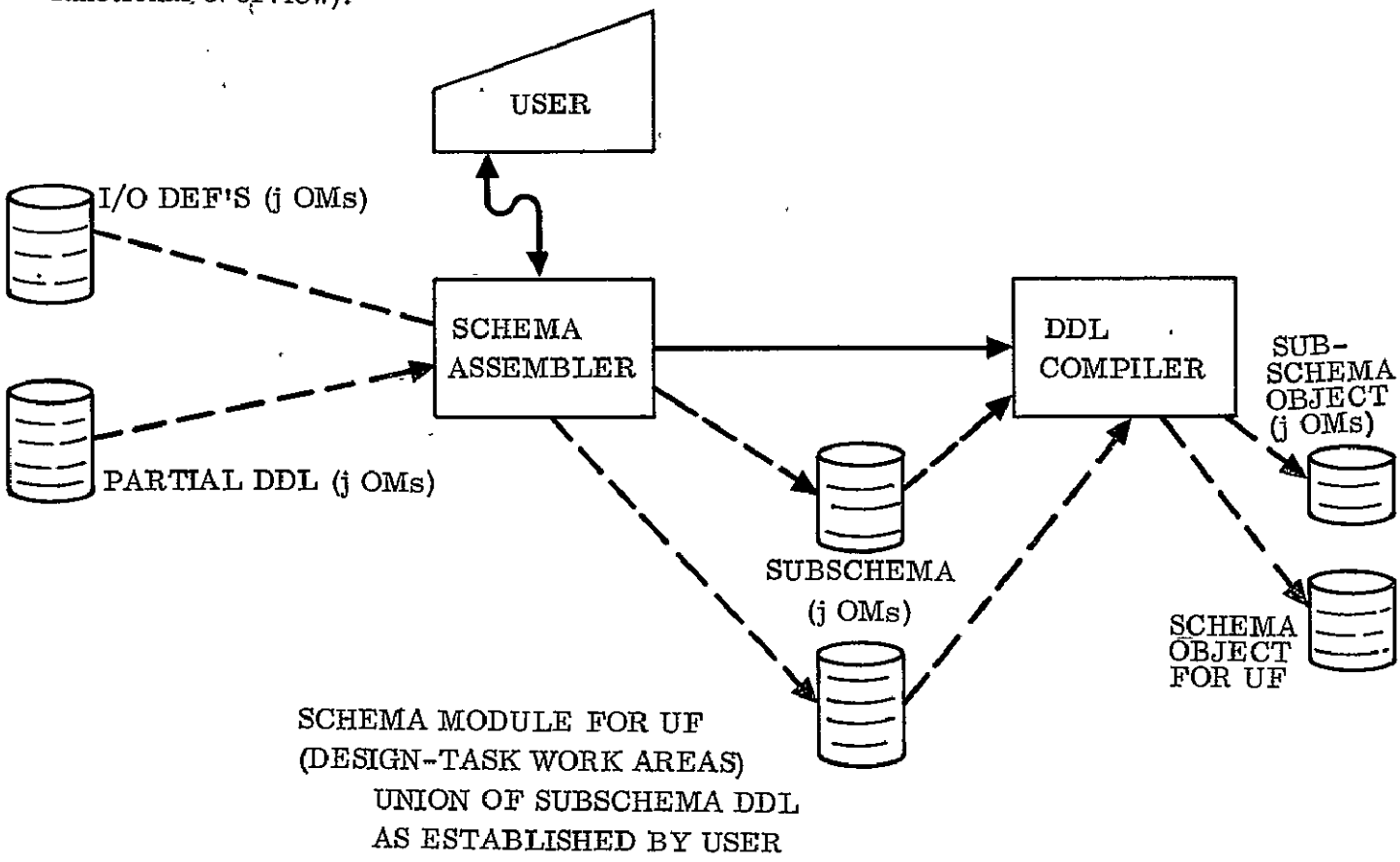


Figure 7-2. OM Interface Resolution Per Design Task Through Definition of a UF in the SCHEMA

The DDL complier produces object versions of the individual SUBSCHEMAs, and an object version of the UF SCHEMA-module to be dynamically appended to the current object SCHEMA.

8 RESTRICTIONS

The intent of the IPAD system design is to provide an environment encompassing as much as possible of the current capabilities at any given installation. The restrictions imposed by the design are consequently very few.

8.1 Restrictions on OMs

An objective of IPAD is to incorporate and enhance the present level of capability in the computerized design process. Consequently the incorporation of all OMs at a given installation, basically without change, was a constraining factor in evaluating design decisions.

Within the IPAD environment, the OMs interface with two software entities not present in their current environment. Restrictions imposed by IPAD therefore relate to the IPAD EXEC and the manufacturer supplied DBMS.

The EXEC permits the full use of the operating system under which the OMs were developed but provides the IPAD user more flexible control through the TCSs. A restriction is imposed here in that the execution of an OM may not alter the TCS which controls its execution.

Interface with DBMS is required to achieve the IOF objectives. This imposes source language restrictions, prohibits asynchronous operations within the OM, and requires source code modifications and extensions. Special Purpose Utilities (SPUs) are provided to assist in constructing the DBMS interface (Section 7).

In order for an OM to interface with DBMS, certain enhancements must be made to the general capabilities of the source language used in developing the OM (see Section 5). The required enhancements for COBOL have already been specified (Reference 4) and the initial IPAD release will require implementation of a DML-enhanced FORTRAN. Other languages will be enhanced as the IPAD community grows and demands language development.

8.2 User Restrictions

From the viewpoint of the system designer, IPAD imposes no restrictions on the user; the user is free to employ any computer capabilities that he might employ without IPAD. The intent of the system design is to provide these capabilities as an integrated

system featuring direct, interactive interface with the user.

Effective use of the system, however, will require some relearning on the part of the individual user and restructuring of project organization:

1. The user must learn - in addition to the normal computer facility operation - the control languages of the utilities which provide the direct, interactive interface. Also, to configure a task oriented UF, he must become familiar with the concepts associated with integrated data bases.
2. The project organization must provide system specialists to perform the DBA function (see Section 4.3).

8.3 Restriction on the Use of IPAD

The thrust of system design was directed toward providing capabilities to be used in any situation involving the systematic use of computer facilities. Restrictions on its use will be imposed only by administrative needs. One such need, obviously, is security.

DBMS provides for guaranteeing the privacy of data designated as private by the DBA. This guarantee does not satisfy DOD security regulations, consequently DOD regulations will prevail. (See Volume IV, Section 5.6 of Part I for a discussion of DOD regulations.)

The same situation may apply with respect to proprietary work. The decision in each case is administrative, not a function of the IPAD design.

TRANSFERABILITY, HOW IT IS OBTAINED

Preliminary considerations concerning transferability were presented in Volume IV, Section 6 of Part I:

1. Transferability was distinguished from transparency.
2. Factors affecting transferability were introduced.
3. Transparency was treated at length and recommendations were proposed.
4. The subject of this discussion was stated: to identify and resolve areas in which transferability can be achieved effectively and areas where the results are not worth the costs.

In the manner of the iceberg concept presented in Volume IV, Section 7 of Part I, IPAD can be decomposed into the following areas:

1. The OMs incorporated into IPAD at a particular installation. The goal of transferability is applicable to general purpose OMs but their transferability is essentially unrelated to the IPAD design excepting that new designs should employ the DBMS approach.
2. Computer system manufacturer-supplied software which, of course, is not considered a candidate for transferability. The related issue to be resolved however - before transfer is attempted - is that manufacturers must provide equivalent capability and support. This software includes:
   a. The host operating system.
   b. The host timesharing subsystem.
   c. Compilers for:
      - Programming languages used, principally FORTRAN, upgraded for DML enhancements.
      - DDL.
   d. A DBMS - per the DBTG recommendations - with any upgrading required by language development (See Section 5).
   e. A QP to provide the user interactive interface with DBMS (as discussed in Section 5.5).

3. An integrating interface, embodied in:

   a. DDL, which defines a data base and permits OMs and utilities to interface with the data base, and

   b. TCSs (and TCSSs) and their subsets (e.g., QPSs and QPSSs), which are prefabricated sequences of command language to control all pertinent executable code.

      At any given installation, the investment in this interface area will probably be greater than in IPAD-special executable code. Consequently, tranferability of this non-executable code will be discussed in Section 9.1.

4. Executable code developed especially for IPAD and required by the integrated framework. This area consists of three software groups:

   a. The EXEC and related utilities are basically intended to interface with a particular machine/system, so intuitively it would be expected that little practical transferability would be achieved (as discussed in Subsection 9.2.3.1).

   b. The SPUs, which function to reduce the impact on human engineering objectives of the DBMS interface languages (DML and the DDLs). Since these languages are to be standardized to a high degree, some degree of transferability will be achieveable (as discussed in Subsection 9.2.3.2).

   c. The GPUs, which support the interactive user by augmenting the capabilities of OMs. Transferability would currently be difficult to achieve because of the variation in graphics capabilities among manufacturers. However, a standard GGL is anticipated (see Section 5.6) which will alleviate this difficulty. Otherwise, the nature of the GPUs is very user-oriented, indicating a high degree of transferability can be achieved (as discussed in Subsection 9.2.3.3)

9.1 IPAD Non-Executable Code

This is the interface with supporting system software which integrates general capabilities into a functional entity. Prior to the concept of a DBMS as recommended by CODASYL, the non-executable interface consisted almost entirely of command or control languages (the OSCL). The scope of the investment in coded sequences of these was such that transparency rather than transferability was the goal. An industry standard DBMS and its standard interface languages change this situation somewhat:

1. A standard DBMS is effectively the same as a transferable DBMS.

2. The task of developing a DBMS (and an IOF capability) tailored to IPAD requirements has been replaced by the task of developing an IPAD interface with the standard, general DBMS.

3. Without this body of software interface there is essentially no IPAD implementation.

Transferability of this code can be achieved by the development and implementation of standard languages as discussed in Section 5.

9.2 IPAD Executable Code

This is the IPAD-special software embodied in modules of source code which - when compiled - results in object code corresponding to computer instructions. This is the area for which transferability was postulated in the conceptual design as an important objective.

The complete set of factors affecting transferability of these system modules, and the relationships among these factors is much too large to deal with here. However, the more important factors include:

1. Choice of appropriate language and proper use of each module.

2. Compatibility of compiler support.

3. Compatibility of hardware/software architecture.

Of these three, only the choice of language (and many choices concerning how it is to be used) is controllable by the IPAD design.

9.2.1 Choice of an appropriate language. - Five computer languages are considered as candidates for the IPAD system modules: FORTRAN, ALGOL, JOVIAL, PL/1 and COBOL. In addition, the various assembly languages are considered as a separate class for comparison purposes. The choice of an appropriate language for each system module depends mostly on the modularity specified in the final build-to specifications. These specifications should permit matching the requirements of the module to the language features as follows:

1. Matching the modularity specified to the modularity attainable in each language.

2. Matching the function required to the applicability of a language to meet that function:

   a. Math applications.

   b. Character manipulation.

c. Logic and control.

d. Functional interface with, and dependency on, the operating system.

e. I/O operations.

3. Matching the programming skills generally available to the programming skills associated with a language as indicated by:

   a. Extent of current (domestic) usage.

   b. Ease of learning.

   c. Ease of use (once learned).

   d. Ease of modification.

4. Matching the transferability required to the transferability attainable in each language.

5. Matching the efficiency required of a system module to the attainable efficiency of the compiled code.

Before a language can be chosen through these considerations, it may be rejected from consideration because it is unavailable for the implementation. Availability of the five languages on the three target systems may be summarized as follows; all languages are available on all three target systems except: PL/1 is currently available only on IBM systems and JOVIAL is currently available except on IBM systems. This situation may change* by the time a language selection must be made, so both language are retained in subsequent discussions.

Having chosen a language for each module based on all the above considerations, the degree to which IPAD software will be transferable is mostly influenced by matching the functions required to the applicable languages. All the higher-level languages considered have a high degree of transferability attainable. However, tranferability is often not achieved because the design application neglected to match functional modules to applicable languages. As a rule, one language is used, rather a mixture of languages, to code an entire software capability. Transferability is further degraded when each programmer employs his own techniques to overcome deficiencies of the single language.

9.2.2 Candidate language comparison. - The pertinent features of the six candidate languages are compared in Table 9-1. Each language is subjectively rated on the features presented in Subsection 9.2.1. The rating reflects the appropriateness of the language to an IPAD system module application with respect to each feature. The ratings are

* For example, CDC is said to be working on a PL/1 compiler.

picked from a numerical scale of 1 (low) to 10 (high) in order to indicate a relative comparison. No measurements are implied by the quantification; the ratings are assigned on the basis of a "consensus" of opinion in the literature and among programmers. Each language is briefly discussed in the subsections which follow.

TABLE 9-1

CANDIDATE LANGUAGE COMPARISONS

	FORTRAN	ALGOL	JOVIAL	PL/1	COBOL	ASSY
MODULARITY	10	6	4	9	9	10
APPLICABILITY TO:						
1. Math Applications	10	10	9	10	6	10
2. Character Manipulation	1	1	6	8	8	10
3. Logic and Control	6	8	8	8	7	10
4. OS Interface	1	4	4	4	1	10
5. I/O Operations	4	4	6	6	8	10
LEVEL OF SKILL MEASURES:						
1. Current Usage	9	2	4	6	10	10
2. Ease of Learning	8	4	4	6	8	1
3. Ease of Use	6	6	6	8	6	5
4. Ease of Modification	6	1	1	3	6	3
TRANSFERABILITY ATTAINABLE	9	7	8	6	9	1
EFFICIENCY OF COMPILED CODE	8	6	8	4	8	10

9.2.2.1 FORTRAN: FORTRAN is among the most widely known and used languages in the scientific field and is available on all large (and most small) computers. Due to its long usage and wide acceptance it has undergone many revisions to become quite efficient. Most FORTRAN compilers produce efficient code. FORTRAN is strong in mathematical processing and good at logical decision making. The I/O features are good but, as in all higher-level languages, these are often implemented by assembly language subroutines of rather low efficiency.

FORTRAN is weak or deficient in scaled fixed-point arithmetic, address or pointer manipulation, run-time storage allocation, recursive or reentrant programming, and capability in processing part-word data, such as character manipulation.

9.2.2.2 ALGOL: ALGOL enjoys wide usage in Europe but is not widely used in this

country. It has many of the features of FORTRAN and includes some reentrant and recursion capability. The unfamiliar reader has difficulty following the instruction flow due to a feature which includes deeply nested BEGIN...END blocks. There are a number of ALGOL compilers available but most are inefficient.

9.2.2.3 JOVIAL: JOVIAL is an outgrowth of ALGOL and is principally used by the Air Force. It has the advantageous feature of declaration-based design which allows high transferability. It has a cumbersome interprogram communication method called COMPOOL (which functions somewhat like unnamed common) which complicates program modification. JOVIAL compilers are not widely available; for example, there is not known to be one for the IBM 360/370.

9.2.2.4 PL/1: PL/1 is principally available on IBM 360/370 Series computers. It is very close to containing all the best features of all the other languages discussed. It does have the disadvantage of being extremely complicated as is ALGOL and is difficult to follow by an unfamiliar reader. It has extensive data declarations (increased transferability) which languages such as FORTRAN do not contain. Unfortunately compilers do not generally exist for other than IBM computers.

9.2.2.5 COBOL: COBOL is basically a business language but has strong capability in character manipulation and file handling which could be of some use to IPAD. It is a relatively simple language but somewhat tedious to use because of its verbosity. It is widely known and used - perhaps even more than FORTRAN - and compilers are available on most large computers.

9.2.2.6 Assembly Language: If a job can be done at all on a computer it can be programmed in its assembly language. Assembly language permits the highest degree of efficiency attainable for a particular machine. It offers very powerful facilities such as synbolic parameters, conditional assembly, and macros which make it possible to reconfigure very general programs simply by changing definitions and reassembling. These features are rare among higher-level languages. Most assemblers offer the helpful facility of printing comments on the same line with instructions which most compilers do not.

A high degree of skill in programming is always evident in the degree of efficiency attained through use of an assembly language.

9.2.3 Overview of functions to be performed. - As discussed in Subsection 9.2.1, the primary factor in obtaining overall transferability is matching required functions to applicable languages. This discussion characterizes the groups of IPAD syste software in terms of types of functions listed in Subsection 9.2.2.

9.2.3.1 The IPAD EXEC and related utilities: The essential function of the EXEC is to augment the features of the host operating/timesharing system. Primarily this entails interfacing closely with the existing capabilities of the host computing system. The functions to be accomplished are often so closely related to the host system that at times a functional requirement must be achieved via a modification to the host software as opposed to incorporating the code in the EXEC. (e.g., CDC's INTERCOM 4.1). As is evident, attempts to make the EXEC a transferable entity would probably not be worth the cost.

EXEC-related functions include the TCSS Writer and TCSS Expander. These functions are primarily concerned with manipulating character strings to produce valid control language sequences to control the activities of the operating/timesharing system as well as any interactive capability. A secondary activity is to provide tutorial assistance to the casual user concerning the requirements of any particular control sequence. Further, reduction of these functions to logical modules should identify modules in which character manipulation is independent of the control language being produced, and the tutorial assistance modules are independent of the lesson being taught. However, the total contribution of these modules would trade badly with the inefficiency inherent in providing such small modules.

Considering the three functions as three monolithic entities, no available high-level language seems to be well adapted to coding the EXEC and related utilities. FORTRAN is very weak on character manipulation and completely lacking in recursion, although both of these deficiencies could be overcome by assembly-language subroutines. ALGOL provides recursion but is no better than FORTRAN for character manipulation; COBOL is strongly oriented toward character manipulation but lacks recursion. PL/1 has all the necessary capabilities but compilers are generally available only on IBM systems. The only practical approach, then, is to use assembly language with no transferability, i.e., program the EXEC in the assembly language of each "target" host computing system series.

9.2.3.2 The SPUs: The nature of the SPUs is similar to the TCSS utilities of the IPAD EXEC; they are primarily concerned with manipulating character strings in order to produce valid source code, and are secondarily concerned with tutorial assistance. However, important differences should be noted:

1. Standard languages are involved:
    a. FORTRAN (e.g., programming source).
    b. SCHEMA DDL.
    c. SUBSCHEMA DDL per standard host language (FORTRAN).
    d. DML per standard host language (FORTRAN).

2. Tutorial content as well as the method of presentation could be transferable.

3. Less stringent efficiency requirements would be tolerable:

   a. DML Insertion Preprocessor and SUBSCHEMA Assembler are used only for non-recurring tasks.

   b. SCHEMA Assembler is used approximately once per user task (as described in Section 1.3 of Part III).

This is an area, then, where very nearly total transferability is practicable.

9.2.3.3 The GPUs: The GPUs are the only IPAD software concerned with the ultimate use of IPAD, that is with actual design engineering. The EXEC and SPUs are equally applicable to any computerized activity. Consequently, the development of the GPUs requires the marriage of engineering and programming skills found in the FORTRAN community. FORTRAN will undoubtedly be the optimum choice for the majority of the GPU modules, even if it is not the best choice in the overall trade presented in Subsection 9.2.1. The point to be resolved here then is how to obtain transferable FORTRAN coding.

It is unrealistic to classify FORTRAN as a highly transferable language; however, a fair degree of transferability may be attained if certain sensitive areas are avoided such as character manipulation, use of shifts or logical operations on numeric operands, or other operations which vary widely in execution on various machines. Modularity is the mechanism by which sensitive areas are to be avoided. Some of the more obvious problems and pitfalls are discussed below.

Part-word data manipulation.- The biggest problem area is part-word data manipulation. There are two main reasons for this. The operators and functions used for accessing and working with part-word data (AND, OR, SHIFT, etc.) are not part of ANSI-standard FORTRAN, and their syntactic forms and semantic effects vary from one compiler to another. (The operators .AND., .OR, and .NOT. are defined in ANSI FORTRAN but only for use with logical variables, not for use with numeric variables.) The other reason is that whatever operators are actually used, accessing part-word data via inline code is completely dependent upon word width. Each of the three machine families being considered for IPAD has - as might be expected - a different size word: 32 bits for the IBM 360 or 370 series, 36 bits for the UNIVAC 1100 Series and 60 bits for the CDC Cyber 70 (6000) Series.

One major use for part-word data is in character manipulation, which is burdened with some extra problems of its own as well. The number of bits per character and the number of characters per word also vary from machine to machine: four 8-bit characters per word for IBM, six 6-bit or four 9-bit characters (depending on the

application) for UNIVAC, and ten 6-bit characters for CDC. The internal character codes, collating sequences, etc., naturally differ as well.

Character manipulation in FORTRAN should be avoided by employing character manipulation modules to be written in assembly language.

Numerical precision.- The classic transferability problem area - the one that most programmers worry about first when they have to move a program to another computer - is numerical precision. This problem is so well known that it is probably not necessary to do more here than merely point it out, and admit that it cannot very well be avoided. If the problem to be solved needs a given level of precision then it must be run on a machine that will provide at least that level. In some cases the transfer can be made by changing REAL variables to DOUBLE PRECISION, but not much can be done if the variables that need it are COMPLEX or INTEGER or are already DOUBLE PRECISION. It might also be noted that cases exist in which a mathematical model contains marginally stable equations. For a certain application the numerical precision yields satisfactory results but added precision may not.

Here again, when numerical precision becomes important in providing a capability, the design should specify that the required precision be achieved through separate precision-arithmetic modules.

Number-representation.- Differences in number-representation can also cause problems when EQUIVALENCE statements are used to overlay different types of data onto each other. Thus EQUIVALENCE should be used only to set up shared storage, not to trick the compiler into accessing data of one type as if it were some other type (e.g., referring to a floating-point word as an integer).

There are still some problems even when the EQUIVALENCE statement is used only for storage sharing, but the three "target" computer families considered will not be affected. On them, as on most current large machines, an INTEGER, REAL or LOGICAL data item occupies one word and a COMPLEX or DOUBLE PRECISION item occupies two words. This keeps all length and subscript relationships fairly simple, either one-to-one or two-to-one, when any type is EQUIVALENCEd onto any other. However, some newer computers are quite large but have relatively short words (16 - 24 bits); on these, an INTEGER or LOGICAL item occupies one word, REAL two or three words, DOUBLE PRECISION three or four words, and COMPLEX four to six words. This leads to rather complicated relationships for calculating relative lengths and subscripts.

This class of problems should not arise since the build-to specifications should prohibit them. For this particular problem, the effect of mapping one data type onto

another can be achieved by a set of no-operation modules to trick the compiler.

Restrictions on language features.- To maintain transferability between different compilers it is essential to use only those language features that are available with essentially all FORTRAN compilers. In practice this will probably mean restricting the coding to the ANSI-standard language set, which will mean giving up quite a few useful features. For example:

1. Statements like NAMELIST, ENCODE, DECODE, PARAMETER, BUFFER IN, BUFFER OUT, IMPLICIT, etc.
2. Semantic extensions to the language, such as right-adjusted character constants and format conversions, multiple entries to subprograms, error returns from subprograms, etc.
3. Syntactic extensions, such as names longer than six characters, quoted character strings in constants and formats, etc.
4. Built-in functions like SHIFT, AND, OR, FLD, etc.

Built-in functions can cause trouble even when they are not intentionally used. If a program that calls a user-defined function named, say, XXX is transferred to a system whose compiler provides a built-in function named XXX, the built-in function will be used instead. To avoid this problem, all user-defined functions should be declared in EXTERNAL statements.

9.3 Conclusion

The areas in which transferability can be obtained at a cost commensurate with benefits consist of:

1. Non-executable software interface code, principally interfacing with QP/DBMS.
2. The SPUs.
3. The GPUs.

Transferability of the non-executable code can be obtained only through standardization of the languages and compliance by manufacturers.

The primary technique by which transferability of the executable code will be obtained is through modularity to isolate types of functions and employing applicable languages for:

1. Math applications.
2. Character manipulations.
3. Logic and control.
4. Interface with the operating system.
5. I/O operations.

Transferability of the SPUs can be very nearly perfect if the software interface languages are standardized.

Transferability of the GPUs will be degraded somewhat by the necessity to employ FORTRAN almost exclusively. The maximum-practicable transferability will be achieved through programming standards which avoid sensitive areas of the language. A suggested list of standards is contained in Section 1.4 of Part III.

10 CONCLUDING REMARKS

The role of DBMS - as envisioned by CODASYL's DBTG - is undoubtedly the foundation of the IPAD design as presented. This first became apparent in Section 3 and became the central theme of the subsequent sections. DBMS and its interactive user-interface (QP) is the major manufacturer-supplied software being exploited by IPAD.

The intent of Part II was "to present a viable system design consistent with the objectives put forth in the Conceptual Design" (Section 1). Section 10.1 reviews the Conceptual Design (Section 2 of Part I, Volume IV) to determine the extent of meeting these objectives. Section 10.2 reviews the dependence on DBMS.

10.1 The Conceptual Design Revisited

The system design formulated in this part met the Conceptual Design formulated in Section 2 of Part I, Volume IV in every respect except one (viz., the earliest release possible, see Subsection 10.1.5). Slight variations in design implementation are noted in the discussion of the subsections which follow. These subsections make direct reference to corresponding subsections in Part I, Volume IV.

10.1.1 The objectives as they related to the host operating system interface (Subsection 2.2.2 of Part I). - The objectives related to the host operating system interface are:

1. "The IPAD system design shall be open-ended: limitations shall arise only through the host computer's hardware/software constraints rather than IPAD's design approach":
   a. OM incorporation is essentially unrestricted (see Section 8).
   b. The IPAD EXEC permits any host computer operating system facility available without IPAD (Section 2) providing it is compatible with DBMS.
   c. The DBMS interface permits flexibility unlimited (Reference 4).
   d. "IPAD's design approach" is to reduce the tedium of the man/machine interface. There were no restrictions imposed on this approach.
2. "The developed IPAD software shall be as transferable to other computer installations as is practicable":

a. Areas amenable to transferability are identified and the practicability delineated (Section 9). (Refer also to Section 6 of Part I, Volume IV.).

3. "Maintenance and modifications required by IPAD to achieve increased capability or retain an acquired capability during a computing system upgrade shall be minimized":

   a. Every attempt has been directed to exploit host software rather than duplicate it; thus maintenance and required modification are minimized.

   b. System dependencies are isolated to the EXEC and standard interfaces where required modification can be more easily provided.

10.1.2 The user (Subsection 2.2.3 of Part I). - The IPAD user is principally accommodated through the GPUs (Part III) and the flexibility of organizing his task (Section 1.3 of Part III). Virtually any existing OM is at his disposal (Section 8).

10.1.2.1 Applicability (Subsection 2.2.3.1 of Part I): The design approach is applicable to virtually any user directing any computerized activity.

10.1.2.2 The role of interactive computing (Subsection 2.2.3.2 of Part I): Interactive user control is emphasized throughout the design except that no provision is made to provide user interactive interface within his non-interactive OMs. OMs run within IPAD with the same interactive capabilities they have without IPAD:

1. Graphics capability is provided through a GPU and through any OM featuring graphics.

2. Tutorials are provided within the GPUs, SPUs, and EXEC functions. Any OM may provide its own tutorial support if desired (Section 8 of Part III).

3. Standard units/coordinates are not imposed on the individual user. Transformation support is provided to permit the user to work with the systems he requires. (Section 9 of Part III).

10.1.3 The TCS, a command structure (Subsection 2.3.3 of Part I). - The TCS - as envisioned in the Conceptual Design - has been split into the TCS associated with the EXEC and the QPS associated with QP. Both are supported by skeleton files (TCSSs and QPSSs) and the TCSS Expander (Subsection 2.2.1).

The primary feature of the TCS is that it is an interactive user interface which is isolated from and thus independent of executable code. The EXEC and QP provide the capability to store, retrieve and execute prefabricated command strings (TCSs and

QPSs), thus providing:

1. The ability "to execute complicated task steps automatically" by issuing a single command.

2. "Unlimited flexibility in the arrangement of the OM execution sequence"- sequences of subtasks (see task integration, Section 7.3) - although some limit is inherent in the executable code of the OMs both in arranging sequences and in monitoring the execution.

3. A common command structure is provided for interactive or batch mode operation (see TCS Interpreter, Section 2.4).

4. "Full control over the design process" is maintained by the user. Task integration permits the user to provide for a control session between subtasks (OMs).

5. Each task is a user-organized entity (Sections 7.3 and in Part III Section 1) which is "readily adaptable to change, thus improving or extending its useful life."

10.1.3.1 User-organized system (Subsection 2.3.3.3 of Part I): The user-organized system which became the basis for the IPAD system design was described as possessing two features:

1. "Data paths for OMs softwired (constructured by user) during checkout following initial OM incorporation." The "data paths" are actually embodied in the SUBSCHEMA DDL as provided by the SUBSCHEMA Assembler during OM incorporation (Section 7.2).

2. "Data Paths for OMs modified as required by user during use." This modification of the "data paths" is accomplished by the SCHEMA Assembler interactive SPU during task integration and/or by QP (interactively or via QPSs) during task execution (Section 7.3 and Section 1.3 of Part III).

The data path linkage is actually provided by DBMS during OM execution.

10.1.3.2 An example of a TCS (Subsection 2.3.3.4): "TCSs are written with the help of IPAD utilities using the interactive capabilities of IPAD". The contruction of a TCS or QPS (a form of a TCS) can be accomplished via the TCSS Expander (Subsection 2.2.1), or via the TCS Interceptor (Subsection 2.2.3), and constructing a QPS can be accomplished utilizing QP (Section 3.6).

Part I presents typical steps of a TCS (much of which are steps of a QPS). The capability to reverse a step implied in the figure and related text is the

"rollback" capability provided by QP/DBMS (Section 6.2)

10.1.4 Incorporation of the OMs (Subsection 2.3.4 of Part I). - The objectives with respect to OMs are completely met:

1. Incorporation - preparation of OM to make it available to IPAD users. This is covered in Section 7.2:

   a. Stated objective is met; simple modifications to the OMs are however required to provide the DBMS interface. User confidence in his OM is retained because:

      - No extensive changes, logic and mathematics need no modifications.
      - SPUs assist with modifications required, however provision is only made for FORTRAN OMs. (OMs in other languages must be modified by hand, viz. no SPU is provided at this time.)

   b. Replacement of an existing OM with a modified version is permitted and may be transparent to the user, depending on the OMs themselves. Modifications for efficiency generally take the form of complete I/O redesign for efficient RECORD management. Other modifications can enhance capability. Certain related changes may be required such as:

      - Interface with the operating system (TCSs and TCSSs).
      - Interface with DBMS (SUBSCHEMA).
      - Interface with user (altered interactive capabilities).

   c. No modifications to IPAD software or organization is required to add, delete, or replace capabilities in the form of OMs as new techniques develop.

2. Deployment of OMs by the user is covered in Section 7, and in Section 1 of Part III:

   a. The user, with the help of tutorial aids (Section 8 of Part III), selects OMs to provide the capabilities required for his task and expands associated TCSSs to configure a sequence of OMs/utilities.

   b. The user employs the SCHEMA Assembler to configure a UF appropriate to the data requirements of the selected sequence of OMs/utilities.

   c. QP provides the capability to initialize the UF, and QP, (as a utility) is available to be configured into the OM/utility sequence as the user requires.

3. The IDEFs and ODEFs are organized slightly different than presented in Section 2.3.4 of Part I and the related text. Specifically, neither the IDEF nor the ODEF contains the location/format of the variables; these are in the related SUBSCHEMA. The remaining portions of the IDEF and ODEF are provided only for the user (Subsection 3.5.6.2).

10.1.4.1 The I/O Formatter (IOF) utility (Subsection 2.3.4.1 of Part I): The IOF utility envisioned in the Conceptual Design has not been provided as such. The prominent place occupied by this utility in the Conceptual Design has been pre-empted by the capabilities provided by QP working through DBMS. Both QP and DBMS are to be provided by computer manufacturers. This is explained at great length in Section 3.

10.1.5 IPAD System software (Subsection 2.3.5 of Part I). - Not all the objectives related to the IPAD system software were achieved:

1. "Strive for the earliest release possible for the IPAD system consistent with satisfying the system objectives and immediate user needs". This objective was deemphasized in the interest of providing a viable system with low cost and risk. The exploitation of a standard DBMS entails the process of:
   a. Language developement (Sections 5.2, 5.3 and 5.5).
   b. Implementation of DBMS/QP by the manufacturers.
   c. Implementation and checkout of IPAD on representative computing systems.

Several of the objectives with respect to IPAD system software (as distinguished from OMs) are also met by adopting the industry-standard DBMS:

2. "Minimize the impact of future computer hardware/software development."
3. "Avoid (where practicable) non-standard software development."

The remaining objectives are covered with the programming standards for IPAD executable code (Section 9.2, and Section 1.4 of Part III):

4. "IPAD software shall be modular to the function level."
5. "In both design and implementation, all machine dependent code shall be clearly identified and isolated."
6. "The IPAD system software shall be structured modularly to aid in reducing the time and effort required in transfering IPAD software to different hardware or software installations."

Note however that development of the IPAD EXEC to comply with all the listed objectives related to IPAD software is not practical because:

1. Every hardware/software future development will impact the EXEC.
2. The EXEC is tailored to a particular version of a particular operating system, so cannot be coded as standard software.
3. To isolate all hardware/software dependencies would be a self-defeating effort.
4. The EXEC is considered non-transferable, and must be tailored to a specific computing system.

However, it is envisioned that the quantity of executable code associated with the EXEC is small enough to warrant overturning the stated objectives.

10.1.6 The data bases (Subsection 2.3.6 of Part I). - All objectives of the conceptual design with respect to the data base are met by exploiting the QP/DBMS facilities. These objectives mesh perfectly with the objectives outlined by the DBTG in Reference 4 and will not be listed again here.

The conceptual design specified essentially that every "file" in the system be self defining; that it should contain:

1. A definition of its (arbitrary) structure.
2. A directory of its (arbitrary) contents.
3. A glossary of its variables.

The intent of this concept was that these should be explicit and available rather than submerged in executable code. The intent is more than adequately provided for in SCHEMA/SUBSCHEMA DDLs (Section 3) and Tutorial Aids Support (Section 8 of Part III).

All of the data bases envisioned in the Conceptual Design were easily provided for via SCHEMA DDL (See Section 4 and Appendix F).

10.1.6.1 Multidisciplinary Data Bank (MDB) (Subsection 2.3.6.2 of Part I): The various "versions" of the MDB as discussed in Section 2.3.6.2 of Part I are easily accomplished by the SET relation (Subsection 3.7.4).

10.1.6.2 Miscellaneous file types (Subsection 2.3.6.4 of Part I): The restriction of various "file parts" (AREAs) to "be transient from tapes or private disk packs" is provided for in the DBMS concept (Reference 4, p. 25) via the Device/Media Control Language (DMCL, ibid, p. 22).

10.1.7 Summary of features of the Conceptual Design (Subsection 2.3.7 of Part I). - The features and operating philosophy listed were fully met. The generality of the "structure, type and contents of the file" (Item 5 of Subsection 2.3.7 of Part I) was met physically through the SCHEMA description and separately met functionally through the SUBSCHEMA description.

As a point of clarification, the IPAD EXEC "is aware of the attached device's I/O limitations" (Item 10 of Subsection 2.3.7 of Part I) through the sign-on procedure the user must employ. Thus the device type, including batch (i.e., no device), is available to the EXEC and each IPAD GPU as required.

10.2 Dependence on Manufacturer-Supplied Software

The IPAD system design - as presented - depends to a larger degree on manufacturer-supplied software than typical developments. This circumstance arose through the desire for

1. Transferability (low machine dependence).
2. Low cost (both developmental and operational).
3. Long life.

Each of these will be briefly discussed.

Transferability (or portability) is classically achieved by coding in such a manner as to operate successfully on any of the target computers. This in turn necessitates avoiding advanced features that are available on some but not all of the target computers. This approach leads to operational inefficiencies (adverse operating costs) that are particularly dramatic in the areas of executive functions (viz. poor response time) and data base management (viz. extreme run-time costs). A functional alternate to transferability avoids the issue entirely by providing separate code for each of the target computers in the most sensitive areas. This separate but functionally equivalent code exploits the existing operating system features of a given manufacturer thus providing the best operational characteristics on his machines.

The disadvantage of this separate but equal approach is creating or drawing upon special purpose software. This special purpose software - unlike, say, a FORTRAN compiler - does not become the responsibility of the computer manufacturer during computing system upgrades (as does the FORTRAN compiler). Thus the creation of special purpose software can (and does) lead to substantial refurbishing costs in the

long run, typically being several times the developmental costs for successful systems (viz. those which have a reasonable economic life). These costs can be considered a part of operational costs and thus contribute to a shorter economic life.

The strategy then is to exploit - to the extent practicable - the operating system through standard software, like the FORTRAN compiler. This standard software is generally under the jurisdiction of one (or more) of the standardization bodies (e.g., ANSI or CODASYL) and typically results in functionally equivalent, manufacturer-supplied software to support that function. Typically the manufacturer-supplied compilers fall into this category.

The design as presented exploits standard (or soon to be standard) manufacturer-supplied software to the maximum extent practicable:

1. DBMS
2. DDL compiler(s) supporting DBMS.
3. DML enhanced compilers, specifically:
    a. FORTRAN
    b. COBOL

It also exploits existing (or soon to exist) manufacturer-supplied software that is a candidate for future standardization, but currently provides functionally equivalent (albeit operationally different) capabilities:

4. Text (context) editors (see Section 3 of Part III).
5. QP (operating through DBMS).

However, there remains certain special purpose software, the responsibility for which cannot be shifted to the manufacturer (cannot, that is, until acceptance and usage reaches a high level among the manufacturer's computer users):

6. The core of IPAD's EXEC.

It is this latter category that represents the principal refurbishing costs associated with this IPAD design.

Naturally, exploitation of existing (or soon to exist) developments will beneficially influence both developmental as well as operational costs (the latter through more efficient code specific to the intended function). Although often not apparent, this exploitation contributes as well to long life through low-(or no-) cost IPAD upgrades through computer operating system upgrades. This provides for an increasingly beneficial tool with low maintenance costs. Life - or more properly economic life -

terminates when the system can no longer provide cost-competitive service to the prevailing user needs. More realistically, system life is somewhat foreshortened due to user dissatisfaction with the system, usually through impatience or boredom.

In an economic sense, it is initial acceptance and long life which is the primary objective for IPAD. In this way can the developmental and maintenance costs be amortized over a long, productive time span. The secondary objective must be to see it eventually incorporated into the then existing computer operating systems (shifting the maintenance burden to the manufacturer and, hence, to the broad base of users).

A more detailed discussion of the dependence on manufacturer-supplied software and the developments begun by CODASYL may be found in Section 2 of Volume VI.

REFERENCES - PART II

1. Richardson, D. G.; Schappelle, R. H.; and Yoshihara, S.: Program P4006C, A Six Degree Digital Program, Volume II, Program Handbook. Report GDC-DDE-021, General Dynamics Convair Aerospace Division, July 15, 1968.

2. Lloyd, J. R.: Users Manual for Structural Analysis Program P4137, Report SA-72-03, General Dynamics Convair Aerospace Division, April 5, 1972.

3. Anon.: Control Data 6000 Computer Systems 274 Interactive Graphics System. SCOPE 3.3 Reference Manual 17303600, Control Data Corporation, Revision D, October 15, 1971.

4. Jones, J. L.: CODASYL Data Base Task Group Report, (no report number), Conference on Data Systems Languages, April 1971.

5. Anon.: UNIVAC 1100 Series Data Management System (DMS 1100). (no report number) Revision 2, UNIVAC (Roseville, Minnesota), June 1972.

6. (System Design & Sizing): Functional Specifications of Query Processor, Revision 1. (no report number), UNIVAC (St. Paul, Minnesota), (no date).

7. Westgaard, R. E.: DDL Version 1.0. External Reference Specification TO39:1.0 - E013*3.4.1, Control Data Corporation, August 4, 1971.

8. Semegran, S. D.: Query-Update Version 2. External Reference Specification TO38:2.0 - E013*3.4.1, Control Data Corporation, August 4, 1971.

9. Anon.: Query Update Version 2. (no report number), Control Data Corporation, (no date, but after Reference 8).

10. Westgaard, R. E.: ASP, Associative Set Processor Preliminary Design. (no report number), Control Data Corporation, August 11, 1972.

11. Anon.: Record Manager Reference Manual, Models 72, 73, 74 Version 1, 7600 Version 1, 6000 Version 1. SCOPE Reference Manual 60307300, Control Data Corporation, Revision B, July 31, 1972.

12. Anon.: Query Update Reference Manual, 6000 Version 1, Models 72, 73, 74 Version 1. SCOPE 3.4 Reference Manual 60307500, Control Data Corporation, Revision C, March 3, 1972.

13. Anon.: QUIDDL Reference Manual, Models 72, 73, 74 Version 1, 6000 Version 1. SCOPE 3.4 Reference Manual 60327200, Control Data Corporation, Revision C, February 25, 1972.

14. Willner, S.; Saukaitas, B. J.: The COMRADE Query Processor. COMRADE Design Note DM06, Naval Ship Research and Development Center (NSRDC) April 1972.

15. Haas, M. E.: VIM-16 Proceedings. Sixteenth Semi-Annual VIM Conference. VIM, INC., April 1972, pp. 754-788.

16. Stacey, G. M.: A Proposal for a Standard DDL Sub-schema Framework for All Host Languages. Edinburgh Regional Computing Centre document, University of Edinburgh, Scotland, August 1972.

17. Stacey, G. M.: A Data Description Language for a FORTRAN Sub-schema: Specification for Discussion. Edinburgh Regional Computing Centre Notes No. 4, University of Edinburgh, Scotland, June 1972.

18. Anon.: Graphic Subroutine Package (GSP) for FORTRAN IV, COBOL, and PL/1. IBM System/360 Operating System Reference Library Manual 360S-LM-537, Nov. 1968.

19. Anon.: UNIGRASP, UNivac Interactive GRAphics Support Package, User's Guide. UNIVAC Division of Sperry Rand, Publication No UC-1000, 1971.

20. Anon.: MARINER 1557/1558 Advanced Graphic Display System, Programmers Reference. UNIVAC Division of Sperry Rand, Publication UME-7657, (no date).

21. Anon.: Terminal Control System User's Manual. Information Display Products, PLOT-10 Document 062-1438-00, May 1971.

22. Anon.: Advanced Graphing User's Manual. Information Display Products, PLOT-10 Document 062-1439-00, Nov. 1971.

23. Batdorf, W. J.; and Kapur, S. S.: FLING-A Fortran Language for Interactive Graphics. Paper presented at Navy Conference on Structural Mechanics (University of Illinois), Sept. 1971.

24. Haas, M.: Consolidated Graphics Processing. NSRDC Internal Memo 1833: MH:am, Aug. 1972.

PHASE I, TASK 2

PART III - GENERAL PURPOSE UTILITIES

1 IPAD SYSTEM OVERVIEW, SYSTEM INTERFACE AND OPERATING PHILOSOPHY: AN INTRODUCTION TO GENERAL PURPOSE UTILITIES (GPUs)

The General Purpose Utilities (GPUs) are a collection of programs that:

1. Are created (or are designed from existing code) specifically for IPAD.
2. Provide general capabilities to support the interactive IPAD user and augment specific capabilities provided by his Operational Modules (OMs).

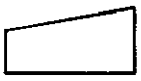
The IPAD system design presented in Part II could be summarized as a software framework intended to reduce the time and labor expended by a user in accomplishing a given task. The prime objective is to relieve him of burdens through improved interface with the supporting software and provide the functional capability to which he is accustomed (viz, his own OMs).

The GPUs, on the other hand, will present the typical user with a whole new dimension of functional capability. There is no major breakthrough in technology here. The capabilities have been available for some time. However, they have not been available in an integrated system. Attainment of even a task-related portion of these capabilities by a typical IPAD user in todays computing environment would require considerable computer-oriented work incidental to his assignments and often not be transferable to other - even similar - tasks.

Specific utilities will be discussed in later sections. This section includes:

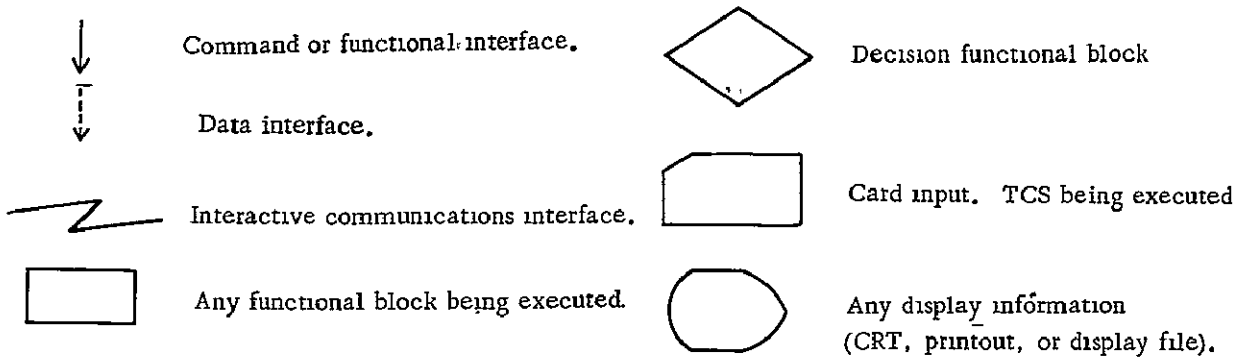
1. An overview of the integrated system (IPAD).
2. The relationship of GPUs to the system and how they are incorporated.
3. Methods of linking OMs and GPUs into a task-oriented sequence of OMs.
4. Programming standards applicable to the GPUs.

The Figures of this volume generally adhere to the following symbolism for the convenience of the reader:



Any interactive console. Any IPAD user at an interactive console

Any file on disk.



The reader is referred to Appendix A for a concise glossary of acronyms and special terminology used throughout this report.

1.1 System Overview

The architecture (or design) of IPAD is primarily a comprehensive plan to exploit computer manufacturer-supplied software to achieve IPAD objectives. Where human interface with the supporting software is so cumbersome as to interfere with IPAD objectives, an IPAD utility is provided to buffer and reconcile tedious requirements with the problem-oriented user.

From the user's point of view, IPAD is a framework which supports and augments the capabilities of his computerized analytic tools. From this viewpoint, the framework is composed of a number of interfacing capabilities:

1. The IPAD EXECutive function which provides control of the full capability of the host operating system/timesharing subsystem (especially to invoke OMs/GPUs) interfaced by:
   a. Tutorial aids.
   b. Prefabricated Task Control Sequences (TCSs).
   c. The ability to fabricate (and save) TCSs:
      - By performing a task.
      - By expansion (specifying task dependent parameters) of TCS Skeletons (TCSSs).

2. The Query Processor (QP) which provides interface with a project oriented central data base and the Data Base Management System (DBMS). To the user, the data base and QP provide for accurate efficient communication with respect to task assignments and task status, and efficient access to pertinent design data and design tools (OMs/GPUs and associated tutorial aids).

3. A task integration capability which permits:

   a. The construction of a task oriented User File (UF) appendage to the database (i. e. to provide for the I/O requirements of selected OMs/utilities).

   b. The construction of a DBMS interface to:

      - Share data among the OMs.
      - Resolve conflicting requirements of the OMs/utilities with respect to the UF.

4. Incorporation capability which assists the programming oriented user in preparing existing OMs to operate within the IPAD framework. Preparation consists of:

   a. Converting conventional I/O procedures to DBMS interface procedures.

   b. Providing the basis for a declarative DBMS interface (i. e., defining I/O requirements) to be completed and tailored to each task.

1.2 Relationship of GPUs to the Overall System

An important objective of IPAD is to provide control* of the system to the user at all times, thus encouraging innovations. However, it is envisioned that the user will evolve some orderly approach to each task assigned to him. It is further envisioned that capabilities provided by the GPUs will assist in this organization.

Evolution of a typical user's task within IPAD includes the following system/user interface sequence:

1. Interrogation (via QP) of the Task Status/Action File (TSA) for notification of a task assignment.

---

*CONTROL was ranked second or third by the groups constructed in the IPAD User Survey. See Table 1 of Section 3.4, Part I, Volume IV for details.

2. Planning (possibly with Tutorial Aids Support) a sequence of capabilities as provided by OMs/GPUs to accomplish the task.

3. Constructing (via the SCHEMA assembler SPU) a task entity; i.e. a UF appendage to the data base and a DBMS interface with the OMs/GPUs.

4. Initialization of the UF by mapping (via QP) values from the central, controlled data base - i.e., the Multidisciplinary Data Bank (MDB) - or local data bases such as the Disciplinary Library File (DLF) or User File (UF).

5. Execution (via a TCS) of the sequence of selected OMs/GPUs interfacing the UF.

6. Mapping of task results (via QP) into the Presentation Review File (PRF).

7. Inserting a message (via QP) into the TSA to notify the Engineering Review Board Coordinator (ERBC) of task completion.

As the task evolution indicates, the role of the GPUs is the same as that of the OMs; to provide capabilities within a planned sequence, interfacing a coordinated UF through DBMS. DBMS provides data transformations when the OMs/GPUs access the UF, thus (effectively) providing for OM/GPU to OM/GPU interface.

It should also be noted that every stage of task evolution other than execution of the TCS involves user interface with DBMS or preparation of OM/GPU interface with DBMS.

It becomes evident that DBMS is the central coordinating software and an overview of DBMS is necessary to proceed with GPU incorporation.

1.2.1 Overview of DBMS. - The DBMS is a manufacturer supplied implementation of a system envisioned by the Data Base Task Group (DBTG) of the Conference On DAta SYstem Languages (CODASYL). The final report of the DBTG (Reference 1) does not contain explicit software specifications but rather language specifications and descriptions of capabilities to be provided in support of these languages. The languages specified have the syntax of COBOL, but the architecture explicity provides for interfacing through many host languages. The Foreword to their report (Reference 1):

1. Recommends implementations in support of the DBTG specified languages.

2. Indicates that language development will continue under the auspices of CODASYL.

3. Invites the scientific community to participate in this language development.

Manufacturers of two IPAD target computing systems (CDC and UNIVAC) as well as four other computing system manufacturers (XEROX, HONEYWELL, PHILLIPS, and BURROUGHS) are tentatively committed to such an implementation. The IPAD tasks of providing for I/O Formatting and data base software were consequently realigned to provide plans for exploiting the proposed DBMS and to analyze the need for language development.

The DBTG report introduces a new vocabulary for discussions of data base software as well as the computer system interface languages. Throughout this report, the special terminology of CODASYL's DBTG appear in caps (e.g. RECORDs or AREA).

1.2.1.1 Major concepts and terminology: The following terms are defined in the DBTG report (Reference 1, page 13, 14) to lend precision and consistency to their language specifications. The definitions are liberally paraphrased here to present an overview.

SCHEMA	- a source language (human readable) description of the data base (i.e. all data to be handled via DBMS).
SUBSCHEMA	- a source language description of the data in the data base to be accessed by an OM/utility (through DBMS) in form that that OM/utility expects to use it. To be useable, it must represent a valid subset of the data base.
AREA	- a named subdivision of the SCHEMA. For orientation purposes an AREA may be considered similar to a file, but no physical implications apply.
RECORD	- a named type-definition of a data structure within an AREA. Structure consists of the names and attributes of DATA-ITEMs and DATA-AGGREGATEs. This term has no relationship to the characteristics of I/O devices. RECORD TYPE is used to refer to the definition of the named structure; RECORD OCCURRENCE is used to refer to the contents of an occurrence of the defined structure.
DATA ITEM	- the smallest unit of named data. DATA-ITEM is the name, DATA-ITEM OCCURRENCE is the value.
DATA-AGGREGATE	- a named collection of DATA-ITEMs, vector or repeating group.
SET	- a named collection of RECORD TYPEs. Establishes the relationship among the RECORD-TYPEs and other characteristics of the collection.

Data Description Languages (DDLs) are the source languages of the SCHEMA and SUBSCHEMAs. They define the relationship and characteristics of occurrences of DATA-ITEMs, DATA-AGGREGATEs, RECORDs, SETs, and AREAs.

SCHEMA DDL is conceptually independent of any particular programming language since it is a description of logical relationships and management requirements related to actual occurrences of data.

SUBSCHEMA DDL is host language dependent since it describes the data configurations required by OMs/utilities written in the specific host language.

Data Manipulation Language (DML) is the language by which the programmer causes data to be transferred between a program and the data base. This includes commands at the "file control" level and specifications of record selection criteria as well as commands at the FORTRAN READ/WRITE level. Note that all data transfers are performed through DBMS response to DML commands.

Data Base Administrator is the human (group function) whose primary responsibility is to mediate the conflicting data requirements of programs sharing the central data base and effect a compromise. This does not imply that the individual programmers concur (see variations between SCHEMA and SUBSCHEMA below). The compromise is made to optimize overall system operation. It is a DBMS software function to reconcile variations between the SCHEMA and an individual SUBSCHEMA during execution of DML commands.

1.2.1.2 Functional overview: The functions performed by DBMS fall into three categories:

1. Control of the I/O functions of the operating system to satisfy DML requests issued by programs in execution.
2. Transformations to reconcile differences in SCHEMA/SUBSCHEMA descriptions of the data. This is basically restructuring, mapping, and conversion (e.g. integer to floating point) of data but also includes invoking object code (provided in the data base) to transform data.
3. Enforcement and maintenance of management concepts detailed by the DBA with respect to:
   a. Integrity of the data.
   b. Logical structure of the data.

1.2.1.3 Variations between SCHEMA and SUBSCHEMA: The following is copied verbatim from Section 2.2.2, page 18 of Reference 1.

A SUBSCHEMA may differ from a SCHEMA in several important respects:

1. At the DATA-ITEM level:
   a. The characteristics of DATA-ITEMs may be different.
   b. PRIVACY LOCKs may be changed.
   c. Descriptions of specific DATA-ITEMs may be omitted.
   d. The ordering of DATA-ITEMs may be changed
2. At the DATA-AGGREGATE level:
   a. Descriptions of specific DATA-AGGREGATEs may be omitted.
   b. PRIVACY LOCKs may be changed.
   c. The ordering of DATA-AGGREGATEs may be changed.
   d. Vectors may be redefined as multi-dimensional arrays.
   e. DATA-ITEMs or DATA-AGGREGATEs may be selected and given a group name.
   f. Additional structure mapping may be provided by the facilities of a particular SUBSCHEMA DDL.
3. At the RECORD level:
   a. Descriptions of specific RECORD TYPEs may be omitted.
   b. PRIVACY LOCKs may be changed.
   c. RECORD OCCURRENCEs included in specific AREAs may be omitted, while other occurrences of that RECORD TYPE are included.
4. At the SET level:
   a. Descriptions of specific SET TYPEs may be omitted.
   b. PRIVACY LOCKs may be changed.
   c. Different SET selection criteria may be specified.
5. At the AREA level:
   a. Descriptions of specific AREAs may be omitted.
   b. PRIVACY LOCKs may be changed.

A SUBSCHEMA must, however, be a consistent and logical subset of the SCHEMA from which it is drawn [in the sense that they describe the same data].

The following additional points are important to an understanding of the concept of the SCHEMA and SUBSCHEMA:

1. An object version of the source code SCHEMA may be "compiled" independently of any user program or any SUBSCHEMA.
2. Object versions of a source code SUBSCHEMA may be "compiled" independently of any user program and stored in a library.
3. An arbitrary number of SUBSCHEMA may be declared on the basis of any given SCHEMA.
4. The declaration of a SUBSCHEMA has no effect on the declaration of any other SUBSCHEMA and SUBSCHEMAs may overlap one another.
5. Each SUBSCHEMA must be named.
6. A user program invokes a SUBSCHEMA.
7. The same SUBSCHEMA may be invoked by an arbitrary number of programs.
8. Only the AREAs, RECORDs, DATA-ITEMs, and SETs included in the SUBSCHEMA invoked by a program may be referenced by that program.
9. Since SUBSCHEMAs are host-language-oriented, a program must invoke a SUBSCHEMA that is consistent with its source language. [End quote]

1.2.2 Incorporation of a GPU into IPAD. - The utilization of a particular GPU by an IPAD user requires preparation of the GPU so it may be integrated into a task-oriented entity. The objective is to decouple the capability provided by the GPU from any requirement for a programmer's skills; i.e., to provide within IPAD facilities all the user will need in order to employ the GPU.

The task of providing the required material for a given OM/GPU process consists of:

1. Expressing the I/O requirements of a process in terms of SUBSCHEMA DDL. This DDL specifies:
   a. AREAs (files) required.
   b. RECORD names within the AREAs.
   c. DATANAMEs within the RECORDs.

d. Structure of RECORDs and attributes of DATA.

e. SETs, i.e. logical relationships among individual RECORDs.

The DDL contains complete correct specification of the above but it is not a complete SUBSCHEMA at this point. A complete SUBSCHEMA specifies relationships of SUBSCHEMA names to SCHEMA names.

2. Expressing I/O imperatives (procedures) in terms of DML. This provides the execution-time interface between DBMS and a process.

3. Providing tutorial data to assist the user in integrating the process into a task.

4. Providing a Task Control Sequence Skeleton (TCSS). Functionally this is the same as a sample control card setup. It provides the basis of a specific Task Control Sequence (TCS) of operating system/EXEC level commands to

   a. Execute the process.

   b. Allocate I/O devices.

   c. Supply parameters and analyze completion codes.

Figure 1-1 illustrates the incorporation procedure as applied to an existing OM. A preprocessor reads existing OM source code replacing conventional I/O statement with equivalent DML, simultaneously generating DDL specifications for the structure of RECORDs involved. A cognizant programmer/user continues the process, inserting DML to apply to files and SETs, and DDL specifications of RECORD, SET, and AREA names. With respect to the GPUs, this preliminary work is included in the original creation of source code. In the figure, the programmer/user produces tutorial data (I/ODEF) accessible to all of the OM's users, and inserts DDL EQUIVALENCEs to equate internal variable names to the design data names meaningful to the user.

The final responsibility of the programmer at incorporation is to make TCSSs available to all users. This is a two stage operation involving two system utilities:

1. The TEXT EDITOR assists in creating, editing, and duplicating a general class of symbolic sequential "files".

2. The TCSS Writer provides for a database update creating a TCSS, edited for acceptability by the TCSS Expander. The TCSS Expander is a utility which assists the IPAD user in creating and modifying TCS strings to control the IPAD EXEC.

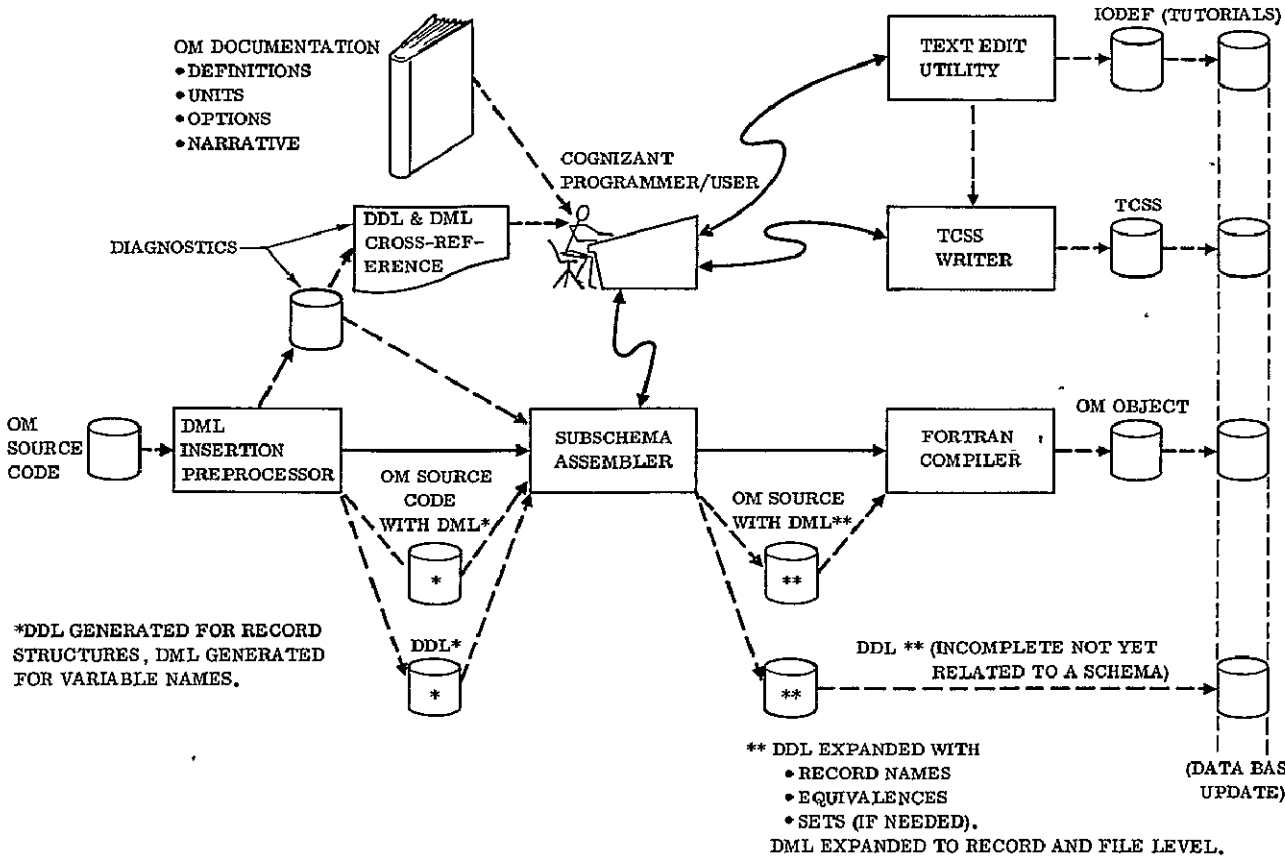


Figure 1-1. Initial OM Incorporation Into IPAD.

1.3 Process Integration

Given a new design task, the user (with the help of utilities) assembles the components of an IPAD task:

1. A Task Control Sequence (TCS) consisting of EXEC commands (and data) to execute a particular sequence of OMs/GPUs and the Query Processor (QP) to initialize, interrogate and update the data involved.
2. A Users Task Trajectory (UTT) to provide a history log of user activity in accomplishing the task.
3. Query Processor Sessions (QPSs, a special form of TCSs) to provide access/update directives to QP. These may be deferred and generated by actually performing data manipulation.
4. A module of the SCHEMA called a User File (UF) to satisfy the I/O requirements of the sequence of OMs/GPUs/QP.

5. A SUBSCHEMA for each OM/GPU in the task to permit it to access a portion of the UF.

6. A SUBSCHEMA for QP to permit initialization of the UF from appropriate sources in the database.

The tedious details of these components have already been supplied by programmers responsible for the individual OMs (GPUs). The user's contribution is to make top level, problem-oriented decisions resulting in a correct assembly:

1. TCS - the user specifies an ordering of the TCSSs supplied with the individual programs and specifies values for dummy arguments relating to the UF.

2. UTT - provided by the DBA when the UF is included in the database. The contents of UTT are automatically provided by the EXEC. The user need not be aware of UTT, cannot modify UTT, but may interrogate his UTT for job status information.

3. QPSs - will typically be saved by QP (at user's request) during a "dry run" of data manipulation.

4. UF module of SCHEMA - begins as a simple assembly of all the I/O requirements expressed in SUBSCHEMA DDL for all OMs/GPUs involved. The DDL facilities provide several options for resolving OM (GPU) to OM (GPU) interface problems:

   a. Redundant DATA OCCURRENCEs may be eliminated by:

      - Specifying, in UF DDL, the source of input which is provided by some OM output and specifying that the input is "virtual" data (viz, data that will exist only as a consequence of calculations from existing data).

      - Renaming within the SUBSCHEMA DDL to define correspondence of DATA-ITEMs within the SCHEMA DDL.

   b. Redundant DATA OCCURRENCEs may be provided by:

      - Specifying in UF DDL the source of inputs as above and specifying that the input is "actual" data (via data which has been stored as a consequence of calculations from existing data).

      - Not specifying the source in DDL but explicitly mapping values onto the DATA-ITEMs via QP.

5. SUBSCHEMA for each OM/GPU is provided - essentially complete - by the programmer. This may be modified to cross-reference data as defined by other OMs/GPUs.

6. SUBSCHEMA for QP to access sources outside the UF (e.g., the MDB). This results from copying source DDL from the UF DDL and the source DDL from the SCHEMA currently in use.

1.3.1. Design optimization, an example of process integration. - An optimization study involves:

1. An initial approximation to a set of design data.
2. A sequence of OMs which produce evaluations of functions of the design data.
3. An optimizer GPU which analyzes the functional values produced by the OMs and modifies the design data accordingly.
4. Looping through 2 and 3 until a criteria function has been met.

In preparing to accomplish the optimization task, it is the user's responsibility to determine (and express to IPAD software) the total relationship of the OMs and the optimizer GPU.

1. What OMs will accomplish the task?
2. What is the sequential relationship of the OMs?
3. What are the logical (set) intersections of the I/O data requirements ?
4. What are the sources of the initial approximations?

Figure 1-2 illustrates this procedure:

1. Select and organize the group of OMs/GPUs themselves. This step scans the TCSSs set up by the cognizant programmer for each OM/GPU (including the optimizer) and uses the TCSS Expander to derive a Master TCS defining the optimization loop.
2. Interrogate the I/ODEF for each OM/GPU and direct the SCHEMA Assembler Utility (an SPU) to produce SCHEMA DDL to satisfy the totality of I/O requirements. In this step the user may cross reference the data such that:
   a. Redundant copies of the same logical data will be eliminated.
   b. DBMS will be directed to provide any required transformations at execution time.

   Or he may establish an expedient provision that the I/O data requirements do not intersect. This approach must be reflected in the TCS in that the

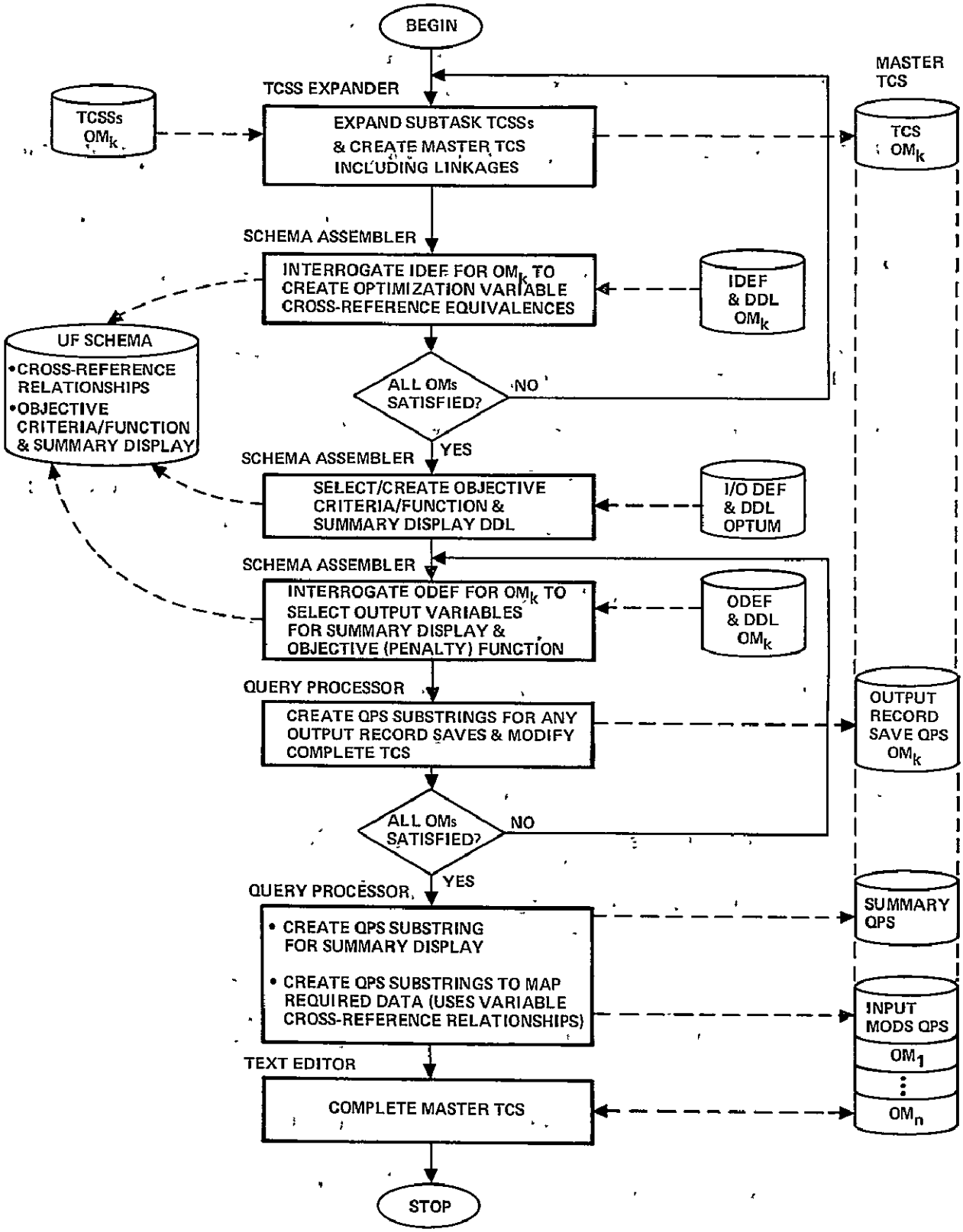


Figure 1-2. Interactively Preparing a Sequence of OMs for Optimization (User Not Shown)

Query Processor (QP) must be invoked between each pair of OMs/GPUs in the loop so that data is mapped from AREA to AREA within the UF.

3. Satisfy the I/O requirements of the optimizer GPU.
   a. Select/create the objective criteria/function and summary display DDL.
   b. Scan I/O DEFs for all other OMs/GPUs and select the outputs which contribute to the evaluation.
4. If the user does not choose to cross-reference data in the SUBSCHEMA DDLs, he imposes a requirement for QP to explicitly map data from AREA to AREA within the UF. In this case he must supply QP directives (a QPS) to accomplish the mapping. There are two ways to supply these directives:
   a. With the TEXT EDITOR utility and familiarity with QP he can set up complete sessions (QPSs) outside the execution loop to be referenced by the TCS and executed by QP.
   b. He can interactively issue directives to QP in an initial execution of the TCS and direct QP to save the sessions (QPSs) for reexecution.
5. At the end of the integration process, the following data has been derived:
   a. A master TCS through which the IPAD EXEC will call the sequence of OMs/GPUs.
   b. Any required QP directives, which are available as QPSs and identified for execution in the master TCS.
   c. A source description of the UF in SCHEMA DDL.
   d. For each OM/GPU, a source description of that portion of the UF required by the OM/GPU in SUBSCHEMA DDL.

Source language for the UF module of the SCHEMA and source language for each SUBSCHEMA are compiled into object tables usable by DBMS. Depending on the user's foresight and procedural preference, the UF will be some compromise between the following extremes:

1. Each AREA specified in the collection of OM/GPU SUBSCHEMAs shall be separately provided to the UF SCHEMA exactly as described. This requires that the user intercede between each pair of OMs/GPUs and map data from the output AREA(s) into the input AREA(s) of the subsequent OM/GPU.

2. The UF SCHEMA shall consist of one logical pool which is the set union of all I/O requirements at the DATA NAME level; mapping and transformations to be provided by DBMS during process execution. This makes it unnecessary for the user to access the data during the sequence of OMs/GPUs.

Whatever the compromise devised by the user, his responsibility includes:

1. Providing a description, in the UF SCHEMA, of data that logically corresponds (i.e., is mappable) to data required by each OM/GPU of the sequence.

2. Specifying, in each SUBSCHEMA, what the correspondence (mapping technique) is.

3. Providing initial ("pump priming") occurrences of input data prior to execution of the sequence of OMs/GPUs.

The OMs/GPUs are then interfaced through DML with DBMS which arranges the mapping and any required data transformations.

Note that the originator of an OM/GPU is not faced with requirements to search for data or recognize special file names or data names, nor to provide interface with another OM/GPU.

Figure 1-3 illustrates the execution of the OMs/GPUs in the optimization loop after UF initialization.

The user, through Query Processor (QP) populates (provides initial "pump priming" data values to) that portion of the database corresponding to his UF module by:

1. Mapping from the Multidisciplinary Data Bank (MDB).

2. Mapping form his Disciplinary Library File (DLF).

3. Mapping from other UFs as required.

4. Inserting parameters via keyboard input.

Depending on the Master TCS he assembles for the task:

1. Some sequence of OMs/GPUs will interact with the data.

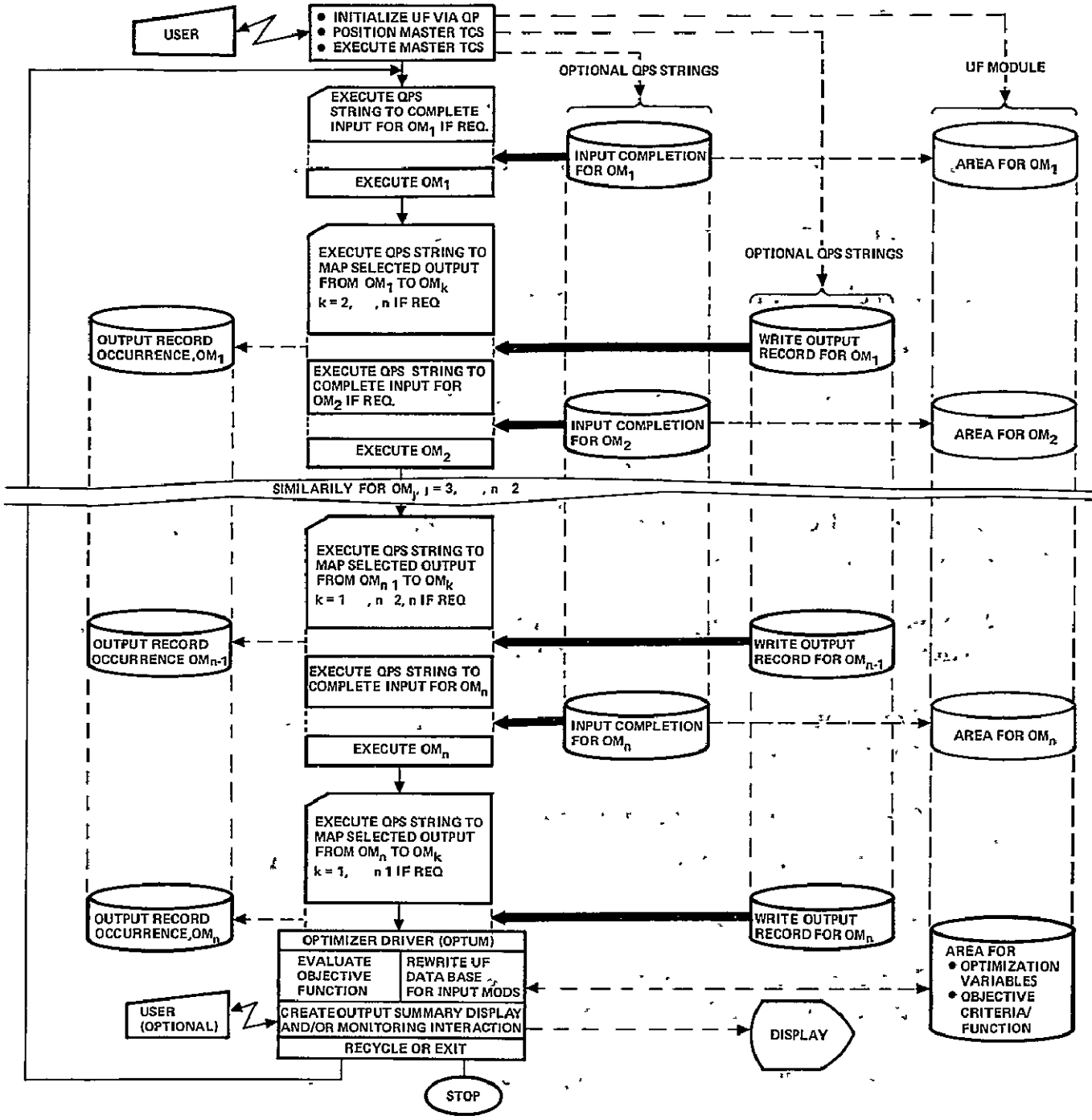


Figure 1-3. Cycling Through an Optimization Loop with Optional Interactive Monitoring

2. The user, via QP may interact with the data between the execution of any pair of OMs/GPUs. The interaction is shown as a QPS in the batch mode. This is recommended as being more practical, however, a QPS may be replaced at any time by a "live" session.

1.4 GPU Programming Standards

GPUs are designed and coded to conform to standards intended to:

1. Insure compatibility with IPAD objectives with respect to the user.

2. Exploit existing software including:

   a. Software developed concurrently with IPAD.

   b. Incremental improvements after implementation.

3. Insure portability of source code.

There is no direct correspondence between these objectives and the programming standards. Each subsection below contributes to one or more of the objectives above.

1.4.1 Standard source language. - This is to be essentially a subset (with respect to any particular computing system) of FORTRAN IV:

1. Only those features supported by all, or nearly all, the major manufacturers will be used.

2. Code enhancements are required in the form of:

   a. DML to interface with an implementation of CODASYL's DBMS.

   b. General Graphics Library (GGL) as applicable.

3. Machine dependent considerations (e.g. dimensions of arrays for text information) will be isolated and identified for ease in modification.

4. Machine/operating system dependent functions (e.g. character manipulation, multiprecision arithmetic) will be delegated to interface subprograms for ease in modification.

1.4.2 Continuity. - It is to be expected that a utility program will be interrupted (through system failures, abort conditions within the utility, or user termination)

before a task is finished. The utilities must be designed to minimize the loss of work (from the user's viewpoint) in case of interruption.

There are two classes of sudden interruptions: total failures and annoyances. The total failures are covered elsewhere in this report (Section 6 of Part II). The design of each GPU is to account for the annoyance class of interruptions. The following design features will minimize annoyance:

1. Each utility will be subdivided into a large number of subtasks and control over the sequencing of these subtasks will be provided to the user. This permits the user to resume at a point very close to the point of interruption.

2. All "files" will actually be permanent AREAs within the UF; structured to permit individual RECORD identification. The utility can resume work on the same RECORD(s) as when interrupted. DBMS will ensure that previous DML "writes" were made permanent.

1.4.3 Modularity. - The GPUs will each be divided into a number of code modules for several reasons:

1. Close user control of an interactive task requires that it be broken down into a large number of subtasks (modules).

2. Continuity of a task over one or more interruptions requires modularity (as discussed above).

3. Proper segmentation of source code permits each IPAD installation to exploit machine/operating system features, e.g.:

   a. Multi-level (tree structure) overlay techniques.

   b. Dynamic overlay techniques.

   c. Memory paging or virtual memory techniques.

Modularity must also include:

1. Separation of executable code from data.

2. Separation of logically distinct tasks.

3. Separation of read-only code from modifiable (e.g. data) code.

1.4.4 Execution mode. - All GPUs (except for the General Design Module, GDM) are to run either interactively or in the batch mode as suits the user. If run batch, the user must supply interactive input as data intermixed with the job control language (the TCS). The GDM is to run interactively, mainly supported by a minicomputer.

1.5 Conclusions

The IPAD design provides a framework for executing and interfacing sequences of OMs, i.e. supporting the user with the capabilities to which he is accustomed. The GPUs augment these capabilities within the same framework, thus contributing significantly to the user's effectiveness.

The capabilities provided by the GPUs have been under development (and usable to some extent) for a considerable period of time. However, as a practical matter, their use has not been generalized due to the difficulty of adapting their specialized implementations and of providing data interface requirements. IPAD alleviates the difficulty by incorporating truly general purpose utilities (GPUs) in such a manner that the same interface resolution assistance applies to all OMs (and every GPU).

An example of OMs and GPUs in a single task sequence is given by a design optimization loop. A sequence of OMs evaluate functions of design data, a GPU (OPTUM) applies an optimization technique to the evaluations and modifies the design data accordingly. The sequence is repeated until an optimization criteria is met. IPAD will thereby provide capabilities permitting a relatively unsophisticated user to configure and exercise this type of sophisticated software capability.

The GPUs are to be IPAD-specific developments. As such they will be developed subject to additional programming standards not necessarily applicable to the OMs. The programming standards are derived from IPAD objectives with respect to the user (e.g., the user will be in close control of the process, will be given opportunities to correct errors and proceed, etc.) and with respect to supporting software (i.e., maximum exploitation of manufacturer-supplied software). The standards anticipate developments in supporting software and interfacing languages; in particular, a DBMS to support CODASYL specified interfacing languages, and to interface with graphics capabilities to support a GGL.

2 STATISTICAL UTILITY MODULE (STATUM), A GPU

The purpose of the Statistical Utility Module (STATUM) is to provide the engineer with a statistical package that can be used at an interactive terminal. The statistics contained in STATUM cover most of the typical needs of an engineer such as standard deviations, means, correlation coefficients, regression coefficients, etc. Menus of the statistical subject matter are made available to him with accompanying tutorial information to help him find the statistic of his choice. Tutorial suggestions are offered on how a particular statistical program might be used.

The philosophy behind STATUM is that the engineer shall retain complete control over his problem each step of the way. He can check to make certain intermediate results look reasonable. After he has selected his statistical program, and given his input to the program, he is next presented with the outputs available from the computations. If anything looks incorrect, he can go back and check or redo his work. If the results look good, he can continue, using additional subprograms of STATUM and developing more complex statistics. This visibility is particularly important in regression analyses where raw observation data is manipulated into a variety of statistics. If the engineer discovers that the standard deviations of a set of observations are unreasonably high, he can go back and check his input data before proceeding further instead of forging ahead and creating results which might be erroneous and totally misleading.

The statistical options included in STATUM are data screening, computations of tolerance intervals, regression and correlation analyses, setup and measurement of testing hypotheses, comparisons of histograms with best fit classical distributions, analysis of variance, and non-parametric statistics. A subject menu tells the analyst what output quantities are available in each of these options. Using this information as a guide, he can browse through the available menus to find a specific statistic if he is uncertain as to what he actually wants. If the analyst is not familiar with the statistical quantities appearing in the subject menu, the Macro and Micro menus can tell him more completely what outputs are available from a particular choice.

The statistical utility module is designed with the user in mind. He doesn't need any special knowledge to find his way through STATUM. The tutorials help him through every step.

2.1 STATUM User Interface

Once the analyst has called STATUM, he will find himself unknowingly in contact with the STATUM USER INTERFACE. This INTERFACE subprogram acts as the host and guide to the user during the time STATUM is active. The STATUM USER INTERFACE is the driver for the activities that take place within STATUM. Thus, the STATUM USER INTERFACE plays the key role in acquiring information from the user, responding to his selections and presenting him with the output results.

The flow diagram showing the functions that the STATUS USER INTERFACE performs is given in Figure 2-1 which should be referred to in the following discussion. The first thing the STATUM USER INTERFACE determines is whether or not this is a new task. In some cases the analyst may have broken off his statistical work, and after a period of time wants to continue. If the task is new, he is asked if he wishes to delete occurrences of data in STATUM's AREA of his User File (UF).

The SUBJECT MENU tells the analyst what statistical options are available. A selection gets him into the correct family of statistics. Adequate information accompanying the SUBJECT MENU helps the analyst make his choice. If he doesn't see what he wants, he selects the T - (Terminate) DISPOSITION item on the menu. This gives him the option to go back to the beginning and try again or else to initiate the termination of STATUM. After the user has selected a specific statistical subject from the SUBJECT MENU, he is shown the MACRO MENU which lists the main programs available under the selected statistical option. If none of the listed topics appeals to him, he selects the T-DISPOSITION choice to terminate his search.

Once the main program has been selected, the STATUM USER INTERFACE retains it and the user is shown the MICRO MENU which contains the subprograms available to him under the main program. These subprograms perform the actual statistical computations under the supervision of the main program selected from the MACRO MENU.

When the analyst has selected his main program and one of the subprograms on the MICRO MENU, he is then placed in contact with the PREFACE to the selected subprogram. He is asked for input data, such as bounds, values of parameters, number of variables, location of the observation matrix and related topics. This extracted information is saved until the analyst instructs the subprogram to execute.

After execution, the analyst is shown the output quantities available on a OUTPUT QUANTITY MENU. He is asked what he wants done with any selected output quantity. An OUTPUT DISPOSITION MENU is presented so he can choose to display (to list or graph), punch cards or similarly disposition any output quantity. Each output quantity

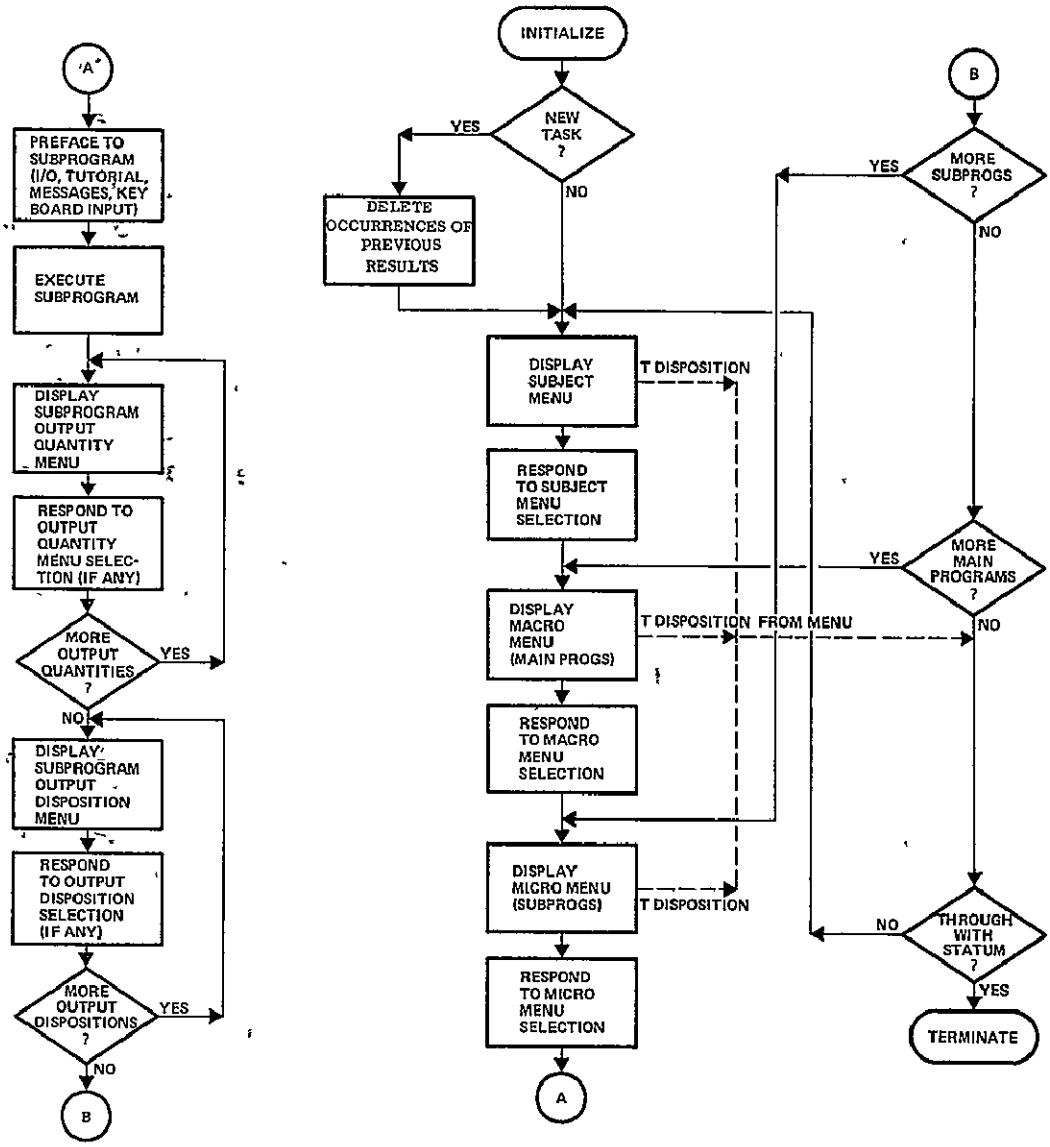


Figure 2-1. Function Flow Diagram, STATUM

that he selects is treated separately. For example, if there are five output quantities, he must ask separately for each of them. He must decide what disposition is to be made separately. In other words, he can only make one selection at a time from a menu. The logic of the STATUM USER INTERFACE permits the analyst to return to a desired menu for further selections before proceeding to a new menu.

After he finishes computing results with a subprogram, the analyst is asked if he wants to return to the MICRO MENU to choose another subprogram. If he does, and selects a new subprogram, he must answer the input interrogation from the new subprogram PREFACE as before. A new OUTPUT QUANTITY MENU, corresponding to the new subprogram, is shown. The OUTPUT DISPOSITION MENU he sees is the same as before; this menu is the same for all subprograms.

When he has finished with all the subprograms under a previously selected main program, his final results are contained in a UF (User File) area of the Data Base.

Since the analyst may not be through with STATUM when he finishes one set of statistical computations, he is given the opportunity of returning to the SUBJECT MENU and starting another sequence of activity. He will not be terminated from STATUM until he indicates that he wants to be terminated.

Two examples are given to illustrate the sequence of activities in STATUM. The first example, shown in Figure 2-2, is the simple case where the analyst wants to screen the data contained in an observation matrix and to obtain the means, standard deviations, and range of the observations for each variable.

The second example, shown in Figure 2-3, illustrates the steps taken to perform a multiple linear regression. The analyst is interested in obtaining the correlation coefficients between his observation variables.

2.2 Statistical Programs Within STATUM

The statistical programs appearing in STATUM are mostly derived from available subroutines described in Reference 2*. These subroutines are programmed in Fortran IV. Nearly all of the subroutines will have to be modified to some extent to be compatible with STATUM. Some will be changed significantly and others will be combined. A relatively small number of subroutines will have to be created from inception to fulfill the needs of STATUM.

* These are quite similar to those 123 statistical subroutines in the IMSL package of about 246 general purpose mathematical subroutines coded in FORTRAN IV. This library is available on IPAD's three target computing systems (see References 3, 4 and 5) and is available for the modest price of $840.

- USER CALLS FOR STATUM
- USER SELECTS DATA SCREENING FROM SUBJECT MENU
- USER SELECTS DASCR FROM MACRO MENU
- USER SELECTS TALLY FROM MICRO MENU
- PREFACE TO SUBPROGRAM, TALLY, ASKS USER FOR
  NAME OF OBSERVATION MATRIX

OUTPUT QUANTITY MENU IS PRESENTED

1. VARIABLES LIST OF TOTALS
2. VARIABLES LIST OF AVERAGES
3. VARIABLES LIST OF STANDARD DEVIATIONS
4. VARIABLES LIST OF MEANS
5. VARIABLES LIST OF MINIMA
6. VARIABLES LIST OF MAXIMA
7 VARIABLES LIST OF RANGE

OUTPUT DISPOSITION MENU IS PRESENTED

1. DISPLAY
2. LIST
3 PRODUCE CARDS
4. PRODUCE FILE (FILE NAME)
5 HARD COPY OF DISPLAYS

- USER IS ASKED IF HE IS THROUGH WITH STATUM, IF YES, TERMINATE STATUM

Figure 2-2. Simple Case· Data Screening

- USER CALLS FOR STATUM
- USER SELECTS REGRESSION & CORRELATION ANALYSES FROM SUBJECT MENU
- USER SELECTS REGRE FROM MACRO MENU
- USER PERFORMS IN SEQUENCE (BY SELECTION FROM MICRO MENU)

  CORRE – GETS MEANS, STANDARD DEVIATIONS, CORRELATION COEFFICIENTS

  ORDER – CHOOSES WHICH VARIABLES ARE INDEPENDENT & WHICH IS DEPENDENT FROM A LARGER SET OF VARIABLES

  MULTR – GETS REGRESSION COEFFICIENTS, T-VALUES, & OTHER CONFIDENCE MEASURES
- OUTPUT QUANTITIES MENU FROM MULTIPLE LINEAR REGRESSION IS PRESENTED
- USER SELECTS DESIRED INTERMEDIATE OUTPUT QUANTITIES ONE AT A TIME

1 MEANS
2. STANDARD DEVIATIONS
3 SUM OF CROSS-PRODUCTS OF DEVIATIONS FROM MEANS
4. CORRELATION COEFFICIENTS
5 REGRESSION COEFFICIENTS
6 STANDARD DEVIATIONS OF REGRESSION COEFFICIENTS
7. T-VALUES
8 INTERCEPT OF REGRESSION LINE
9 MULTIPLE CORRELATION COEFFICIENT
10 STANDARD ERROR OF ESTIMATE
11 SUM OF SQUARES ATTRIBUTABLE TO REGRESSION (SSAR)
12 DEGREES OF FREEDOM – SSAR
13. SUM OF SQUARES OF DEVIATIONS FROM REGRESSION (SSDR)
14. DEGREES OF FREEDOM – SSDR
15 MEAN SQUARE OF SSDR
16 F-VALUE

Figure 2-3 Typical Case: Performing a Multiple Linear Regression

There are two levels of programs which are responsible for performing the statistical computations. The first level which is under the control of the STATUM USER INTERFACE is called the MAIN program. The main program is the driver which supervises the computational programs or the subprograms, as they are called. There are at least two subprograms under each main program except in the case of DISCR - Discriminant Analysis - which is just the combined main program and subprogram. The MAIN programs appear on the MACRO MENUS. The subprograms appear on the MICRO MENUS. In actual operation, the MAIN program controls only one subprogram at a time. When the computations are completed for the first subprogram, the MAIN program is ready to accept instructions from the STATUM USER INTERFACE for driving the next subprogram.

A list of the MAIN programs and their accompanying subprograms are given below with a brief description of each.

2.2.1 DASCR-Data screening. - This main program supervises the screening of a set of observations. Under its direction, bounds can be placed on observations within each variable. Histograms can be made from the frequencies of the observations within given intervals. Totals, means, standard deviations, minimum and maximums, as well as range can be determined for each selected variable.

2.2.1.1 TALLY: This subprogram computes totals, means, standard deviations, ranges, minima and maxima.

2.2.1.2 BOUND: This subprogram selects from a set of observations, those observations which are under, between and over two given bounds.

2.2.1.3 SUBST: From certain conditions imposed on the variables, this subprogram creates a subset of observations that satisfy those conditions.

2.2.1.4 ABSNT: This subprogram is used to test for missing data in an observation matrix or to test for zero values in the observation matrix.

2.2.1.5 TAB 1: This subprogram is used to tabulate (for a selected variable in an observation matrix) the frequencies and percent frequencies over class intervals where upper and lower bounds are imposed. After the frequency is obtained, then each frequency is divided by the total number of observations to obtain the frequency in percent.

2.2.1.6 TAB 2: This subprogram is used to perform a two-way classification of the frequency and percent frequency for two selected variables in an observation matrix where upper and lower bounds are imposed. In addition, it computes the totals, means and standard deviations for each class interval for variables 1 and 2.

2.2.1.7 SUBMX: This subprogram copies from a larger matrix of observation data a subset matrix of those observations which have satisfied certain conditions.

2.2.2 TOCI - Tolerance intervals. - This main program supervises the computations for tolerance limits, confidence limits and prediction limits. The limits are the two edges of the interval.

2.2.2.1 TOINT: This subprogram is used to compute tolerance intervals and limits. Tolerance intervals are computed from a sample to show where most of the population can be expected to lie within a given confidence level.

2.2.2.2 COINT: This subprogram computes confidence intervals and limits.

2.2.2.3 PREINT: This subprogram computes prediction intervals. It computes the interval within which the value of the dependent variable from a regression is expected to fall given values of the independent variable.

2.2.3 REGRE - Multiple linear regression. - This main program supervises the development of a multiple linear regression. Under its direction means, standard deviations, simple and multiple correlation coefficients, regression coefficients and T-values are computed.

2.2.3.1 CORRE: This subprogram computes means, standard deviations, sums of cross-products of deviations from means, and correlation coefficients from an observation matrix. Some subprograms are popular and used with a number of main programs. A typical case is CORRE. It is used with main programs REGRE for multiple linear regressions, STEPR (see Subsection 2.2.5.1) for stepwise multiple regressions and MCANO (see Subsection 2.2.6.1) for canonical correlations.

2.2.3.2 ORDER: This subprogram constructs, from a larger matrix of correlation coefficients, a subset matrix containing the independent variables and a vector containing the intercorrelation of the independent variables to the dependent variable. (See also Subsection 2.2.4.2.)

2.2.3.2 MULTR: This subprogram performs a multiple regression analysis for a dependent variable and a set of independent variables. It computes the regression coefficients, the multiple correlation coefficient and various measures of confidence, e.g. standard deviations of regression coefficients, the variance and the standard error of the estimate (see also Subsection 2.2.4.3).

2.2.3.4 MISR: This subprogram computes means, standard deviations, third and fourth moments, correlation coefficients, regression coefficients, and standard

errors of regression coefficients when data is missing.

2.2.4 POLRG - Polynomial regression. - This main program supervises the development of a polynomial regression. Powers of an independent variable are generated to calculate polynomials of successively increasing degrees. If there is no reduction in the residual sum of squares between two successive degrees of polynomials, the search for higher powers stops. Under its supervision means, standard deviations, correlation coefficients, and regression coefficients are computed along with various confidence measures.

2.2.4.1 GDATA: This subprogram generates independent variables up to the highest degree polynomial specified and calculates means, standard deviations, sums of cross-products of deviations from means, and product moment correlation coefficients.

2.2.4.2 ORDER: This subprogram constructs, from a larger matrix of correlation coefficients, a subset matrix containing the independent variables and a vector containing the intercorrelation of the independent variables to the dependent variable (see also Subsection 2.2.3.2).

2.2.4.3 MULTR: This subprogram performs a multiple regression analysis for a dependent variable and a set of independent variables. It computes the regression coefficients, the multiple correlation coefficient and various measures of confidence, e.g. standard deviations of regression coefficients, the variance and the standard error of the estimate (see also Subsection 2.2.3.3).

2.2.5 STEPR - Stepwise multiple regression. - This main program supervises the development of a stepwise multiple regression. Stepwise multiple regression analyzes the relationship between a dependent variable and a set of independent variables, and selects the independent variables in the order of their importance. The criterion of importance is based on the reduction of sums of squared errors and the independent variable most important within this reduction in a given step is entered in the regression. Under its supervision, means, standard deviations, correlation coefficients, and regression coefficients with various confidence measures are computed.

2.2.5.1 CORRE: This subprogram computes means, standard deviations, sums of cross-products of deviations from means, and correlation coefficients from an observation matrix (see also Subsections 2.2.3.1 and 2.2.6.1).

2.2.5.2 MSTR: This subprogram is used to restructure the storage mode of a matrix. For example, the upper triangular elements of a general matrix are used to form a symmetric matrix.

2.2.5.3 STPRG: This subprogram performs a stepwise multiple regression analysis for a dependent variable and a set of independent variables. It computes the regression coefficients and measures of confidence for each step of regression.

2.2.6 MCANO - Canonical correlation. - This main program supervises the development of a canonical correlation. An analysis is performed of the interrelations between two sets of variables measured on the same subjects. The canonical correlation gives the maximum correlation between linear functions of the two sets of variables. Under its supervision, means, standard deviations, correlation coefficients, and canonical correlation coefficients are computed.

2.2.6.1 CORRE: This subprogram computes means, standard deviations, sums of cross-products of deviations from means, and correlation coefficients from an observation matrix (see also Subsections 2.2.3.1 and 2.2.5.1).

2.2.6.2 CANOR: This subprogram performs the canonical correlation analysis between two sets of variables. It computes the canonical correlations and coefficients.

2.2.7 ALLTST - Measurement of testing hypothesis. - This main program supervises the testing of an initial hypothesis. It analyzes a sample for the purpose of testing a null hypothesis about a population.

2.2.7.1 TTEST: This subprogram computes T-statistics on the means of populations under various hypotheses. For example, a null hypothesis could be as follows: The population means of B equals the population mean of A, given that the variance of B is not equal to the variance of A. (Where A and B are input lists of data.)

2.2.7.2 FTEST: This subprogram computes the F-statistics on the null hypothesis that the ratio of variances of two normal populations is 1 at the significance level X, on the basis of a sample size N1 from population 1 and an independent sample of size N2 from population 2.

2.2.7.3 UTEST: This subprogram tests whether two independent groups are from the same population by means of the Mann-Whitney U-test. The scores for both groups are ranked together in ascending order. Tied observations are assigned the average of the tied ranks.

2.2.7.4 QTEST: This subprogram determines the Cochran Q-test statistic from a matrix of dichotomous data. It tests whether or not three or more matched groups of dichotomous data differ significantly.

2.2.7.5 SIGNT: This subprogram performs a non-parametric sign test, given two

sets of matched observations. It tests the null hypothesis that the differences between each pair of matched observations has a median equal to zero.

2.2.8 KOLM - Kolmogorov-Smirnov tests. - This main program supervises the one-sample and two-sample Kolmogorov-Smirnov tests. In the one-sample test it determines from what probability density function the sample is most likely drawn. In the two-sample test, it determines whether the two samples were drawn from the same population.

2.2.8.1 KOLMO: This subprogram tests the difference in absolute value between an empirical distribution and a theoretical distribution using Kolmogorov-Smirnov's limiting distribution. It is used to determine from what probability density function a particular sample is most likely drawn.

2.2.8.2 KOLM2: This subprogram tests the difference in absolute value between two empirical distributions and a theoretical distribution using Kolmogorov-Smirnov's limiting distribution. It is used to determine whether two independent samples were most likely drawn from the same population.

2.2.9 HBFCD - Histograms compared with classical distributions. -This main program supervises the comparison of a histogram of frequencies against various classical distributions. A residual sum of squares provides the criterion for best fit among the various distributions.

2.2.9.1 NDTR: This subprogram tests against the normal distribution function.

2.2.9.2 BDTR: This subprogram tests against the Beta distribution function.

2.2.9.3 CDTR: This subprogram tests against the Chi-square distribution function.

2.2.9.4 NDTRI: This subprogram tests against the inverse of normal distribution function.

2.2.9.5 BNDTR: This subprogram tests against the binomial distribution function.

2.2.9.6 PODTR: This subprogram tests against the Poisson distribution function.

2.2.9.7 CADTR: This subprogram tests against the Cauchy distribution function.

2.2.10 ANOVA - Analysis of variance. - This main program supervises the analysis of variance. This analysis permits the variance to be broken into several portions: a portion caused by experimental error, a portion caused by varying several parameters simultaneously, and a portion caused by varying a single factor.

2.2.10.1 AVDAT: This subprogram places data for analysis of variance in properly distributed positions of core storage.

2.2.10.2 AVCAL: This subprogram performs the calculus for the general k-factor experiment. Deviates for an analysis of variance are computed using the special operators Σ and Δ.

2.2.10.3 MEANQ: This subprogram performs the mean square operation for the general k-factor experiment. It pools the deviates from AVCAL and computes sums of squares, degrees of freedom, and mean squares.

2.2.10.4 TWOAV: This subprogram determines the Friedman two-way analysis of variance statistic from a matrix of groups and cases. It is used to decide whether a number of samples are from the same population.

2.2.11 RANCO - Rank coefficients. - This main program supervises the computation of rank coefficients for two variables (or groups) whose elements are ranked. The rank coefficients are a measure of the correlation between the two variables (groups).

2.2.11.1 KRANK: This subprogram computes the Kendall rank correlation coefficient. It is used as a measure of the correlation between two variables.

2.2.11.2 SRANK: This subprogram computes the Spearman rank correlation coefficient. It is used as a measure of the correlation between two variables.

2.2.11.3 TIE: This subprogram seeks out ties in ranked observations and computes a correction factor.

2.2.11.4 WTEST: This subprogram computes the Kendall coefficient of concordance to test the degree of association among a number of variables.

2.2.11.5 CHISQ: This subprogram calculates the Chi-square and degrees of freedom for a contingency table containing observed frequencies within certain groups and conditions.

2.2.12 DISCR - Discriminant analysis. - This main program supervises the classification of new individuals into one of several groups. This main program contains the subprogram which does the computing. A set of linear functions is calculated from data on many groups. The classification of an individual into a group is performed by evaluating each of the calculated linear functions, then finding the group for which matchup is best.

2.3 Input/Output Requirements

The input and output quantities of STATUM are expressed in three forms — scalars, vectors and matrices. To give broad visibility of the different kinds of scalars, vectors and matrices used in the various subprograms, two cross-reference charts were prepared. Figure 2-4 shows the kinds of input quantities needed to support the STATUM

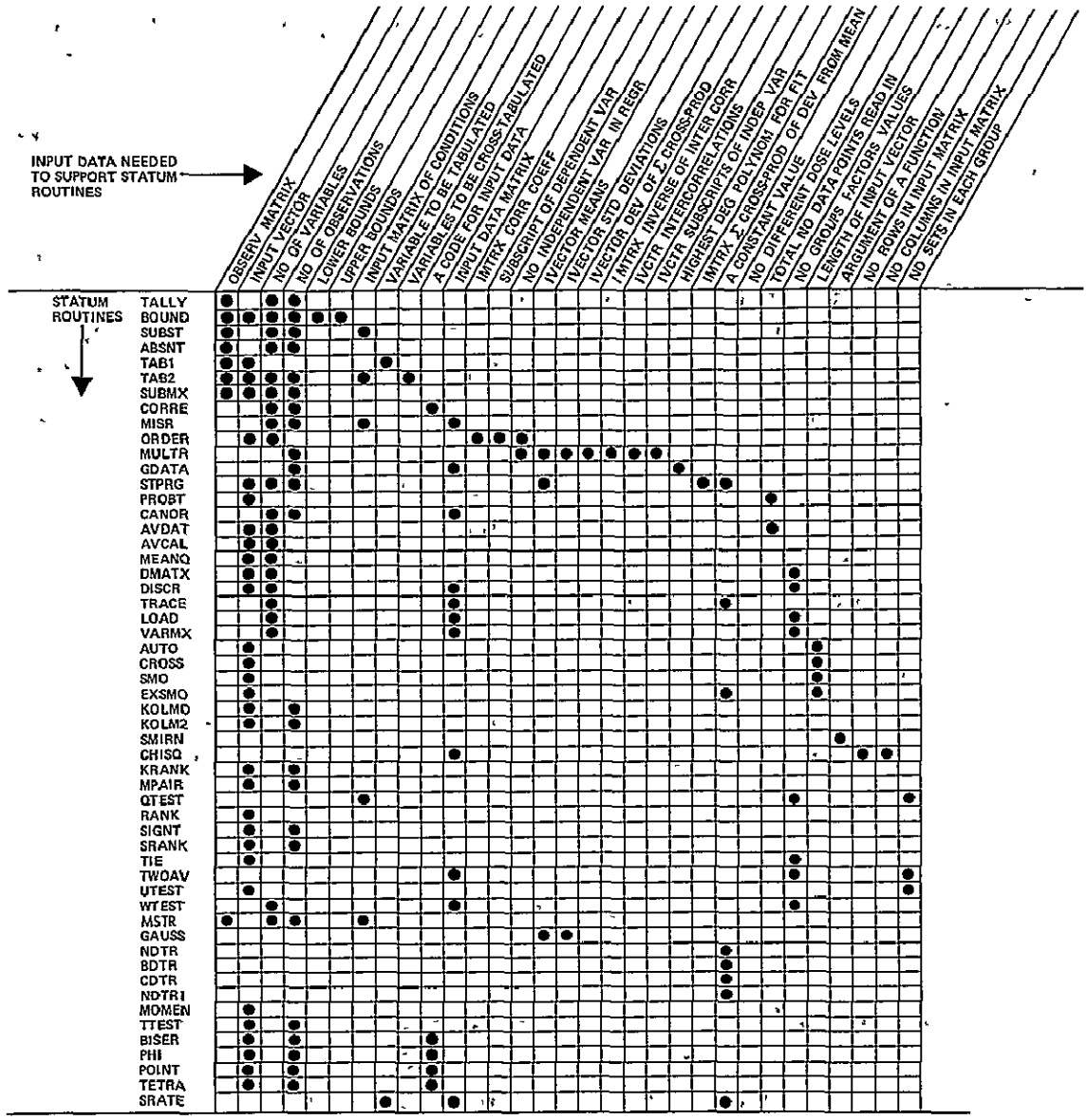


Figure 2-4. Input Information Cross-Reference Matrix

ORIGINAL PAGE IS
OF POOR QUALITY

subprograms. For example, subprogram TALLY requires as input an observation matrix and the dimensions of the matrix in terms of the number of variables involved and the number of observations.

The subroutines listed along the left-hand side of Figure 2-4 are those available from the IBM 360 Scientific Subroutine Package (SSP), Reference 2. STATUM, as presented here, will use many of these statistical subroutines but not all of them. Some subroutines could come from other sources, others may be created expressly for this purpose. For example, the new subroutines TOINT, COINT, and PREINT (which do not appear in Figure 2-4) are used in computing tolerance intervals, confidence intervals and prediction intervals; these subprograms are quite small and can be created by using the formulas given in any standard statistical reference manual (e.g. Reference 6).

The output quantities produced by the STATUM subprograms are presented in Figure 2-5 which is directly comparable to Figure 2-4. As an example, the output quantities generated by subprogram TALLY are an output vector of totals, output vector of means, an output vector of standard deviations, an output vector of the minimum value of each variable, and an output vector of the maximum value of each variable.

The analyst has to make certain his input data is available to STATUM before starting his statistical work. The data can be mapped by QP into a separate AREA just prior to using STATUM or, if the data already resides in an identifiable AREA in storage, he can proceed directly.

The STATUM USER INTERFACE will provide a communication link to the analyst and ask for the name and AREA of the input quantities required for running the main programs and the subprograms. The output quantities will be inserted in storage and will be available for whatever disposition the analyst desires.

2.4 STATUM Menus

The key to successful operation of the statistical programs is good communications between the Statistical Utility Module and the analyst. To help him gain insight to the statistical options available to him, the analyst is given a tutorial on each menu.

The material presented in this section on the STATUM MENUS follows the STATUM functional flow diagram (Figure 2-1). The menus show the options available to the analyst as he progresses through the flow diagram.

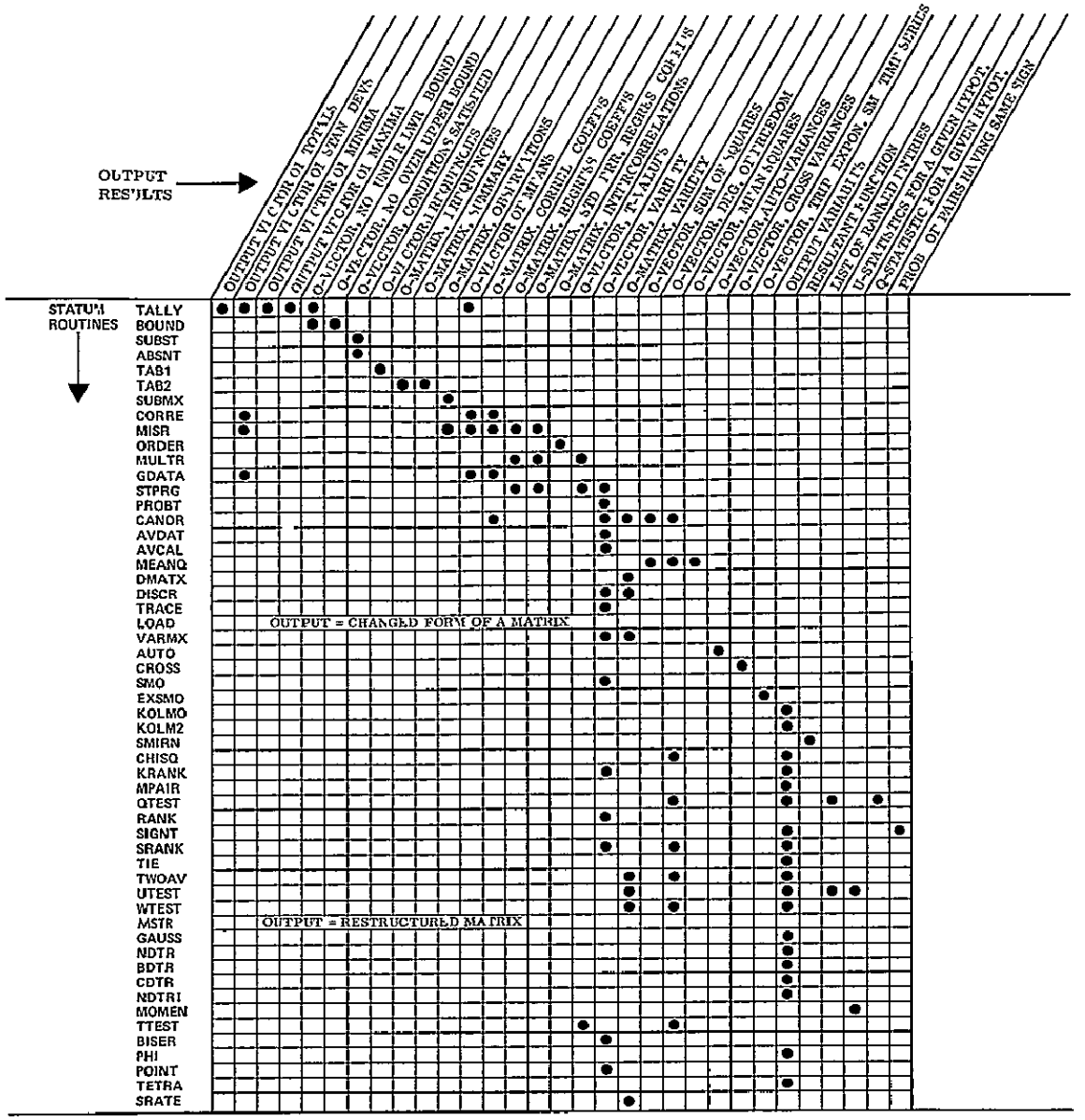


Figure 2-5. Output Information Cross-Reference Matrix

2.4.1 STATUM subject menu. - The STATUM SUBJECT MENU tries to focus the analyst's attention on the major categories of statistical topics available in STATUM. Figure 2-6 shows how the STATUM SUBJECT MENU might appear on the interactive terminal. Each SUBJECT MENU topic is meant to guide an inexperienced user in selecting the statistic that he wants to use. The subject titles are meant to be suggestive. If the user finds the subject titles uninformative, he can browse through the programs under each subject and try to discover what he wants. He can browse, for instance, by calling for the DATA SCREENING macro menu and its supporting MICRO MENUS. Continuing along the same vein, he can call for viewing the TOLERANCE

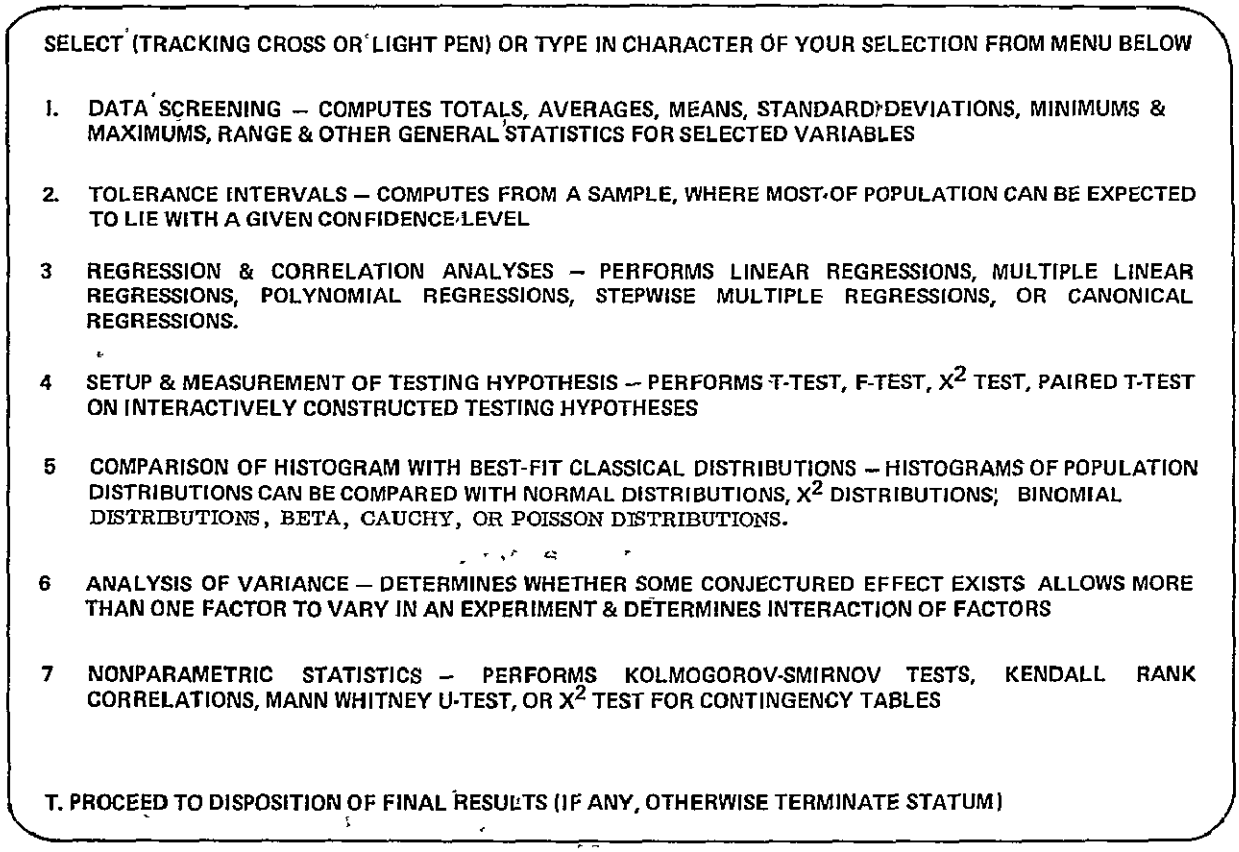


Figure 2-6. STATUM Subject Menu

INTERVALS MACRO MENU and its supporting MICRO MENUS; the REGRESSION AND CORRELATION MACRO MENU and its supporting MICRO MENUS; etc. Note that the selection of an item in the SUBJECT MENU does not initiate any computations, it just guides the analyst to get him started.

2.4.2 STATUM macro menus. – By making a selection on the SUBJECT MENU, the analyst automatically gets the MACRO MENU corresponding to that selection. For each item in the SUBJECT MENU, there is a distinct and separate MACRO MENU. The MACRO MENU contains the name of the main program and a brief tutorial describing the quantities computed under the supervision of the main program. An example of a MACRO MENU is shown in Figure 2-7 as it might appear at the interactive terminal. Tutorial information came from Section 2.2.

There are seven MACRO MENUS. Each contains from one to four selections of main programs. In Figure 2-7, the MACRO MENU for regression and correlation analyses, are shown the four possible activities that can be selected from this menu, namely, Multiple Linear Regression, Polynomial Regression, Stepwise Multiple

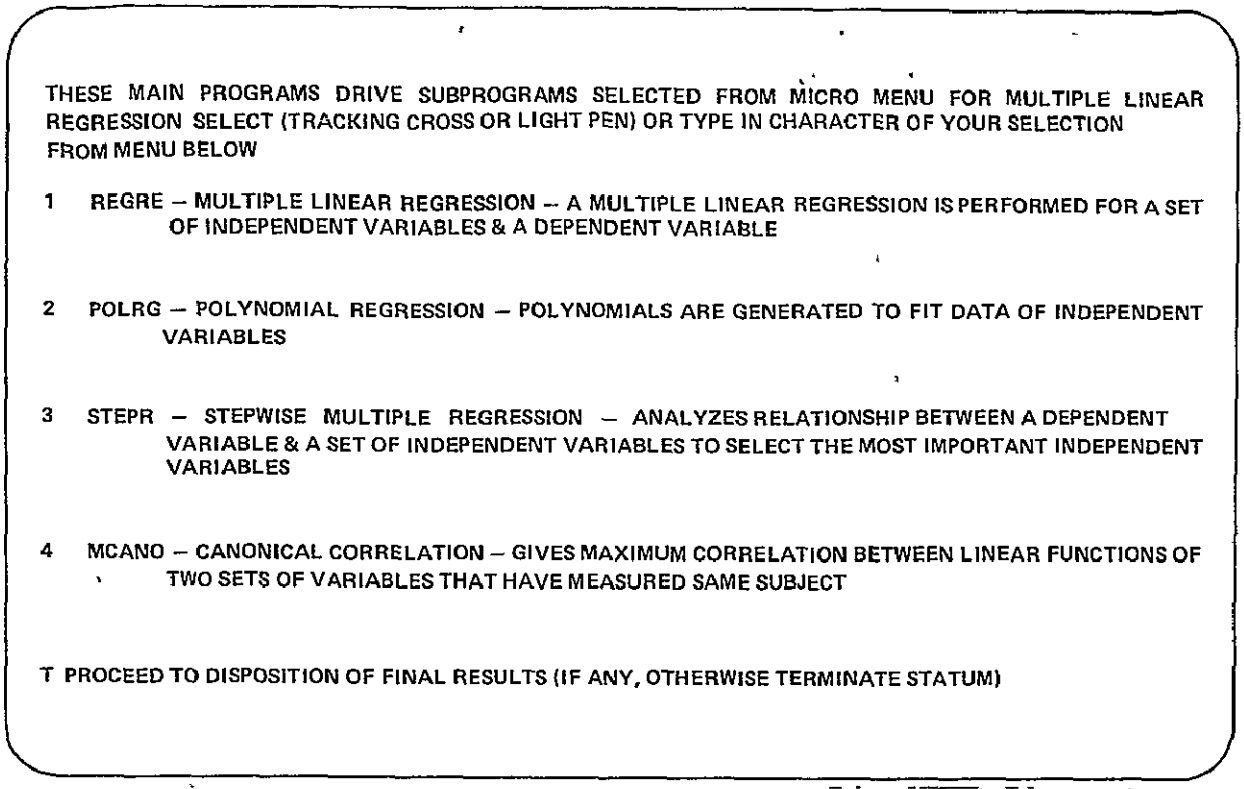


Figure 2-7. STATUM Macro Menu for Regression and Correlation Analyses.

Regression, and Canonical Correlation. The seven MACRO MENU groupings are as follows:

1. DATA SCREENING
   a. DASCR - Main program for Data Screening
2. TOLERANCE INTERVALS
   a. TOCI - Main program for Tolerance Intervals
3. REGRESSION AND CORRELATION ANALYSES (see Figure 2-7)
   a. REGRE - Main program for Multiple Linear Regression
   b. POLRG - Main program for Polynomial Regression
   c. STEPR - Main program for Stepwise Multiple Regression
   d. MCANO - Main program for Canonical Correlation
4. MEASUREMENT OF TESTING HYPOTHESES
   a. ALLTST - Main program for Measurement of Testing Hypotheses
   b. KOLM - Main program for Kolmogorov-Smirnov Tests

5. COMPARISONS WITH BEST-FIT CLASSICAL DISTRIBUTIONS

   a. HBFCD - Main program for Histograms Compared with Classical Distributions

6. ANALYSIS OF VARIANCE

   a. ANOVA - Main program for Analysis of Variance

7. NONPARAMETRIC STATISTICS

   a. RANCO - Main program for Rank Coefficients

   b. DISCR - Main program for Discriminant Analysis

2.4.3 STATUM micro menus. - The MICRO MENUS contain the computational programs which produce the desired statistical results. The MICRO MENU appears automatically to the analyst after a selection has been made from the MACRO MENU. The MICRO MENU contains the names of the subprograms and a brief tutorial describing the quantities being computed. The subprograms contained in the MICRO MENU are generally independent of each other. There are cases, however, where the subprograms must be used sequentially and the output results of one are used as input to the other. Where this sequential dependence may arise, a tutorial statement advises the analyst of the correct sequence. An example of a MICRO MENU with its tutorial statement (see Section 2.2) is shown in Figure 2-8.

MULTIPLE LINEAR REGRESSION IS NORMALLY PERFORMED BY CALLING IN SEQUENCE CORRE, ORDER & MULTR SELECT (TRACKING CROSS OR LIGHT PEN) OR TYPE IN CHARACTER OF YOUR SELECTION FROM MENU BELOW

1. CORRE – TO FIND MEANS, STANDARD DEVIATIONS & CORRELATION MATRIX (IF NOT PREVIOUSLY ACCOMPLISHED)

2 ORDER – TO CHOOSE A DEPENDENT VARIABLE & A SUBSET OF INDEPENDENT VARIABLES FROM A LARGER SET OF VARIABLES

3 MULTR – TO COMPUTE REGRESSION COEFFICIENTS B(O), ..B(M) & VARIOUS CONFIDENCE MEASURES

4 MISR – CONSIDERS THAT OBSERVATION DATA MAY BE MISSING COMPUTES, MEANS, STANDARD DEVIATIONS, THIRD & FOURTH MOMENTS, CORRELATIONS, SIMPLE REGRESSION COEFFICIENTS & THEIR STANDARD ERRORS

T PROCEED TO DISPOSITION OF FINAL RESULTS (IF ANY, OTHERWISE TERMINATE STATUM)

Figure 2-8. STATUM Micro Menu for Multiple Linear Regression (REGRE)

2.5 Output Quantity Menus

After a particular subprogram has been executed, an OUTPUT QUANTITY MENU is presented to the analyst. He can select key quantities for monitoring his intermediate results before proceeding further. Or, for those output quantities he wants to disposition, he makes his selection and states what he wants done with them when they are presented on the OUTPUT DISPOSITION MENU.

A typical example, the OUTPUT QUANTITY MENU is given for subprogram TALLY. The following information is displayed to the analyst.

STATUM
OUTPUT QUANTITY MENU FOR TALLY
Select (tracking cross or light pen) or type in
character of your choice from menu below.

1. Variables list of totals
2. Variables list of means
3. Variables list of standard deviations
4. Variables list of minima
5. Variables list of maxima
6. Variables list of range

T. Proceed to disposition of final results
(if any, otherwise terminate STATUM)

The OUTPUT DISPOSITION MENU is always the same for all of the subprograms. The analyst has the identical options for disposing of his output quantities regardless of the subprogram. The OUTPUT DISPOSITION MENU contains the following options:

1. Display (text or graph)

2. Hard copy displays (lister or microfilm)

3. Produce cards

4. Produce a special file (file name)

The output quantities for the various STATUM subprograms were shown in the output information cross-reference matrix of Figure 2-5. In this matrix the STATUM subroutines appear on the left and the output results across the top. As presently envisioned, STATUM will use many of the subroutines shown in Figure 2-5 but not all of them. The output results shown across the top do not contain all of the output information that is actually available. There are so many quantities available that

they could not all be listed on one sheet of paper. Consequently, there are general output quantities shown, e.g. OUTPUT VECTOR (variety) and OUTPUT MATRIX (variety) to catch the unusual details and to place them in a summary vector or matrix.

Additional output quantities must be provided when the new subprograms TOINT, COINT, PREINT, FTEST, BNDTR, PODTR and CADTR are designed. These output quantities are summarized below:

1. The following statistics are available from TOINT
   a. Tolerance interval and tolerance limits
2. The following statistics are available from COINT
   a. Confidence interval and confidence limits
3. The following statistics are available from PREINT
   a. Prediction interval and prediction limits
4. The following statistics are available from FTEST
   a. List containing degrees of freedom associated with the F-statistic
   b. F-statistic for a given hypothesis
5. The following statistic is available from BNDTR
   a. Measure of the best fit of a histogram to the binomial distribution function
6. The following statistic is available from PODTR
   a. Measure of the best fit of histogram to the Poisson distribution function
7. The following statistic is available from CADTR
   a. Measure of the best fit of a histogram to the Cauchy distribution function.

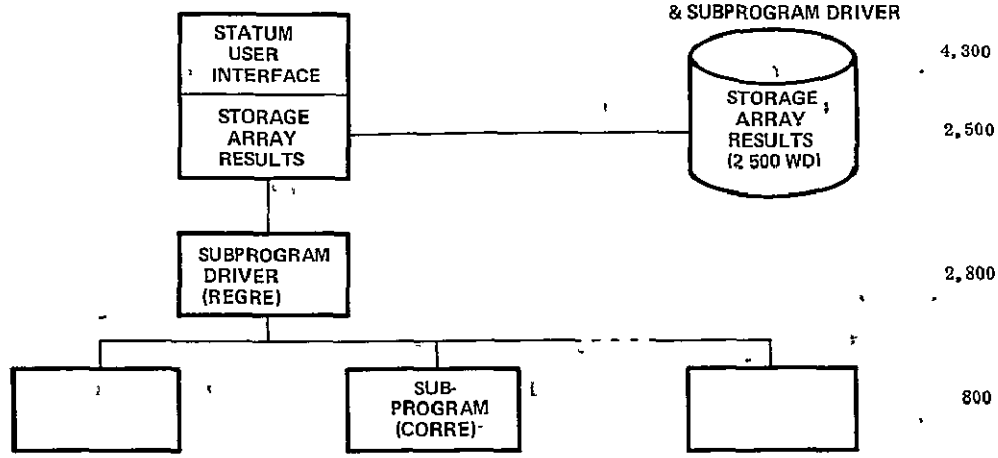
2.6 Operating Requirements for STATUM

Now that the general features of STATUM are known, it is possible to estimate the storage requirements and computational times for various STATUM statistics. These are summarized in Figure 2-9.

2.6.1 Incore storage requirements for STATUM. - The statistical activity requiring the largest array of output results and therefore the largest incore storage array is

Subject	Typical No of Programs	Typical Compute Time* (sec)	Typical Storage of Results (words)	Maximum Program Size** (words)
DATA SCREENING	8	1 4	1,000	3,500
TOLERANCE INTERVALS	4	0 3	300	1,200
REGRESSION ANALYSIS	11	1 4	2,500	5,000
TESTING HYPOTHESES	7	1 7	1,000	3,400
BEST FITS	9	1 7	1,000	3,300
ANALYSIS OF VARIANCE	4	0 4	500	4,000
NONPARAMETRIC STATISTICS	7	1 7	1,000	3,000

*CDC 6400 **INCLUDES SUBPROGRAM PREFACES & SUBPROGRAM DRIVER



IN-CORE BUFFERS & DATA MANAGEMENT	3,000
I/O IN-CORE BUFFERS	2,000
GRAPHICS SUBROUTINES	1,500
SYSTEM SUPPORT ROUTINES	1,000
GROWTH DURING IMPLEMENTATION	2,500
STATUM OVERALL SIZE	20,400 WORDS

Figure 2-9. Estimated Operating Requirements for STATUM

Regression Analysis. It requires approximately 2,500 decimal words* of incore storage (Figure 2-9). These estimates are based on Figures 2-4 and 2-5.

The largest main program that must be handled in core is REGRE (Regression Analysis). It contains approximately 2,800 decimal words. The largest subprogram operating under the direction of REGRE is CORRE (800 words) which computes (among other things) correlation coefficients. The STATUM-USER INTERFACE which directs all the activities of STATUM while communicating with the analyst, is estimated to require 4,300 words of incore storage. This estimate is based upon the number of functions it has to perform and by comparing it with similar programs.

In addition to the programs that control and compute statistical quantities, there are additional host computer operating system support subroutines that must be accounted for to complete the core storage requirements for STATUM. These are

*These requirements are based upon CDC 6000 series 60 bit words. Comparable requirements for 8-bit byte machines can be obtained from Figure 2-9.

interface programs that link STATUM to other functions to handle such things as input/output data, graphical displays, etc. The incore buffers and data management are estimated to require an additional 3,000 words of storage, the input/output incore buffers require 2,000 words, the graphical display subroutines themselves require 1,500 words, and the other support subroutines that might be needed to complement STATUM incore are estimated at 1,000 words.

It is anticipated that there may be a growth of storage requirements during the implementation of STATUM. To account for this growth, an additional 2,500 words (14 percent) of core storage growth potential has been added. The maximum incore size of STATUM is estimated to be 20,400 decimal (47,660 octal) words. This is below the 50,000 octal words that might normally be targeted for such a utility program. It is low enough that it could be used on smaller computer if desired.

2.6.2 Computing times for STATUM. - To get an idea of how long typical problems would take on a CDC 6400 using STATUM, running times were taken from Reference 2 where typical statistical problems were performed on an IBM 360/30. This information was extrapolated to give the typical compute times shown on Figure 2-9. It can be seen that computing times ranging less than 2.0 seconds can be expected on a CDC 6400 computer. This length of compute time is perfectly compatible with activities at an interactive terminal; the analyst has negligible wait time and can perform efficiently.

2.6.3 Modification and creation of new subroutines for STATUM. - Although many of the subroutines are readily available for STATUM, a number of new programs must be created and others need to be modified. The new programs required are as follows:

1. STATUM USER INTERFACE
2. TOCI - Main program for Tolerance Intervals
3. ALLTST - Main program for Measurement of Testing Hypothesis
4. HBFCD - Main program for Histogram Comparisons to Classical Distributions
5. RANCO - Main program for Rank Coefficients
6. TOINT - Subprogram for computing Tolerance Intervals
7. PREINT - Subprogram for computing Prediction Intervals
8. COINT - Subprogram for computing Confidence Intervals
9. FTEST - Subprogram for computing F-statistics

10. BNDTR - Subprogram for Binomial Distribution Comparison

11. PODTR - Subprogram for Poisson Distribution Comparison

12. CADTR - Subprogram for Cauchy Distribution Comparison

The technical requirements for these routines can be obtained from Reference 6.

The programs of Reference 2 that will require significant modifications are the following:

1. KOLM - Main program for Kolmogorov-Smirnov Tests

2. DISCR - Main and subprogram for Discriminant Analysis

3. MULTR - Subprogram for Multiple Regression Analysis

4. CANOR - Subprogram for Canonical Correlations

All of the subprograms must be modified to include a PREFACE of about 125 words. The PREFACE contains tutorial statements and questions to help the analyst satisfy the needs of the subprogram. The PREFACE for each subprogram will be different and only have meaning when used in context with its host MAIN program.

3 IPAD TEXT EDITING, A HOST UTILITY

This portion of the study was conducted to determine the text editing capabilities required by IPAD and to make a document survey of the existing text editing software supplied and maintained by the major computer system manufacturers. The survey was limited to editing capabilities of interactive software subsystems provided under standard operating systems. The text editors studied in detail are those currently supplied for the CDC Cyber 70 and 6000 series, the IBM 360 and 370, and the UNIVAC 1100 series computers (References 7, 8 and 9, respectively).

Two additional large scale systems (Reference 10), the HONEYWELL 6000 (GE 600) and the UNIVAC 70 (RCA Spectra 70), and a medium scale system (Reference 11), the DEC PDP-10, were briefly examined to ascertain the extent of text editing capability provided under their timesharing operating system. This was done primarily to ensure that the three text editors studied in detail were representative of the range of computer systems on which IPAD is likely to be implemented.

3.1 Text Editing Concepts

The two main functions of text editing are (1) the creation of new information files and (2) the editing of existing information files. "Information" is used here to cover a variety of computer stored documents: programs written in any kind of source language; input data files; specifications; and other types of arbitrarily formatted text. In short, any character-coded, sequentially-ordered file can be created or edited by an interactive terminal user through the text editors under discussion.

To service the two functions of creating new files and editing existing files, text editors operate in either an input or edit mode (or both simultaneously). In the input mode, each editor has an initial command (e.g., CREATE, INPUT) to open a new edit file for the user to enter lines of text from his terminal. Depending on the design of the editor, line numbers are either automatically generated as each line is entered, or the user may create a file without line numbers.

In the edit mode, various commands may be used to locate, modify and manipulate existing text. These commands may be performed upon entire files, lines or groups of lines within a file, and characters or character strings within a line. Each of the editors discussed here may perform either line-number editing or context editing. Each editor has an edit file line pointer that refers to the line to be edited. In line-number editing, the pointer moves to the line number referenced

by the command. In context editing, lines may be located by a search for specific characters within a line. When the line is located, the edit file pointer points to the line to be edited. In addition to this means of locating a line in context editing, some editors provide commands for positioning the pointer relative to its present line position or to the first or last line of the file.

Once the proper position is established, characters, lines, or blocks of lines in the edit file may be deleted, inserted, or replaced by information specified within the command, from the terminal, or from other files.

3.2 A Review of Text Editing

IPAD will employ a text editor as a primary facility for creating and modifying a number of character coded files:

1. Program source code development.
2. Task Control Sequences (TCSs) and TCS Skeletons (TCSSs).
3. Query Processor Sessions (QPSs) and QPS Skeletons (QPSSs).
4. Tutorial and other textual data.
5. Display or presentation data, e.g. for the Engineering Review Board (ERB).

Since all editors worthy of the name perform roughly the same functions on files of information and are not concerned with the subject matter of these files, IPAD presents no unique design requirements or constraints to existing text editing software. However, the ease of use, power and flexibility of the editor subsystem and its commands will be important to both the IPAD system implementer and the IPAD user. These and other points concerning design philosophy of the three text editors are discussed in Section 3.3.

Text editing capabilities required by IPAD may be broken down into file manipulation, line manipulation, character string manipulation, and formatting and general utility functions. These functions are described below and are cross-referenced in Table 3-1 to specific commands that perform similar functions under the CDC, IBM and UNIVAC text editors. This table in turn cross-references the page number of the manufacturer-supplied literature listed in those references appearing in the table. Thus, considerable information may be readily obtained on the workings and specific implementations of the functions that follow. The table additionally provides (left hand edge) a cross-reference back to the subsections below.

TABLE 3-1. - MAJOR EDITING FUNCTIONS AND COMMAND COMPARISONS

Refer to Section 3.2		FUNCTION TO BE PERFORMED	CDC EDITOR (Reference 7)	IBM EDIT (Reference 8)	UNIVAC ED (Reference 9)
.1 FILES	.1	Set up to create a new file from terminal	CREATE (p.4-9)	INPUT (p.103)	INPUT (p.18-10)
	.2	Load and sequence local file into edit file	EDIT (p.4-12)	MERGE (p.111)	ADD (p.18-8) (first option)
	.3	Combine all or part of a file with edit file	SAVE (p.4-14) (merge option)	MERGE (p.111)	ADD (p.18-8) (second option)
	.4	Save edit file upon exit from editor	SAVE (p.4-14)	SAVE (p.121)	Edit file saved unless OMIT (p.18-11)
.2 LINES	1	Delete specified lines from edit file	DELETE (p.4-20)	DELETE (p.91)	DELETE (p.18-9)
	2	Insert a line or lines into edit file	ADD (p.4-18)	INSERT (p.105)	INSERT (p.18-8)
	.3	Delete/Insert/Replace a single line from terminal	Line No.=Text (p.4-11)	Line No.=Text (p.107) Current line=Text	Not available in ED as a single command.
	.4	Resequence all or a portion of edit file	RESEQ (p.4-22)	RENUM (p.115)	Not available in ED as a single command.
	.5	Position edit file line pointer	Not available, line is found by number or string in line.	BOTTOM, DOWN TOP, UP (p. 85, 93, 127, 129)	± (p. 18-14) LAST (p. 18-10)
.3 STRINGS	.1	Locate a character string within a line	SAVE (p.4-14) LIST (p.4-16) DELETE (p.4-20)	FIND (p.97)	LOCATE (p.18-1) FIND (p.18-9)
	.2	Replace or modify a string within lines	/text 1/=/text 2/ (p. 4-23)	CHANGE (p. 87)	CHANGE (p. 18-9)
.4 I/O AND GENERAL	.1	Define line length, tabs and tab key	FORMAT (p.4-6)	TABSET (p.125) EDIT line (p.76)	PLIMIT (p.18-12) TAB (p.18-14) SET (p.18-13)
	.2	Display lines from edit file on terminal	LIST (p. 4-16)	LIST (p. 109)	PRINT (p.18-12) QUICK (p. 18-13)
	.3	Display lines when located by pointer	LIST (p. 4-16)	VERIFY (p. 131)	PRINT (p. 18-12)
	.4	Compile and execute program under editor	RUN (p. 4-26)	RUN (p. 117)	Not available, Must leave ED
	.5	Terminate, and exit from editor	BYE or BYE,BYE (p. 4-5)	END (p. 95)	EXIT (p. 18-9)
	.6	Check for syntax errors in source code as line is entered	Not available in EDITOR	SCAN (p.123)	Not available in ED
	.7	Display command tutorial or usage information	TEACH (p. 3-5)	HELP (p. 141)	Not available in ED.

NOTE: Page numbers following commands refer to the manufacturer's literature listed in the references quoted.

3.2.1 File manipulation functions. - These functions relate to either the editor's edit (random) file, input files to be edited, or the editor's output files.

3.2.1.1 Setup: When the user desires to create a new file, a command is required to direct the editor to enter the input mode and accept everthing typed at the IPAD user's terminal. The command should specify the beginning line sequence number, the line number increment, and whether the line is to be displayed back to the user.

3.2.1.2 Load: In loading an existing file into the edit file for the user to modify, a command must be given which will allow an entire file to be moved into the edit file working area with line sequencing and resequencing options. These line numbers are editor-generated and are appended to all lines as they are stored in the edit file.

3.2.1.3 Merge: In the creation of a new file, the editor should have the ability to merge existing files (or selected portions of existing files) into a specified area of the edit file. Through the use of these commands the user inserts previously stored information into any desired new files.

3.2.1.4 Save: The contents of the edit file are operated upon in a temporary storage area. In order to save the edit file or selected portions of the edit file, a command which allows the user to assign a file name and write the edit file contents onto that file must be available.

3.2.2 Line manipulation functions. - These functions relate to the specific lines of text being created or modified.

3.2.2.1 Delete: The IPAD user has the ability to delete lines anywhere in the edit file. The line or lines to be deleted are located by line number, line pointer or by a character string search. Options include the ability to display the lines before deletion occurs so the user can determine if the deletion is desirable.

3.2.2.2 Insert: The user requires the ability to insert lines created from the IPAD terminal between existing lines in the edit file. This insert command does not result in overwriting of existing edit file lines unless a user-specified option is provided.

3.2.2.3 Replace: The above line deletion and insertion capabilities allow the scanning of a file with deletions or insertions occurring within line positions specified by command parameters. In addition, the user replaces a single line with terminal input information by simply typing in a line number and the associated text any time in the editing process. This allows simple correction of errors or omissions made by the terminal user.

3.2.2.4 Resequence: At completion, or at any point during the editing of a file, the user has the ability to resequence the entire edit file or any portion of it. The command allows the user to specify the starting number and number increment to be used.

3.2.2.5 Position: CDC's EDITOR assigns a line number to every line. However, the other systems provide for the positioning of a line pointer in files not sequenced by line numbers. This offers an advantage for files (e.g., report text) where line sequencing serves no useful function.

3.2.3 String manipulation functions. - These functions are related to character strings embedded within lines.

3.2.3.1 Locate: The text editor allows the user to locate a character or set of characters within lines of text in the edit file. A single line, lines within specified limits, or the entire file is searched at the option of the user. The user modifies the lines as they are found or displays the entire set of lines containing the search character string.

3.2.3.2 Modify: The editors provide a command that allows the user to replace a set of characters with another set of characters within edit file lines. Both the search and replacement character strings are command parameters. Characters in all lines between specified limits or in lines throughout the entire edit file can be so modified.

3.2.4 Formatting and general utility functions. - These functions are related to formatting or general functions not covered above.

3.2.4.1 Format: The format of the lines in the edit file is under the control of the editor user in much the same manner as manuscript typing is under control of the typist, or card punching is under the control of the keypunch operator. Commands exist in the editor which will allow the user to define his own format in terms of line length (in number of characters), tab positions and terminal tab key definition. He selects a frequently used format (e.g., FORTRAN source statement) from a list of predefined formats provided by the editor.

3.2.4.2 Display, general: The user displays the entire edit file or any portion of it at any time by issuing a single command. This command allows him the options of displaying the file either with or without the editor-generated line numbers.

3.2.4.3 Display, selective: In addition to displaying lines between specified line numbers, the editor displays all lines containing a specified character string as they occur in the edit file or as the line pointer points to the line. This allows the user to limit output to lines containing items of immediate interest to him.

3.2.4.4 Compile, load and execute a source program: The user has the ability to compile, load and execute a source program created or modified in the edit file without leaving the control of the editor. This command allows debug runs and user corrections to be made under the editor until a working program is produced.

3.2.4.5 Terminate: The editor provides a terminating command to exit from the editor and return the user to the control of the operating system. If the edit file is automatically destroyed upon exit, the editor provides a prompting message for the user to save the edit file unless he intentionally wishes to destroy it.

3.2.4.6 Syntax analysis: IBM's EDIT provides the ability to perform a syntax analysis on program source entries as the line is being entered. This is a very valuable aid in programming since a large number of initial program errors are simply errors in syntax.

3.2.4.7 Tutorials: Both CDC and IBM provide tutorial information in response to a user request. IBM's implementation is the more useful in that it selectively provides the pertinent information relative to the command being questioned. An autotutorial capability is highly desirable in an IPAD environment.

3.3 A Comparison of Text Editors

The three editors compared in this study represent the current versions, and the literature referenced is the latest available from the manufacturers at the time of the study. Both IBM and UNIVAC representatives say that new releases are due before the end of the first quarter of 1973. Both indicated that these releases will contain editor improvements.

Each editor has unique design characteristics and unique features. The CDC EDITOR allows the user to use intermixed EDITOR, INTERCOM and SCOPE commands or control statements without leaving the control of EDITOR. This allows the user to have a very flexible and powerful set of commands and reduces exits, reentries and file saving. The IBM EDIT does not allow this much flexibility but does allow the user to compile and execute a newly created or edited program without

leaving EDIT. In addition, EDIT can provide syntax checking of each source program statement as it is created. The UNIVAC ED under the EXEC8 system for the 1100 series is the most isolated from the operating system of the three. The ED command list contains more than forty separate commands or over twice the number in either the CDC or IBM editors. This is due to differing design philosophies. UNIVAC employs many, simple, single-function commands while CDC and IBM use fewer commands having multi-function options to accomplish many of the same editing requirements.

With the FORMAT and MERGE commands provided by IBM as a program product (at an additional fee), the IBM EDIT package is the most powerful general text editor of the three for the general IPAD user. However, from the standpoint of the user whose primary use will be the creation and editing of source programs as opposed to textual information, the CDC EDITOR appears to be the most flexible and easy to use. The UNIVAC editor, in its present form, appears to be the least flexible of these three editors for IPAD implementation.

3.4 IPAD's Text Editor

The editing functions examined in Table 3-1 will be available to the IPAD user. All are implemented in the current CDC, IBM and UNIVAC editors and may either be accomplished by a single command or by a combination of two or more commands. Although the document survey indicated that these were significant differences among the manufacturer supplied editors, any of the three editors reviewed in detail met the IPAD requirements. Rather than supply a separate text editor specifically for IPAD, it was determined that the existing editors could supply this capability, although each in a slightly different manner.

However, since the Query Processor (QP) is also concerned with interactively creating/modifying data, a comparison of the two capabilities is in order:

1. A text editor is intended to perform character string operations on sequential files only. QP is intended to perform arithmetic operations on data base information regardless of its organization.

2. A major objective of a QP is to permit the user to interactively devise complex search strategies to access/update data. A text editor provides one search strategy based on relative or absolute line numbers, and one based on character content within a line.

3. A text editor is supplied by the computer manufacturer independently of IPAD and DBMS, consequently (in initial implementations) it operates on conventional files only. QP is provided with DBMS and operates on data base information as well as conventional files.

The conclusion is that the functions of the text editor and QP overlap to some extent but neither is an adequate replacement for the other. The IPAD user need employ both in manipulating textual information. In particular, when information to be edited exists in the data base or text editor output is to be dispositioned to the data base, QP may be used to interface the text editor with DBMS. That is QP will act as an I/O Formatter (IOF) to map data base information to a conventional file and/or a conventional file into data base information. Eventually, it is anticipated that all text editors will be updated to optionally operate on data base information directly through DBMS.

4 OPTIMIZER AND PARAMETERIZER MODULE (OPTUM), A GPU

The IPAD Optimizer/Parameterizer is a collection of General Purpose Utilities (GPUs) for performing mathematical optimization or one of its subfunctions (viz, parameterization or sensitivity extraction). The optimization methods listed in the Sections 4.2 are currently regarded as the best techniques for solving the pertinent optimization problems. The particular method selected for use from the collection becomes the object code driver OPTUM for that particular optimization task. A sequence of OMs within the IPAD system can be optimized by OPTUM with respect to some objective function defined by the user.

4.1 Introduction

The IPAD optimizer operational module collection must contain a selection of multivariable search techniques which will modify the initial estimated values of several independent "design" variables to improve the value of an objective or merit function. Some of the techniques will result in a set of values for the independent variables for which the objective function attains a relative optimum. This mathematical optimum is usually achieved at the expense of many objective function evaluations.

For situations where objective function evaluation is expensive, techniques must be provided to modify an initial guess at the independent variable values so that the objective function value is improved. Typically, the interactive user will make a judgment as to whether it is practical to continue the computations to improve the objective function or whether the expected improvement is not worth the computer expenditure. In this mode of operation, only an improvement will be achieved; the techniques used are approximate optimization methods which yield only an approximate optimum.

The practicality of carrying out total vehicle optimization using a detailed synthesis program in the necessary disciplines appears to be questionable. The difficulty is not in the unavailability of a method but in the computer time involved. A more practical approach is to carry out optimization at the subsystem level by suboptimizing the appropriate basic design variables for each subsystem. A limited number of design variables could then be optimized for the whole system.

Each optimization method must potentially interface with all OMs and be interchangeable with other optimization methods so that the appropraite method can be matched with the particular task configuration of OMs selected for optimization. Query Processor (QP), which is discussed in detail in Section 1.3.1, can be made

reponsible for the interface between the optimization routine used, and the OMS. QP can be delegated the responsibility of taking the design variable values generated by the optimizer and placing them where the OMs can access them; QP could also select the desired results generated by the OMs and place them where the optimizer can find them. Thus QP permits great flexibility in the choice of parameters used in optimization studies.

Virtually any input parameter can be modified to optimize the objective function. Similarly, any parameter generated by an OM can serve as the objective function to be optimized. This means that an optimization can be performed on any process where the objective (merit) function varies as a function of the basic design variables. In the case where more than one parameter is selected as an objective in the optimization problem, a weighted sum of all the desired parameters can serve as the objective function. Again, QP (via a QPS) can provide this weighted-sum objective function evaluation.

4.2 Optimization Methods

There are many excellent optimization methods available in the literature. Although they each differ in the procedure used to optimize, they all share characteristics in common. For example, they all need some initial guess at the optimal values of the independent variables. These values (guesses) are then refined by the optimization technique; this is done by determining what changes occur when the variables are increased or decreased. In some methods this information is used to generate gradients; in other methods this information provides local geometry information for approximation techniques. In still other methods the lowest objective function value is retained from the exploration of the independent variable influences on the objective function.

Additional relationships can be seen graphically by classifying each optimization method by its attributes. Each technique can be classified as falling on a certain branch of the overall optimization technique evolution tree. Figure 4-1 presents a functional flow diagram of the evolution tree which classifies the current best technique according to their attributes. The figure illustrates the fact that each optimization method is not distinct from all others but is related.

The two primary branches of the tree divide the optimization methods into random and nonrandom methods. Since the major emphasis has been on the development of nonrandom methods (because they are faster and more accurate than random techniques), there are many more methods on the nonrandom primary branch. The most powerful methods are gradient-based methods which is the first of the secondary branches.

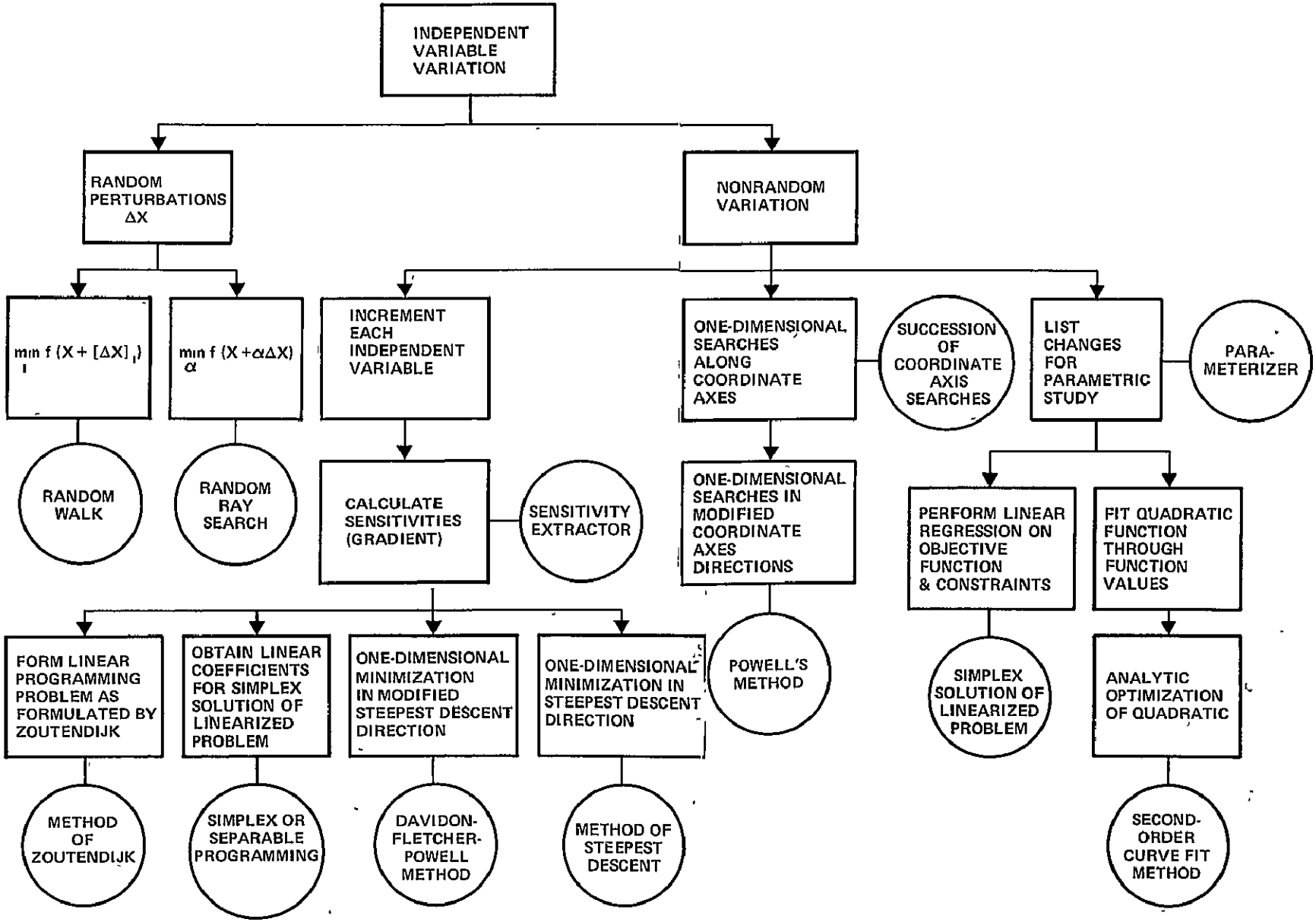


Figure 4-1. Functional Flow Diagram of Optimizer Evolution

The other two secondary branches are specialized methods which apply to problems with discontinuous gradients, or to problems where only an improvement of the objective functions is desired.

The parameterizer and sensitivity extractor appear in the figure even though they are not optimization methods. The computation required by these techniques, however, is identical to a subset of the computations required for some of the optimization methods. Sensitivity extraction uses the gradient evaluation subroutines from an optimization technique to obtain sensitivities; parameterization uses the linkages setup for the optimization process to evaluate the OM sequence with input parameter variations. For this reason they naturally fall in the evolutionary tree of optimization techniques and are therefore treated here.

Each of the proposed multivariable search techniques is discussed individually below. The problem formulation to which each applies is given. References, tutorial aids, I/O requirements, and sub-computations required are discussed. The order of presentation is organized around Figure 4-1. The gradient-based methods are discussed first, followed by one-dimensional search methods, list-change methods, and finally random-perturbation methods. The last three subsections describe techniques which are supplementary to the previously described techniques, one-dimensional search algorithms, linearization of the objective function and constraints, and penalty function techniques.

4.2.1 Gradient methods. - In this subsection, optimization methods which require computation of gradients are discussed.

4.2.1.1 Steepest descent method: The method of steepest descent (Reference 12) is the simplest of the unconstrained optimization methods of the gradient type. It requires that continuous derivatives of the objective functions exist. To obtain these derivatives for a general case a finite difference method is required. The user must make a judgment concerning the stepsize used in order to obtain accurate gradients. The user will also be required to indicate when the process should terminate. The best way to do this is to monitor the optimization process interactively by viewing plots of the objective function as it decreases and observing the independent variable values versus iteration cycle. These curves should approach some stationary value near convergence; the user can then judge whether additional iterations are necessary or not. Figure 4-2 presents a functional flowchart of steepest descent search method.

A one-dimensional search subroutine can be used to obtain the maximum benefit from a given steepest descent direction. The one-dimensional search requires some convergence criteria with which the user can interact; the results could be graphically displayed as an option so that the user could then judge convergence and give direction to the program. Optionally the program could use convergence tolerances to terminate the search.

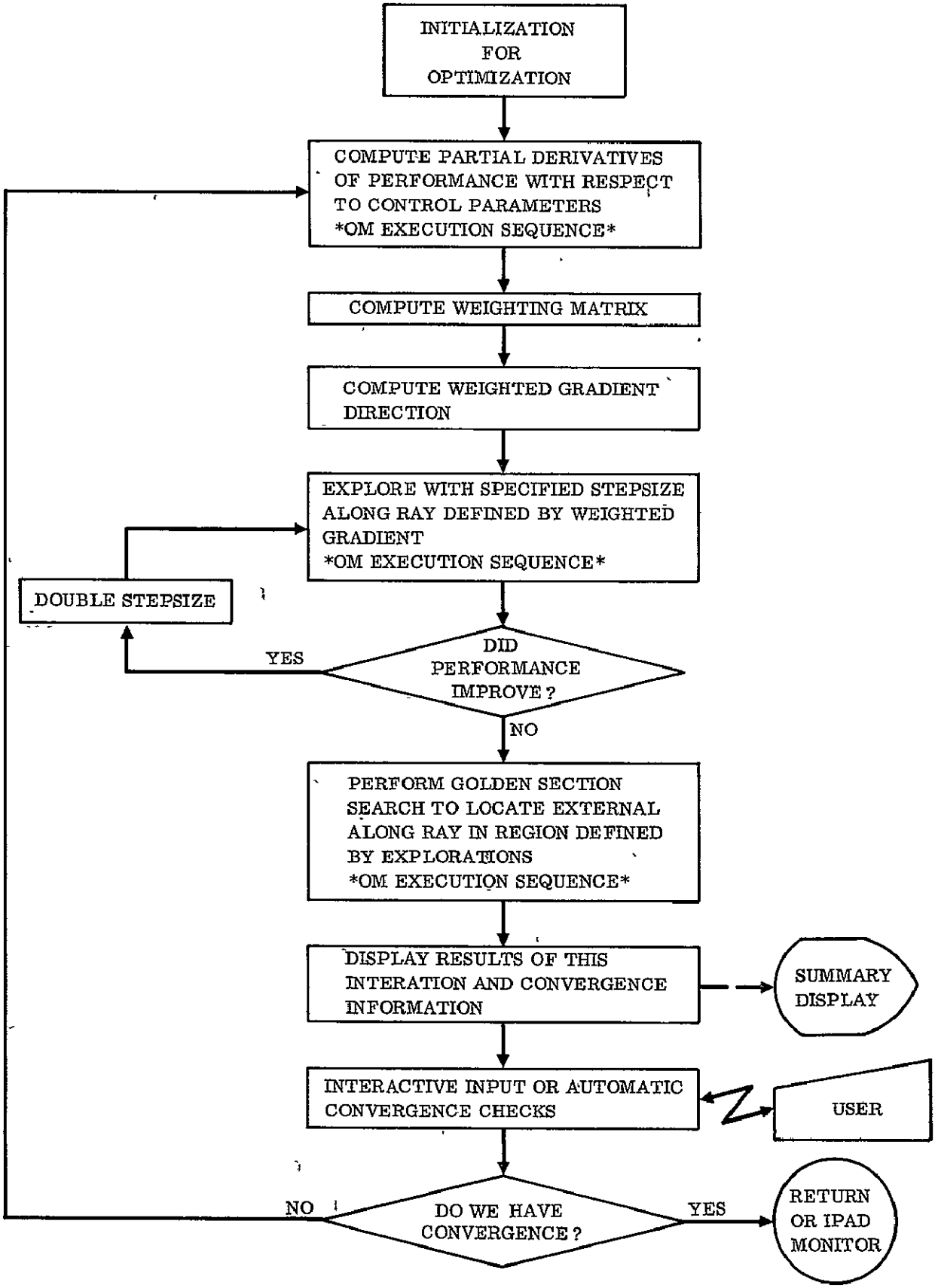


Figure 4-2. Functional Flowchart of Steepest Descent Method

Printed output typically includes the most important parameters after each one-dimensional search, and complete OM output for the final design achieved.

4.2.1.2 Davidon-Fletcher-Powell (DFP) method: The DFP method (Reference 13) has been recognized in the literature as the best of the gradient type unconstrained optimization techniques. The method (Figure 4-3) modifies the steepest descent direction by using an approximation to the Hessian matrix to ultimately achieve more rapid convergence. The user interaction and I/O requirements for DFP method are similar to those of the steepest descent method.

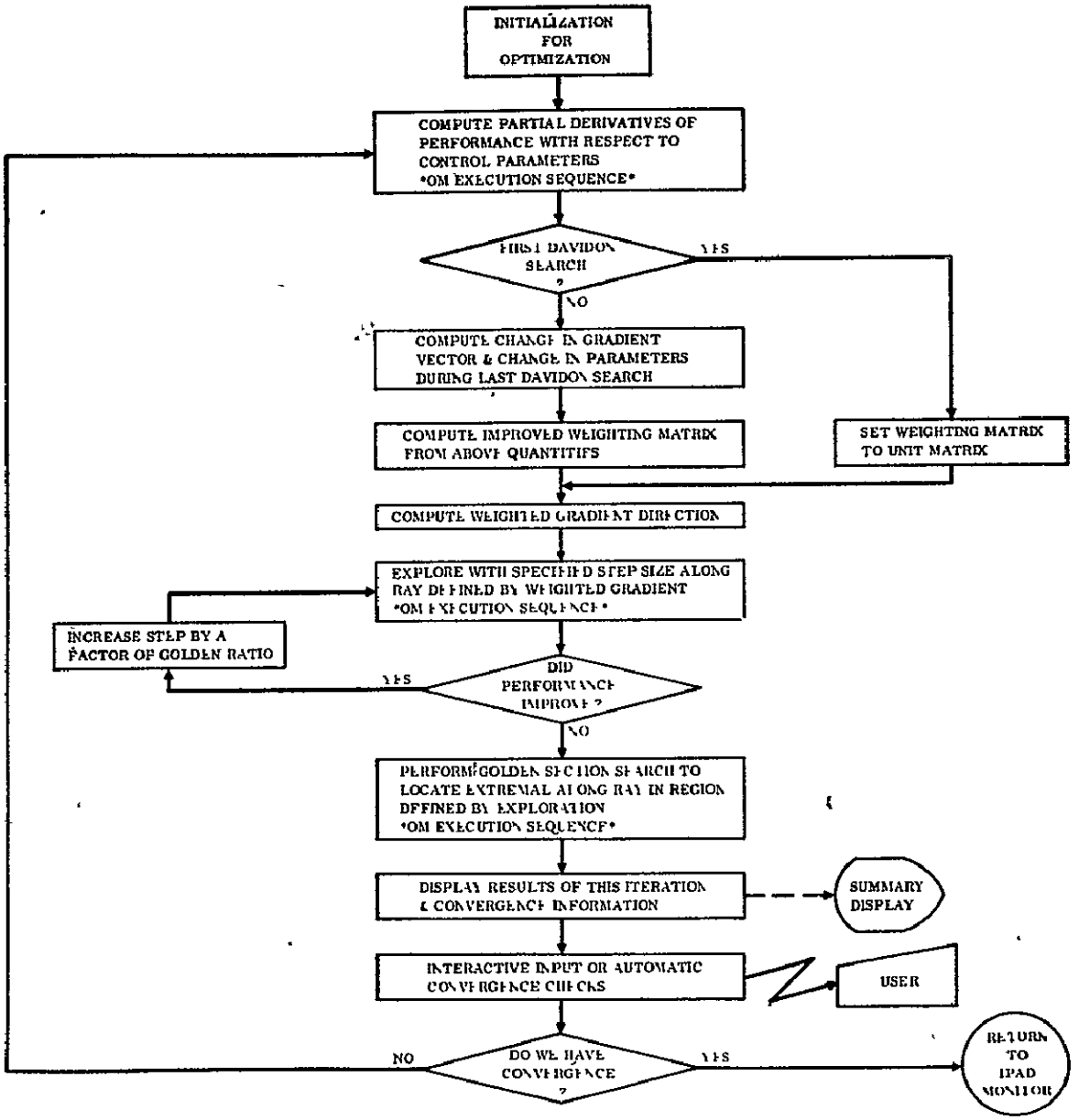


Figure 4-3. Functional Flowchart of Davidon-Fletcher-Powell Method

4.2.1.3 Simplex method: The simplex method (Reference 14) applies only to constrained optimization problems where both the constraints and the objective function are linear. The technique achieves the mathematical optimum of the constrained problem with relatively little computer time. Unless the analysis modules are specifically written to supply the linear coefficients for the simplex method, they may be obtained by evaluating the gradients of the objective and constraint functions desired.

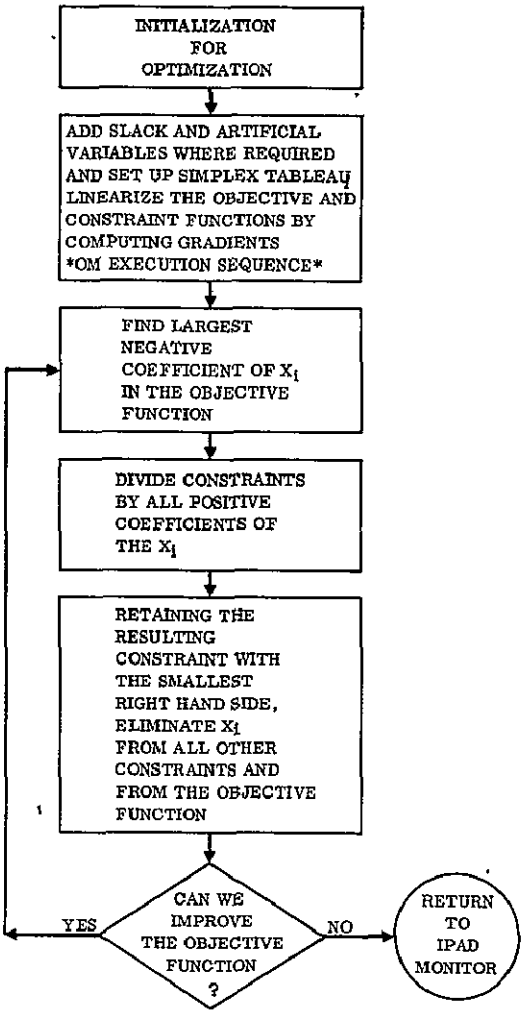


Figure 4-4. Functional Flowchart of Simplex Method

Figure 4-4 shows the function flow of the simplex method. The user has no interaction opportunities for this method. Once the linear coefficients of the constraints and the objective function have been obtained (either directly or by linearizing the equations), the simplex method can proceed to optimize without any additional OM sequence execution. Since the simplex method is a closed algorithm, there is no useful interaction.

4.2.1.4 Separable programming: The technique of separable programming (Reference 15) applies to the inequality-constrained optimization problem where the objective function is linear and the constraint functions are quasi-linear. The technique consists of approximating the nonlinear inequality constraints by linear segments and solving the resulting linear programming problem by the simplex method. Linear equality constraints could also be handled by this method.

The user could interact with the program by specifying the way in which the nonlinear constraints are linearized, however, for higher dimensions this becomes difficult.

4.2.1.5 Method of Zoutendijk: The method of Zoutendijk (Reference 16) solves the inequality constrained optimization problem by breaking it into a sequence of linear

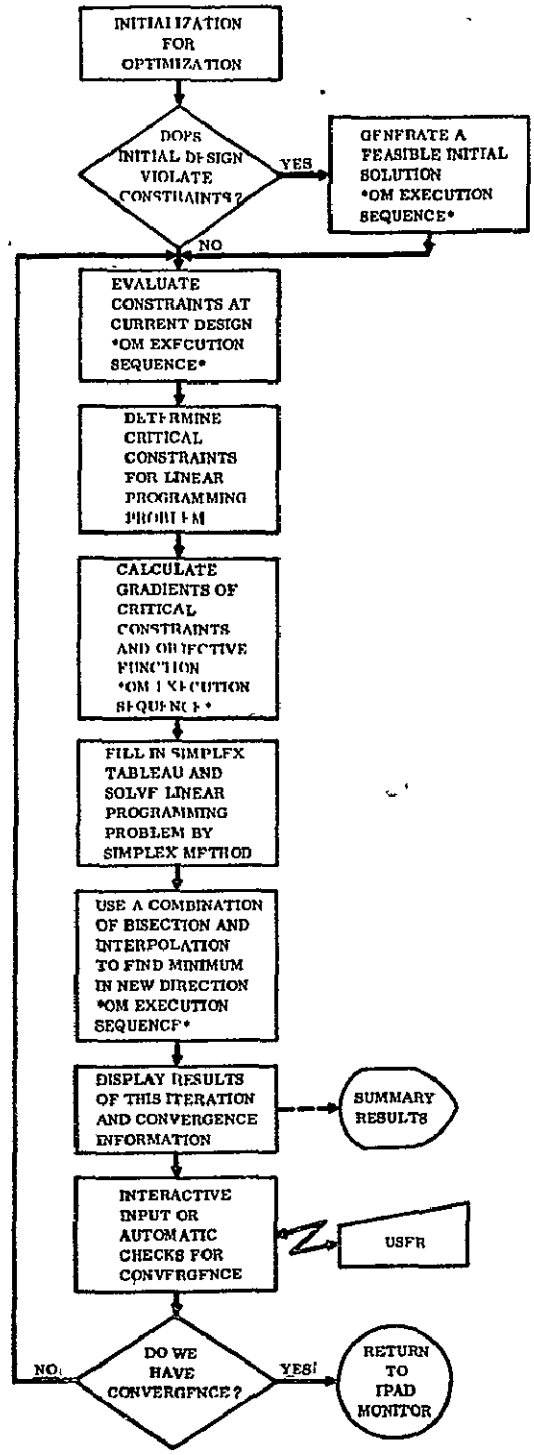


Figure 4-5. Functional Flowchart of the Method of Zontendijk

programming problems. The simplex method is then used to solve the linear programming sub-problem. The method works best for problems that are almost linear. Figure 4-5 is the functional flow chart of this method.

The user can interact by designating for each iteration the constraints that are most critical at that time. He can also judge convergence based on plots of objective function and independent variable values versus iteration cycle.

4.2.1.6 Sensitivity extraction: Sensitivity extraction utilizes that portion of the optimizer that calculates the sensitivities (partial derivatives) of the objective function with respect to the independent variable identified. The normal mode of operation will be to call the OM sequence with the independent variables perturbed one at a time. The resulting objective function values can also be used to estimate the partial derivatives of the objective function.

4.2.2 One-dimensional search methods: - Discussed in this subsection are optimization methods which reduce the multi-variable optimization problem into a sequence of one variable optimization problems.

4.2.2.1 Powell's method: The method of Powell (Reference 17) with modifications by Zangwill (Reference 18) is a quadratically-convergent multivariable search technique for optimizing unconstrained objective functions. The method (Figure 4-6) has the attractive feature that gradients are not required. Thus, for functions with discontinuous derivatives, this method is a good choice. It consists of a sequence of one-dimensional searches in conjugate directions. The one-dimensional searches can be handled effectively by a simple

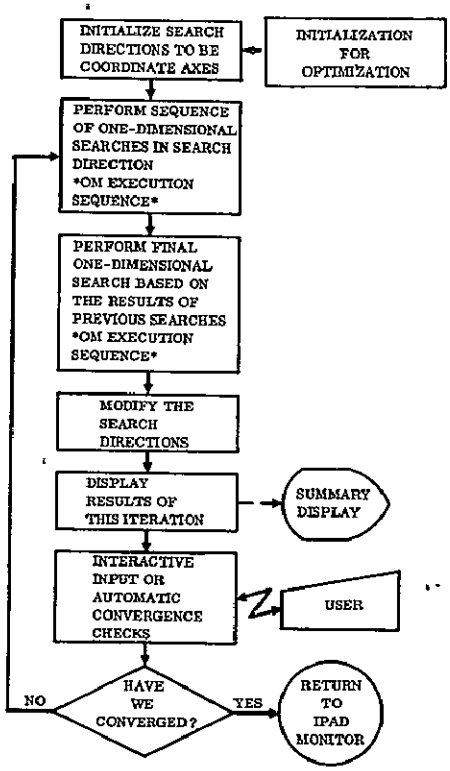


Figure 4-6. Flowchart of Powell's Method

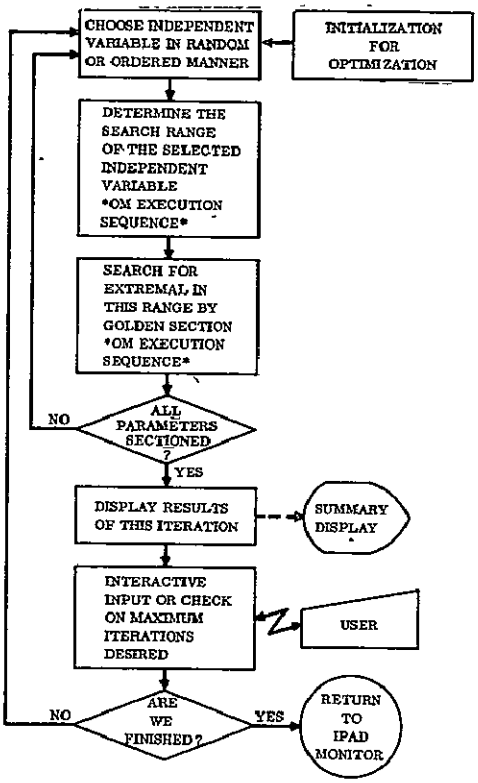


Figure 4-7. Flowchart of Succession of one Dimensional Searches Along Coordinate Axes

curve fit to several objective function values and its analytic optimization. The user can interact by judging convergence of the overall technique and stopping the procedure when practical convergence has been achieved.

4.2.2.2 Succession of one-dimensional searches along coordinate axes: This method should be used when the unconstrained objective function evaluation is expensive and does not possess nice analytic properties. This method could also be used to obtain a good starting point for a more powerful optimization method. The technique is not designed to optimize but simply to improve the objective function value. Figure 4-7 shows the functional flow diagram of this method.

The user can interact to control the one-dimensional search convergence, or to select the order in which the independent variables are optimized.

4.2.3 List changing techniques. - The methods which involve evaluation of the OM sequence for specified values of the independent variables, as an end in itself or for subsequent optimization, are discussed below.

4.2.3.1 Second-order multivariable curve fit of objective function and its minimization: This technique should be used whenever evaluations of the objective function are expensive. With only a few function values, a second order surface is fit to the function, and its optimum obtained analytically. This optimum will generally represent an improvement in the objective function value. This procedure can be repeated as often as desired. A program employing this method is described in Reference 12. Figure 4-8 shows the functional flowchart of this method.

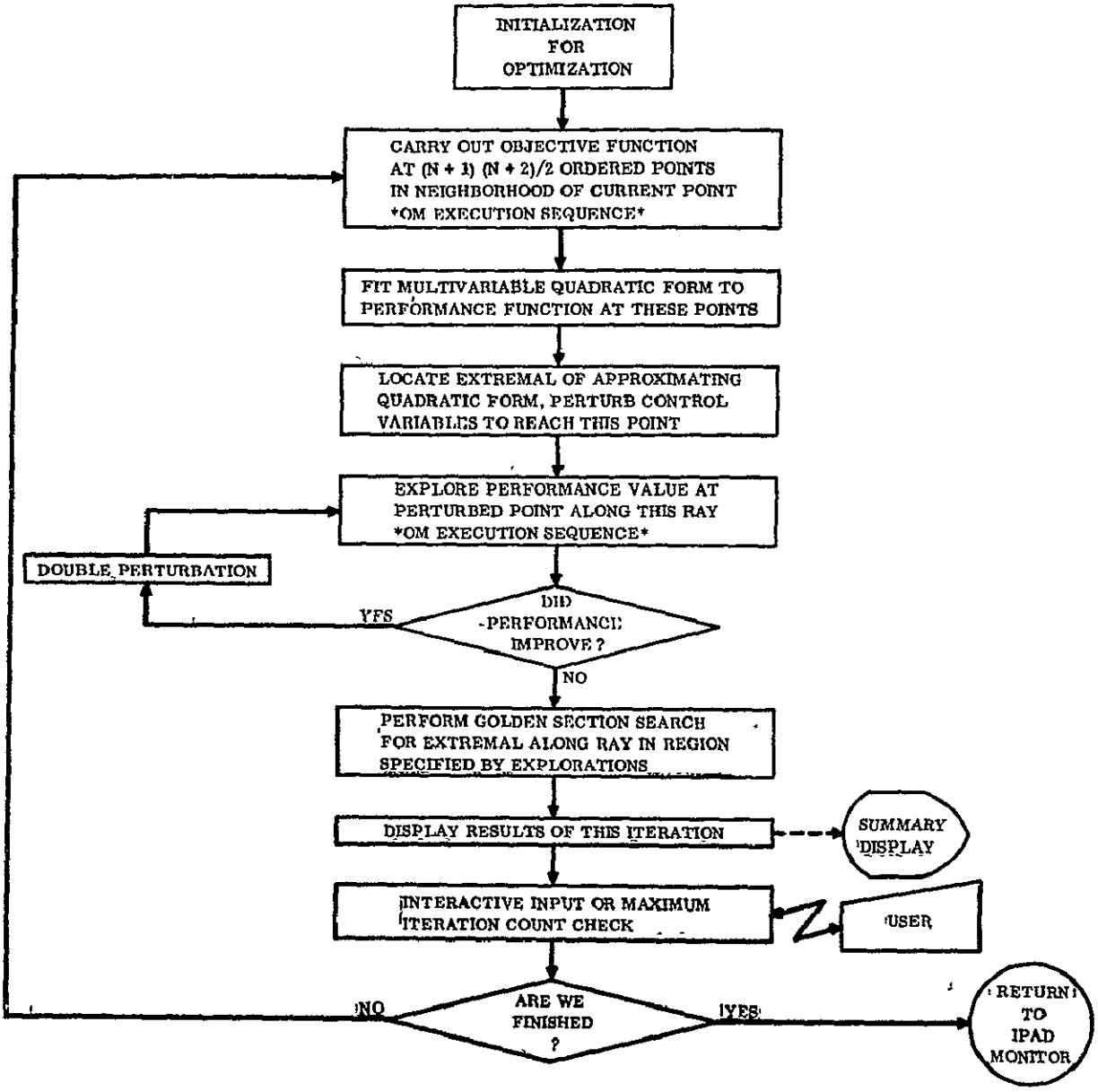


Figure 4-8. Second-order Fit Approximate Modeling Technique Flow Chart

4.2.3.2 Parameterizion: Parameterization provides the capability of exploring the output from the sequence of OMs (or an objective function behavior) as a function of prescribed changes in the independent variables. Prescribed sequences of changes (list changes) will be accommodated.

The user can interact with the parameterizer by specifying the list changes interactively. By observing an objective function behavior, the user can modify the list changes and continue to explore the behavior of the function.

4.2.4 Random variation. - The following techniques involve the use of random numbers to arbitrarily change the independent variables. This shotgun approach is included for the situation where the user has no idea what variations should be made in the independent variables to improve the objective functions.

4.2.4.1 Random walk: This is an approach to improve the unconstrained objective function value. Independent variable values are selected at random in an attempt to find a better value of the objective function (Figure 4-9). This technique is most properly used to provide an initial guess to a more powerful optimization procedure.

4.2.4.2 Random ray search: This is a crude method designed to improve the unconstrainted objective function value (Figure 4-10). The user can apply his knowledge to the process by selecting a ray which will change certain variables more rapidly than others. He can also control the convergence of the search along the ray. A description of this method and a program employing it can be found in Reference 12.

4.2.5 Methods for performing the one-dimensional searches which are required by most of the preceeding techniques. - Many of the optimization methods require the minimization of a function of one independent variable. This usually constitutes a major portion of the computation time. The following is a discussion of the two techniques which have been most successfully used for this purpose.

4.2.5.1 One dimensional search by polynominal approximation: This method improves the value of an unconstrained objective function of one independent variable. Four functional values are fitted by a cubic polynominal whose minimum is found analytically. Depending upon how closely the objective function behaves to a cubic, the resulting minimum point can be close to optimal or conversely not even an improvement in the objective function value.

The user can interact by selecting the four points mentioned above.

4.2.5.2 Golden section search: The golden section search method (Reference 19) can determine the optimum of an unconstrained objective function of one independent

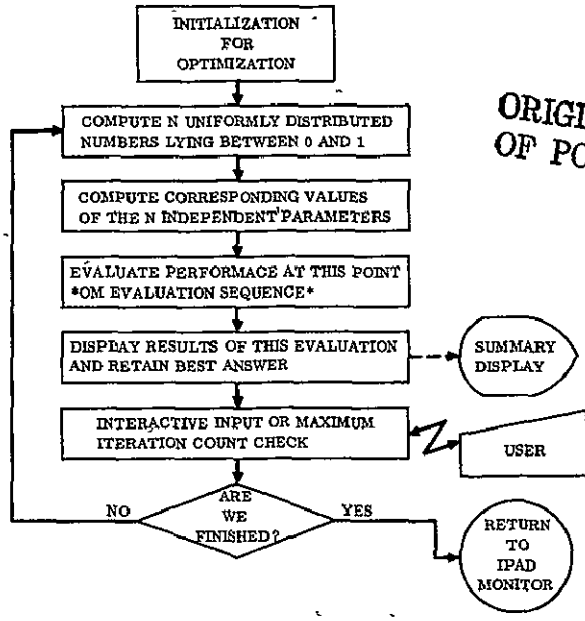


Figure 4-9. Random Walk Flowchart

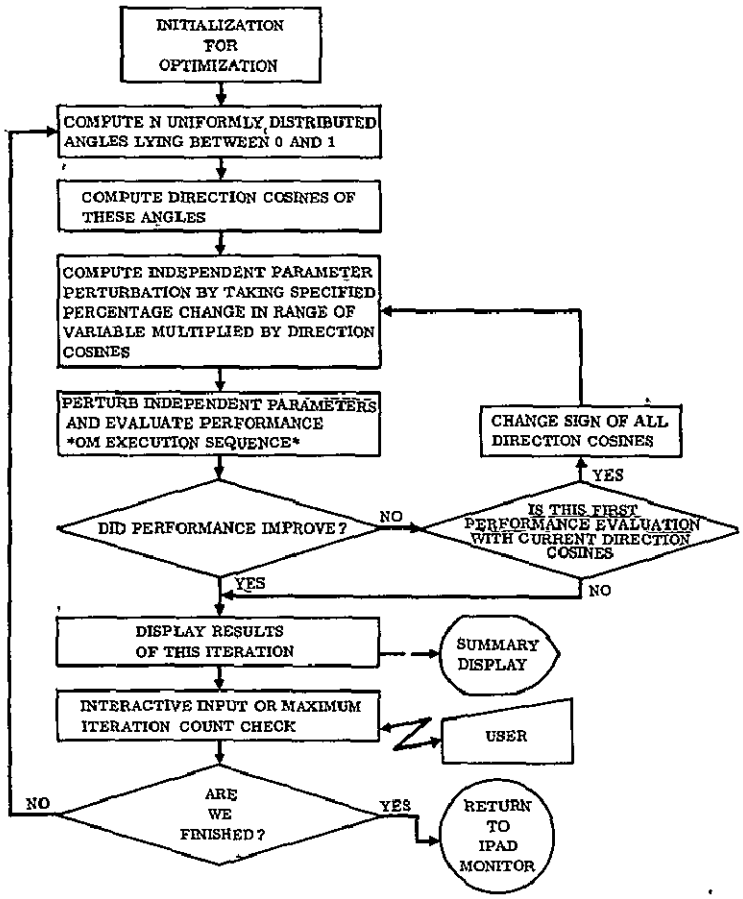


Figure 4-10. Flowchart of Random Ray Search

variable. It is an efficient and reliable technique which will work even for discontinuous functions.

The user can interact with the method in determining convergence.

4.2.6 Linearizing the objective function and constraints.- This procedure should be used whenever the objective function evaluation is expensive and the objective function and constraints are quasi-linear. Linear functions are fit in the least squares sense to the constraints and objective function; the simplex method is used to solve the resulting linear programming problem. A program for this method is described in Reference 12.

The user can interact by selecting the independent variable points at which the original objective function and constraints are evaluated.

4.2.7 Transformation of constrained problems to unconstrained problems by way of penalty functions. - For highly constrained nonlinear optimization problems, the penalty function techniques are most successful. Two of the best methods for transforming the constrained problem into an unconstrained problem by using penalty functions are discussed below.

4.2.7.1 Fiacco-McCormick method: The method of Fiacco-McCormick (Reference 20) will solve the general nonlinear programming problem of optimizing an objective function subject to equality and inequality constraints, all of which may be nonlinear. The method involves transforming the constraint functions to penalty terms which are added to the objective function. The resulting combination is then optimized without constraints by any reliable unconstrained optimization technique. Figure 4-11 shows the functional flow of the general penalty function nonlinear programming technique. The method of Fiacco-McCormick belongs to this class of methods.

The user has many opportunities to interact with this method. The penalty functions involve parameters which may be interactively selected or modified. Scaling and normalization constants are important to the technique and could be specified interactively. Finally, the unconstrained optimization method could permit additional interaction.

4.2.7.2 Exponential penalty functions: The method of Allran and Johnsen (Reference 21) is a penalty function technique which can solve the general nonlinear programming problem. The method uses a different form for the penalty terms than Fiacco-McCormick. Otherwise the same comments apply here as were given for the Fiacco-McCormick method.

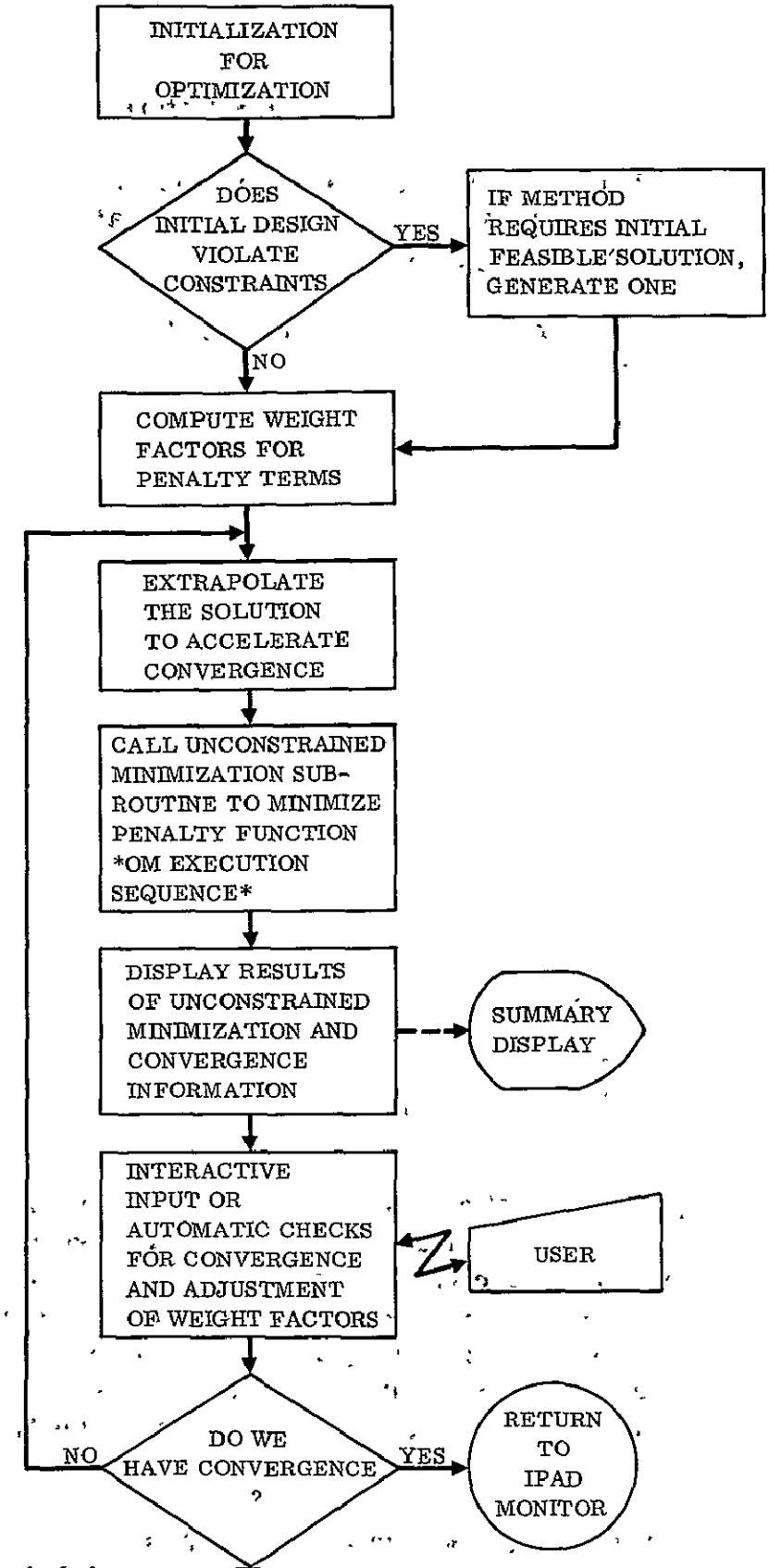


Figure 4-11. Functional Flow of Penalty Function Nonlinear Programming Techniques

4.3 Optimization Method Selection

The problem that faces the user is the determination of which method of optimization is best for a given problem. The type of mathematical formulation of the problem is important in solving this problem.

4.3.1 Classification. - Optimization problems can be classified according to the following characteristics:

1. Type of objective function:
    a. Linear.
    b. Nonlinear.
    c. Continuous
    d. Discrete.
2. Type of constraints:
    a. Unconstrained.
    b. Equality constraints.
    c. Inequality constraints.
    d. Equality and inequality constraints.
    e. Constraint functions.
        - Linear
        - Nonlinear
3. Type of gradients
    a. Continuous
    b. Discontinuous
    c. Nonexistent

4.3.2 Diminishing returns. - The choice of the best method for a particular problem depends upon the accuracy desired in the answer. Most frequently the accuracy criterion is based upon practical considerations. The longer an optimization technique executes on the computer the better the evaluation of the objective function becomes until an optimum point has been reached. This is the desirable property of stability which is characteristic of most optimization methods. Therefore, the cost of computer time versus the benefit of expected improvement can be the basis for the decision of when the solution is accurate enough .

Expected improvement must be extrapolated from previously observed improvement versus the number of iterations or versus the computer time expended. One of

the important contributions that an interactive user can provide is the practical judgment of when sufficient and economical optimization has been accomplished.

4.3.3 Applicability. - A great deal is known concerning the applicability of optimization techniques to specific engineering problems. This information can be used in tutorial form to aid the engineer in selecting the appropriate optimization method for his particular problem. However the question of how much suboptimization should be performed before overall system optimization is attempted depends on the particular problem. Trade studies will be necessary to determine the best strategy in each case.

4.3.4 Implementation approach. - Tutorials lead the user step-by-step in the optimization process. When the appropriate method has been selected, the object code driver for that method (along with tutorial aids for their use) are configured into the sequence of OMs being optimized. The tutorial aids for the specific method selected lead the user in selecting appropriate iteration counts, tolerances and available options.

4.3.4.1 Tutorials: The user will have (1) specified the OMs that he wishes to use, (2) selected the design variables which will be subject to optimization and provided their initial values, (3) selected or created the variable which will serve as the objective function, and (4) selected the constraint variables. (This process has been described in Subsection 1.3.1) . With the mathematical formulation of the problem in mind, tutorial aids which will assist in the selection of the appropriate optimization technique should be employed.

The multivariable search techniques proposed can be classified according to the type of optimization problem which they solve. For example, a user might want the unconstrained optimum of an objective function. There are several excellent techniques for this purpose. However, if certain values of the independent variables are not acceptable, then the space from which these values are selected must be constrained. The constraints are usually expressed as equality or inequality function relationships which must be obeyed. Depending upon the linearity or nonlinearity of the constraints and objective function, there is a method which will best handle the particular formulation.

The following is a list of proposed multivariable search methods which are classified according to problem formulation:

1. Gradient methods for unconstrained problems:
    a. Steepest descent.
    b. Davidon-Fletcher-Powell.
    c. Sensitivity extraction (as opposed to optimization).

2. Nongradient methods for unconstrained problems:

   a. Powell's method with Zangwill's modification.

3. Methods for linear objective function with linear constraints:

   a. Simplex method.

4. Methods for linear objective function with nonlinear constraints:

   a. Zoutendijk.
   b. Separable programming.

5. Methods for nonlinear objective functions with nonlinear constraints:

   a. Fiacco-McCormick
   b. Exponential penalty functions.

6. Approximate modeling methods:

   a. Second-order multivariable curve fit of objective function and its minimization.
   b. Linearizing objective function and constraints by regression analysis and then using the Simplex method.

7. Approximate optimization methods which give an approximate optimum:

   a. Succession of one-dimensional searches along coordinate axes.
   b. Random walk.
   c. Ray search in various directions chosen from previous search or chosen at random.
   d. Parameterization (as opposed to optimization).

8. One dimensional searches:

   a. Polynomial approximation with analytical optimization.
   b. Golden section search with cubic extrapolation.

Some of these techniques require some of the other techniques to solve a subproblem encountered in using the method. For example, the Fiacco-McCormick method and the exponential penalty function method both require an unconstrained optimization method for part of their computation. Methods under parts 1, 6 or 7 above would be applicable. Methods under parts 1 and 7 require methods under part 8. The basic idea is that the various techniques are modular so they can be employed whenever necessary. With this modularity maintained, it is easy to add additional modules as improved techniques become available or when it is recognized that a certain type of problem is best solved by an existing method which has not yet been included as a module.

Figure 4-12 is a flow diagram of the important features in the proposed tutorial aids TCSS to guide the user, if he so desires, in selecting a good multivariable search method to solve his particular optimization problem. The method to be selected (terminating circles in figure) are those listed above. Some considerations involved will be the following:

1. How many independent variables (n) are to be used?
2. Is Parameterization (P), Sensitivity Extraction (S) or Optimization (O) being sought, or is merely an improvement in the objective function desired?
3. Is the problem constrained? If so:
   a. How many equality constraints (e) are there?
   b. How many inequality constraints (i) are there?
4. Is the objective function linear? Approximately linear?
5. Are the constraints linear? Approximately linear?
6. Approximately how many central processor seconds does it take to execute the sequence of OMs in evaluating the objective function (ℓ)?
7. Is this magnitude of cycle time acceptable?
8. Are the design variable derivatives continuous?

4.3.4.2 User interaction: When the problem has been completely defined, the user can indicate that execution is desired. The response time and the degree of interaction must now be considered. The response time is entirely dependent upon the execution time of the sequence of OMs that were specified in the problem definition. (The optimization algorithm's execution time is usually trivial in comparison.) If this time is excessive, then the value of interaction is questionable. However, in cases where response time is reasonable the user is shown the results of the objective function evaluations.

In the event of a failure in the optimization technique, the user can use his judgement in selecting a good recovery method. Tutorial aids should be provided in the individual optimization drivers for this purpose. The user should ultimately judge the convergence of the optimization technique based upon the practical considerations previously mentioned.

4.4 Operating Requirements for OPTUM

The central memory storage associated with the optimization techniques being considered is small (e.g., 5,000 octal) in comparison with the storage requirements

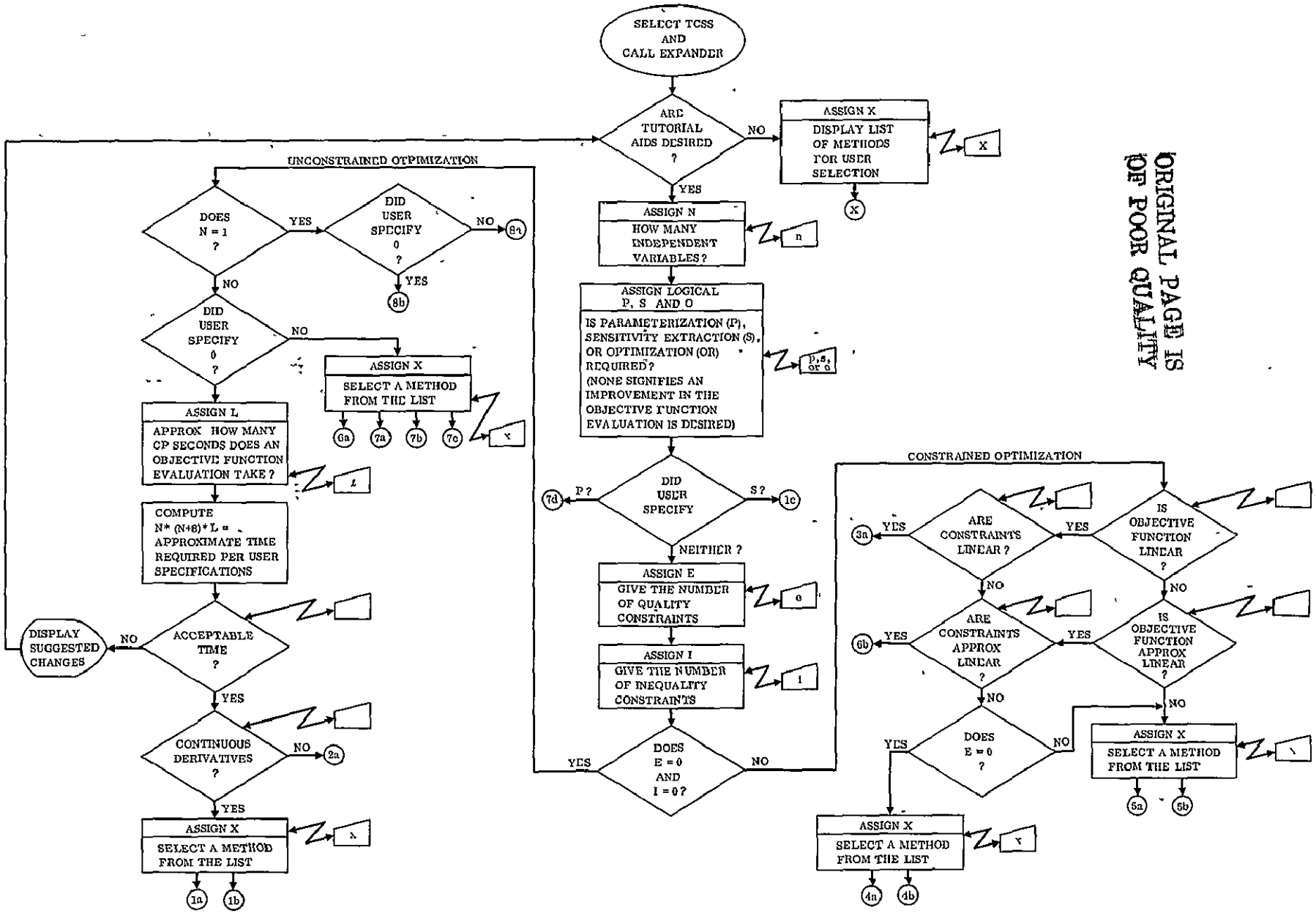


Figure 4-12. Proposed Tutorial Aids TCSS for Selecting OPTUM utilizing the TCSS EXPANDER

expected for the OMs being optimized. In this case, as in response time, the problem definition will determine the computer central memory storage required.

4.5 Conclusion

The optimizer GPU as proposed consists of a collection of optimization methods to give the user sufficient opportunity to match a relatively good, efficient method to the particular problem at hand. The best known current methods will be included, and the modular structure of the design will permit additional optimization techniques as they become available.

No one optimization method is best for all problems. For the situation where the objective function evaluation is time-consuming, approximate optimization techniques have been provided for efficiency. For problems where the OM sequence has limited fidelity and possible discontinuities, methods have been provided that are not continuity dependent. For problems with OM execution sequences which have moderate execution times and which are continuous, there are many good methods depending upon the type of problem constraints.

With the provided tutorial aids, the user of the IPAD optimizer is led to selecting the appropriate method for his problem and the appropriate parameters for the method selected. In this way he saves a great deal of time which is ordinarily spent seeking a good optimization method for his application. This, of course, is one of the basic features of the IPAD concept which has been implemented in the design of the optimizer.

The more complex optimization problems will require careful planning and control through feedback of results as the optimization process evolves. The following is a list of the formidable problems associated with optimization methods:

1. The large number (n) of design variables (equivalent to the combinational complexity, C_I^n). This is what Dr. Richard Bellman (Reference 22) termed "the curse of dimensionality".
2. The many design disciplines involved (long cycle time through the sequence of OMs).
3. Limited OM fidelity (e.g., small angle approximation).
4. Conflicting objective junctions (i.e., there is no single optimum — all optima depend directly on the selected objective function, for which there are often conflicting disciplinary interests).

5. Discontinuous operating hypersurface (multi-overlapping patch-quilt surfaces with large voids) where multiple external points mask the global optimum -- this is what Dr. Bellman termed "the menace of the expanding grid". (op cit.).

6. A poorly defined constraint space.

7. Exceptionally poor visibility in complex problems.

In nontrivial problems there is no a priori way of determining that an indicated extremal point will lie within the constraint space nor is global within that space. The global optimum cannot be distinguished from local optima except by exhaustion. For these and similar problems there can be no simple solution. The optimization analyst must be fully cognizant of the pitfalls to be avoided and be highly competent at his task.

5 GENERAL GRAPHICS PLOTTER (GGP), A GPU

The General Graphics Plotter (GGP) is the cornerstone in the foundation of IPAD. It addresses the problem of producing for all users the geometric, graphical and pictorial displays required in any design process. Each user in the design cycle requires visualization not only to verify that his design or analysis is feasible but also to document any realizable configuration. In fact, each user must always have uppermost in his mind the actual object he is designing and analyzing. This object, be it the smallest bolt or the most complex digital autopilot, must function; its aesthetic value on paper is of no value. Regrettably, the engineer often must use a tool (e.g. a digital computer) which cannot make subjective judgements and moreover cannot be taught to understand when a design is "really optimal". The engineer must always be alert to recognize this and continually check that the results (numerical) found by this tool are reasonable. GGP provides the vehicle for pictorially representing these numerical results.

The philosophy behind GGP is to allow the user complete freedom to define his display. He may wish a standard plot of two variables or an isometric view of a complex vehicle. In either case, he can control the appearance of the picture within the physical limits of the image device he is using. Moreover, he can manipulate the picture (rotate, translate, zoom, etc.) just as if he were holding the object in his hand.

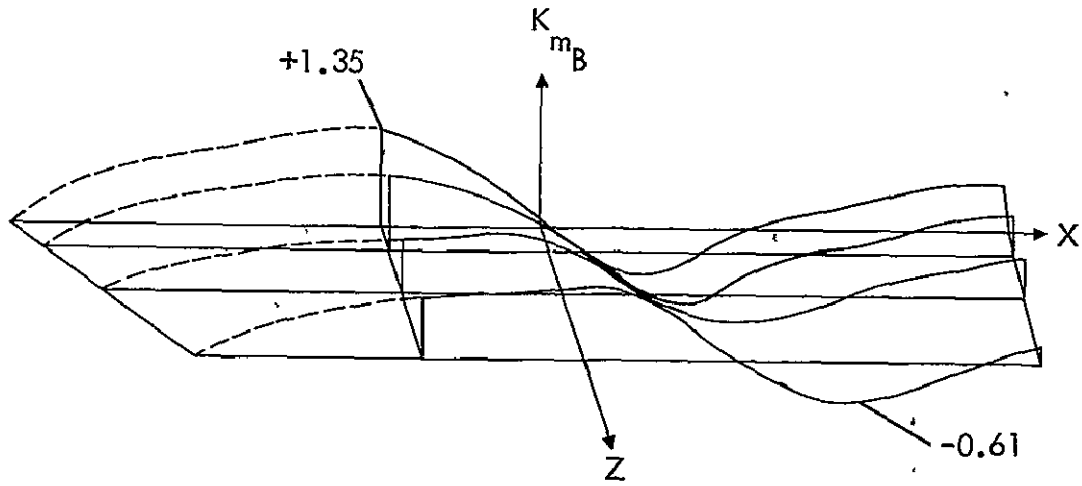
Although hard copy capability is directly available to the user of GGP at an interactive graphics terminal, nothing precludes him from also using any of the existing offline vector drawing hard copy devices (e.g. S-C4020 micro film recorder, Gerber or Calcomp paper-ink recorders). The interactive user would be assisted by GGP in using any of the many hard copy devices at his discretion.

5.1 Conventional Graphical Output

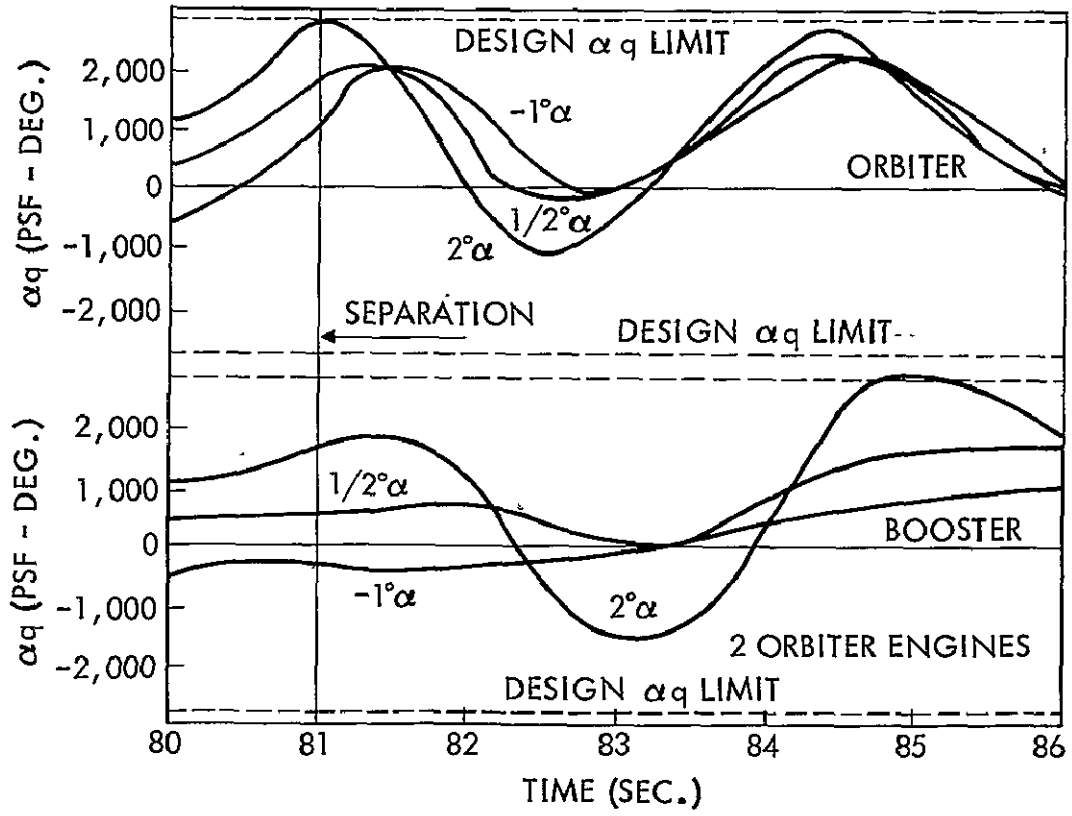
In the OM Questionnaire results, discussed in Section 4.5 of Part I of Volume IV, practically three quarters (76 percent) of the respondents indicated that they could use a generalized graphics plotter to advantage. To further define the requirements for GGP, detailed discussions were conducted with key individuals to identify how GGP might be used and what it should do. This section presents a collection of ideas and applications obtained from talking with a representative cross-section of engineers about how the interactive graphics plotter could help them get their tasks done better and faster. It concludes with a list of requirements for GGP.

5.1.1 Graphing. - The ability to graphically portray analytical results of an OM is a fundamental requirement of GGP. Several applications are discussed next.

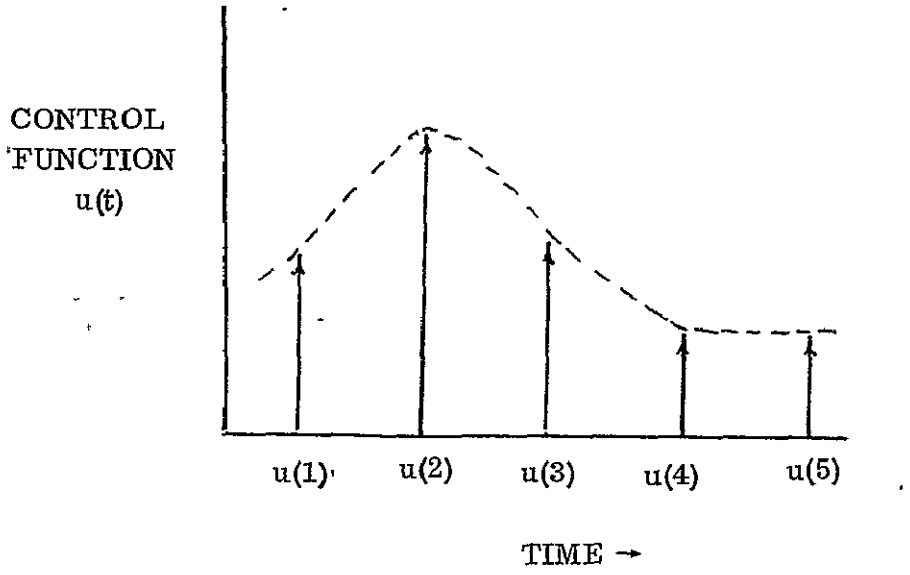
APPLICATION: Perform dispersion studies at the graphics terminal. The dispersion studies focus on small changes and correspond to fine changes in vehicle's performance. Many parameters are held fixed and only one or two parameters are varied at a time.



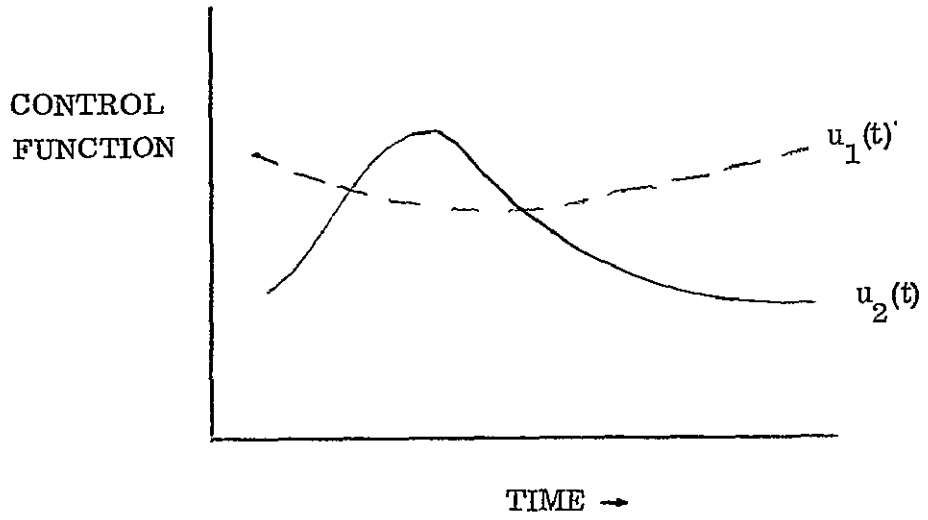
APPLICATION: From the graphics terminal, have the ability to select the desired parameters and have them plotted against each other.



APPLICATION: To give more visibility into the optimization of trajectories, show the control function u(t) as it varies with time. The control function may reflect the combined performance of ten or more of the major parameters.

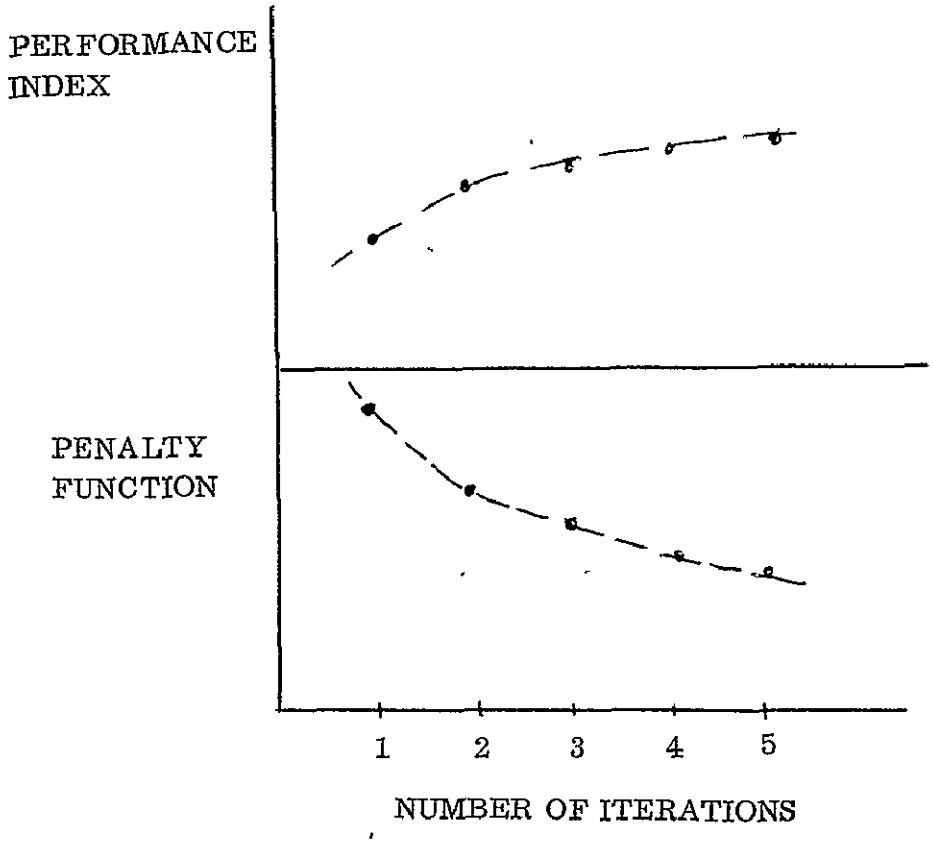


Compare an old control function $u_1(t)$ with a new control function $u_2(t)$.

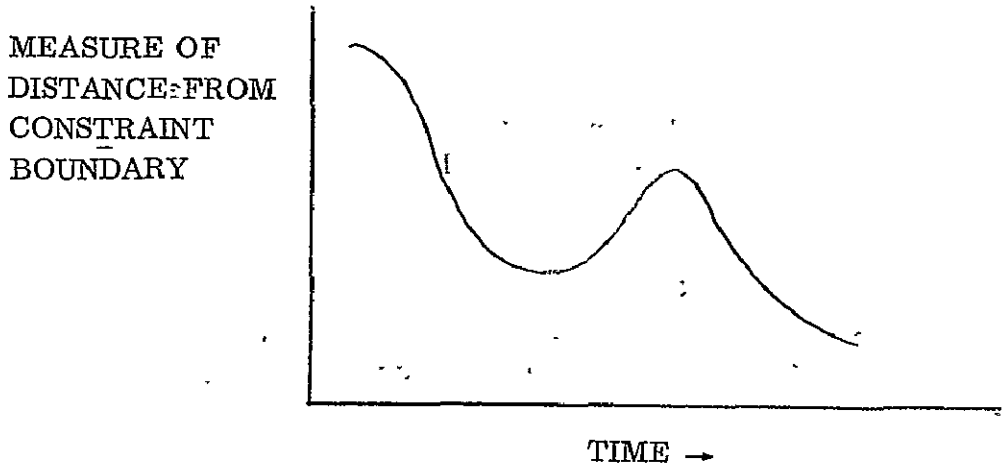


The gradient of the performance index gives an indication of how the control function should be changed to improve the optimization.

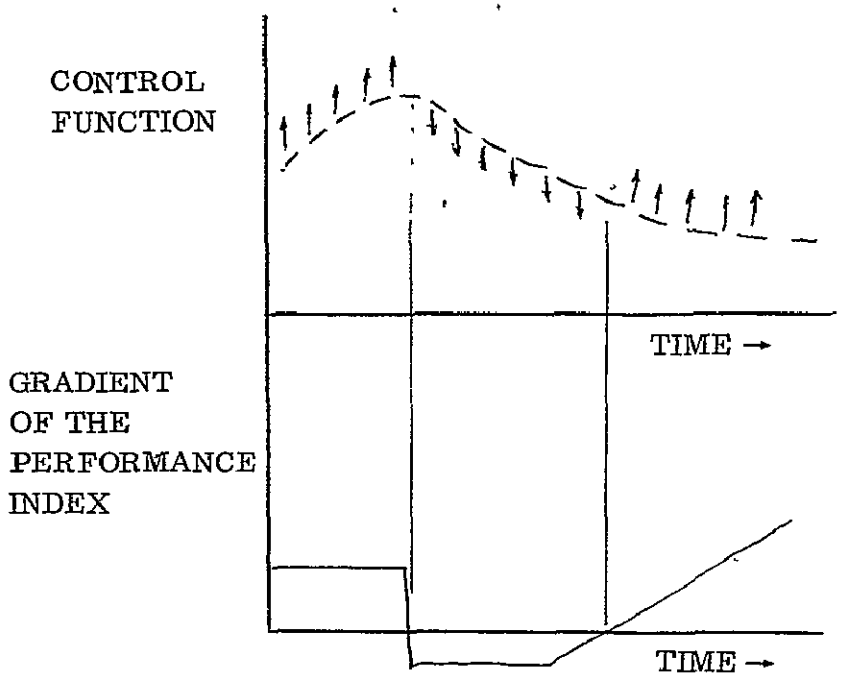
Compare the performance index with the penalty function as a function of iteration number so the convergence of the optimization can be followed.



Give visibility to the distance of a parameter from a constraint boundary as a function of time.



Compare simultaneously the gradient of the performance index with the control function.



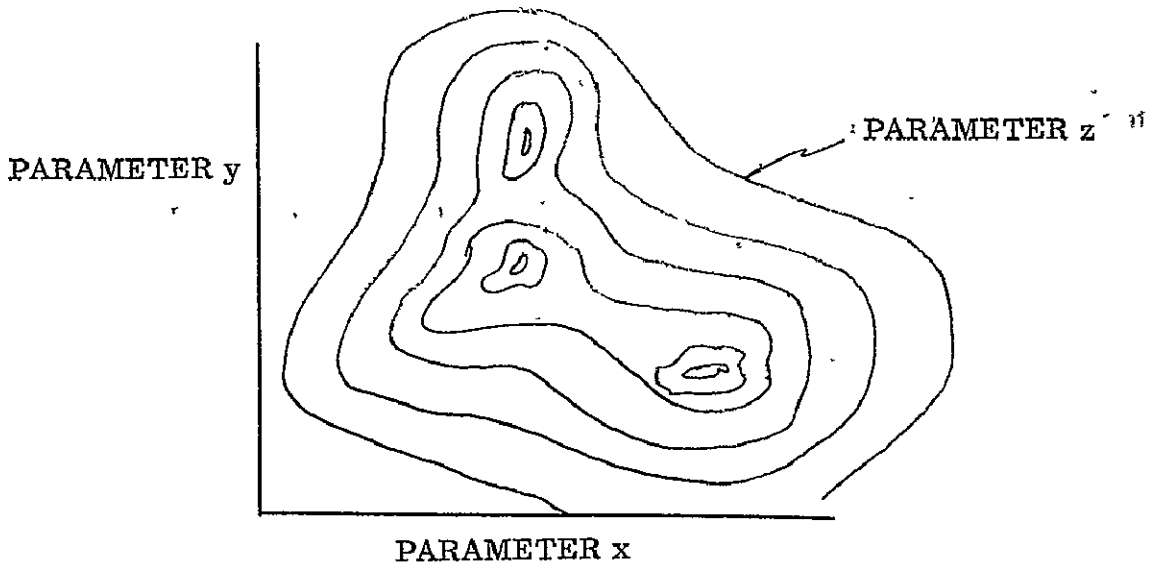
APPLICATION: Use the graphics terminal to compare previous vehicle performance with that of the vehicle under examination. In this way if performance results look dubious, a data base can be queried to call up similar results from past performance to see if the results in question fall within the expected range.

APPLICATION: Display the results of a booster/payload synthesis program. Be able to change a major parameter and immediately see the consequences in the booster's configuration and performance variables. Being able to examine such possibilities at a graphics terminal is highly desirable when many configurations need to be examined quickly.

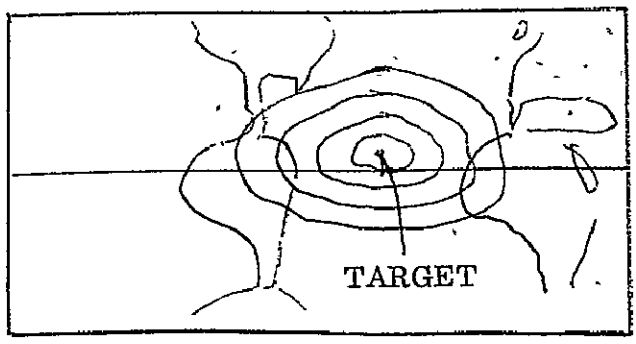
APPLICATION: Sketch the desired configuration at the graphics tube and insert the dimensions and parametric relationships for preliminary design analysis. By varying one major parameter, the resulting new configuration is displayed with new weights, dimensions, etc.

5.1.2 Contour plotting. - Although essentially graphical, contour plotting is a special case of grapning and often displays the results on a pictorial background. Several applications are discussed in the following paragraphs.

APPLICATION: Show constant performance lines for different performance or trajectory characteristics.



APPLICATION: With transmitting antenna in orbit pointing towards a point on the earth, superimpose on a Mercator projection map the constant power lines as seen at the earth's surface.



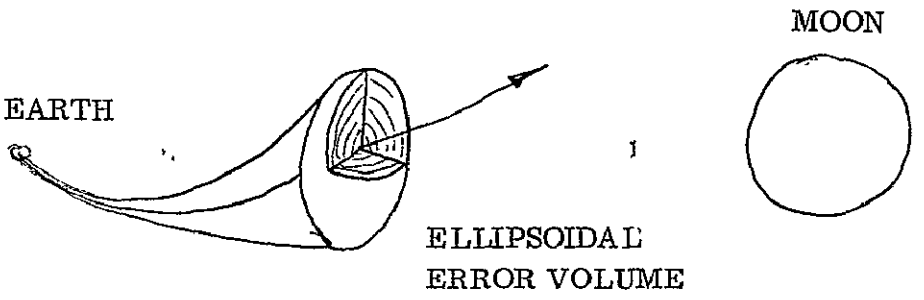
APPLICATION: The electronic engineer wants to see the side lobes of antennas in 3D. He also wants to see the pattern of a radar beam and FM sidebands.

APPLICATION: Show the thermal contours on a satellites surface. Accentuate the hot spots since they can cause instrumentation failures. By rotating the satellite about its axes, show the constant temperature lines under specific sun-radiator conditions.

APPLICATION: Problems can sometimes be solved in one domain but not in another. If the ability existed to show how simple curves map into the desired domain, it would give better understanding of the problem and its potential solutions.

5.1.3 Pictorial displaying. - The ability to convey information by drawing a picture is also a fundamental requirement of GGP.

APPLICATION: Show in 3D the guidance error buildups for lunar and planetary trajectories as an aid to concept visualization.

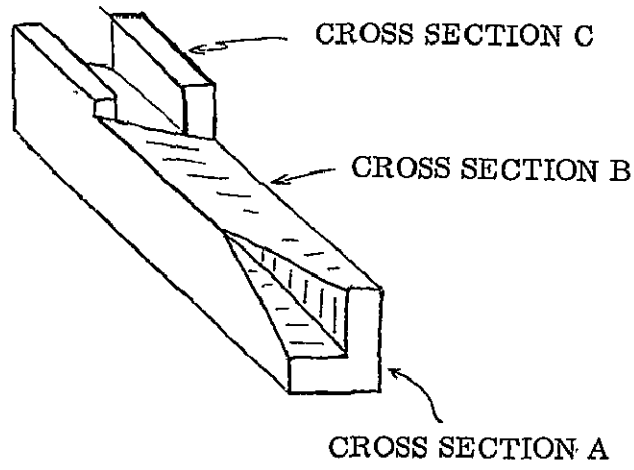


APPLICATION: Be able to do shading (the equivalent of smearing charcoal on paper to get different shades of black and gray) to make selected surfaces stand out in a 3D portrayal. This might be done by using dots or contour lines at various separation intervals.

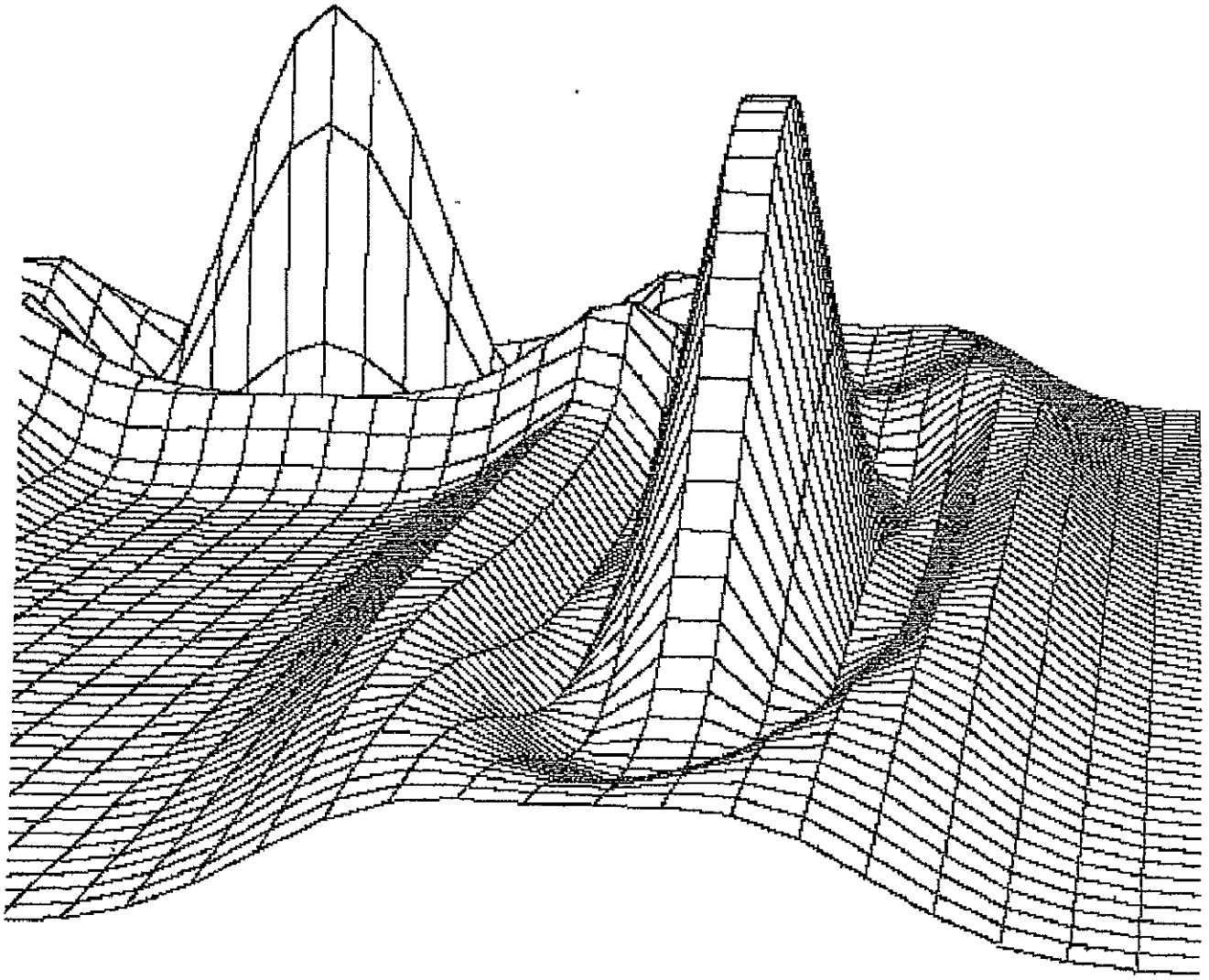
APPLICATION: Simulate what is visible from the pilots eye position to help with canopy design. This includes objects in the cockpit area, along the aircraft's surface, in the sky and on the ground.

APPLICATION: In n-space problems, hold n-3 variables fixed, vary the other 3 and show a 3D plot of the solution to an optimization problem.

APPLICATION: In designing longerons for an aircraft, the engineer computes the cross sectional areas needed at each station to support the expected loads. The loads vary at each station, therefore the cross-sectional areas vary. The designer has to fair these different cross-sectional areas into one another. Need the ability to do this fairing from the graphics terminal.



APPLICATION: In the visualization of complex 3D surfaces - for example in optimization analysis - be able to rotate the reference axes so that the surface is in the prop perspective. Be able to exaggerate the scale on one axis to make the display more visible.



APPLICATION: Provide perspective in the 3D portrayal to assist visualization. There are two kinds of 3 dimensional portrayal.

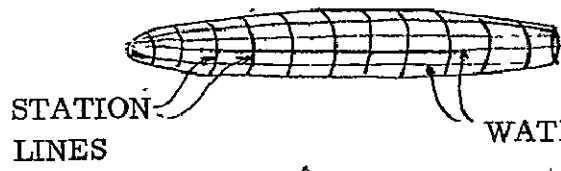
1. Geometrically accurate (isometric) for measurement purposes.
2. Perspective, non-orthogonal to aid in visualization.

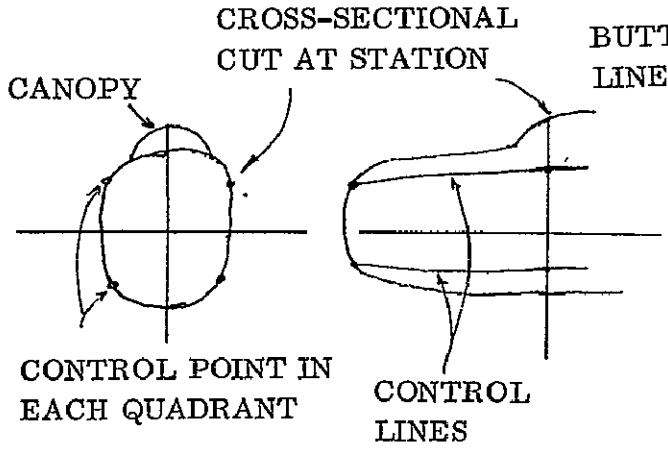
5.1.4 Configuration display. - Perhaps the most used application of GGP in an IPAD environment will be to provide configuration definition to the user at a graphics terminal. This is a special case of pictorial displays.

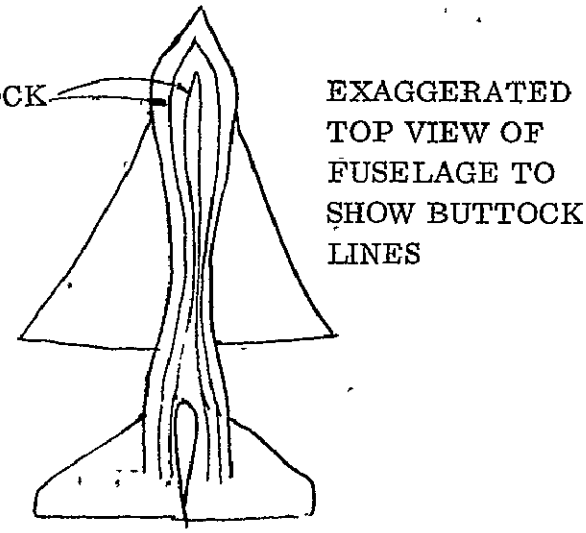
APPLICATION: On variable geometry aircraft, visibility is needed of the wing's surface contour. Be able to use station lines, water lines, buttock lines and control lines on visible surfaces to indicate 3D surface curvature.

3D LINES

STATION LINES
WATER LINES
BUTTOCK LINES
CONTROL LINES





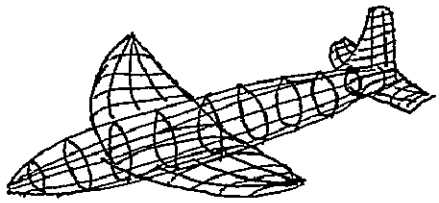


APPLICATION: Display the true-length of the structural elements, etc. at the cross-section of a cut through an aircraft's wing.

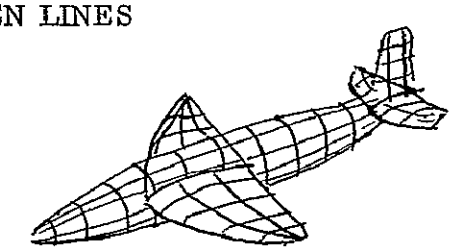
APPLICATION: Find the true-length of a structural member by identifying the member from the graphics terminal. Or, equivalently, find the clearance between two points on a structure identified from an interactive terminal.

APPLICATION: There are some cases where the analyst wants to see the hidden lines even though the figure being displayed becomes cluttered. This is particularly true in structural analysis where the appearance of the hidden line gives confidence that a member which is hidden from view is included in the picture. In other cases, where only a 3D portrayal is wanted, the fewer hidden lines showing, the easier it is for the observer to visualize what the object looks like. Since removing all hidden lines takes too much computer time, there are cases where 3D views can be most efficiently portrayed by leaving in some hidden lines.

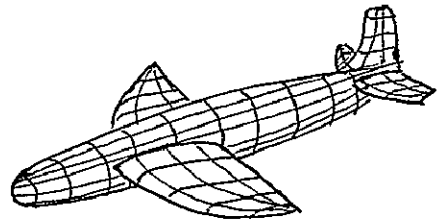
3D EFFECT WITH HIDDEN LINES



1. ALL HIDDEN LINES SHOWING



2. SOME HIDDEN LINES STILL SHOWING



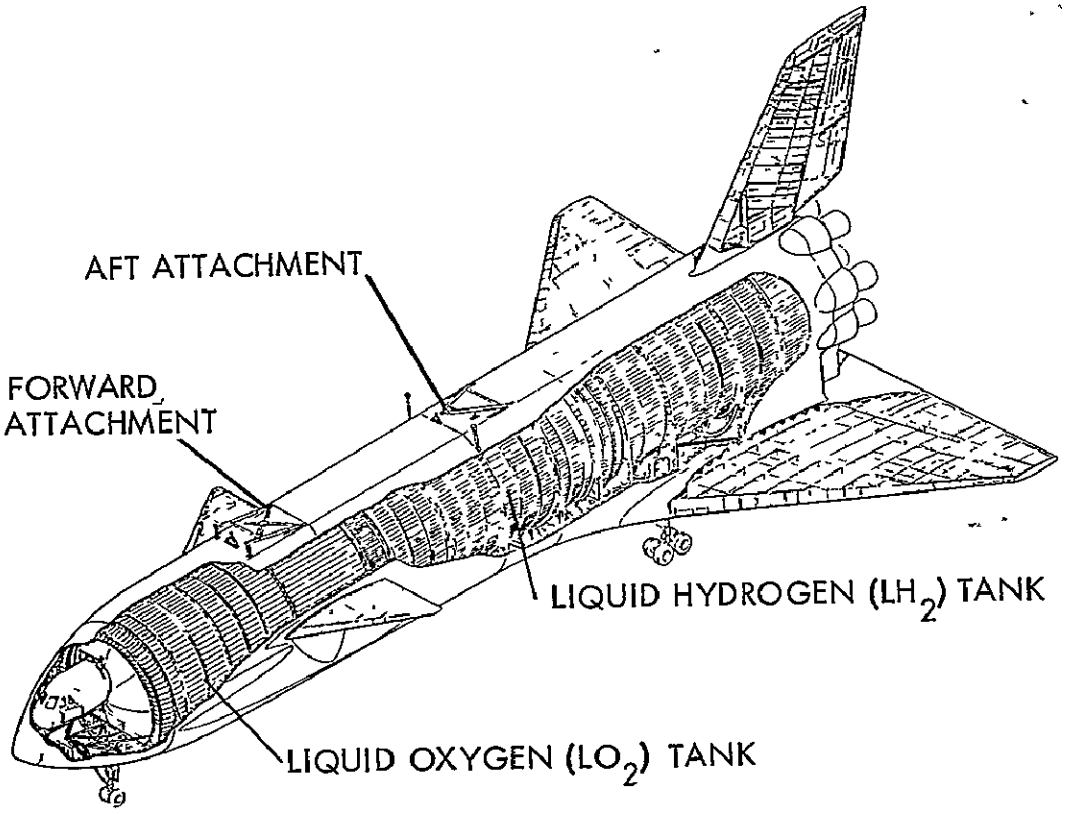
3. NO HIDDEN LINES SHOWING

USE VECTOR RESULTANTS TO FIND WHICH LINES ARE HIDDEN AND WHICH ARE VISIBLE. THE VECTORS ARE (A) THE LINE-OF-SIGHT FROM VIEWER TO THE POINT ON THE AIRCRAFT; (B) THE NORMAL TO THE SURFACE AT THE POINT OF EYE CONTACT.

APPLICATION: In preliminary design work, each contributor to the overall design wants to see how his portion is fitting in. With the ability to rotate a 3D model about its axes, the engine analyst, for example, could be given a view of the engine inlet which normally might be obstructed in a regular drawing.

APPLICATION: The designer needs to see where propellant lines go, where cross-feed takes place, and where best to put the valves.

APPLICATION: Display structural information from a data base. Have several levels of detail available. Be able to call for specific structural members. Be able to call for features like the cross-sectional area, moments of inertia, etc. Be able to call for the material properties of the member, such as its metallic composition, modulus of elasticity, etc.



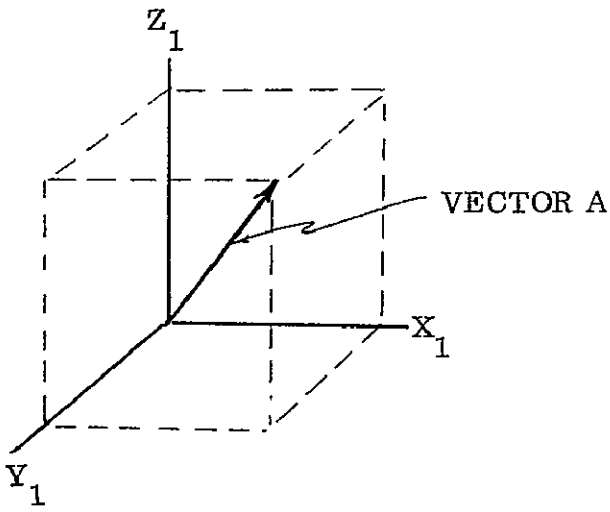
APPLICATION: Have the ability to reconfigure a baseline design to get a better understanding of the favorable design values for a parameter. For example, during the preliminary design investigations, when the wing of a cruise missile is made larger, it starts moving towards the rear until it over-runs the position occupied by the air-breathing engine. The designer needs to know just how big the wing can get before its size starts to impinge on the engine space.

APPLICATION: Be able to see the effect on the cross-sectional area rule by changing canopy shapes, fineness ratios, taper ratios, tail, sweep of the wing while sitting at the graphics terminal.

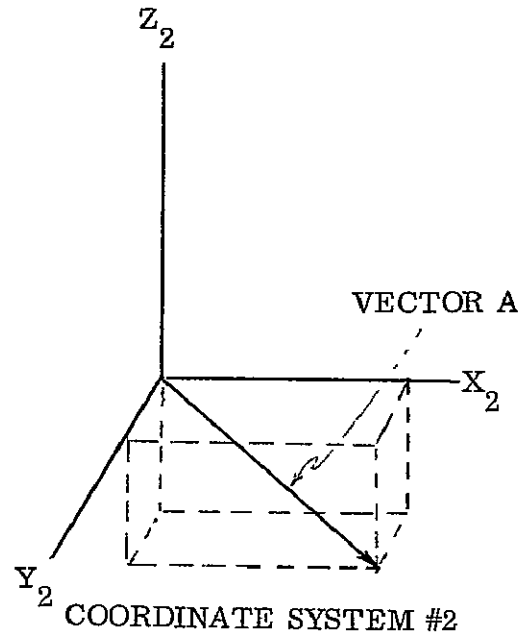
APPLICATION: Have the ability to zoom-in on a detail from a larger picture and examine the region in the vicinity of the detail.

5.1.5 - Coordinate system visualization. - This could be considered a special application of a pictorial display.

APPLICATION: Visibility is needed by the engineer to follow a vector starting with one set of components and passing through a multitude of coordinate transformations. As an example, consider the Centaur upper stage mounted on a Titan booster. The Centaur inertial guidance system controls the motion of the Titan engines during the boost phase. There are about 8 coordinate transformations in getting a control vector from the guidance computer to the engine gimbals. The engineer needs to understand and verify that each step is correctly made.



COORDINATE SYSTEM #1



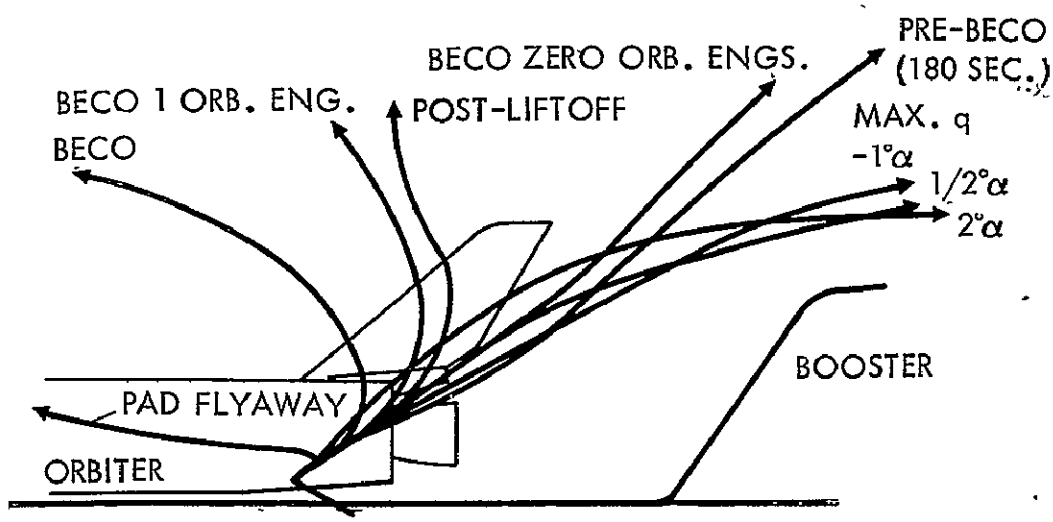
COORDINATE SYSTEM #2

APPLICATION: Same as above with two different coordinate systems but one coordinate system is rotated so that vector A appears once as a common vector.

APPLICATION: When an upper stage booster and its payload are moving unpowered through a coast phase, the pair must eventually be maneuvered prior to firing for the next powered flight portion of the trajectory. The vehicle must be rolled, yawed and pitched at various times so it will have the proper orientation when the engines are ignited. These maneuvers are hard for the analyst to visualize. A 3D portrayal will verify that the preset maneuvers are correct. If vector representation of the orbital planes, vehicle axes, etc., are shown, it is possible to verify that the vehicle starts the maneuvers with the correct orientation and ends up with the correct new orientation.

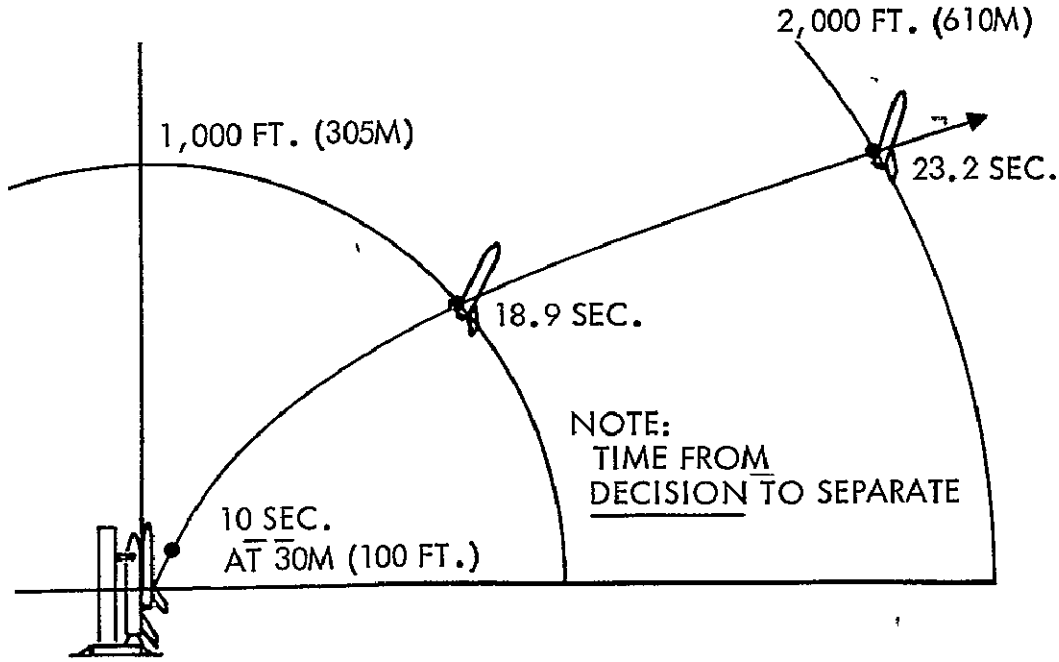
5.1.6 Clearance presentation. - Clearance presentations between objects generally combine pictorial information with either clearance callouts or graphical displays as well.

APPLICATION: Blast impingement and clearance when space shuttle orbiter separates from a winged booster. Include callout details as shown.



*BECO = BOOST ENGINE CUT OFF

APPLICATION: Illustrate the clearance obtained as a function of time.



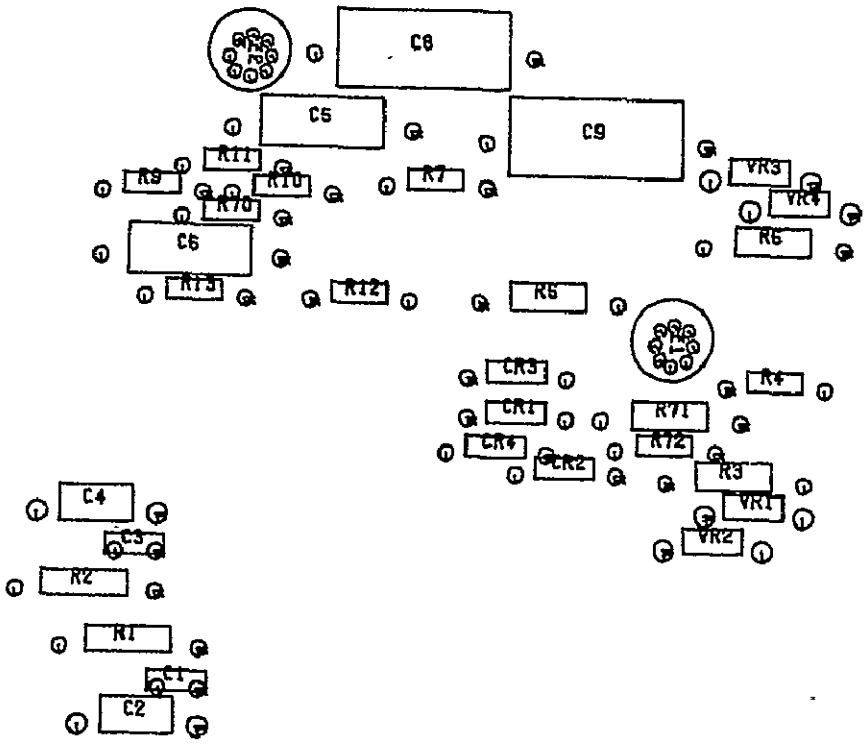
APPLICATION: Visibility is needed when a booster leaves its supports on the launch pad. Animation in 3D can help to determine the best time to start the initial roll steering commands. There is always an uncertainty as to the best time to start initial maneuvering. Starting too soon can produce collisions; starting too late wastes energy.

APPLICATION: Usually when a booster separates from its payload, it is important that no engine blast particles impinge on the payload. Need to have animation in 3D showing the payload separating from its booster. Need to be able to rotate about various axes to get different sepatation views.

APPLICATION: Use the 3D portrayal to simulate a pilot landing his aircraft on an aircraft carrier. Use different sea states and wind velocities.

5.1.7 Packaging and routing. - The packaging problem presumes the ability to control the placement of items within the display. Routing additionally presumes to control the interconnection of these items, sometimes to the extent of specifing length and route-path curvature.

APPLICATION: The 2D transfer of components in a printed circuit board packaging design. (The sketch is hard copy from one such design program.)



APPLICATION: In a space shuttle design with twelve engines, it was a requirement to keep the propellant feed lines of equal length. The large amount of plumbing that had to occupy the same space made it almost impossible to visualize whether or not adequate clearance was available for everything as well as to ensure that the feed lines were of the same length. A 3D portrayal with the ability to rotate the field of view to examine the crowded regions would have been extremely helpful.

APPLICATION: On satellites, need to package instruments to keep the center of mass near the middle of the satellite, yet have unobstructed viewing angles for all the instruments. There is a need for 3D visualization on placement of the instrument packages to help meet all of the constraints.

APPLICATION: In following conduit or piping through an aircraft's wing box, the ability is needed to rotate the wing box at various angles to see if the piping can be inserted in the wing box without hitting other objects or being blocked by some object.

APPLICATION: In building an aircraft wing, visibility is needed of the available space so actuators, hydraulic lines, electrical lines, etc. can be placed without interference. To have 3D rotational capability would make this possible.

APPLICATION: The designer needs the ability to switch instruments, crew stations, plumbing, etc. from one place to another within a volume envelope while still maintaining the center of gravity near the physical center of the volume. The designer would identify which package is to be moved and where it is to be moved.

APPLICATION: To simplify the volume-packaging problem, it is convenient to create 3 or more spheres or cubes of different sizes that can be assembled in various ways. The packaging of people and/or equipment within each solid can be handled separately.

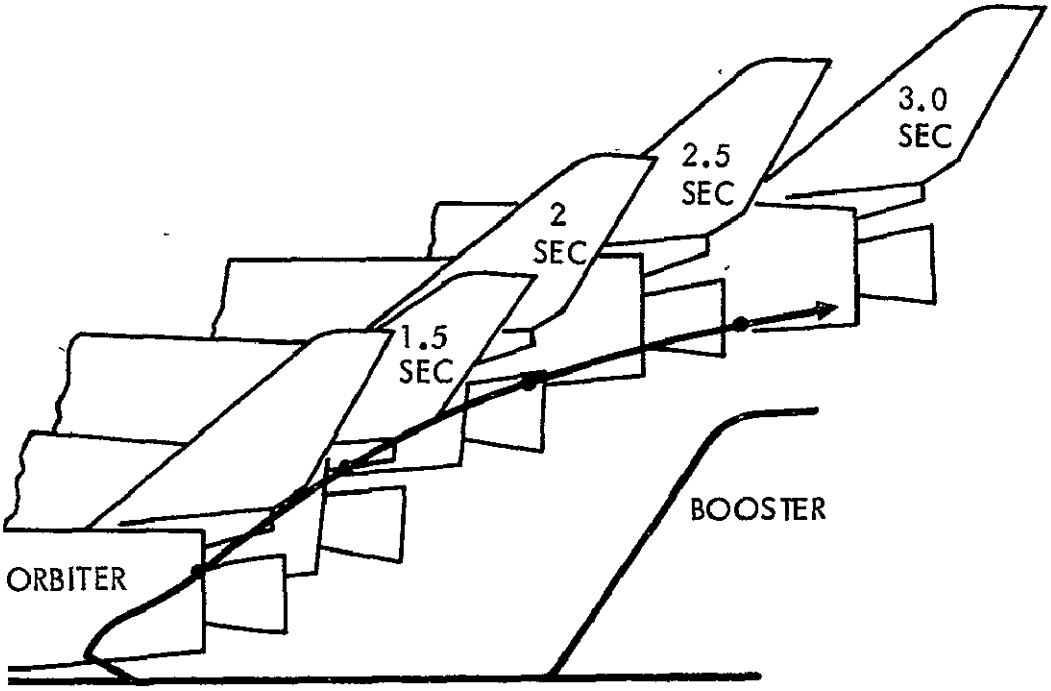
APPLICATION: Be able to prepare flowcharts.

5.1.8 Animation. - Animation is an invaluable aid in visualization, combining visualization and time (i.e. sequenced visualization).

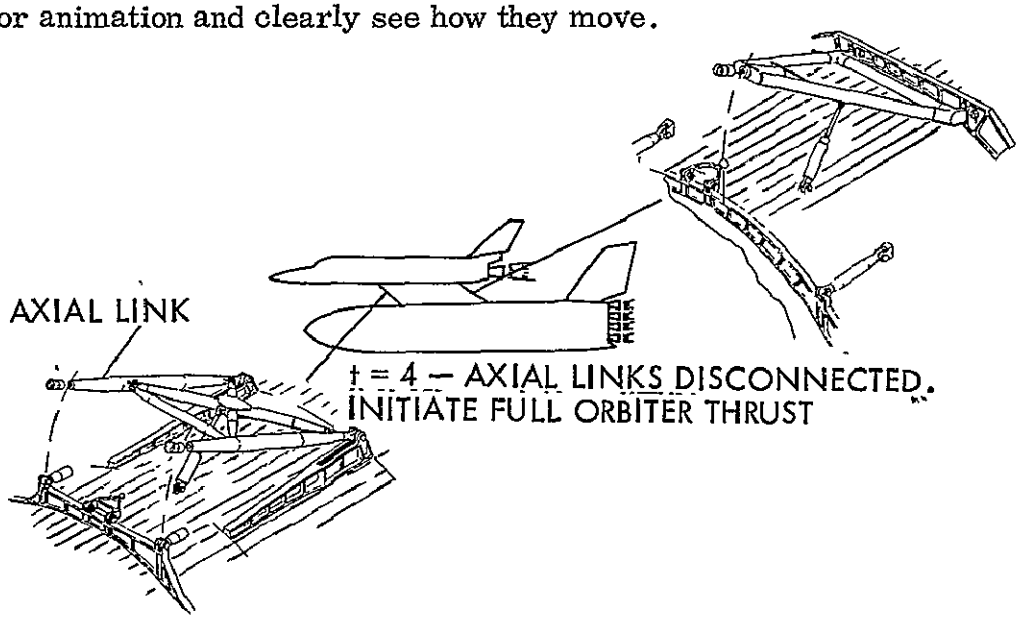
APPLICATION: Need to give visibility to a conceptual idea. For example, show how the cruise missile comes out of its launch tube, how its wings unfold, how the mechanisms move. The complex motion of mechanism need to be viewed; it is too difficult to visualize whether the linkages are moving the way they are expected to move.

APPLICATION: Four-bar linkage problems are hard to visualize but with animation they become most understandable.

APPLICATION: Hard copy pictures from a graphics terminal can be used to show sequences of activities for launching a vehicle, separation of payload and booster, extension of wings, antennas, etc. Microfilm (16 mm) can also be made into animation movies when properly sequenced.

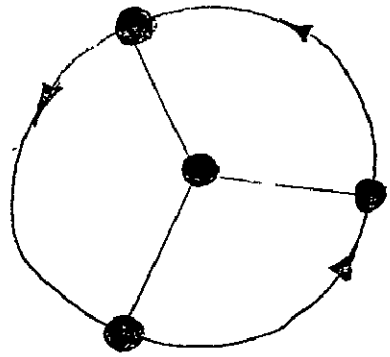


APPLICATION: Store in a data base the equivalent of a handbook of linkage mechanisms and their animated motions. The engineer can call for this "linkage mechanisms" package and review them for ideas. For the linkages of interest, he can call for animation and clearly see how they move.



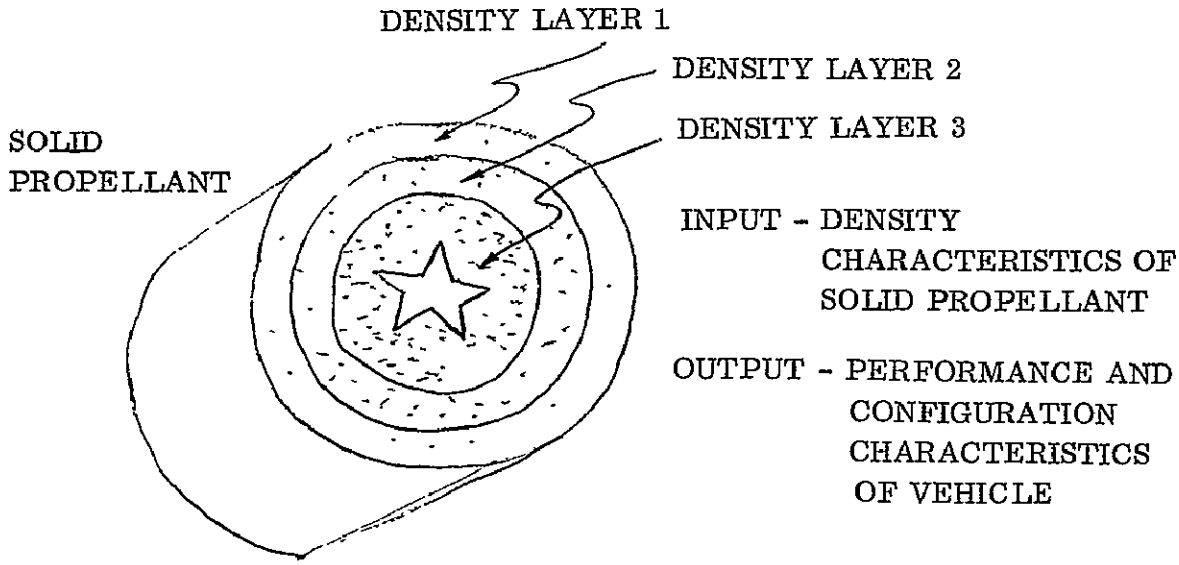
5.1.9 Special applications. - This subsection is a collection of various applications illustrating use of one or more of the above capabilities in a specific role.

APPLICATION: Need animation to show how a constellation of four satellites are placed to form a revolving Y, where the outer three satellites retain a Y-configuration and appear to revolve around a central satellite.

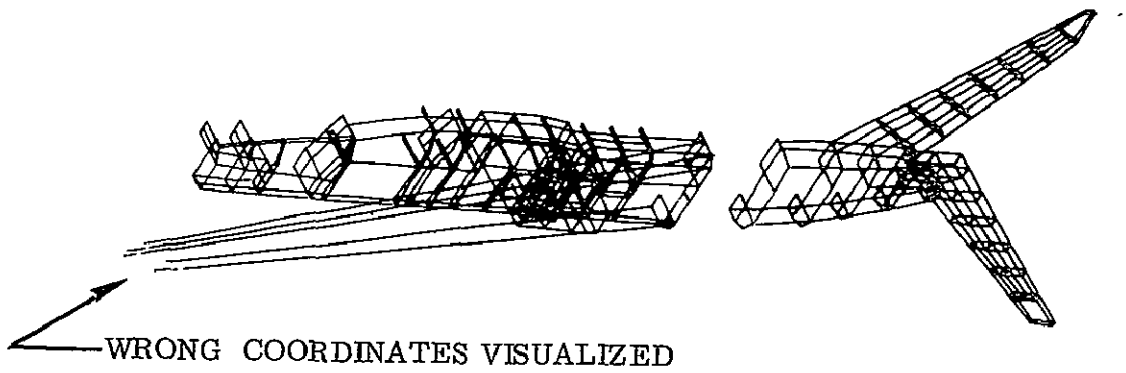


Revolving Y Satellite Configuration as Seen from a Point on Earth.

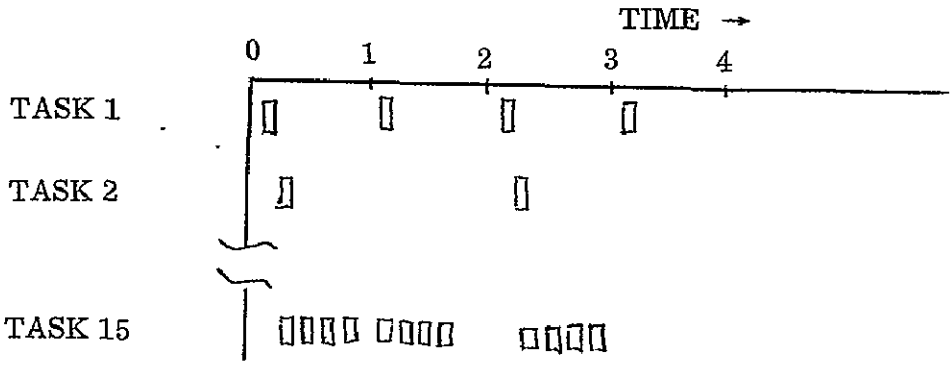
APPLICATION: When vehicle performance and design are tied closely together, use graphics for quick turn-around and visibility of consequences of the design change or performance change. For example, take the case of designing a solid propellant. When the density of the propellant layers are altered, different burning characteristics emerge and consequently different thrust/weight characteristics appear.



APPLICATION: Need pictorial 3D display of a vehicle model to verify that the model is correct. For example, a card-deck composed of x, y, z points represents the vehicle's surface. If any of the x, y, z coordinates are wrong, the surface is wrong. A direct visual display could quickly point out the errors.

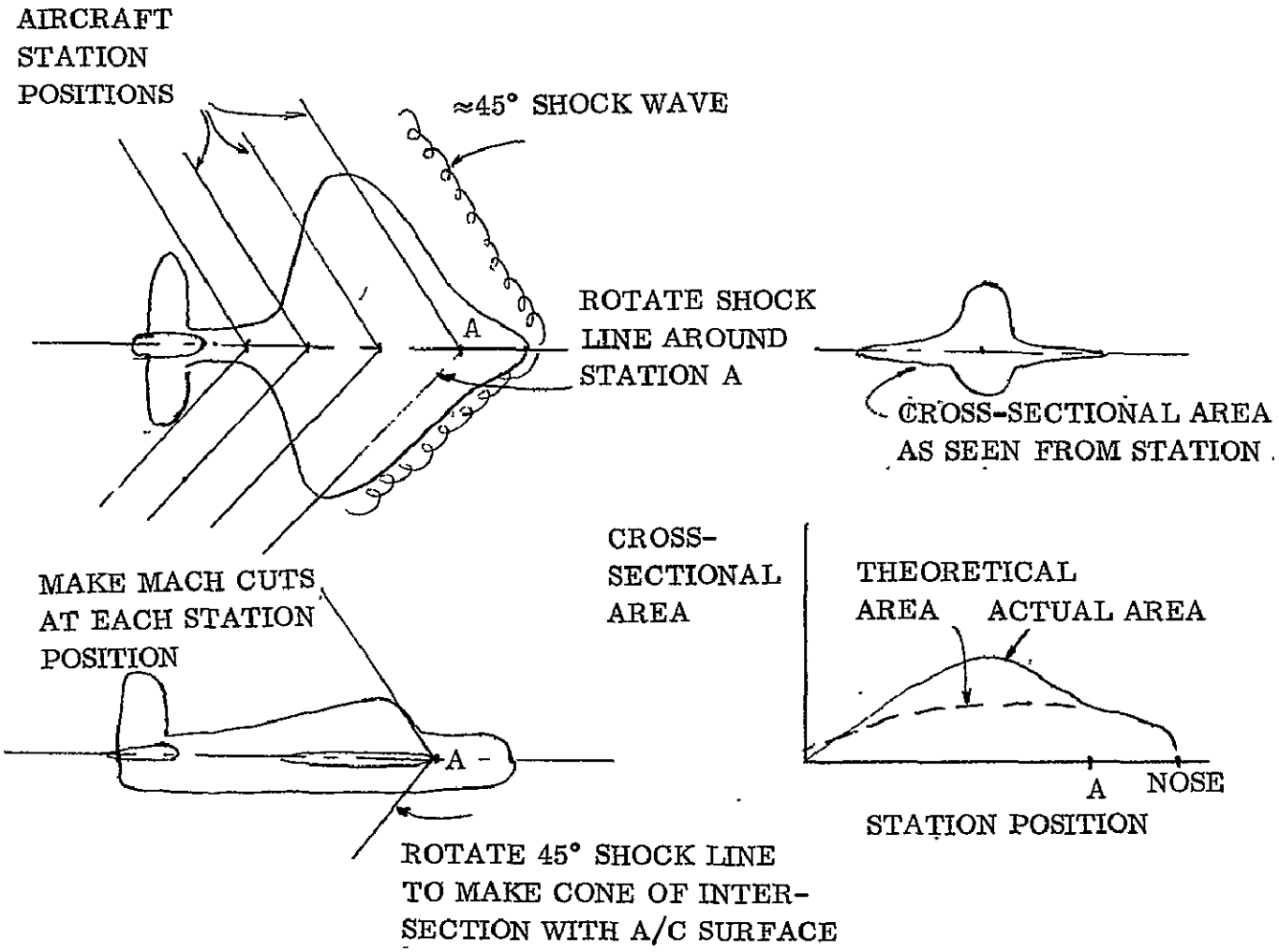


APPLICATION: When multiple tasks are being performed in an airborne computer, it is necessary to know where each task stands. A graph that shows how the tasks are progressing would be helpful.



APPLICATION: Need visualization to make certain that shadowing between antennas and solar panels is not occuring, that viewing angles are clear and unobstructed, and that shielding from radiation is adequate.

APPLICATION: Compare the cross-sectional area from Mach cuts with theoretical area-rule. A Mach cut is the cross-sectional area of an aircraft as seen by lines parallel to the shock wave cutting through the surface of the aircraft. These cross-sectional cuts change for different Mach numbers and angles of attack.



It is necessary to compare the actual cross-sectional areas from the Mach cuts with the theoretical areas. The closer the actual areas matches the theoretical values the better the aircraft will perform at higher Mach numbers.

APPLICATION: Show frequency response curves.

APPLICATION: Show the load distributions graphically over the length of a structural member or of the complete structure. Have an outline of the structure showing important station lines. Above this outline have a load distribution curve.

APPLICATION: Need visibility of complicated rib truss loadings. Need to be able to apply loads and see what deflections take place. Since the deflections are small, they need to be amplified for visibility. These deflections can provide the clues for improving the rib truss design.

5.1.10 Requirements for a General Graphics Plotter. - The following summary was obtained from the canvass and presents results not explicitly displayed above.

5.1.10.1 Graphical plotting requirements, general:

1. Cartesian coordinates.
2. Rectangular, polar, cylindrical and spherical coordinates.
3. Logarithmic and semi-logarithmic coordinates with up to 5-cycle log scales.
4. Histogram coordinates.
5. Bar charts and pie charts.

5.1.10.2 Pictorial plotting requirements, general:

1. Have three different 2D views visible at the same time.
2. Starting with a baseline design, be able to change a parameter while at the graphics terminal and display the resulting configuration and performance characteristics.
3. Plot contours (isotherms, isobars, constant power lines, constant altitude lines, etc.) on a background map or on a surface.
4. Provide 3D isometric display of an object.
5. Be able to translate, in 3D, an object along one of its principal axes from terminal input.
6. Be able to rotate an object about one of its principal axes from commands initiated at the terminal.
7. Be able to display a true-length view of a dimension or the clearance between two points. This involves several rotations for proper viewing.
8. In 3D representation, use station lines, water lines, buttock lines and control lines on visible surfaces to indicate surface curvature.

9. Be able to sequence 3D pictures to create animation. Let sequence of pictures be controlled by either time or the value of a selected parameter.

10. Be able to display two dynamical 3D bodies interacting with one another at different viewing angles. For example, a payload separating from its booster, where both may be tumbling.

11. Be able to display in 3D two separate and different rectangular coordinate systems, each showing components of the same vector.

12. Be able to superimpose two rectangular coordinate systems with the origin of the coordinate systems located at the same point so they can describe the same vector. This means one coordinate system must be rotated with respect to the other.

13. Be able to show an ellipsoidal error representation in 3D where the semi-major and semi-minor axes are the standard deviation of the errors.

14. Be able to display the interaction between the ellipsoidal error envelope moving along the nominal trajectory and another body travelling on an impact or near-miss course.

15. Be able to relocate 3D packages from one stowed position to another within a volume envelope from terminal input. Have computer account for the constraints imposed on the location of these packages.

5.1.10.3 Special capabilities:

1. Zoom-in display capability (less area, more detail).

2. Zoom-back display capability (more area, less detail).

3. Need ability to display exaggerated motions (e.g., deflections) to make the motions more visible.

4. Be able to create objects of various dimensions in 3D and permit the designer to arrange them arbitrarily. This helps him attack the volume-packaging problem.

5. Need ability to display a window view of objects, e.g. objects seen from the pilot's eye position. This includes objects in cockpit area, along the aircraft's surface, in the sky and on the ground.

6. Display schematics for propellant lines, wiring, etc. Provide routing capability.

7. Display 3D information, graphs, extending mechanisms, etc. called from a reference data base.

8. Need ability to fair one 3D shape into another. (Like carving a block of wood).

9. Need ability to shade a surface (equivalent of smearing charcoal on paper). Shading might be done with dots or lines or both.

10. Show how a straight line or other feature appears when transformed into another domain, e.g. conformal mapping.

11. Perspective presentation in 3D to assist visualization.

12. In optimization problems, need 3D portrayal of global slopes and specific local slopes. For N-dimensional problems, hold N-3 variables fixed and display the other 3.

5.1.11 Concluding Remarks. - During the canvass, much enthusiasm was displayed by the various engineers when they found out how an interactive graphics terminal could help them with their tasks. The possibilities of using the terminal for some applications had never occurred to many before. In general, they were elated at the prospect of being able to eliminate the drudgery associated with design. Included in the drudgery are the repetitive calculations of bending moments, moments of inertia, areas of wetted surfaces, volumes, etc. whenever a single design change is made.

One of the biggest potential time savers the graphic terminal can provide is to reduce the turnaround time in converting small design changes into new performance characteristics. In preliminary design, where many parameters are changed - usually one at a time - with a subsequent performance reevaluation for each change, much time is consumed. With a graphics terminal to accelerate this iterative cycle, the time can be reduced by an order or magnitude in some cases. More importantly, especially for the predesigner whose time is invariably limited, he can examine more basic vehicle configurations for the same amount of elapsed time. He can spend his time on innovations rather than on iteration drudgery.

Another powerful use of the interactive graphics terminal that emerged from the canvass is the ability to provide 3D visual animation so the designer can display his conceptual ideas. What may be clear in his head cannot be readily transmitted to his fellow workers, his supervision, the project management or the customer. But with the interactive graphics tool he can make his ideas abundantly clear. This is another verification of the old adage that "A picture is worth a thousand words."

5.2 Graphical Output with Topological Input

There are a class of problems for which the primary method of describing the problem formulation is by a (generally two-dimensional) topological diagram. Figure 5-1 presents and identifies typical examples taken from engineering design and analysis. Although each engineering discipline has specialized names and attributes associated with the identifiable elements of the diagram, the diagram's intent is general and simply to lend visibility to a problem formulation in a pictorial form. Not only are these topological diagrams convenient forms for problem formulation and communication with the analyst, but they are also easily interpreted by the computer. In the OM Questionnaire results discussed in Volume IV, Section 4.5 of Part I, 39 percent of the respondents indicated that they had a requirement for topological input manipulation.

The following subsections develop the requirement for topological input manipulation (TIM).

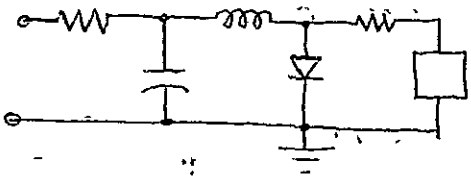
5.2.1 Fundamental requirements for TIM. - The fundamental information to be embodied in the topological diagram are the identification of:

1. Primitives - the fundamental building blocks or "symbols" of the topological diagram.
2. Connectivity - the ways in which the primitives are interconnected (including, of course, the allowable ways).
3. Attributes - the additional descriptives associated with the primitive (besides its identification) and its connectivity.

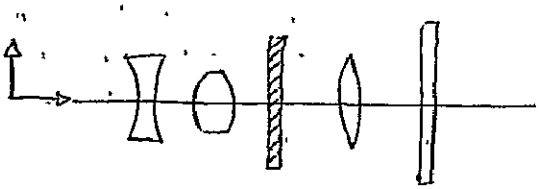
The following subsections identify these three components for some of the symbols in the examples presented in Figure 5-1.

5.2.1.1 Electrical schematics: Electrical schematics of the type presented in Figure 5-1a are usually two-dimensional schematics composed of passive and active elements (primitives), e.g., resistors () capacitors (), inductors (), diodes () and even integrated circuits (). Other symbols (also primitives) denote the state of the circuit at various points in the network, e.g., an open node () or a circuit ground ().

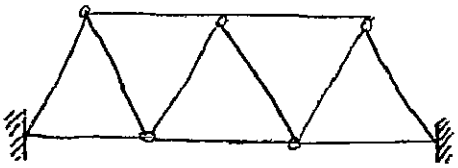
A resistor primitive () has two identical connections - one at each end - called nodes. In theory any number of connections can be made at either node; at least one at each node is required. Generally three attributes are associated with the resistor:



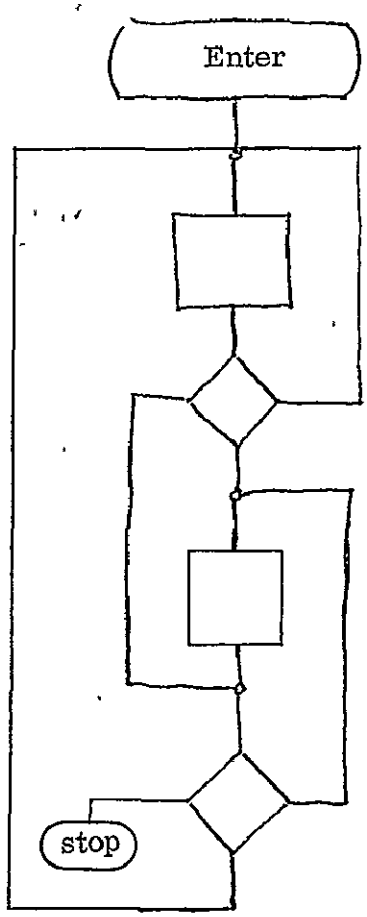
a. Electrical Schematic



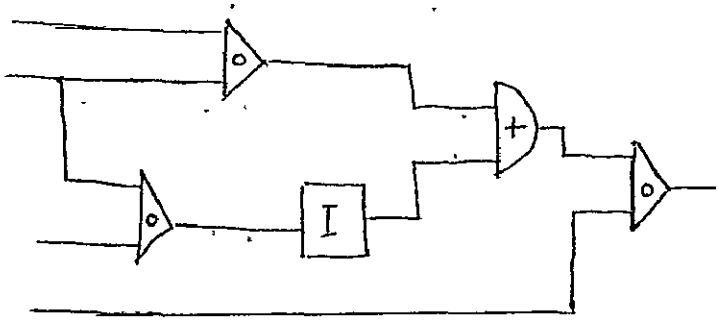
b. Optical Schematic



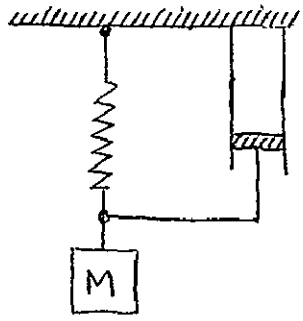
c. Structural Schematic



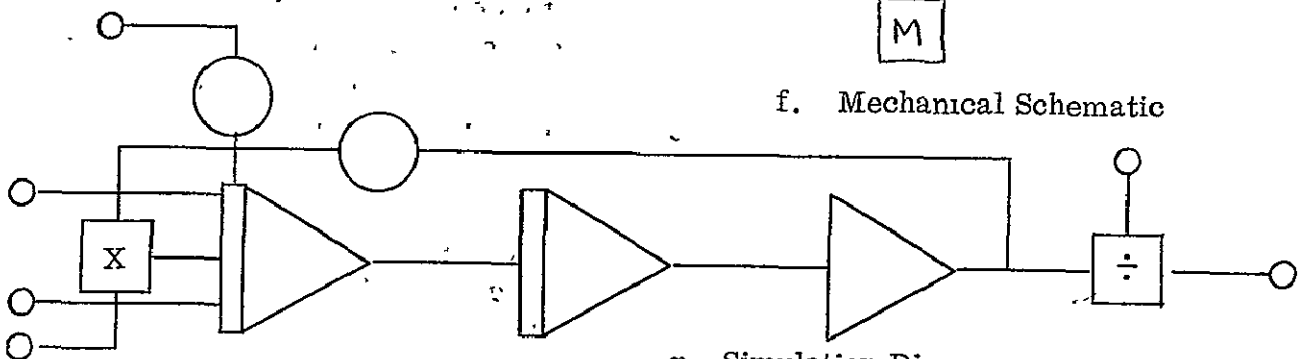
e. FORTRAN Flow Diagram



d. Logical Schematic



f. Mechanical Schematic



g. Simulation Diagram

Figure 5-1. Examples of Problem Formulation Schematics

1. Resistance - expressed in ohms.
2. Rating - expressed in watts.
3. Accuracy - expressed in percent of resistance.

Other attributes can be associated with a resistor if desired (e.g., capacitance, inductance, leakage, temperature coefficient) but these are of lesser importance.

A capacitor primitive (⊣⊢) has two non-identical connections (in general), one for each voltage polarity. Like the resistor, any number of connections can be made at either node with at least one required. The principal attributes are:

1. Capacitance - expressed in farads.
2. Rating - expressed in volts.
3. Accuracy - expressed in percent of capacitance.

Like the resistor, capacitors can also possess resistance, inductance and leakage attributes. Leakage is often included among the principal attributes. The primitive ⊣⊢ is used to represent a capacitor with two identical connections (non-polarized).

A ground (⏚) has only one connection and generally no attributes. (Resistance, capacitance, etc., however, could be associated with a "low quality" ground.) Any number of connections can be made to this node.

An integrated circuit (-□-) can have any number of non-identical connections (at least one is required) and can be configured in many different ways (⋈ , ⊐□-, ⊐□_) as suits the analyst. Any number of attributes can be associated with an integrated circuit.

For high frequency applications, the lead lengths of nodal connections become important and must be included in the attributes associated with connectivity.

5.2.1.2 Optical schematics: Optical schematics are usually two-dimensional schematics composed of lenses, filters, prisms, and refraction gratings arranged along an optical axis (Figure 5-1b).

Lenses may be concave (⌶), convex (O), or composites (⌶O) treated as a single lens. They typically have focal length (positive or negative) and aperture as their attributes. Depending on the requirements, the attributes can be expanded to include the lens material or those attributes providing the material properties (e.g., refraction index).

Filters ( ) typically have associated with them a function of attenuation (percent) versus wavelength (angstroms).

The "connectivity" of optical schematics is the placement of the primitive along the optical axis and includes their relative separation (distance). Placement lateral to the optical axis (offset) can also be taken into account.

The capability for ray tracing specifications, although peculiar to this application, could be contained with a special primitive, i.e., a RAY.

5.2.1.3 Structural schematics: Structural schematics are usually three-dimensional schematics composed of such elements (primitives) as bars, beams, panels, linear springs, clock springs, dampers and the like. Simplistic examples are presented in Figures 5-1c and 5-1f).

Connectivity of structural primitives are specified by their ability to react moments, shear or both. For example, a pinned connection can react only shear, a clock spring primitive only moments, and a clamped connection can react both. This gives rise to type of connectivity as well as the connection; connectivity type, however, is an attribute of the connection. Some elements can be connected in several ways, e.g., the end of a beam (Figure 5-1c) can be clamped, pinned or free (unconnected). Some elements can only have one connection attribute, e.g., both ends of a linear spring (, Figure 5-1f) must be pinned.

The attributes associated with structural elements are typically such things as spring rate or stiffness (springs and beams), mass (mass nodes or distributed mass elements, e.g., tapered beams) and structural damping coefficient. Beam stiffness is frequently composed of still other parameters.

5.2.1.4 Computer diagrams: Computer diagrams come in two types. There are flow diagrams for procedural code, e.g., FORTRAN as illustrated in Figure 5-1e. There are also representative diagrams embodying the intended solution, as the simulation diagram in Figure 5-1g.

FORTRAN flow diagrams (Figure 5-1e) are two-dimensional diagrams composed of interconnected functional blocks (e.g., , ,) and nodes ( ). A node may have any number of connections whereas the functional blocks may have only one connection per connect point. (If the connection is to a node, the effect is to have many connections, however these connections are identified to the node.)

Every element except the node has input connections and output connections. An input connection may only be connected to an output and vice versa. The node has

only the single multiple connection. An entry element (ENTER) has only an output connection and a return (RETURN) or exit (STOP) has only an input connection. By convention, all input connections are located at the top of the element.

A node (o) has only one attribute, a statement number (e.g., 100). In general, functional blocks have an arbitrary amount of information (the computer source code statements) associated with the block. Decision blocks (◇) come in two types: two-way branches (-◇-), as with logical IF tests, and three-way branches (-◇-) as with arithmetic IF tests.

Simulation diagrams (Figure 5-1g) are an outgrowth of analog computer circuit diagrams. There are a variety of computer programs which use such two-dimensional diagrams as the basis for problem formulation (see Appendix B, Subsection B.1.1, Simulation Programs).

Simulation elements (primitives) typically have a fixed number of inputs and unlimited outputs. (This was not formerly true for a potentiometer (-○-) of an analog circuit due to resistive loading; however this is no longer a restriction for its replacement, which merely represents a constant multiplier.) Besides conventional input, some element have special input, e.g., the "initial condition" input of an "integrator" (▷) and the divisor input of the divider (÷).

The attributes associated with a simulation element are typically associated with its connections, e.g., multiplicative "gain" including sign. The element itself may have associated with it a non-unity "gain".

5.2.2 Functional requirements. - The analyst requires the following functional capabilities:

1. View any element (primitive) associated with the problem formulation technique, and:

   a. Examine inputs and their requirements.

   b. Examine outputs.

   c. Examine connectivity conditions.

   d. Examine attributes to be (normally) supplied.

   The interfacing software should contain tutorials assisting in this examination process.

2. Create and place an element or move (or remove) an element already existing and placed within the diagram. When moved, all connectivity

should automatically move with the element. When creating an element, the element number should be automatically assigned.

3. Establish connectivity or alter existing connectivity. The routing of the connective lines (if required) should be automatically accomplished but alterable by the analyst.

4. Establish, add to, take from or change attributes associated with an element or its inputs. This includes the capability to display any attribute for inspection. (Attributes should not be displayed unless requested, to obviate excessive clutter.)

5. Reorient the entire diagram (or that portion of the diagram being viewed) on the display. For three-dimensional diagrams, the analyst must be able to reorient the display to obtain better views.

In addition to these, the analyst also must have the capability to:

6. Specify collections of interconnected primitives (subsets of the diagram) to be automatically reproduced and added to the diagram. Element numbers should automatically be assigned and the relative placement among the elements should be preserved.

7. Selectively alter the names, shape, attributes and connections of primitives, and thereby establish new primitives. (For example, a new Integrated Circuit could be created from an existing one.)

8. Zoom-in on a display (less area, more detail) and zoom-back on a display (more area, less detail).

9. Window a display (i.e., looking at a portion of a large diagram as if it were contained behind the CRT but such that only that portion directly behind the CRT is viewable).

10. Have three different planar (2D) projections of any portion of a 3D diagram viewable at the same time, as an aid to visualization.

11. Superimpose a grid structure (2D or 3D as applicable) to enhance visualization and to aid the user in "regularizing" the schematic on uniform grid lines.

5.2.3 The analysis OM. - The intent of describing a problem formulation by a topological diagram is to ultimately provide the required input to an analysis OM. Examples are circuit analyzers (e.g., SCEPTRE), simulation network precompilers (e.g., MIDAS IV and CSMP), structural analyzers (e.g., NASTRAN), control system analyzers (e.g, CSAP), and printed circuit board (PCB) packaging programs. (See Appendix B for examples of these and other engineering applications deriving input principally from

topological diagrams.) Once constructed, the diagram with its established primitives, connectivity and attributes supply all that is required for the analysis OM.

This information is stored in the data base and is accessible by the OM through:

1. Its SUBSCHEMA directly.
2. By mapping selected information to another portion of the UF via:
   a. A QPS.
   b. A DBMS callable function module (code).
   c. A special interfacing program.

It is assumed that the analysis OM (or OMs) exists for each application of the topological diagram under discussion and that the required information (primitives, connectivity and attributes) has been provided in the data base. The analyst need not be aware that his actual interface is a GPU; it will appear that he is providing input directly to an OM tailored to his requirements.

5.3 Design Synthesis

It should be noted that pictorial plotting and topological input manipulation (TIM) share many features in common. Note that TIM must provide pictorial displaying as must the graphics plotter, e.g.:

1. Display items obtained from the data base.
2. Enable the translation and rotation of displayed items.
3. Provide for different projected views of 3D displays.

Conversely, two general graphics plotter requirements specified in Subsection 5.1.10.3 are actually topological input manipulation, viz.:

4. Be able to create objects of various dimensions in 3D and permit the designer to arrange them arbitrarily. This helps him attack the volume packaging problem. (Item 4 of Subsection 5.1.10.3).
5. Display schematics for propellant lines, wiring, etc. Provide routing capability (Item 6 of Subsection 5.1.10.3).

Thus topological input manipulation and the pictorial plotting capability of the graphics plotter are essentially inseparable. This is why they have been combined as the General Graphics Plotter (GGP) GPU.

5.3.1 . Data base implications. - The data base support to graphical or pictorial plotting is trivial. DATA AGGREGATEs (usually vectors) are contained in the data base in a form specified by the SCHEMA module representing the user's UF. Also contained in that SCHEMA module are the names by which these DATA AGGREGATEs are known to the user. By selecting an output AREA of his UF, the user may examine the names of the various output variables and plot (crossplot) selected variables or visualize pictorial displays.

The data base support to TIM is more complex in that provision must be made to provide the topological schematic as well as display it. This in turn requires that the SET relationships be established during execution time via appropriate DML. The procedure is outlined as follows. The reader is referred to Tables 5-1 and 5-2 for reference throughout this discussion.

TABLE 5-1. DATA BASE SUPPORT TO TIM: RECORD TYPES

PRIMITIVE

ID (e.g., ground, wire, resistor, node, etc.)
DISPLAY (prefabricated sequence of display commands per menu)
TUTORIAL (attributes, terminals, etc.)

ORIENTATION

LOCATION (coordinates)
ROTATION, SCALING } to be applied to prefabricated display sequences

ATTRIBUTES

N
ATTRIBUTE (i), i = 1, N

TERMINALS

N
CONNECTION (i), i = 1, N (location relative to orientation, DATA-BASE-KEY (DBK) of the other end (TERMINAL))

ELEMENT

ID (e.g., element number)
DBK of PRIMITIVE

OPERATION

ID
DBK of PRIMITIVE (optional)

DIAGRAM

ID

TABLE 5-2. DATA BASE SUPPORT TO TIM: SET TYPES

COMPONENT

OWNER = ELEMENT
MEMBER = ORIENTATION

CIRCUIT

OWNER = OPERATION
MEMBER = ORIENTATION
MEMBER(s) = ELEMENT(s)

SPECIFICATIONS

OWNER = ORIENTATION
MEMBER = ATTRIBUTES
MEMBER = TERMINALs

NETWORK

OWNER = DIAGRAM
MEMBER(s) = ELEMENT(s), OPERATION(s)

During initialization of the UF, the user identifies the primitives for his application and maps occurrences of primitive type RECORDs into a menu AREA defined for the UF. These RECORD occurrences contain an ID field, reference pointers to interface with tutorial aids, and prefabricated sequences of commands to direct the graphics support software to display the corresponding symbols in a menu.

In order to incorporate a symbol from the menu into a diagram, the user picks the desired symbol from the menu and indicates its orientation in the diagram portion of the screen. TIM then generates a new occurrence of an "element" type RECORD and an "orientation" type RECORD. The "element" RECORD contains an ID field (element number) and a reference to the primitive RECORD (which contains display commands). The "orientation" RECORD defines modifications to apply to the display commands in order to display the indicated use of the primitive. Since this is all the information required to display the use of a primitive (assuming that connecting lines are also defined as primitives) it is reasonable to define a "component" SET consisting of the "element" RECORD as OWNER and the "orientation" RECORD as the only MEMBER. TIM can process occurrences of this SET which represents a minimum of data required to generate and manipulate the diagram.

The OWNER-MEMBER relationship means that each occurrence of the "element" type RECORD establishes an occurrence of the "component" type SET. DBMS maintains the association of the "orientation" RECORD of each "element" RECORD.

If a diagram is built up from primitives, each use of each primitive is a separate display item. The number of distinct display items that the graphics support software can handle will be limited. Also, in a complex diagram, the same primitives must be used in the same way a number of times. Consequently it is necessary to group several occurrences of the "component" SET into one entity or display item which can be manipulated as a unit. To accomplish this, the DDL might define yet another RECORD type exactly like the "element" RECORD but with a different name, such as "operation". This would be the OWNER of a second SET type, named "circuit" for instance. This SET would contain MEMBERs "orientation" and "element" (an arbitrary number of occurrences of this MEMBER). The command sequences associated with a number of primitives are passed to the graphics software as one sequence, thus the entire SET "circuit" is one display item. The "orientation" RECORD of this SET specifies transformation parameters to be added to the initial orientation of each primitive. That is, the primitive is oriented with respect to the circuit, then with respect to the orientation of the circuit.

When a circuit is to be duplicated, TIM generates a new occurrence of the "operation" type RECORD, thus establishing a new occurrence of the "circuit" type SET, and generates the associated "orientation" type RECORD MEMBER. Then, for each "element" type RECORD MEMBER in the first "circuit" SET, TIM generates an "element" type RECORD MEMBER in the second "circuit" SET. Each of these "element" occurrences establishes an occurrence of the "component" type SET, and TIM generates (copies) the associated "orientation" type RECORDs for the new components.

Note that the lumping together of display command sequences to make one display item is reversible. Given a "circuit" as a display item, the data base representation still consists of references to the command sequences at the primitive level. Consequently the user can modify individual occurrences of a basic "circuit".

In order to have an overall definition of the display as a group of display items - i.e., to specify which components are pickable items and which are buried within "circuits" - a network type SET is defined. The OWNER RECORD, named "diagram" for instance, consists only of an ID field. The MEMBER RECORDs are "elements" (OWNERs of "components") and "operations" (OWNERs of "circuits"). TIM inserts MEMBERs into this SET whenever a menu item is picked and oriented or whenever a circuit is to be duplicated. It deletes MEMBERs (of the "element" type) whenever the user defines a circuit, and it inserts a MEMBER (of the "operation" type).

This completes the data base requirements to generate and manipulate a schematic diagram. The display is defined in terms of nested SETs. The overall network is a SET of "circuits" and "components", and a "circuit" is a SET of "components". Either the individual "components" or "circuits" may be (pickable) display items.

The SCHEMA description of the UF would contain the definition of a fourth SET type called "specifications". The OWNER of this SET is "orientation" and MEMBERs are "attributes" and "terminals". Since "orientation" is a logically required MEMBER of the "component" type SET and the "circuit" type SET, DBMS will establish a "specications" type SET whenever a "component" or "circuit" is oriented in the display, and DBMS will maintain the association of each occurrence of "specifications" with a "component" or "circuit". A point to be made here is that this relationship will be maintained whether or not the SUBSCHEMA for TIM defines the "specifications" type SET. It is conceivable that this SET is irrelevant to TIM and need only be processed by the application OM.

This discussion is summarized in Tables 5-1 and 5-2 which present the RECORD and SET types supporting TIM.

5.3.2 Graphical display implications. - In order to design a generalized graphics plotter, it is necessary to define what constitutes a generalized plot or picture.

A generalized picture is always (at least in this context) a two dimensional entity, that is, it exists on a two dimensional surface - a piece of paper, microfilm, a cathode-ray tube, etc. Immediately, one sees there is a big difference between the physical object - a three dimensional entity - and a picture of the object. It is possible to made the picture "look three dimensional" with such techniques as perspective, shading, etc., but the picture is still two dimensional. In fact, mathematically one can be interested in N-dimensional entities - hypersurfaces - but in order to "picture" them a two dimensional display is made. What has just been said is that in the final analysis there is at least one transformation (mapping) necessary to "picture" a physical object or an N-dimensional mathematical entity. This transformation is usually straightforward and easy to define, e.g., projection or cross section. Moreover the transformation once selected can generate "pictures" knowing only the definition of the object in its natural space.

Obviously a general definition of an object in its natural environment is needed. Rather than considering three dimensional items specifically, one can look at N-dimensional entities. Objects in N-space can be digitized and numerically approximated by establishing reference points (coordinate systems, origins, axes, measurement directions, conventions, etc.). The accuracy of this approximation is a function of the completeness of the digitizing. When this digitizing of the entity in N-space is finished, the approximation is considered to be the definition of the object. In its simplest elements, the model consists of points, connectivity, and/or textual information. (Really each text character could be considered as a predefined collection of lines but for simplicity here a character is considered as a primitive unit.)

The digitizing of the object produces points in N-space. Associated with each point identified is a connectivity relationship which defines how that point relates to all other points in the model. For example if two points are "connected" then in the picturing of the model which represents the entity in N-space there should be a line displayed. (Here the hidden line problem appears and the solution of which is not considered so important as to completely diminish the effectiveness of the picture.) These connectivity relationships when transformed to the two dimensional picture surface are what generate the picture.

Often times the picture of an N-dimensional item only takes on meaning when seen relative to some of the chosen reference points. This usually takes the form of axes, legends, labels, sign conventions, titles, or other aids. The presence of this textual information is specific and not arbitrary. Thus to completely define a general N-dimensional item one must have available in the definition all of the supporting alphanumerics required with the respective locations of each. Sometimes the text needed to understand the picture is not properly identified with the item but with a particular view of the item. This is not only acceptable but to be expected.

Now that the general definition of a single object in N-space is realized, the production of a generalized picture can continue. Occasionally a user may wish to see the "picture" of a single N-dimensional item. For example assume the object is a nodal (x,y,z) representation of an airplane fuselage and the user wishes to see pictures of various projections of the nodal model. He may even wish to translate, rotate, or zoom some of the pictures. In general, a user-defined picture is made up of a composite of various N-dimensional items. He may wish to view that same airplane fuselage with a grid superimposed on it, or contours reflecting lines of constant stress or pictures of any of hundreds of other supporting objects. Thus, in general, the juxtaposition of the pictures produced by the tranformations of several N-dimensional entities defines the generalized picture that the user wishes to see.

5.3.3 Design implementation. - The implementation of the design approach must meet the design objectives and furthermore be user oriented to encourage its interactive use. The user must be able to control the transformation process and produce the pictures he needs to do his job. GGP exists as a utility to be used by all disciplines. It facilitates the pictorial presentation of data. The data itself exists in the data base possibly as output from an OM or as the current description of the configuration under study. Thus the existence of the N-dimensional entity is assumed.

In addition to user peculiar items to picture, there exists supporting N-dimensional items. For example, the general representation of a grid with labeling, axes, and reference points can be defined. The various coordinate systems - cartesian, polar, spherical - all can be represented as points with assigned connectivity relations and

textual information. In a sense these types of entities represent overlays or meaningful pictures to a particular user. This library of user defined items can be expanded as more "standardized" picture presentations are desired.

The Generalized Graphics Plotter produces pictures in the infinite (at least infinite to maximum representable computer floating point word) two dimensional space herein called the viewable space (Figure 5-2). This space is like a drawing board or display screen which has no limit - horizontal or vertical. Any picture is finite since the data is finite and exists somewhere in the viewable space. The user first defines in what region in the viewable space the picture will appear and then defines the items which will be "pictured". It is this viewable space which is eventually mapped to the device being used. Here also the interactive user of GGP can actively control this transformation and present on the terminal that region of the viewable space desired.

The final act of mapping the subset of the viewable space to the hardware device itself is the most restrictive. Throughout the rest of the approach no hardware dependence or limitation need be mentioned. In this way, the various I/O devices can be interfaced to new OMs with the minimum of modification. Moreover once a picture is generated and saved in a protected region of the viewable space recall of past displays is trivial - no regeneration of complex pictures are needed.

The functional flowchart (Figure 5-3) of GGP describes the hierarchy of elements. Entities are N-dimensional items or objects. Pictures are regions in the viewable space which consists of images of mappings of some number of entities (possibly zero). Displays are mappings of pictures onto a physical I/O device.

A key factor in the usage of this utility is the completeness of the data base. GGP retrieves from the data base the entities that the user desires to "picture". Also in the data base are the often used overlays which can be superimposed over the "pictured entity" to give greater understanding, for example grids for graphing or station lines for reference points on a vehicle. Many entities have built-in connectivity relationships. For instance an array of two dimensional values (say TIME versus ALTITUDE for a series of time values) could have the natural relationship that pair i is connected to pair $i+1$ and pair $i-1$ only.

5.4 Operating Requirements

The user of GGP will generally have his own specific entities that he wishes to picture. He itemizes these and provides them to GGP for future usage. Along with these entities are the "library" entities - grids, axes, labels, station lines, etc. - that are used by a cross-section of IPAD users. From this set of entities the user can now construct pictures and displays.

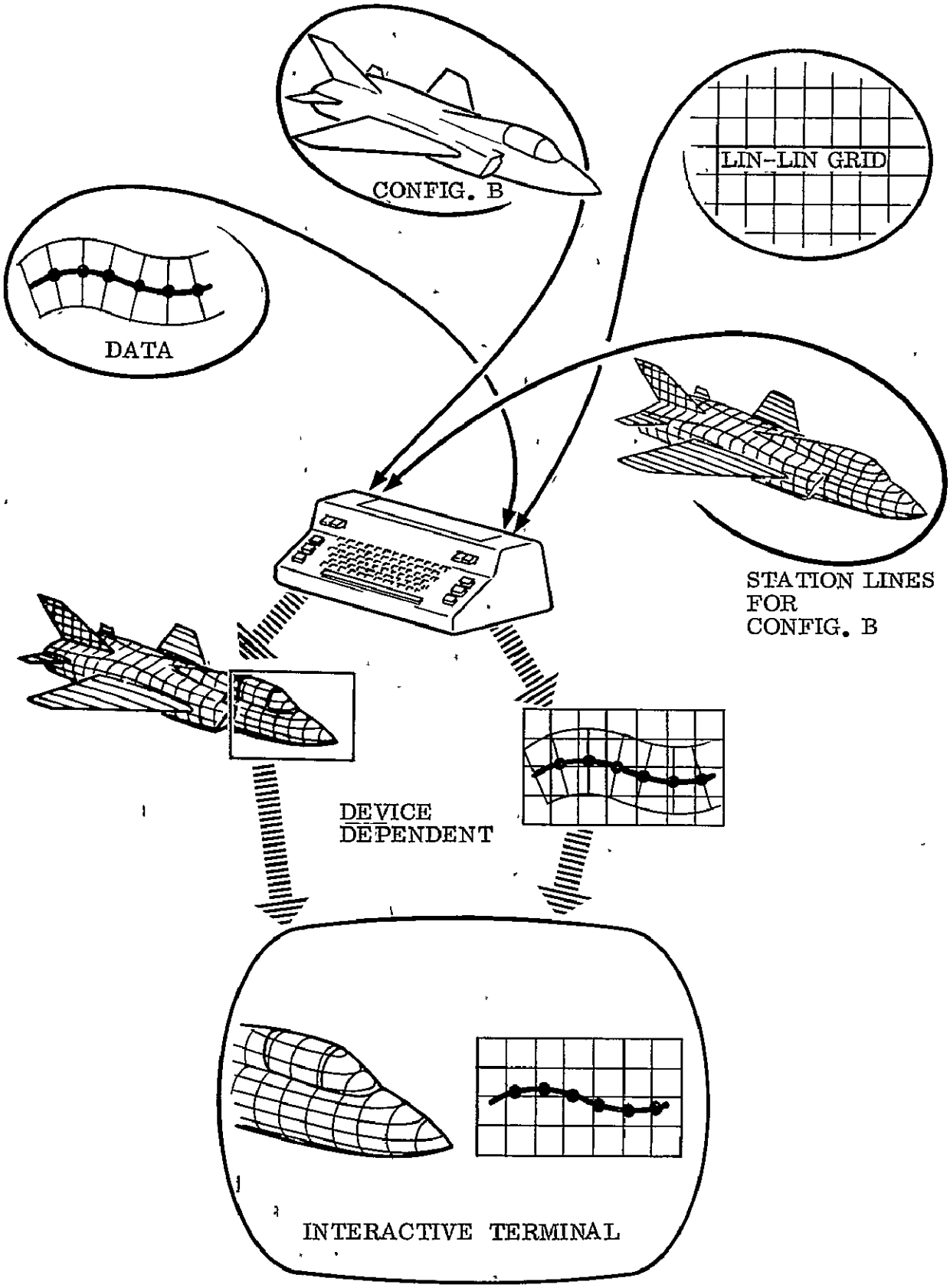


Figure 5-2. GGP's Viewable Space

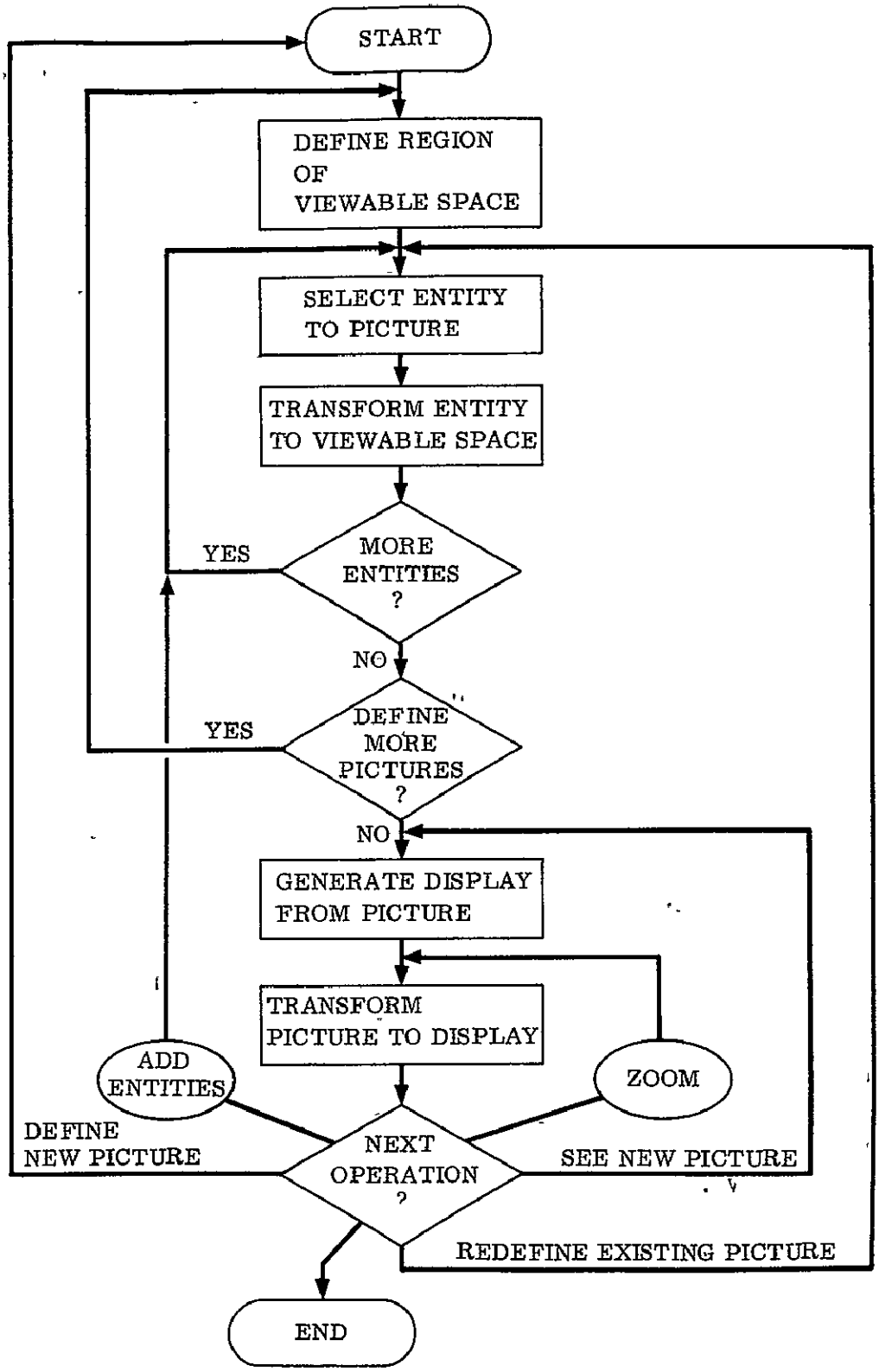


Figure 5-3. Functional Flowchart of GGP

The first step in defining a picture is to assign it a region of the viewable space. The user can do this himself or have an internal automatic scheme take over this assignment. This step is necessary to preserve (if desired) the integrity of past pictures and prevent the complete regeneration of previous images that are recalled for review. An example menu of this function is seen in Figure 5-4. Note that under the list of current pictures, the user changed the default name from frame three to BOOSTER.

Once the region in the viewable space is assigned, the user can begin to select entities to be "pictured" here. As each entity is chosen and validated the picture takes form. Eventually the user will have defined the totality of entities that makes up the picture. See Figure 5-5 for a typical menu the user might be confronted with. In the case depicted a series of plots are accessible with a variety of grids to select from. In the figure displayed, the user has indicated that he wishes to define a picture consisting of two entities - TIME versus MAXQ and CARTESIAN GRID.

As soon as the transformations of the selected entities are completed, a picture exists in the viewable space. The user now interactively defines what subset of that picture is to appear at his terminal. The user may construct a display made up of subsets of several different pictures. When the user directs GGP to generate the display, the device dependent routines peculiar to the terminal in use will be called to map the display to the I/O device. With these options, the user can selectively define what is to appear on the terminal screen from the viewable space.

Currently, prototype programs along these lines have been developed. One deals exclusively with the production of plots from files of user data (n-tuples). Another addresses the problem of "picturing" a physical object described with a nodal relationship, i.e., nodes, bars between nodes, etc. The object can be viewed in any orientation or rotated through user specified directions Both these programs run on a CDC 6000 series computer and use the CDC 274 interactive graphics console. They require only 17000 decimal words of core storage and provide moderate to good response in most cases. With this historical insight, it is believed that GGP can be written to reside in less than 20000 decimal words of incore storage. However, due to the generality introduced it is expected that response time will suffer. The initial construction of pictures in the viewable space will undoubtedly increase the delay. An important benefit however is that once a picture is created, a display of it or a subset of it is a much simpler operation. The hard copy capability is an immediate by-product since the chosen device is just another output unit. Moreover, movie making or animation is not a special task. In its simplest terms, a movie is nothing more than a series of individual pictures. As each of these pictures is routed to a film recorder and saved, a movie is made.

SELECT (TRACKING CROSS OR LIGHT PEN) OR TYPE IN CHARACTER OF YOUR SELECTION FROM MENU BELOW. IF DEFAULT VALUES (SPECIFIED WITH ASTERISKS) ARE DESIRED TO BE OVERWRITTEN, TYPE IN REPLACEMENT VALUE.

1. PICTURE REGION NUMBER *5*
2. PICTURE REGION NAME *FRAME 5*
3. LIST CURRENT PICTURES

PICTURE NUMBER	PICTURE NAME
1	FRAME 1
2	FRAME 2
3	BOOSTER
4	FRAME 4

C. CONTINUE TO NEXT MENU

T. TERMINATE

Figure 5-4. Define Picture Region

SELECT (TRACKING CROSS OR LIGHT PEN) OR TYPE IN CHARACTER(S) OF YOUR SELECTION FROM MENU BELOW IF DEFAULT VALUES (SPECIFIED WITH ASTERISKS) ARE TO BE OVERWRITTEN, TYPE IN REPLACEMENT VALUE.

1. BEGIN LIST OF AVAILABLE USER ENTITIES

   A. TIME VERSUS ALTITUDE
   B. TIME VERSUS RANGE
   C. TIME VERSUS MACH
   D. ALTITUDE VERSUS RANGE
   E. TIME VERSUS MAX Q

2. PAGE FORWARD USER LIST

3. PAGE BACKWARD USER LIST

1A. BEGIN LIST OF LIBRARY ENTITIES

   AA. CARTESIAN GRID
   BA. LOG-LOG GRID
   CA. LIN IN X - LOG IN Y GRID
   DA. LOG IN X - LIN IN Y GRID
   EA. POLAR GRID

2A. PAGE FORWARD LIBRARY LIST

3A. PAGE BACKWARD LIBRARY LIST

CURRENT PICTURE DEFINITION

P1. E -- TIME VERSUS MAX Q
P2. AA -- CARTESIAN GRID

C. CONTINUE TO NEXT MENU

T. TERMINATE

Figure 5-5. Construct Picture

GENERAL SURFACE (CURVE) FITTING, AN APPLICATION FOR OPTUM AND GGP

General curve or surface fitting is the task of reducing empirical data to a usable form. This task appears constantly in the work of an engineer who must analyze and understand large quantities of data. The data may be raw data, experimental data or precise, complex and voluminous mathematical calculations from a digital computer. The engineer's role is to correctly interpret the implications of this data; the capability of surface fitting in an interactive mode can greatly enhance this analysis process.

Historically, application programs have been written to assist a user in segments of this task. Often these programs were also discipline-peculiar and hence limited in scope. These approaches ignored the fact that the understanding and analyzing of data is a fundamental concern of engineers in all disciplines. The requirements of curve fitting are similar for all disciplines and can be thought of as a minimization of the sum of the squares of errors between the data being fitted and the proposed mathematical model. This minimization can be readily accomplished by the optimizer GPU (OPTUM) by utilizing unconstrained optimization methods, some of which are specialized methods for the most common types of curve fitting. (For example, polynomial curve fitting is very common because of its simplicity.) The computation involved in minimizing the sum of the squared errors can be carried out by using the necessary conditions for the optimization of a function. The resulting numerical computation involves the solution of a system of linear equations. The specialized optimization procedure can easily be incorporated into OPTUM as one of its component procedures. This is the approach taken to provide a curve (surface) fitting capability to IPAD.

6.1 The Data Analysis Process, An Overview

The utilities OPTUM and GGP include all the often used techniques encountered in data analysis and reduction. There are several distinct phases that may be used in a given data analysis operation. Each of these must be considered as a necessary segment in the overall task.

6.1.1 Data manipulation. - The first phase of the data analysis process can be considered manipulation and is a special application of the GGP utility (See Section 5.2). Here the user studies the data to gain an overall understanding for how it should best be analyzed. The user attempts to identify spurious input data that should be removed. He may also judgmentally add points to areas where data is sparse. In fact, he may completely transform the data mathematically (using a QP directive) to get a new set of data to work with. Another technique often used is segmentation or grouping of data

values together to be considered as a unit. Weighting of points is another option often employed by data analysts. All of these options are basically preliminary to the actual analysis process and are briefly treated in the subsections to follow.

6.1.1.1 Data transformations: The user often defines data transformations which might be advantageous as a prelude to curve fitting attempts. For example, consider the following situation.

A user has an array of (x, y) data values and, furthermore, he knows that the dependent variable is periodic and hence possibly it is a polynomial function of SIN (x). By computing SIN (x) for each x value and using the new array of points (SIN(x), y) as the basis for a polynomial least squares curve fit, the solution for $a_0, a_1, \ldots, a_n$ in

$$y = a_0 + a_1 \text{SIN}(x) + a_2 [\text{SIN}(x)]^2 + \cdots + a_n [\text{SIN}(x)]^n$$

gives a polynomial fit of order n in SIN(x). The user need not only use elementary functions such as LN, LOG, SIN, EXP, etc., but can also use algebraic combinations of these. With the transformation features of the Query Processor (QP), a user can experiment with many possible approaches that he had never previously considered in the search for an accurate interpretation of the data.

6.1.1.2 Data alterations: Besides transforming the basic data, the user can selectively alter either the data or transformations of the data via GGP (see Section 5.2). Herein is included the requirement to be able to discard obviously erroneous points and judiciously add data points. The removal of points for curve fit situations is commonplace due to any of the following reasons: data redundancy, lack of uniformity, inconsistency, errors in input, etc.

Note however that all of these operations affect the interpretation of the data and certainly influence any subsequent curve fitting operation. These alteration actions can always be made reversible and the insight gained by the results of various alterations greatly aid the data analysis process.

6.1.1.3 Segmentation: Another technique employed prior to the curve fitting is segmentation. Here the user can arbitrarily treat the data as separate entities with different functional characteristics. Each part of the data (viz, each segment so defined) can then be fit separately. Various segmentation strategies for a given set of data with their resultant curve (surface, hypersurface) fits can yield knowledge as to the best interpretation of the data. Here also the user must be concerned with how smoothly the transition from one segment to the neighboring segment occurs.

6.1.2 Data fitting. - The culmination of data manipulation phase is generally a determination of the "best" functional approximation to the data. The next phase can be

considered to be strickly computational, and is a special application of OPTUM. Here the user actually tries various mathematical algorithms to determine how best to interpret his data. The curve fitting options provided within OPTUM must be flexible to support a broad base of users. Some least-squares fitting options which need be present are polynomial, trignometric, exponential, rational fraction, and mixtures of these. Other analysts will prefer to draw upon more refined optimization techniques (available within OPTUM) for their computational needs.

The optimizer utility (OPTUM) must include and accommodate the great majority of curve fitting methods in vogue in various disciplines. Each method should have various measures of the "goodness of fit" to compare different fits of the same data, e.g. maximum residuals, root-sum-square, sum of the residuals, etc. A user defined function may be input via a Query Processor (QP) directive which assigns to a given fit a real number which quantizes it relative to other fits.

In general the data to be analyzed must be assumed to be N-dimensional. Moreover the desired functional relationships may have multiple independent variables. The curve fitting procedures provided within OPTUM must handle the computations in N-space; interface with GGP will provide user-defined displays on the two-dimensional output device (interactive terminal and/or hard copy devices). (The actual transformation to the medium of the picture is a topic that is covered in the discussion of the Generalized Graphics Plotter (GGP) in Section 5.)

As an example, least squares curve fitting requires the use of an optimization procedure. In such cases, the user can link up to the Optimizer and Parameterizer Utility (OPTUM) to assist him. He can then specify the design variables of his problem. All the flexibility of OPTUM is available to him and he can now, if he wishes, do such things as parameter (fit) sensitivity studies.

6.1.3 Evaluation. - The final phase that the user enters is evaluation of the resulting fits. Here the user must begin to subjectively (with some objective mathematical criteria) determine the correct interpretation of what he sees. In this mode, the user is retrieving computational results and comparing different analyses.

GGP is again employed at this stage. The user can begin to consider sensitivity studies (again via OPTUM) to further define the best solution to the analyses process.

6.1.4 Conclusions. - It should be noted that the above division of the data analysis process is by no means rigid and consecutive. In fact, the three phases in general are intermixed and re-entrant at many points in the actual total solution process as illustrated in Figure 6-1. It is this variability in the progression to the solution that is the best argument for man-machine interactive cooperation.

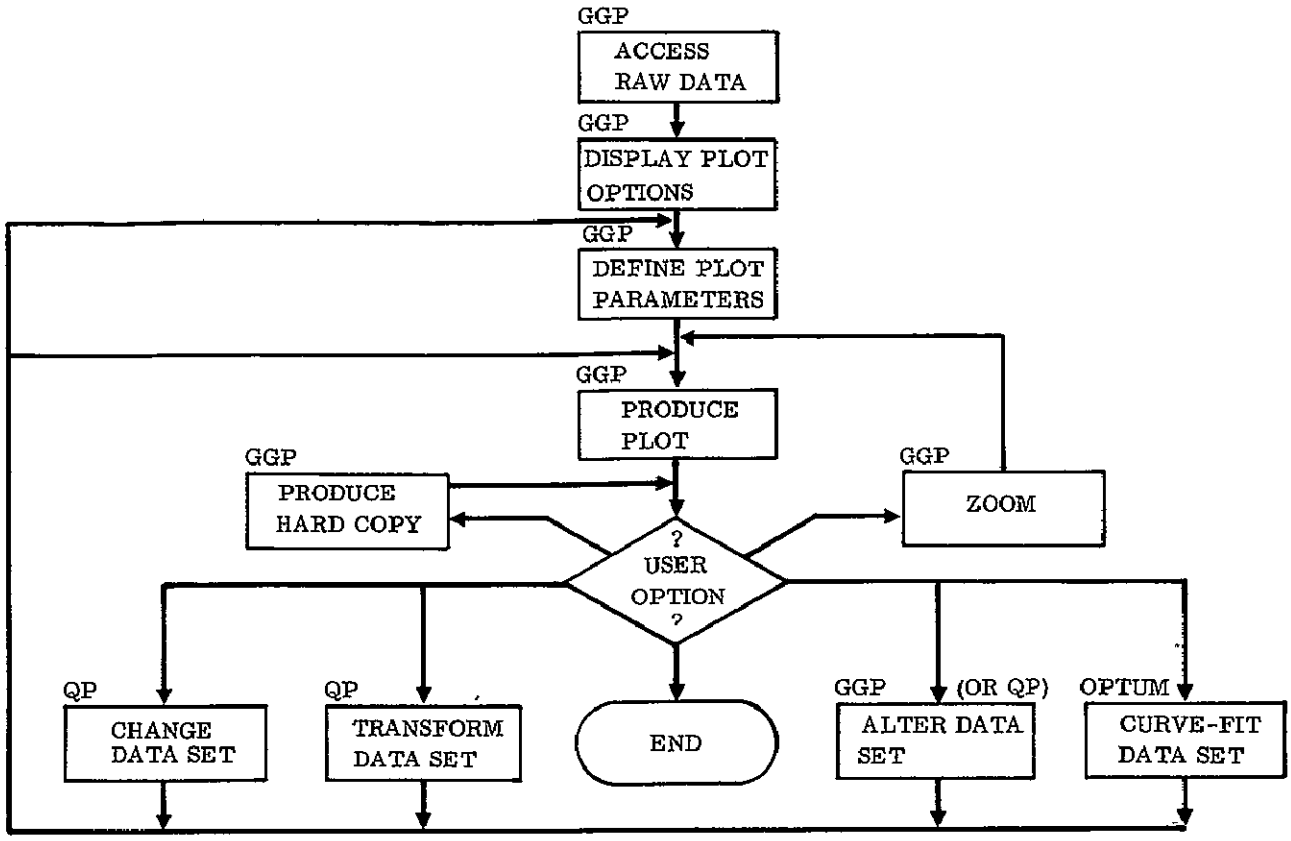


Figure 6-1. Data Analysis Process, Functional Flow

The figure denotes which utility is used at each step in the data analysis process. The use of OPTUM provides complete computational capabilities as well as the selection of the desired options. For instance a user must define his design variables (unknowns) and the objective function to be considered. (Further discussion and treatment of the operations in OPTUM can be found in Section 4.)

GGP lets the user formulate "pictures" of what he wishes to see. (The reader is referred to the Section 5 on GGP for further details and definition of terms.) The user can construct pictures of data with several overlays of different fits to get a visual comparison. He also can window in on areas of interest and produce hard copy records of what he sees at any time. The various entities created by data transformations and curve fits are accessible to the user of GGP and he defines the pictorial makeup of what he sees on his terminal.

The data itself is assumed to exist in the data base - usually in the UF but possibly in an AREA of the MDB. The curve fitting process itself will access the required input data (through OPTUM) for subsequent operations. Any special data manipulations required (e.g. transformations of data, alterational of data points, etc.) will be con-

structed locally via the Query Processor (QP) to expedite the demands for interactive accessing of information.

All operations on the data items can be made reversible, i.e. to not destroy the original representation of the items. This is necessary to provide restart and recovery procedures. Moreover, the iterative nature of the analyst's task makes it mandatory that one be able to get back to selected points in the solution process. Although some operations can be made irreversible, such action is not suggested in general.

6.2 Summary

Curve or surface fitting actually represents an application with numerous computational options which can use the optimizer (OPTUM) to perform the computation and the General Graphics Plotter (GGP) to present the results in pictorial form.

The inclusion of all required curve fitting options in the OPTUM and GGP utilities gives the user powerful tools to assist him in the analysis of data. The man-machine team functions efficiently in conducting the curve fitting process. The man actively directs the machine and takes over completely when subjective judgements are required; the machine does the algorithmic calculations required and supplies the quantitative results to augment man's decision-making processes.

Curve or surface fitting will be principally interactive due to the unpredictability of the analysis process. Moreover, the final act of the analyst, that of determining the correct interpretation of the data, is in general a subjective decision requiring the integration of results of numerical calculations and qualitative considerations. There is, however, the option to execute in the batch mode to perhaps answer independent, rigidly determined questions that do not require interactive monitoring of the output.

7 GENERAL DESIGN MODULE (GDM), A GPU

Design/drafting systems have been under development for many years by various aerospace companies.* These systems have, for the most part, been developed as drafting systems in that they are intended to reduce calendar time, man-hours and cost figures in the development of engineering drawings. Although highly commendable achievements have been made in these efforts (see Appendix B, Section B.2), IPAD's design system is aimed at a broader challenge: the total design process.

7.1 Introduction

In order to truly augment the designer, the General Design Module (GDM) must be a tool carefully constructed to be harmonious with the thought processes of the designer. To be most effective, the automation of the design process must relate as closely as possible to the actual operations in that process. The design process can be diagrammed as shown in Figure 7-1.

The designer begins with a concept or requirement of function. This might be the location of a pulley, or the performance requirements of a jet nozzle; any requirement which is dependent on physical or physically-modeled relationships can utilize the design module. From the concept of function - through the creative talents of the designer - emerges a concept of form. In the case of the pulley location, this concept could be a type of bracket; in the case of the nozzle, a preliminary cross-sectional shape. How does the designer create these initial shapes and designs? Initial designs usually are based on past experience; previous designs and ideas give a starting place from which new ideas evolve. The creative designer finds fault with old designs (while also recognizing their advantages) and from there he either leaps or steps ahead with an evolutionary design.

Recognizing that the design process is usually evolutionary, IPAD's design module is modeled after that concept. The ability to give birth to initial design concepts is a fundamental goal of IPAD's GDM. Four capabilities reveal the emphasis of GDM on design.

* For example, Lockheed Aircraft Co., Burbank CA; MacDonnel Douglas Aircraft Co., St. Louis, MO; Systems, Science and Software, San Diego, CA; and MacDonnel Douglas Aircraft Co., Long Beach, CA.

1. A comprehensive information storage and retrieval (IS&R) subsystem.
2. Three-dimensional, geometric building blocks supporting a design/drafting subsystem.
3. An open-ended analytical design analysis program library.
4. A comprehensive design data structure which - when the design is concluded - provides a legacy to assist in Computer Aided Manufacturing (CAM).

These are briefly described in the subsections which follow:

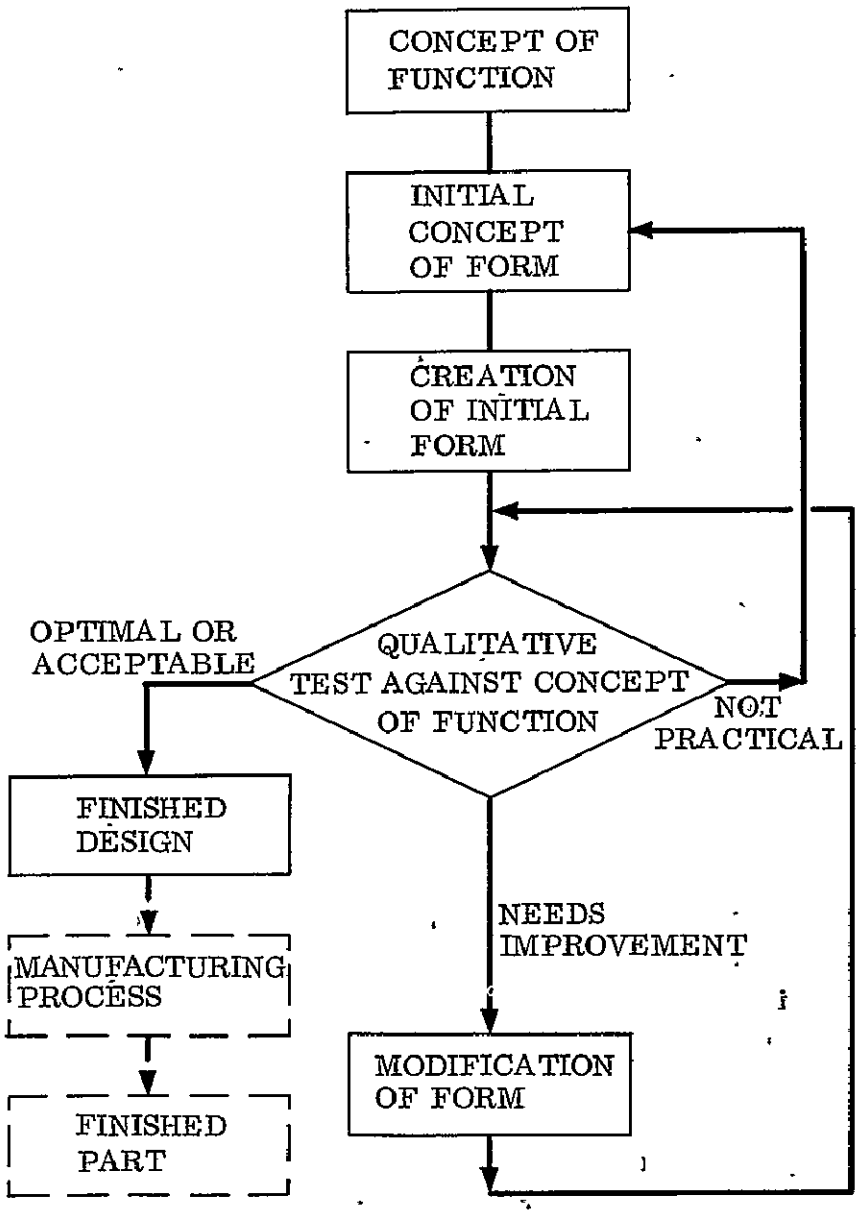


Figure 7-1. Design Process Flow Diagram

7.1.1 IS&R subsystem. - The IS&R subsystem enables old designs (and their attributes) to be readily located. Magnetic tapes and disk data files containing previous solutions to similar requirements are to be retrievable in response to menus of descriptive information. In the case of the pulley, standard or special pulley brackets could be displayed upon request for the scrutiny of the designer. In the case of the nozzle, previous designs which might be of assistance could be reviewed. In either case, the designer's task might be as simple as a local modification to an existing part or it might require a totally new concept. The history of similar designs is an important ingredient in this process.

It should be emphasized that this IS&R system need not contain only information regarding previous part designs, but can contain data histories on complete systems. For example, landing gear geometry, weight, performance data, can be stored for complete classes of aircraft. Attributes of vital commercial components can also be filed (e.g., tires and wheels).

7.1.2 Geometric (3D) building blocks. - The second capability encompasses the capabilities of the existing design/drafting systems plus providing higher-level capabilities. Existing design/drafting systems are modeled after the mechanics of the designer's/draftsman's job; namely the drawing of lines, arcs, etc. The General Design Module (GDM) of IPAD is modeled after the thought processes of the designer/draftman. The designer normally thinks in terms of plate stock, tubing, bar stock, nuts, bolts, or three-dimensional shapes, and only draws lines, arcs, etc., to represent these objects. IPAD's objective is to relate directly to the designer.

This is the concept of shapes and surfaces. As depicted in Figure 7-2, lines and arcs are the lowest level of geometric entity. If the designer can request shapes (e.g., plates, cubes, etc.), the subordinate geometry (surfaces and elements) can be automatically generated (computed) and stored in the data base. Since all geometry cannot be easily developed from shapes (e.g., a spoked wheel), the capability to develop designs from surfaces or primitive elements (lines, arcs, etc.) is also provided.

The designer therefore has a range from the highest to lowest-level geometric building blocks from which to build. The designer's role is to develop the data structure shown in the figure using the most efficient means available. If the designer chooses to modify an old part, it is obvious that a great deal of work is saved since much of the data in the structure (figure) may not need to be changed at all.

7.1.3 Design analysis program library. - The third capability, that of a design analysis program library, is intended to enable the designer to routinely use analysis and synthesis programs in the design modification cycle. It provides the ability to

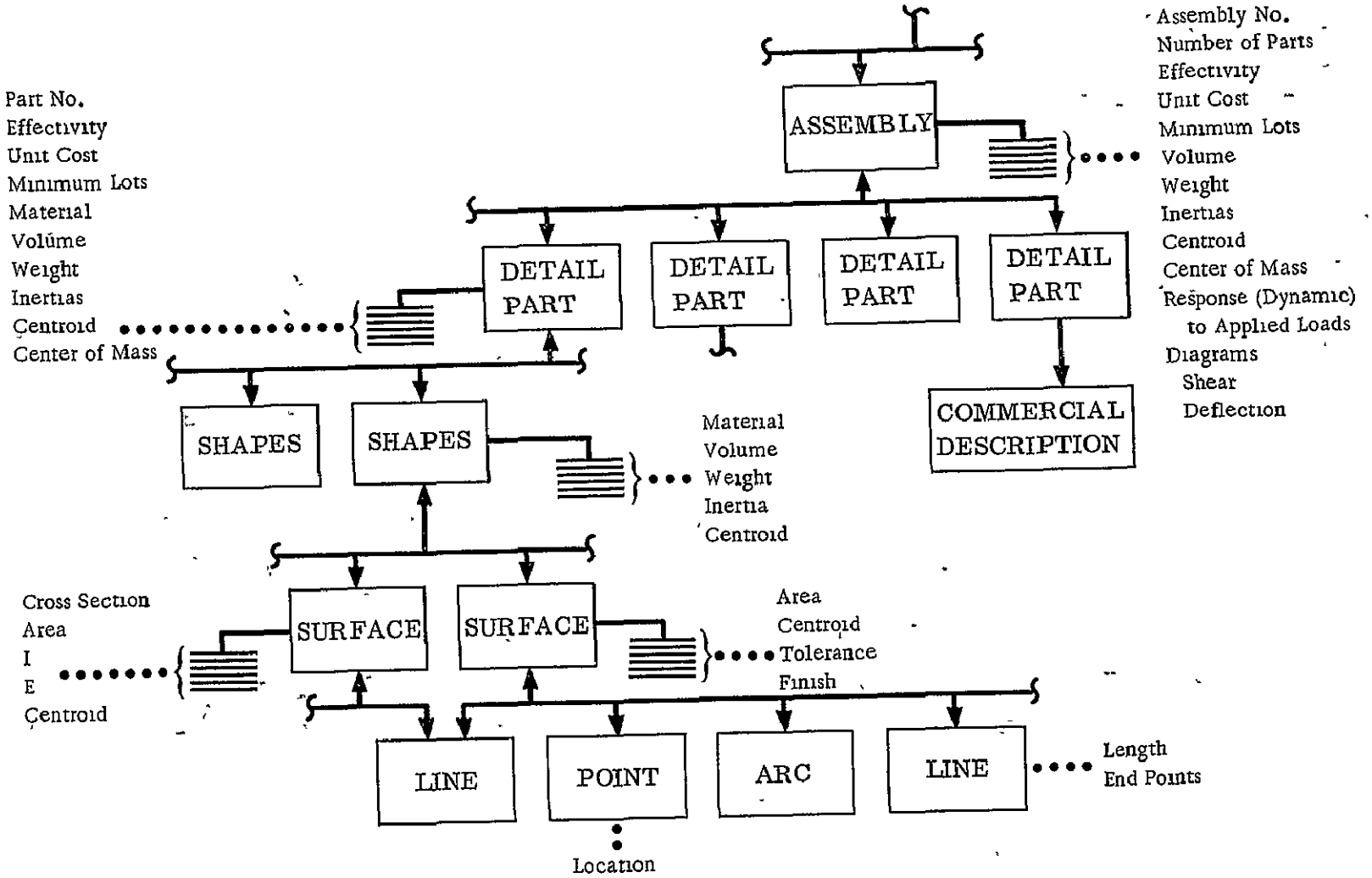


Figure 7-2. GDM Data Structure, Assembly Level

perform the qualitative test against the concept of function (analysis) and to assist in the creation of the initial or modified form. In the case of the nozzle design, an analysis program could provide information to guide the designer in modifying the nozzle profile. A synthesis program might provide an optimization procedure which - having started from a previous design's profile - would create a finished profile.

The library of design analysis programs should encompass many disciplines, e.g., heat transfer, thermodynamics, kinematics of machines, aerodynamics, etc., and should be coupled to the design module in such a way as to enable direct data interaction. For example, kinematics programs would enable animation of landing gear geometry, door linkages, etc.; springs could be automatically sized and drawn; loaded structures can be deflected; aerodynamic and thermal effects can be graphically related to the geometry.

Any analytical algorithm (OM) can be added to the design analysis program library and be made to interface with the geometric data structure.

7.1.4 The legacy to Computer Aided Manufacturing (CAM). - An important part of existing design/drafting systems is the Computer Aided Manufacturing (CAM) interface. Two basic approaches are available to GDM:

1. Provide a capability which interfaces the graphical part description with a programming language (e.g., APT),
2. Circumvent programming languages in their entirety and utilize only graphical interaction to develop tool cutter path movement.

Note that when the data structure (Figure 7-2) is finished and a complete description of the detail part exists, much of the work of a tooling programmer is already completed:

1. Surface information is available and linked to shape descriptions.
2. Surface intersections and bounds are already defined and available.

This is a case where a graphical part display and its backup data structure provides much of the required information.

The future of the numerically programmed tools' cutter location (CL) tape generation lies in total interactive graphic descriptions of cutter paths. Programming languages are no longer necessary nor comparatively efficient, viz. compared with the time and cost reductions envisioned with graphical CAM system interfacing with IPAD's GDM data subbase (Figure 7-2). Further, cutter path descriptions can be more easily and conveniently saved as a portion of the data base than as is presently done by saving program card decks. Since the user is interacting, many assumptions

can be automatically made by the computer (e.g., most likely cutter path) and if incorrect can be corrected immediately by the user. This ability to assume defaults enables many instructional shortcuts not practical when operating in a non-interactive mode.

The key to the applicability of GDM to CAM is the ability to view the component building blocks of the displayed object's substructure (Figure 7-2). This provides a vast quantity of required information no longer available in either the completed design (e.g., conventional drawings) or the manufactured part.

The following sections describe features of the General Design Module (GDM) intended to fulfill the general design requirements just described. Being broader in scope than existing design/drafting systems, it is felt that GDM will fulfill the needs of the designer better while maintaining the advantages already shown eminent with computer-aided design/drafting systems (Appendix B).

7.2 Construction Features

Typical constructions which GDM should be capable of producing are:

1. Machine Parts:
   a. Casting.
   b. Forgings.
   c. Bent-up sheet metal.
   d. Flat patterns.
   e. "Hog-outs".
2. Assemblies and installations.
3. Electrical schematics.
4. Circuit board layouts.
5. Procurement specifications (descriptions of commercial parts, e.g., motors, switches, fasteners, etc.).

The use of computer graphics terminals for design purposes enables many features which aid immensely in constructing geometry. It is these special features which give rise to the impressive cost and time savings inherent with design/drafting systems. Among these features are mirroring and copying, simplified dimensioning along with automatic generation of views whenever possible. It is hoped that the following sub-

sections, in describing some of these features, give insight to the tremendous potential of a well developed design capability envisioned for GDM.

7.2.1 Working with the geometric building blocks. - The geometric building blocks consist of

1. Shapes (volumes) which provide the basis for weight studies, section analysis etc.;
2. Surfaces, which provide the basis for machining instructions,
3. Elements, which are used to create the actual graphic display of the object.

Surfaces and elements will have analytical definitions (mathematical expression) while shapes generally will not. Most design analysis calculations will use the collection of surface expressions for part definition.

Data base support for the GDM must allow for growth and facilitate many different modes of access. The data structure (Figure 7-2) is a model of the geometry of (essentially) an arbitrary physical object, consequently the SUBSCHEMA associated with GDM must define SET representations of arbitrary networks of RECORDs. An individual SET may represent a detail part, an assembly of detail parts, or an entire aircraft. The SET definition must be such that the designer can add to the structure of a SET occurrence indefinitely and/or create an indefinite number of SET occurrences. Furthermore, a secondary SET definition must be provided such that RECORDs defining attributes (tolerances, revision effectivity, etc.) may be associated with any basic geometric building blocks.

Note that the designer may work in any of three modes:

1. Building up complex structures from basic elements.
2. Building down complex structures from intersected shapes.
3. Modifying completed structures obtained through the IS&R capability.

It is an important consideration that the design of a data base for a basic design system be adaptable for implementation of new capabilities. The SUBSCHEMA support to GDM via DDL insures this.

The different ways of creating the design geometry should require minimum designer interaction. The designer should be able to indicate the type of geometry he wants and provide the required geometric data; the computer should determine the appropriate data organization. (E.g., if the designer wants to draw a line, he

need only provide the points, tangent circles, etc., and the computer should find the appropriate definition to fit the supplied data.) All geometry should be defined in a three-dimensional coordinate system. During construction, the third coordinate can be assumed as a default, unless specified.

Capabilities should be included to extend and to bound existing geometric entities. For example, lines and arcs should be extendable to aid in construction of other entities and also be capable of being bounded (i.e., being shortened or segmented) to give flexibility to the designer in the construction mode.

Surfaces should be generated automatically through the generation of shapes. When a shape is specified, its surfaces are automatically identified and stored immediately in the data structure (Figure 7-2); in a similar manner elements should be automatically generated when surfaces are intersected or bounded.

The following descriptions of some basic geometric entities are exemplary and are not intended as complete definitions. A complete set of definitions will require much investigation of utilization methods and human factors and is beyond the scope of this study.

7.2.1.1 Basic geometric shapes (volumes): A tentative list of geometric shapes consists of:

1. Parallelepipeds:
   a. General.
   b. Rectangular.
      - Cubes.
      - Plates.
      - Bars.
2. Solid cylinders (extruded shapes):
   a. General
   b. Solid circular cylinders (rods).
3. Spheres.

Additional shapes can be added for special purposes (or for general usage) as requirements become apparent.

Following is an exemplary list of suggested methods for defining convenient and frequently encountered geometric shapes:

1. Rectangular parallelepipeds could be defined in any one of the following ways:

   a. Define height, width, length. Sides are to be parallel to coordinate axes. Position may be varied as well as orientation.

   b. Identify six orthogonal planes which form the parallelepiped.

   c. Identify edges (points) of the parallelepiped.

   d. Identify corners (points) of the parallelepiped.

   Cubes, plates and bars will have their own more simplified methods of definition.

2. Solid circular cylinders (or rods) are the most generally encountered solid cylinder and could be defined:

   a. Identify axis, radius and length.

   b. Identify axis and point on its surface and length.

   c. Identify circular cylindrical surface and two limiting planes.

   Other types of cylinders will have similar methods of definition.

3. Spheres could be identified simply by a center point and specifing a radius.

7.2.1.2 Basic geometric surfaces: A suggested list of geometric surfaces consists of:

1. Quadratic surfaces
2. Ruled surfaces.
3. Meshes.

Additional surface definitions can, of course, be added as requirements for them become apparent.

A number of ways of defining a surface is necessary for efficient operation. The following is an exemplary list of suggested methods for quadric surfaces:

1. Planes:

   a. Identify a line and a point.

   b. Identify three points.

   c. Parallel to a given plane at a given distance.

   d. Parallel to a given plane through a given point.

e. Perpendicular to two given intersecting planes.

2. Circular cylinders:

a. Identify a line segment as its axis and a point on its surface.

b. Identify a line segment as its axis and specify a radius.

c. Identify any circle and specify its length.

Similar defining techniques are envisioned for the other quadratic surface definitions.

7.2.1.3 Basic geometric elements: The minimum basic geometric elements should consist of points, lines, circles, arcs, and splines. Additional special elements such as ellipses, parabolas, etc., are required. A number of ways of generating each is necessary for efficient operation. The following is abstracted from a set of suggested methods:

1. Points:

a. Positioning of the cross-hair cursor in 3D.

b. Keying in the X, Y and Z coordinates of the point.

c. Keying in the polar coordinates of the point.

d. Intersection of two lines.

2. Lines:

a. Identifing two end points.

b. Parallel to a given line and through an identified point.

c. Perpendicular to a given line and through an identified point.

d. Tangent to an identified arc at a point.

e. Tangent to two identified circles in a plane.

f. Through a point at an angle.

g. At an offset distance parallel to a given line.

Again, similar definitions for other elements are also envisioned.

7.2.2 Semiautomatic dimensioning. - Dimensioning can be greatly simplified using a computer graphics design system. Several operations could be made "automatic", e.g., generation of leader lines, witness lines, arrowheads, and geometric distances. Dimensions must, of course, be readily changable and movable. Nested dimensions can be created with an automatic separation allowance. If a dimension in a nest is

changed or removed, the rest should be repositioned automatically. A capability for providing reference dimensions must also be included.

Datum and dimensional control symbols (concentricity, parallelism, straightness, perpendicularity, etc.) should also be included to enable precision dimensioning. The ability to define tolerances (both general and individual) must be available to the operator. Another feature will permit dimensional compatibility studies (effects of extreme tolerances on function and interchangability of parts).

7.2.3 Isometric and perspective views. - To enhance physical visibility, isometric views are sometimes used by designers. A capability must be included in the design system to display isometric views of a part automatically. This capability should include the automatic detection of hidden lines. Another desired feature allows modification of the part in the isometric view.

Perspective views enhance "visibility" and may be desired either as a construction aid or may be used for presentation purposes. A perspective viewing capabilities should be provided.

7.2.4 Geometric mirroring and copying. - The ability to mirror geometry can greatly simplify the work of the designer. Since the large majority of parts are either totally or mostly symmetrical (about an "axis of symmetry") half of a part may be constructed and then "mirrored" to create the other half. The other half can then be modified to take into account any deviations from symmetry. Since part geometry is three-dimensional, the mirroring operation should mirror about a plane (seen in the mirrored planar view as the line of symmetry) and should create mirrored surface and shape definitions if those definitions have been previously defined.

Copying enables any geometric entity or group of geometric entities to be copied into the data base as positioned on the display. This feature can save multitudes of repetitive work by the designer and is possibly the most powerful feature of any design system. An ability to copy objects already generated - using the basic geometry definitions - is a must for GDM. The designer should be able to group shapes, surfaces, lines, circles, etc., into a single item which can be copied as a single entity. These copied items should, furthermore, be capable of being modified by adding or erasing their geometric details (e.g., a flange might be copied and then the bolt pattern changed to fit a different application). A smaller detail of a geometric entity should also be able to be copied without having to copy the whole entity (e.g., copying a boss from a complicated casting). Once a set of geometric definitions have been made into a group to be copied, further copies should require a minimum of effort; regrouping should not be necessary.

7.2.5 Parts library (IS&R system). - To enable copying geometry from other part constructions, a parts library recall capability should exist. The designer should be able to specify a part number ("drawing number") and get a display of the requested object. The ability to overlay the retrieved part over the present construction and to blank out any non-applicable geometry is required. The designer must then be able to copy, mirror, etc., as if the retrieved construction were part of the present construction. After copying and modifying the retrieved geometry, he must then be able to erase the original and retain only that geometry which has been copied.

An extension of this capability incorporates the IS&R system to allow the designer to search for parts with defined characteristics (via menus) so that these parts may be copied. The ability to request views of standard parts (e.g., NUT PLATES, BEARINGS, etc.) from a menu of parts is a highly desirable feature for GDM. It is envisioned that a designer might , for example, request a stopnut from a list of standard fasteners. He is shown either a military or an internal standard, pictorial description of stopnuts. He is able to select the appropriate size fasteners - as is presently done - from tabulated size information. The callout and appropriately sized views of the fastener would be automatically generated for the designer to position on the display.

Although such a system requires considerable data-storage to include a large number of parts, such a capability is extremely beneficial. GDM must provide the mechanisms to allow the eventual buildup of a large parts library.

7.2.6 Auxiliary views. - Auxiliary views are arbitrarily oriented orthographic views. Since it is often desirable to create auxiliary views to depict a special feature of a part, it is a must that GDM include an auxiliary view capability. Since auxiliary views are additional to the other conventional views, much of the information contained in the auxiliary view should be automatically generated. This view could aid in modification of the part to show the added detail, which then must be reflected back in the other views automatically.

The automatic generation of auxiliary views should include an automatic solution for the hidden lines. Such a feature should be an optional operation because of the time-consuming computer calculations required. The auxiliary view capability must not be limited to any particular orientation or set of orientations and no limit should be placed on the number of auxiliary views per display.

7.2.7 Sectioning. - Generating sections is often necessary to display internal details of parts. GDM should permit section cuts in much the same manner as auxiliary views. This capability should allow a single cutting plane or jagged section cuts (section cuts defined by multiple cutting planes). Sections should show only that portion of the part left after cutting and should be able to detect hidden lines.

GDM should semi-automatically create section lines (the boundaries being defined interactively by the designer) as well as optional totally automatic cross-sectioning.

7.2.8 Assemblies. - The ability to superimpose parts - using library files - facilitates building assemblies from detail parts. By making particular lines, arcs, etc., of the display details dashed (or hidden), or by removing them, the details can be made into descriptive assembly drawings. Dimensions can be added to control assembled positions and tolerances. The dimensional compatibility analysis feature mentioned in the dimensioning subsection (7.2.2) will apply to assemblies as well. This feature will be useful in Quality Control and interchangeability studies.

It is envisioned that the assembler himself might construct his own assembly drawings (hard copies) while viewing a constructed assembly at an interactive terminal.

7.2.9 Flat pattern development. - The usual procedure for creating bent sheet metal parts is to first develop a "bent up" display showing the part in its final configuration after which the part is unfolded to create what is known as a flat pattern. Since the development of the flat patterns is a highly systematic or procedural process, the computer can be used in the conversion of the bent-up configuration back to the unfolded flat pattern. GDM should contain a semi-automatic flat pattern development capability (i.e., the designer may have to fold each bend separately, supplying specific information, such as corner, radius, etc) and an option that unfolds the bends automatically and even performs a checking function (such as tearout distance, bend radius, etc.).

7.2.10 Revisions. - Making revisions to parts on a graphics-based design system can greatly simplify conventional procedures and bookkeeping. The conventional method usually requires a separate form to be completed for each drawing change or set of changes. This form must reflect graphical and numerical changes and end-item applicability (effectivity). To depict a part configuration or to depict an end-item configuration (e.g., a specific airplane configuration) one must consider the drawing and the effectivity requirements of every previous change.

Using GDM, revisions and effectivity characteristics can be stored in the data base (Figure 7-2) and can facilitate viewing configurations for change/effectivity requests. As previously discussed (Subsection 7.2.1) a basic configuration is represented as a SET relationship defining a network of geometry RECORDs (Figure 7-2), attributes are represented as associated (at any branch of the network) SETs of attribute RECORDs. Several versions of any particular RECORD occurrence within either type of SET may be inserted, retrieved, deleted, and maintained. DBMS

architecture provides for many methods of implicitly controlling and assisting this process (through DDL options) and in turn requires additional RECORD selection criteria from programs accessing various versions of a RECORD occurrence.

Much of the conventional paper work for changes can also be automated through a revision mode feature (if desired). The designer can be made to respond to requests for information (e.g., effectivity, reason for change, date, etc.), and notification and distribution could be automatically handled via periodically batch processing information contained in the data base.

7.2.11 Desk calculator. - To aid in various calculations not programmed as part of the design system's code but found necessary by the designer (e.g., subtraction and addition of tolerance limits to assess fits, etc.), the design system must incorporate a desk calculator feature. This enables the designer to add, subtract, divide, take square root, etc., much the same as a simple desk calculator (e.g., Wang, Singer, Friden, Hewlett-Packard, etc.). In addition it permits the user to develop functions or programs to be added in the same manner as for stored-program desk calculators. These functions could be listed in menus built into a logical tree structures, e.g.:

AERODYNAMICS
MECHANICS
TRIGONOMETRY
DERIVATIVES
INTEGRALS
HEAT TRANSFER
STRESS ANALYSIS
:

Selecting any of these options could produce a more detailed menu for the particular discipline. For example if TRIGONOMETRY were selected a menu like the following might appear:

RIGHT TRIANGLES
OBTUSE TRIANGLES
FUNCTIONS OF SUMS OF ANGLES
FUNCTIONS OF MULTIPLE ANGLES
MISCELLANEOUS RELATIONS
LIMITING VALUES AND INEQUALITIES
INVERSE TRIG FUNCTIONS
RELATIONS BETWEEN SIDES AND ANGLES OF ANY
PLANE TRIANGLE
SPHERICAL TRIGONOMETRY
:

Selecting one of these options would then produce the functions desired and allow the designer to choose one of those.

The user can select the desired function and the arguments needed will be assigned to specific registers awaiting keyboard Input.

If several functions were assigned to the keyboard function keys, the user could then utilize the function as if it were a built-in keyboard function. Several functions could be stored simultaneously, the number being limited by the available keyboard function keys. To keep track of the functions a brief list could be displayed to remind the user of the functions assigned and their keys. Tutorial information describing the arguments required will also be available.

7.3 Display Features

The use of interactive computer graphics provides a limited area display tube upon which the design work - which classically has been done on large drawing boards - must be accomplished. To facilitate a humanly adaptive system, several display features are needed to enable display construction and manipulation. The following subsections discuss those envisioned for GDM.

7.3.1 Line styles. - The following line styles are required for a basic design/system:

1. Solid light (construction lines, dimension lines, leader lines).
2. Solid heavy (border lines).
3. Dashed short-short (hidden lines).
4. Dashed short-long-short (center lines).
5. Dashed short-short-long-short-short (phantom lines, section lines).
6. Break lines for tubes, rods and general shapes.

The ability to change the style of any existing line or a portion of a line must be provided. This capability enables the designer to graphically depict hidden lines in the construction of auxiliary views and sections.

The lengths of the short dashes should be a function of the scale of the display. The long dashes are made adjustable to enable the designer to properly "fit" the line to the part.

7.3.2 Text. - The text writing feature enables notes and labels to be created using the alphanumeric keyboard. (The keyboard employed should be configured as closely as possible to a conventional typwriter to simplify typing.) The carriage should return to the starting location of the first line unless otherwise instructed. Semi-automatic generation of leader lines and arrowheads to the geometry are to be provided.

The ability to move text and to duplicate it is also a necessary feature. The character set includes an upper case alphabet and numerals plus common symbols. Alphanumerics are preferably of variable size or have at least two different sizes. A capability for storing a menu of standard notes is a highly desirable feature, along with a similar capability for title blocks and drawing frames to be used with hard copy. A bill of materials and standard parts block capability should be provided which could either stand alone or be printed with the hard copy. Another capability would provide an accounting of materials and standard parts in the data base which permits computerized material summations and standard parts summations, etc.

7.3.3 Move, zoom, rotate and scissor. - Because of the limitations imposed by the small working area of a graphics display tube (as compared to a conventional drawing board), the ability to move and zoom are critically important features. Moving translates a specified point on the object to the center of the screen. This, in effect, shifts the viewing area of the design being generated. Because a designer often works on several parts of a design simultaneously he must be able to move the viewing area quickly and easily. As the viewing area is changed, the part of the view no longer visible must be "scissored" (erased) from view. Zooming is similar to moving except the shifted image is made larger or smaller (usually in 2:1 increments). Rotating causes the entire visible portion of the design to rotate about a specified point (normal to the viewing plane) by a specified amount. This capability can simplify the construction of lines, arcs, etc., which would otherwise be at a more inconvenient orientation. Scissoring must be automatic with moving, rotating and zooming.

7.3.4 Grid. - To enhance the construction mode, a capability should exist which enables the designer to superimpose a three-dimensional grid. The spacing and position should be variable and capable of being blanked out (erased) and redisplayed easily.

7.3.5 Distortions. - The loftsman (or linesman) frequently needs to "fair" two lines or curves. To enhance his ability to "eyeball" this fairing, the ability to distort the shape of the object is desirable. This can be achieved by allowing different scale factors to be applied to the orthogonal axes. This has the effect of exaggerating rough or non-faired areas. If for example a designer wishes to fair two surfaces representing the profile of an air inlet duct for a jet engine, he would first orient the part to obtain the best view of the intersection of the surfaces, and then he would distort the view to exagerate that intersection's roughness.

7.3.6 Erase. - The ability to erase geometric, alphanumeric or any other portion of the construction is a basic requirement. Erasing must be fast and easily requested. Two types of erasing are needed. The first simply blanks out items which can later be redisplayed. The purpose of blanking out parts of a display is to minimize "flicker" and to temporarily remove geometry which may tend to confuse or complicate the picture. The second type of erasing causes the item to be permanently removed; this will affect both the picture and the data base. It is important to safequard a permanent erase sufficiently to insure that the designer has selected the correct erase type.

7.3.7 Hard copy. - Essentially, GDM must be able to provide hard copy for any display it can produce at the interactive graphics terminal, including the complete display, a portion of which is being "windowed" onto the terminal's display area (Subsection 7.3.3). The generation of precise drawings is one of the most critical requirements of a design system. Three basic types of hard copy are deemed necessary:

1. High quality, dimensionally precise and stable, full-size drawings (dimensionless drawings).
2. Good quality dimensionally stable, scaled drawings (mylar originals).
3. Reference copy of any display.

It is expected that the first type must be developed on a high quality, precise automated drafting machine such as a Gerber plotter or Orthomat. Turnaround time to obtain this type of drawing should not exceed one day.

The second type of hard copy serves as an original from which prints can be made for distribution if desired. Revisions are made at the graphics console and a new original produced. A process used to produce this type of hard copy starts from the 35 mm film of a microfilm recorder (e.g., the Stromberg Carlson S-C 4020) which is then projected using a magnified image onto wash-off mylar. High quality equipment is used to obtain good quality copy. The time to obtain a wash-off mylar should not exceed 4 to 8 hours.

The third type of hard copy is used only for intermediate reference or engineering reports and is either immediately available to the console operator or after a short delay. This type need not be full size but should be scalable and have "tooth" to permit writing thereon. It must be of good quality and be legible however. Some devices which may serve this purpose use light sensitive, heat developing paper (dry silver) which is exposed to a special CRT; copies can be obtained within a few seconds (e.g., TEKTRONIX 4610). Other means of providing such reference copy are through the devices previously mentioned (e.g., Gerber and Calcomp pen-ink recorders or microfilm recorders). These latter devices are generally slower but often provide higher-quality report copy.

7.4 Attributes

Since the SET relationships of the data structure (Figure 7-2) actually model the physical shape of the object, the SETs can be extended to model other attributes of the object. For example, surface finishes can be related to surface descriptions, material densities with the shape descriptions, etc. The following attributes are typical of what can be computed and retained for the benefit of the designer.

7.4.1 Volume and mass properties. - The capability of computing volumes and weights of basic shapes must be available in GDM. This capability would permit volumes and weights to be calculated for any arbitrary part, regardless of shape. Included is the capability of calculating various mass properties, e.g.:

1. Moment of inertia about a given axis.
2. Radius of gyration about a given axis.
3. Center of mass.

Retaining the computed weights of parts, components, subassemblies, etc. (i.e., shapes) as attributes in the data structure is potentially valuable since the weight of an entire aircraft can be determined from summing part weights. Weight of total aluminum, titanium, etc., can also be determined by summing appropriately delineated parts separately.

7.4.2 Section properties. - After a cross-section (surface) has been defined or generated, GDM should enable computation of several section properties. These surface properties (attributes) - which are stored within the data structure could include:

1. Area of the section.
2. Moment of inertia of the section about a specified axis.
3. Section modulus.
4. Section centroid.

If the cross-section represents a beam, the capability for computing and storing additional information should be made available, e.g.:

5. Bending moment diagrams.
6. Vertical shear diagrams.
7. Deflection diagrams.
8. Reactions to applied loads.

This capability should be available for the most common beam supporting methods and loading conditions (i.e., concentrated and distributed loads).

7.4.3 Data retrieval. - Since the designer often needs to know numerical information about previously constructed geometry, a data retrieval (or verification) feature is required. This feature retrieves and displays numerical descriptive information (attributes) regarding existing geometric definitions (i.e., shapes, surfaces, lines, arcs, points, etc.), or relationships between them. The following is a exemplary list of information the designer might retrieve:

1. Rectangular parallelepipeds:
   a. Width, length and height.
   b. Corner coordinates.
   c. Volume.
2. Cylinders:
   a. Axis direction cosines.
   b. Axis endpoint coordinates.
   c. Volume
3. Planes:
   a. Equation of plane.
   b. Coordinates of intersection of plane and line or curve.
   c. Normal distance from a specified plane to a specified point.
4. Arcs:
   a. Radius at a point.
   b. Coordinates of center at a point.
   c. Start and end tangent angles.
5. Lines:
   a. End point coordinates.
   b. Length.
   c. Angle between lines.
   d. Distance between parallel lines.
   e. Normal distance between skew lines and end coordinates of normal.
   f. Direction cosines of line.

6. Points:

   a. Coordinates of a point.

   b. Distance between two points.

   c. Perpendicular distance between a point and a line.

All of the above information, where possible, should be available in both the original and viewing coordinates systems.

7.5 Program Linking

Since other graphics (or non-graphics) OMs/GPUs may be useful when linked to GDM, a capability must be provided to facilitate this objective. Some classifications of OMs which might be beneficially linked to GDM are:

1. Linkage analysis.
2. Linkage synthesis.
3. Structural analysis.
4. Redundant member force analysis.
5. Spring design.
6. Circuit analysis.

It is envisioned that linkage will be completely handled through GDM's data base through employing different SUBSCHEMAs for each OM/GPU as required.

7.6 Partitioning of Computing Functions

During the process of establishing the design form, much repetitive interaction exists between the user and his console. Literally hundreds of display variations will be made during the design of a single part. Due to the fact that most of this interactive processing will not require large core storage or large amounts of computing, it is practical to use minicomputers for much of this processing. Using minicomputers can reduce the work load of the host computer and can allow for a much more efficient system*.

* See Subsection 2.2.2.4 in Volume IV, Part I for a discussion of these advantages.

The partitioning of the computing functions (viz. those functions detailed in Sections 7.2 and 7.3) is very subjective and depends heavily on the architecture of the system (GDM) as designed. In general, tasks should be allocated to the minicomputer whenever the estimated response time is acceptable or if it is estimated to be less than that typical on the host computer (providing the capability exist on the mini). The response time for tasks performed by the mini should never exceed one second if possible**. Tasks allocated to the maxi are expected to vary considerably depending upon the complexity of the operation and the maxi's background job mix. If a maxi's task is expected to exceed three seconds, a courtesy (or status) message should be displayed. In general, accesses to the host computer should be minimized to improve response time.

A possible partitioning of some of the design tasks is depicted in Figure 7-3 which has the majority of the computing functions on the minicomputer. Some tasks which are expected to be best suited for the host computer are the sectioning, generating hard copy and data base management functions. Sectioning can require considerable calculations. Hardcopying can be handled separately and draws directly on maxi peripherals (e.g., the microfilm recorder). Data management will require extensive disk searching - specifically the capabilities offered by DBMS (see Section 1).

As the user builds or modifies the design, the data structure, stored on the disk of the host computer, must reflect that design. Modifying the original data structure with each change would cause an excessive amount of host computer accesses. To minimize the number of accesses, the minicomputer software must be designed to store data changes and additions until a block of information is ready to be sent to the host computer. This will both reduce cost and response time due to fewer host computer accesses.

Figures 7-4 through 7-12 provide some additional detail on the functions presented in Figure 7-3 and are self-explanatory. Note that some of the options of the coordinate transformation subroutine (Figure 7-3) - e.g., zooming, rotating and windowing (scissoring) - may be accomplished (at the display-controller level) by the interactive graphics device. (One such device is CDC's new General Purpose Graphics Terminal, GPGT.) In these cases it is anticipated that modular code and installation "parameters" will be provided to allow a specific IPAD installation which has this capability to optionally switch off those functions and drop the corresponding code modules.

** A practical minicomputer would exhibit approximately a one microsecond cycle time, have a central memory of 24,000 16 bit words, direct memory access (DMA), and have a one million word disk.

7.7 Conclusions

The general design system envisioned advances the state of the art by taking a completely different approach. The change from the drawing board to the graphics console must be accompanied by a change in emphasis from drawings to three-dimensional objects and a corresponding detailing of the data base. With the inherent restrictions of classical board design (viz. drafting) lifted, a fresh approach to design is made possible. Designers will be able to quickly transfer their skills to IPAD's GDM since continuity has been maintained. However with their newly established freedoms and abilities, these same designers may well forge ahead with less obstructed thought processes in a stepped-up, creatively-enhanced atmosphere.

IPAD's GDM is evolved around a concept in which physical labor is minimized (in the sense of establishing or documenting ideas) and mental labor is maximized (in the sense of allowing continuous, rigorous thought processes). More creative thought per unit of time must result in better designs at lower cost. Bookkeeping and CAM fall-outs, although not the central objective of IPAD's GDM, can produce an order of magnitude reduction in associated costs.

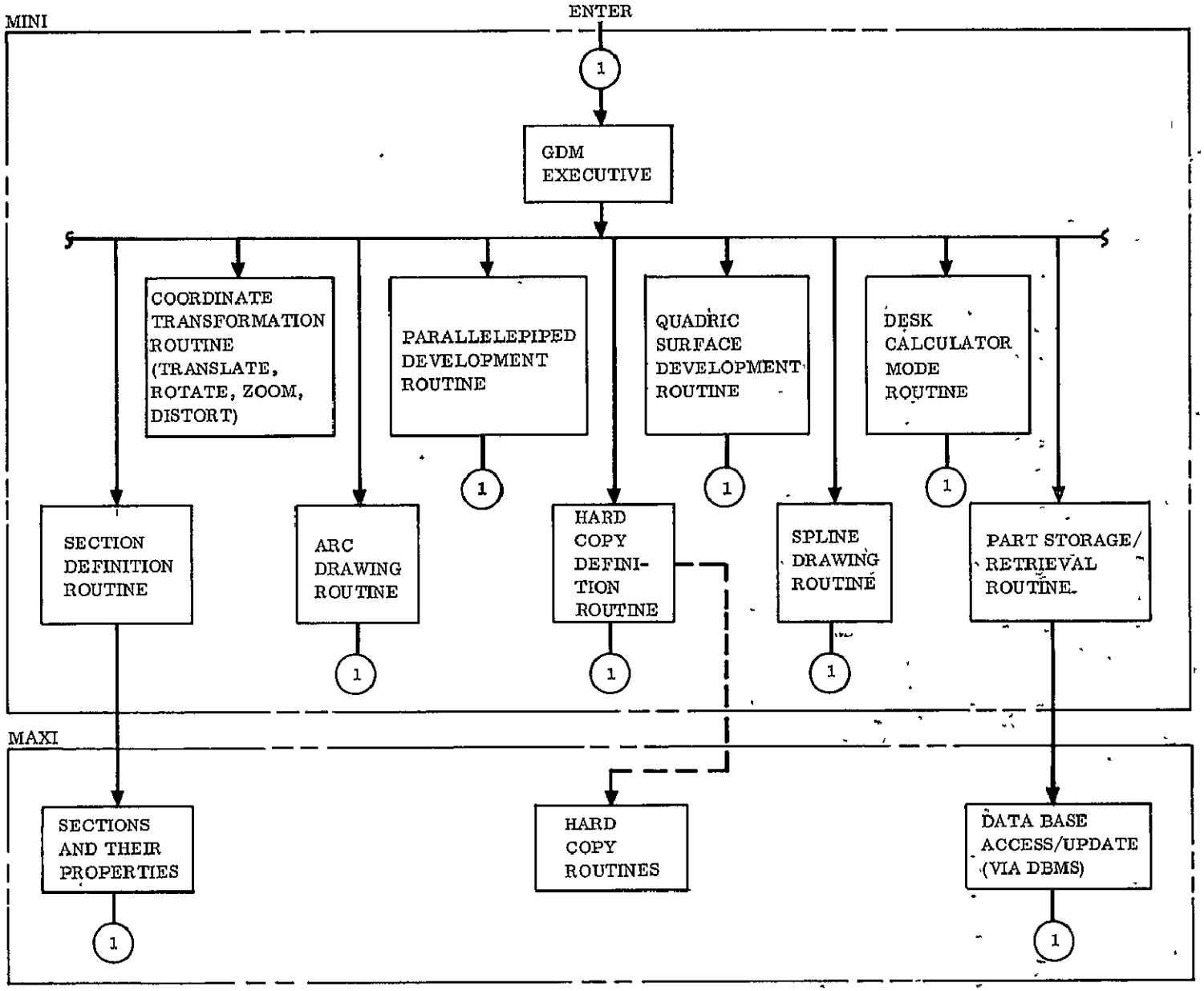


Figure 7-3. Partitioning of Computing Functions Between the Mini and Host (Maxi) Computers

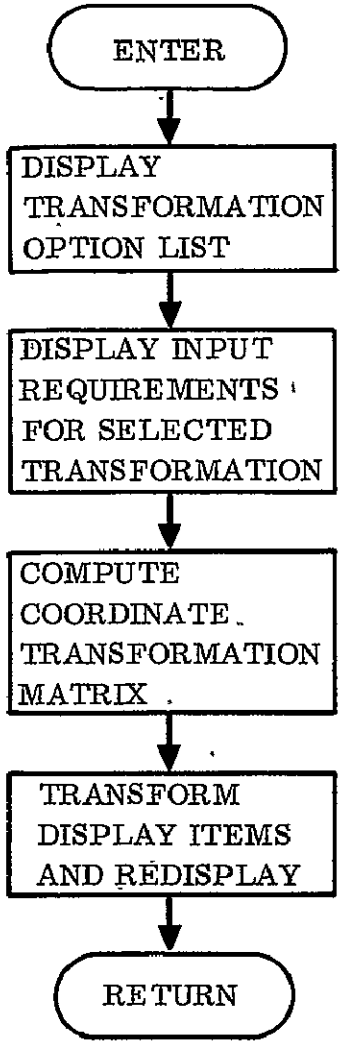


Figure 7-4. Coordinate Transformation Functional Flow Diagram

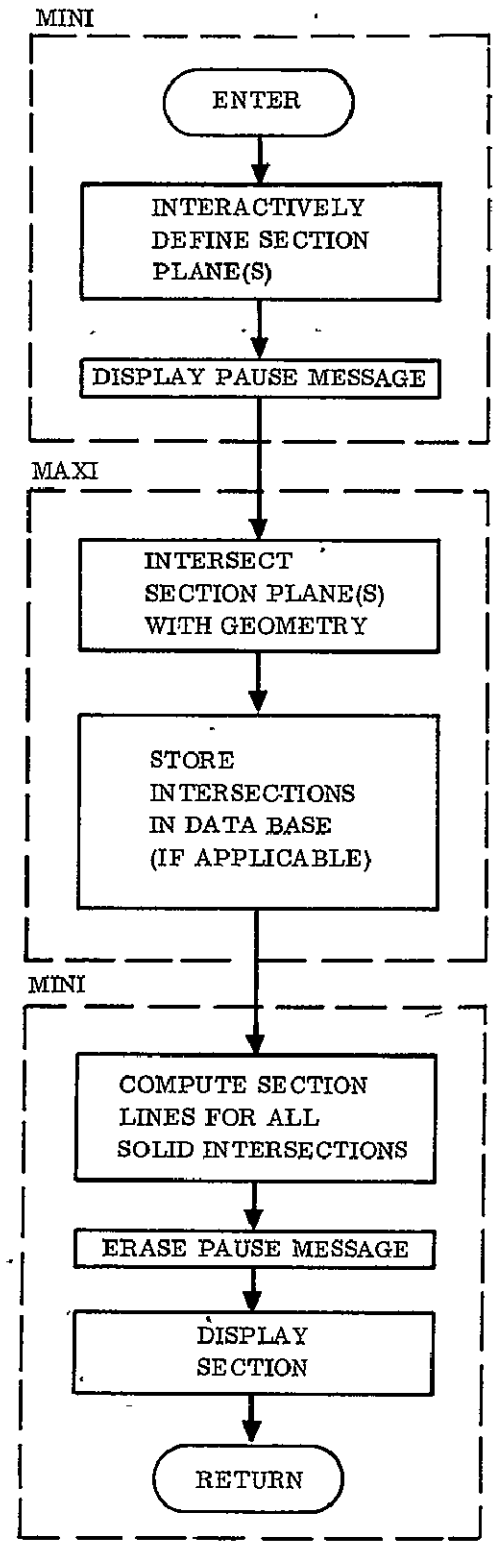


Figure 7-5. Section Development Functional Flow Diagram

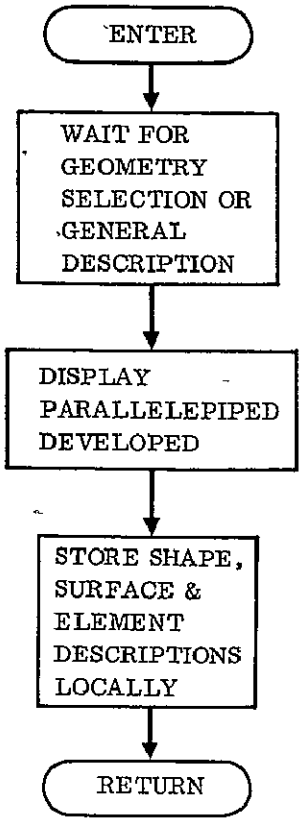


Figure 7-6. Parrallelepiped Development Functional Flow Diagram

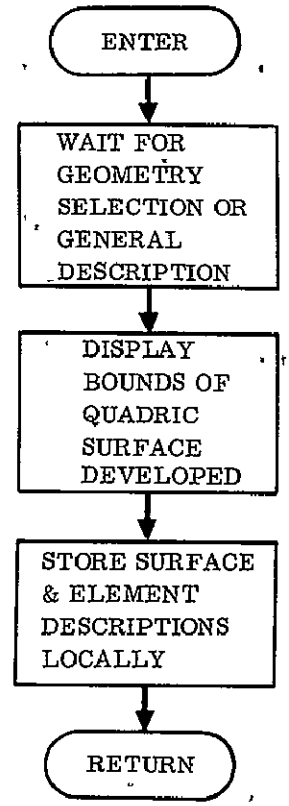


Figure 7-7. Quadric Surface Development Functional Flow Diagram

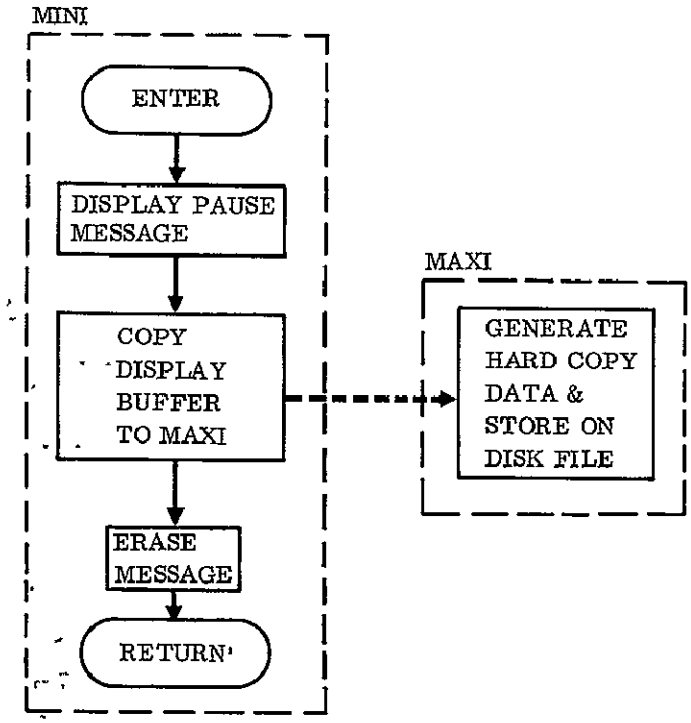


Figure 7-8. Hard Copy Development Functional Flow Diagram

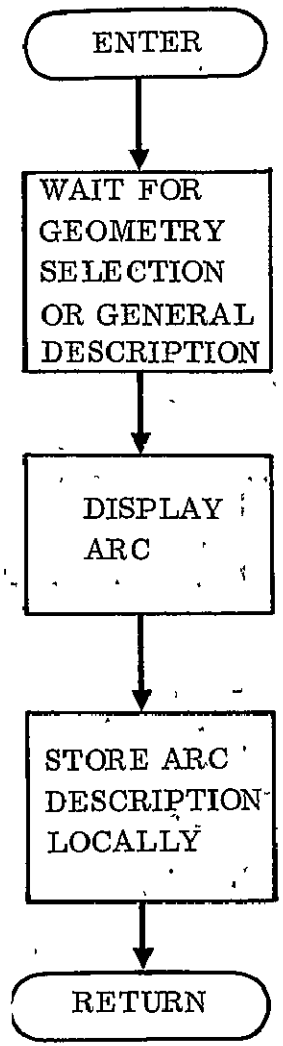


Figure 7-9. Arc Development Functional Flow Diagram

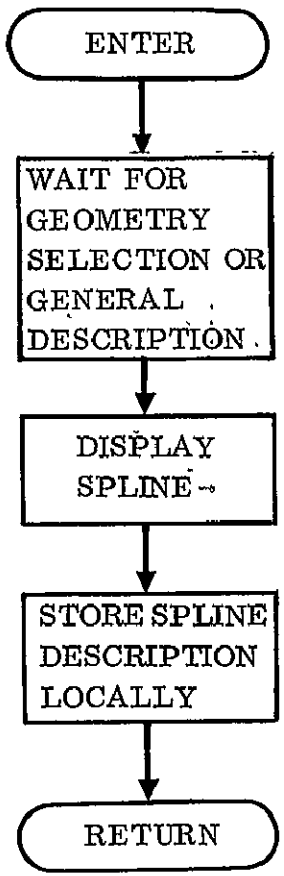


Figure 7-10. Spline Development Functional Flow Diagram

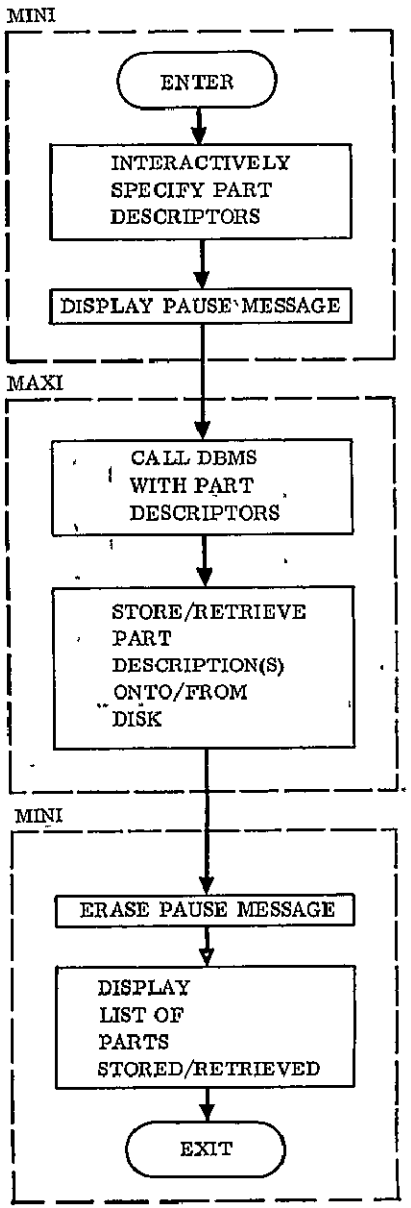


Figure 7-11. Part Storage/Retrieval Functional Flow Diagram

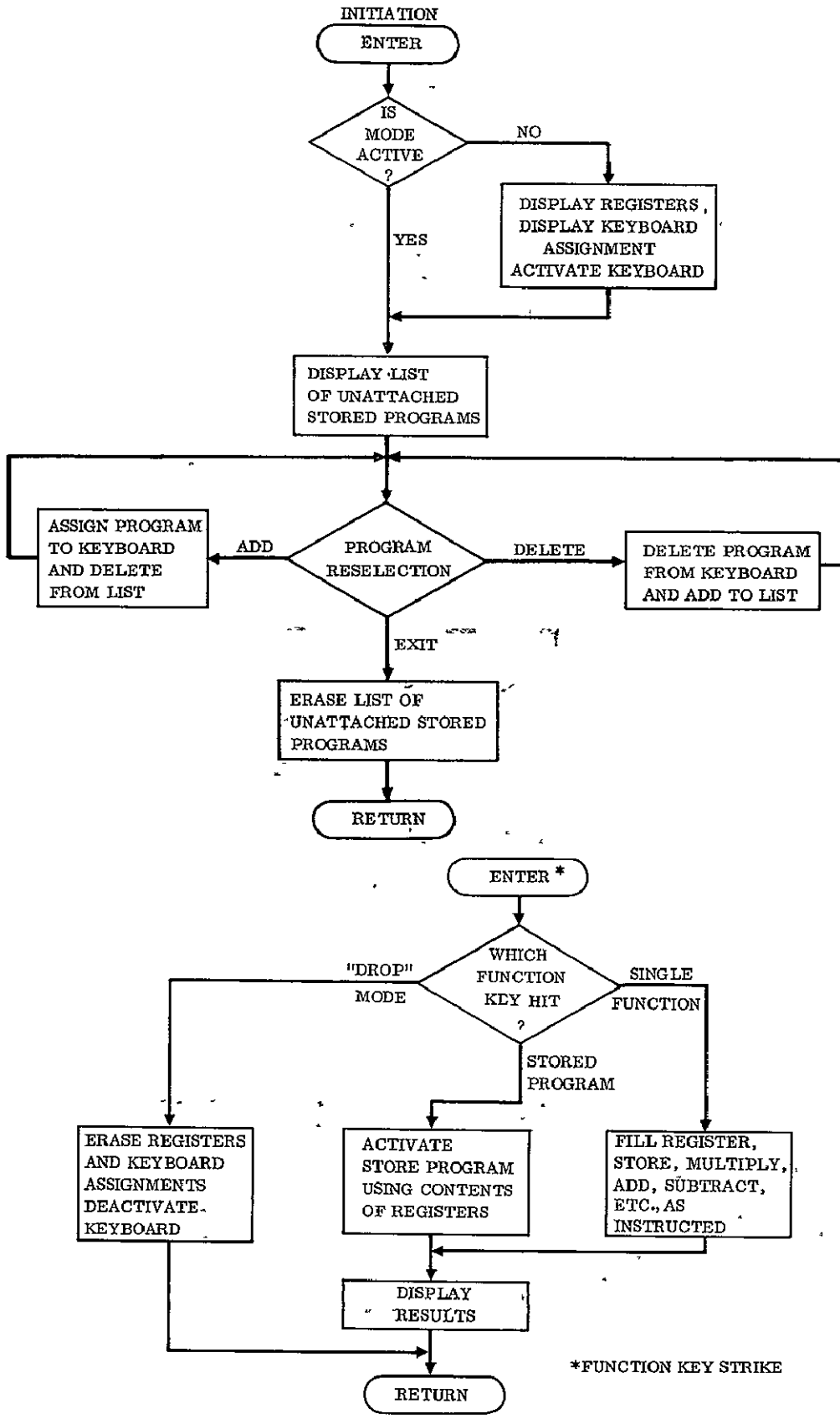


Figure 7-12. Desk Calculator Mode

8 TUTORIAL AIDS SUPPORT (TAS), AN APPLICATION

The purpose of Tutorial Aids Support (TAS) is to provide access - through an interactive terminal - to tutorial aids. This is a general support mechanism as distinguished from specific logical tutorial aids that may be built into any executable module (OM or GPU) with which the user interacts. The intent is to satisfy the user's need for interactive assistance without requiring built in tutorials, and to augment tutorials that are already built in.

The nature of tutorial aids required for incorporation of an OM into the system varies. Some of the factors that affect the requirements are:

1. Complexity:
   a. The number of different capabilities provided by the OM.
   b. The extent of detailed set up for the OM to be performed by the user.
2. General applicability:
   a. The extent to which different users require the capabilities provided by the OM.
   b. The frequency of use of the OM by each user.
3. Extent of built-in tutorial assistance.

Incorporation of the OM is performed by cognizant programmers and tutorial aids are provided as required. Requirements for tutorials vary from none to the equivalent of a program's user's manual, complete with quick reference capabilities. TAS supports this full spectrum of applications and any innovations the programmers may devise.

Since the very nature of the tutorials are to be devised by the programmer in consultation with the users, no rigid preconcieved structure can be imposed. The very act of providing the tutorials with an OM includes providing descriptions of the tutorials' structures. Also, the intent of TAS is to assist the user rather than to burden him with a complex command language to control TAS. Consequently, the tutorials provided must also contain sequences of commands and menus providing for details of control. To summarize, tutorials to be provided by programmers include:

1. Text to be displayed, including menus.

2. A description of the text to provide TAS with access capability.

3. Command sequences to be executed in response to the user's choice from menus.

With this concept of what the programmer must provide for a general TAS capability, it is evident that TAS is not an IPAD special capability, but is in fact an application of the manufacturer supplied Query Processor (QP). A specific application is a collection of QPSs.

9 COORDINATE/UNITS TRANSFORMATION, AN APPLICATION

A primary feature of the IPAD concept is a central data base shared (through independent OMs) by the different engineering disciplines. The system design permits each user to configure a task-oriented UF (Section 1.3) to be shared by the independent OMs employed in a task. Capabilities of the manufacturer supplied DBMS/QP provide for:

1. Construction of the central data base, to include the UFs.
2. Initialization of portions of the data base.
3. Management, maintenance and restructuring of the data base.
4. Interfacing independent OMs with the data base.

In support of interfacing independent OMs, executable code within DBMS provides for transformation of the type implied by variations between the SCHEMA and SUBSCHEMAs descriptions of the same data. (see Subsection 1.2.1.3). These transformations are in the nature of reformatting and restructuring. There is no executable code provided within DBMS to evaluate specific mathematical formulas associated with coordinate/units transformations. The requirement is for a capability to augment the DBMS functional capability.

Although the DDLs do not specify the coordinate/unit systems of variables, there are options through which DBMS can be directed to invoke any augmenting capability (code) provided by an IPAD installation. The DDLs, then, provide the declarative interface (e.g., identification of parameters and variables needed in a transformation calculation) and DBMS provides the functional interface (i.e., invoking the calculation at the appropriate time).

Wherever the nature of the user's problem is such that the DDL specifications are awkward or the IPAD installation has not provided the transformation required, the user can use QP to effect virtually any transformation he desires. Those transformations beyond the scope of QP should be provided in the form of additional code (see Section 9.3).

9.1 Units

Typically, the required units transformation is from one standard system (e.g., English to International units). The coding of a DBMS invokeable function to accomplish

a given transformation and insertion of the code into the data base is a trivial programming task, typically involving one executable FORTRAN statement plus some linkage code.

In addition to the units transformation standards provided by the IPAD installation, the individual user will require the ability to perform transformations not provided by the installation. This can be most easily accomplished through the update capabilities of QP (either interactively or via a QPS).

9.2 Coordinates

A coordinate transformation consists of operating on a set of points (vectors, matrices), one at a time with one or more transformation matrices. This is a more complex problem than unit conversion since the relationship of two coordinate systems must be known in order to construct the transformation matrix. Since the OMs are incorporated independently - without restrictions on subsequent use - the transformation matrix corresponding to a given interface problem cannot be provided at initial OM incorporation into IPAD.

To simplify this, a suitable set of reference systems is chosen for the installation and matrics are supplied at OM incorporation to transform the systems used by that OM into the base (prime) reference systems. This permits the direct interface of independent OMs as follows:

$$T_1 * A = A' \qquad (1)$$

$$T_2 * B = B' \qquad (2)$$

$$A_B = T_2^{-1} * T_1 * A \qquad (3)$$

Given a transformation matrix T_1 which transforms an OM's vector A from its coordinate system into the prime reference system(1), and given a transformation matrix T_2 that transforms a second OM's vector B from its coordinate system into the prime reference system (2), then the matrix $T_2^{-1} * T_1$ transforms vector A from its coordinate system into the coordinate system of vector B (3), suitable for use by the second OM (e.g., as input vector B).

Coordinate systems obviously apply to geometric data but can apply to other types (e.g., aerodynamic) as well. An appropriate prime reference system must

be chosen for each type of data handled (with any reasonable frequency) by a project.

The problem of constructing transformation matrices can be further complicated by the fact that the elements of a matrix may be variables, whose values change with time or other problem data. (E.g., angles of attack and sideslip in aerodynamic transformations.) In this case, a QPS can be provided in addition to the matrix to assist the user in updating the values of these elements within his UF (but see also Section 9.3).

Given the proper matrices, the transformations may be accomplished via QPSs or by installation supplied code called by DBMS as specified in the DDL.

9.3 Implicit Transformation Functions

In addition to describing data structures and logical relationships of data, the DDLs provide specification of many implicit procedures. Many of these specifications involve exits to installation-supplied executable code. Two such exits are of particular interest in arranging coordinate/units transformations. These are the ON clause (Reference 1, page 108) and the ACTUAL/VIRTUAL RESULT clause (op. cit. page 94).

The ON clause is associated with a particular DATA-ITEM or DATA-AGGREGATE and specifies that DBMS is to invoke an identified installation-supplied function code whenever the data is the subject of a specified DML command (or any of a group of commands) executed by DBMS.

The ACTUAL/VIRTUAL RESULT clause specifies that a particular DATA-ITEM is dependent on the value of other DATA-ITEMs and that DBMS is to invoke an installation supplied function code to re-evaluate the DATA-ITEM whenever the independent DATA-ITEMs are changed.

Beyond these capabilities, the installation supplied code may also make reference to data not presently in central memory, e.g., within the UF. In this case, the code must make DML requests of DBMS and must also be supplied with an appropriate SUBSCHEMA. It should be noted, however, that DBMS is not proposed as an executive program. When the intent of installation supplied code exceeds the scope of data management and interface support, the task should be restructured to provide the required code to be called via TCS commands.

9.4 Conclusions

The nature of coordinate/units transformations is simple enough that QP has sufficient capability to accomplish these. An interactive QP Session is, in a sense, a programming session, and a prefabricated QPS is functionally equivalent to executable code.

For frequently used "standard" transformations, the IPAD installation may provide executable code as part of the data base. DBMS is then directed via the DDL to invoke the transformations at appropriate times. This method provides more efficient operation and is less tedious to the user. The supplying of such code is installation-specific and must be provided by the various IPAD projects.

No separate transformation GPU is to be provided in the IPAD system design because of the tasks dependent characteristics at each installation.

10 CONCLUDING REMARKS

What started out as a "requirement" for twelve GPUs - Figure 10-1, previously shown in Section 4.5, Part I of Volume IV - has ended up as only four: STATUM (Section 2), OPTUM (Section 4), GGP (Section 5) and GDM (Section 7).

Each of these GPUs is specific for its purpose and — with the exception of GDM — complementary to one another. GDM is envisioned as a stand alone capability supporting the board designers. Each of the originally conceived GPUs (Figure 10-1) are treated separately in the subsections to follow.

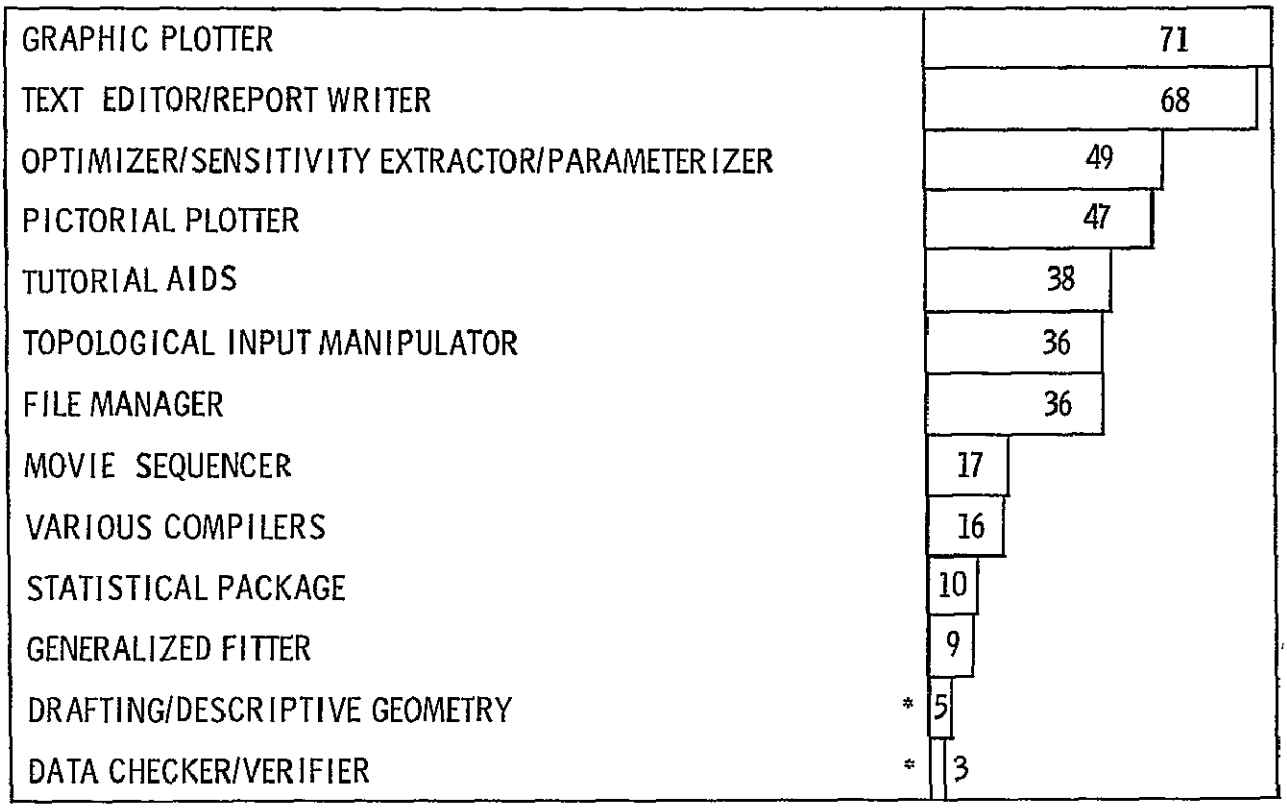


Figure 10-1. Projected Usage of IPAD Interactive Utilities By Questionnaire Respondents

10.1 Graphic Plotter/Pictorial Plotter/Movie Sequencer/ Topological Input Manipulator

The Graphic Plotter (GGP) is the most frequently used utility by the design analyst. It combines the capabilities of graphic plotting, pictorial plotting, topological input manipulating, and movie (pictorial) sequencing (Section 5). One of the 93 OM Questionnaire respondents, 77 (83 percent) would use GGP in one or more of these roles. The role of topological input manipulation is a general capability with widespread implications (see Section 5.2).

10.2 Text Editor/Report Writer

The capabilities of text editing and report writing are met by the host-system supplied software. The host system text editors (Section 3) provides all the capabiliti required by the IPAD user for editing character-coded, sequentially-ordered files suc as report text (e.g., this page) or computer program source code. The host system query processor (QP in IPAD discussions) provides the user with a wide spectrum of report writing capability through the host-system supplied data base management subsystem (DBMS). QP has the capability to provide even the text editing functions, but not as easily or directly as a text editor. Thus, it was found that there was no need to provide a separate GPU for text editing or report writing.

10.3 Optimizer/Sensitivity Extractor/Parameterizer

The Optimizer/Sensitivity Extractor/Parameterizer (OPTUM) was seen to be nothing more than a collection of software modules (each called "OPTUM") representing optimization techniques (Section 4). The organizational structure provided by IPAD allowed for the selection of the correct technique by the user (via the rather strange vehicle of employing a TCSS) and linking these together in a optimization loop (via the TCS resulting from the TCSS and various TCSs resulting from process integration - see Section 1.3). The optimization process optionally drew on GGP for graphic displays. Some element of the collection - viz. some of the many software modules called "OPTUM" - provide an interactive interface with the IPAD user.

10.4 Tutorial Aides

The requirement for tutorial aids to support any IPAD OM was seen to be a direct application of QP (Section 8). A specific application for a given OM is a collection of QPSs.

10.5 File Manager

File management with IPAD became more than envisioned in Volume IV, Part I. File management via DBMS encompasses more than the mere management of "files" (viz. the management of RECORDs, AREAs, SCHEMA modules, SUBSCHEMAs, etc.). QP provides the capability to manage the files under the jurisdiction of DBMS. The use of conventional files is discouraged (Section 6.2 of Part II).

10.6 Various Compilers

The requirement for compilers envisioned in Volume IV, Part I is adequately handled by host system supplied software. (For example, CDC provides BASIC, COBOL, FORTRAN, ALGOL and JOVIAL which are among the most frequently used programming languages. They are also working on a PL/1 compiler.) Beyond these, the system also provides for DBMS support by supplying compilers for the DDLs (as well as DML-enhancements to the various major-language compilers, e.g., FORTRAN and COBOL). Again, no separate IPAD development is required here.

10.7 Statistical Package

The statistical package (STATUM) is an interactive capability to accomplish a wide spectrum of statistical tasks (Section 2). Whereas STATUM could have employed the approach of OPTUM by providing its tutorials via a TCSS, it was reasoned that - due to the complexity of statistical analysis and the diverse background of a potential user - "personalized" tutorials were best supplied interactively as part of STATUM. (Recall that a user of OPTUM will - of necessity - be highly qualified in both optimization techniques and data base management. Refer to Section 4.5 for details.) The minimal demand for STATUM (10 of the 93 respondents or 10.7 percent) makes it an ideal candidate for a phased release of IPAD. Note that all other IPAD system software is recommended for the first release capability.

10.8 Generalized Fitter

The requirement for a general curve or surface fitter was seen to be a rather simple application of an OPTUM software module interacting with the data base and GGP (Section 6). It is envisioned that a special tutorial TCSS would be provided to interface the less qualified user with OPTUM. With this addition, OPTUM becomes the second ranked GPU (behind GGP) with 58 of the 93 OM Questionnaire respondents (62 percent) employing it.

10.9 Drafting/Descriptive Geometry/Data Checker/Verifier

Figure 10-1 is misleading with respect to the requirement for the design (drafti module, GDM. The OM Questionnaire did not adequately account for the board design er since he is not currently adequately represented by computer software (OMs). (See Section 4 of Part I for details.) It is envisioned that GDM will be the most frequently used GPU in a fully deployed IPAD system.

In the case of GDM (Section 7), the approach taken was to provide for both desig and design analysis (of the type a board designer routinely accomplishes) in a single module (GDM). In this sense, GDM is a joint development of a GPU and a collection of OMs for board designers. It is noted that GDM has a significant contribution to Computer Aided Manufacturing (CAM) as well (Subsection 7.1.4). Note that it is GDM only that has the requirement for data checking or verifying.

GDM represented the only GPU within IPAD that had a firm requirement for a minicomputer. (The other GPUs could benefit from use of a minicomputer to service several terminals - see Section 5.3 of Part I, Volume IV - but this merely improved response time at the interactive device.) These requirements lead to a division of work between the maxi and mini (Section 7.6) and to some problems as well.

If the display data is to be kept at the minicomputer, then adequate local mass storage must be provided. This generally raises the terminal cost significantly. While minicomputer costs have been going down (see related discussion in Volume IV, Section 5.3 of Part I) the peripheral devices have not followed at the same rate, and now, they can represent the major part of the cost of a terminal system. If an integrated (computations and display) data structure is maintained at the host computer (via DBMS), then usually higher speed communication lines (adding to terminal cost) are required to handle all the data within response times acceptable to the user (a human factors consideration).

Several other factors that must be included are of an operational nature. If the minicomputer has adequate I/O peripherals and mass storage, it can be used essentially in a standalone mode (independent of host computer load, priorities, failures, etc.) and access the main computer only when the program requirements (analytical, large arrays, fast computation, etc.) dictate the use of the larger computer. This, of course, means programming for two computer, but experience has shown that this can be done with not too much difficulty with compatible languages. An example of this has been shown in the software system designed by Pat Hanratty of Integrated Computer Systems, now a part of Systems, Science and Software (S^3). This system

encompasses CAD drafting, N/C and circuit mask layout programs. It is highly modular, has extensible data base capability and is very machine independent. It has been implemented on REDCOR, IBM 1130, CDC 6600, PDP 9, and other computers including refresh CRTs such as IBM 2250 and Vector General D2, and direct view storage tubes such as COMPUTEK 400 series and TEKTRONIX 4002A and 4010. Regardless, the interface between DBMS and the companion capability on the mini must be addressed by GDM's developer.

10.10 Coordinate/Units Transformations

Not present on Figure 10-1 — but required in an IPAD environment — is a coordinate and units transformation capability. As noted in Section 9, this capability is most adequately provided by QP/DBMS.

REFERENCES - PART III

1. Jones, J. L.: CODASYL Data Base Task Group Report, (no report number), Conference on Data Systems Languages, April 1971.

2. Anon.: System/360, Scientific Subroutine Package (SSP), Version III. Document GH 20-0205-4, Fifth Edition, International Business Machines Corp., Aug. 1970.

3 Anon.: IMSL Library 1 (IBM 370/360). IMSL Manual, Edition 2, International Mathematical and Statistical Libraries (IMSL), Inc., Houston, Texas, 1972.

4 Anon.: IMSL Library 2 (UNIVAC 1100 Series). IMSL Manual, Edition 2, International Mathematical and Statistical Libraries (IMSL), Inc., Houston, Texas, 1972.

5. Anon.: IMSL Library 3 (CDC 6000 Series). IMSL Manual, Edition 2, International Mathematical and Statistical Libraries (IMSL), Inc., Houston, Texas, 1972.

6. Crow, Edwin L.: Statistics Manual. Dover Publications, Inc., 1960.

7. Anon.: INTERCOM Reference Manual, Models 72, 73, 74 Version 4, 6000 Version 4. SCOPE Reference Manual 60307100, Revision B, Control Data Corp., Aug. 1972.

8. Anon.: IBM System/360 Operating System, Time Sharing Option, Command Language Reference. System/360 Reference Manual, File No. S360-36, International Business Machines Corp., Sept. 1971.

9. Anon.: UNIVAC 1100 Series Operating System, Programmer Reference. UNIVAC Programming Manual UP-4144, Revision 2, UNIVAC Division of Sperry Rand Corp., 1971.

10. Jamison, Floyd L.: Comparative Operating Systems, ACM Symposium. (RCA File Editor p. 7 and G. E.'s GECOS III text editor, p. 72). Brandon/ Systems Press, 1969.

11. Anon.: DEC System-10 Users Handbook. DEC System-10 Handbook Series, DEC-10-NGZA-D. Digital Equipment Corp., 1972.

12. Hague, D. S.; and Glatt, C. R.: An Introduction to Multivariable Search Techniques for Parameter Optimization. NAS2-4507, April, 1968.

13. Fletcher, R.; and Powell, M. J. D.: A Rapidly Convergent Descent Method for Minimization. Computer Journal, Vol. 6, 1963.

14. Gass, S. I.: Linear Programming. McGraw-Hill Book Co., Inc., 1964.

15. Hovanessian, S. A.; and Pipes, L. A.: Digital Computer Methods in Engineerir McGraw-Hill Book Co., Inc., 1969.

16. Zoutendijk, G.: Methods of Feasible Directions. Elsevier, 1960, pp. 68-71.

17. Powell, M. J. D.: An Efficient Method for Finding the Minimum of a Function of Several Variables Without Calculating Derivatives. Computer Journal, Vol. 7, 1964.

18. Zangwill, W. I.: Minimizing a Function Without Calculating Derivatives. Computer Journal, Vol. 11, 1967.

19. Johnson, I. L.; and Myers, G. E.: One-Dimensional Minimization Using Search by Golden Section and Cubic Fit Methods. NASA-MSC Internal Note No. 67-FM-172, Nov., 1967.

20. Fiacco, A. V.; and McCormick, G. P.: Nonlinear Programming: Sequential Unconstrained Minimization Techniques. John Wiley & Sons, Inc., 1968.

21. Allran, R. R.; and Johnsen, S. E. J.: An Algorithm for Solving Nonlinear Programming Problems Subject to Nonlinear Inequality Constraints. Computer Journal, Vol. 13, 1970.

22. Bellman, Richard E.: Adaptive Control Processes: A Guided Tour. Princeton University Press, 1961.

APPENDIXES

APPENDIX A
GLOSSARY OF IPAD ACRONYMS AND SELECTED TERMINOLOGY

APPENDIX A

GLOSSARY OF IPAD ACRONYMS AND SELECTED TERMINOLOGY

Throughout the IPAD feasibility study - as concepts were formulated and designs envisioned - acronyms and special terminology were evolved to represent these concepts in a concise and easily recognized form. These acronyms soon became inseparable from the concepts they represented and found their way into all discussions, presentations and documentations concerning IPAD. Although unfortunate from the casual reader's standpoint, the use of acronyms is a tool which the system designers - especially those involved in conceptual design - tend to rely on and incorporate into their thought processes.

It is with these apologetic thoughs in mind that Appendix A, which contains most of the acronyms and special terminology used throughout this report, is presented.

ANSC — AMERICAN NATIONAL STANDARDS COMMITTEE...ANY STANDING COMMITTEE OF ANSI

ANSI — AMERICAN NATIONAL STANDARDS INSTITUTE...AN INSITIUTE FOR INDUSTRY AND

AREA — A CODASYL CONCEPT...A NAMED SUB-DIVISION OF THE ADDRESSABLE STORAGE SPACE IN THE DATABASE AND MAY CONTAIN OCCURRENCES OF RECORDS AND SETS OR PARTS OF SETS OF VARIOUS TYPES. AREAS MAY BE OPENED BY AN OM. THE CONCEPT OF AREA ALLOWS THE DBA TO SUBDIVIDE A DB RATHER THAN CONSIDER THE DB AS A SINGLE UNIT. THE USE OF AREAS ALLOWS THE DBA OR THE DBMS TO CONTROL PLACEMENT OF AN ENTIRE AREA TO PROVIDE EFFICIENT STORAGE AND RETRIEVAL. THE OPENING OF AREAS BY OMS OR TCS ALSO GIVES USERS AN OPPORTUNITY TO OPTIMIZE ACCESS TO THE DB SINCE THE OM HAS NARROWED THE RANGE OF INTEREST IN THE DB TO A RELATIVELY SMALL NUMBER OF SUBDIVISIONS OF THE ENTIRE DB.

ARPA — ADVANCED RESEARCH PROJECT AGENCY.

CGDS — CONSOLIDATED GRAPHICS DATA STREAM.

CIO — CIRCULAR I/O..THE BASIC READ/WRITE PP UTILITY OF THE SCOPE OPERATING SYSTEM.

CM — CENTRAL MEMORY.

CMS — CONVERSATIONAL MONITORING SYSTEM...IBM 370/145,158,168 CONVERSATIONAL TIMESHARING SUBSYSTEM (SOFTWARE)....ALSO USED TO DENOTE CAMBRIDGE MONITOR SYSTEM...IBM 360/67 CONVERSATIONAL TIMESHARING SUBSYSTEM (SOFTWARE).

COBOL — COMMON BUSINESS ORIENTED LANGUAGE... A PROGRAMMING LANGUAGE DEVELOPED BY CODASYL (CIRCA 1960) TO PROVIDE THE BUSINESS COMMUNITY WITH A LANGUAGE SUPERIOR TO THOSE PROVIDED BY COMPUTER MANUFACTURERS AND INDEPENDENT OF ANY MANUFACTURER. PRINCIPAL FEATURES ARE
1. ENGLISH-LIKE SYNTAX,
2. POWERFUL, COMPLETE DATA-DEFINITION FACILITIES ISOLATED FROM PROCEDURE, AND
3. POWERFUL LOGIC AND CONTROL FACILITIES COMBINED WITH CHARACTER MANIPULATION CAPABILITY.

CODASYL — CONFERENCE ON DATA SYSTEMS LANGUAGES...AN INFORMAL GROUP OF PEOPLE BROUGHT TOGETHER THROUGH THEIR COMMON INTERESTS IN DEVELOPING AND STANDARDIZING DATA SYSTEM LANGUAGES. THE GROUP MEMBERS ARE SPONSORED BY THEIR RESPECTIVE COMPANIES AND MEET PERIODICALLY TO REVIEW THEIR WORK AND DISCUSS DEVELOPMENTS. (SEE APPENDIX E.)

CP — CENTRAL PROCESSOR...CDC TERMINOLOGY FOR THE CENTRAL ARITHMETIC UNIT IN A COMPUTATIONAL SYSTEM.

CPU — CENTRAL PROCESSOR UNIT...SEE CP.

CRT — CATHODE RAY TUBE DISPLAY OF INTERACTIVE DEVICE. COULD BE THREE TYPES

DVST — DIRECT VIEW STORAGE TUBE...RETAINS IMAGE WITH SLOW DECAY UNTIL REPAINT

REFRESH — REFRESHED CRT...THE IMAGED IS CYCLED 30 TO 40 TIMES/SEC AND TO REFRESH THE CRT TO PREVENT FLICKER.

TV — TELEVISION (CONTINUOUS RASTER SCAN)...RESOLUTION LIMITED BY RASTER.

CTS — CONVERSATIONAL TIMESHARING SYSTEM...UNIVAC CONVERSATIONAL TIMESHARING SUBSYSTEM (SOFTWARE).

DATA-AGGREGATE — A CODASYL CONCEPT...A NAMED COLLECTION OF DATA ITEMS WITHIN A RECORD. THERE ARE TWO TYPES...VECTORS AND REPEATING GROUPS. A VECTOR IS A ONE-DIMENSIONAL, ORDERED COLLECTION OF DATA-ITEMS, ALL WHICH HAVE IDENTICAL CHARACTERISTICS. A REPEATING GROUP IS A COLLECTION OF DATA THAT OCCURS AN ARBITRARY NUMBER OF TIMES WITHIN A RECORD OCCURRENCE. THE COLLECTION MAY CONSIST OF DATA-ITEMS, VECTORS AND REPEATING GROUPS

DATA-ITEM — A CODASYL CONCEPT...THE SMALLEST UNIT OF NAMED DATA. AN OCCURRENCE OF A DATA-ITEM IS A REPRESENTATION OF A VALUE.

DB — DATA BASE...THE TOTAL REPOSITORY OF IPAD RELATED DATA ON DISC OR OTHER ON-LINE MASS STORAGE WHICH IS UNDER CONTROL OF THE DBMS. MORE SPECIFICALLY, A DATABASE CONSISTS OF ALL THE RECORD OCCURRENCES, SET OCCURRENCES AND AREAS WHICH ARE CONTROLLED BY A SPECIFIC SCHEMA.

DBA — DATA BASE ADMINISTRATOR (FORMERLY MANAGER)...THE PERSON OR GROUP OF PEOPLE RESPONSIBLE FOR THE DB IN GENERAL AND THE MDB IN PARTICULAR. THE TERM DATA BASE MANAGER (DBM) USED IN THE ORIGINAL CONCEPTUAL DESIGN WAS CHANGED TO DATA BASE ADMINISTRATOR (DBA) TO ADHERE TO MORE CONVENTIONAL COMPUTER TERMINOLOGY WHICH USES 'MANAGER' TO MEAN COMPUTER SOFTWARE.

DBMS — DATA BASE MANAGEMENT SYSTEM...A CONCEPT DISCUSSED IN THE CODASYL DBTG REPORT AS THE CORE SOFTWARE SYSTEM SUPPORTING DATA MANAGEMENT.

DBTG — DATA BASE TASK GROUP...THE CODASYL COMMITTEE GIVEN THE RESPONSIBILITY FOR INVESTIGATING THE REQUIREMENTS AND DETAILING THE SPECIFICATIONS FOR A COMPREHENSIVE DATA MANAGEMENT SYSTEM. THESE RECOMMENDATIONS ARE

	CONTAINED IN THEIR APRIL 1971 REPORT. SEE ALSO APPENDIX E.
DDL	DATA DEFINITION LANGUAGE...A CODASYL DBTG CONCEPT. THE DDL IS THE LANGUAGE USED FOR DESCRIBING A DATABASE, OR THAT PART OF A DATABASE KNOWN TO A PROGRAM.
DE	DISCIPLINARY ENGINEER...MEMBER OF ONE OF THE DISCIPLINARY GROUPS.
DG	DESIGN GROUP OR DISCIPLINE GROUP.
DGL	DISCIPLINARY GROUP LEADER...RESPONSIBLE FOR LEADING A PARTICULAR DISCIPLINE.
DLF	DISCIPLINARY LIBRARY FILE...THE DB AREA RESERVED FOR A DG LIBRARY.
DMCL	DEVICE MEDIA CONTROL LANGUAGE...A CODASYL DBTG TERM FOR THE DEVICE OR MEDIA SELECTION FOR DATA AND FOR THE CONTROL OF WHERE AND HOW THE DATA IS TO RESIDE ON THE DEVICE (EG. RECORDING MODE, FORMAT, BLOCKING AND THE LIKE). TO A CERTAIN EXTENT DMCL OVERLAPS OSCL.
DML	DATA MANIPULATION LANGUAGE...A CODASYL DBTG CONCEPT. THE DML IS THE LANGUAGE WHICH THE PROGRAMMER USES TO CAUSE DATA TO BE TRANSFERRED BETWEEN HIS PROGRAM AND THE DATA BASE.
DMS	DATA MANAGEMENT SYSTEM...THE COLLECTION OF DATA MANAGEMENT SUPPORT SOFTWARE FOR IPAD, CONSISTING OF THE DBMS AND QP FUNCTIONAL ENTITIES TOGETHER WITH THEIR SUPPORT UTILITIES AND COMPILERS (FOR DDL AND DML).
DOD	DEPARTMENT OF DEFENSE.
DVST	SEE CRT, DVST.
EOF	END OF FILE...A STATEMENT THAT THE END OF A FILE HAS BEEN REACHED.
EOI	END OF INFORMATION...CDC TERMINOLOGY THAT THE END OF THE LAST FILE HAS BEEN REACHED AND NO FURTHER INFORMATION EXISTS BEYOND THIS POINT.
EOR	END OF RECORD...A STATEMENT THAT THE END OF A RECORD WITHIN A FILE HAS BEEN REACHED.
ERB	ENGINEERING REVIEW BOARD...THE BOARD OR PANEL, A WING OF MANAGEMENT, GIVEN RESPONSIBILITY FOR REVIEWING THE ENGINEERING DESIGN FOR ADEQUACY AND CONTROLLING THE TECHNICAL ASPECTS OF THE PROJECT.
ERBC	ENGINEERING REVIEW BOARD COORDINATOR...THE PERSON RESPONSIBLE FOR SCHEDULING THE ITEMS BEFORE THE ERB AND PREVIEWING THESE ITEMS.
ERS	EXTERNAL REFERENCE SPECIFICATION...A CDC TERM DENOTING A SPEC COVERING SYSTEM SOFTWARE IN DEVELOPMENT BUT WITH RELATIVELY FIRM SPECIFICATION. THE DOCUMENT, WHEN DISTRIBUTED EXTERNALLY TO CDC, GENERALLY CONTAINS THE DISCLAIMER...THIS DOCUMENT IS A WORKING PAPER ONLY AND DOES NOT NECESSARILY REPRESENT ANY OFFICIAL INTENT ON THE PART OF CONTROL DATA CORPORATION.
EXEC	REFERRING TO THE SOFTWARE (CODE) WHICH PERFORMS THE IPAD EXECUTIVE FUNCTION.
EXPANDER	EXPANDER INTERACTIVE UTILITY...ASSISTS USER IN CONSTRUCTING A SPECIFIC TCS (OR QP SESSION) FROM A GENERAL TCSS (OR QPSS).
GDM	GENERAL DESIGN MODULE...THAT IPAD UTILITY THAT PROVIDES THE DRAFTSMAN AND BOARD DESIGNER WITH THE TOOL SUFFICIENT FOR HIS DESIGN TASKS.
GGL	GENERAL GRAPHICS LIBRARY..IPAD'S ANSWER TO SUPPLYING A STANDARDIZATION OF THE FORTRAN CALLING SEQUENCE FOR GRAPHICS APPLICATIONS. THE NOW EXISTING MANUFACTURER SUPPLIED SOFTWARE BECOMES GGL WHEN REPACKAGED IN ACCORDANCE WITH THIS STANDARDIZATION.
GGP	GENERAL GRAPHICS PLOTTER...THAT SINGLE IPAD UTILITY RESPONSIBLE FOR PROVIDING ALL GRAPHIC AND PICTORIAL PLOTTING WITHIN IPAD.
GID	GENERAL INTERNAL DESIGN...A CDC TERM DENOTING A SPEC COVERING SYSTEM SOFTWARE IN DEVELOPMENT BUT WITH SPECS SO FLEXIBLE AS TO NOT JUSTIFY AN ERS. THE DOCUMENT, WHEN DISTRIBUTED EXTERNALLY TO CDC, GENERALLY CONTAINS THE DISCLAIMER...THIS DOCUMENT IS A WORKING PAPER ONLY AND DOES NOT NECESSARILY REPRESENT ANY OFFICIAL INTENT ON THE PART OF CONTROL DATA CORPORATION.
GPGT	GENERAL PURPOSE GRAPHICS TERMINAL...CDC SUCCESSOR TO THEIR LARGE CRT 274 TERMINAL (EXPECTED DELIVERY,FALL 1973).
GPL	GRAPHIC PROGRAMMING LIBRARY...UNIVAC TERM FOR THEIR HIGHER LEVEL INTERACTIVE GRAPHICS SUPPORT SOFTWARE (FORTRAN CALLABLE). GPL PROVIDES FOR EASE OF MANIPULATING DATA STRUCTURES AS WOULD BE REQUIRED TO DEFINE AND DESCRIBE A DISPLAY IMAGE. IT ALSO CONTAINS INTERACTIVE CONTROL AND GRAPHICS DISPLAY CALLS. UNIGRASP IS A FUNCTIONAL SUBSET OF GPL.
GPU	GENERAL PURPOSE UTILITY...CHARACTERIZED AS A PROGRAM OUTSIDE THE SCOPE OF A TYPICAL OM WHICH IS OF GENERAL USE TO THE IPAD USER.
GSF	GENERAL SURFACE (CURVE) FITTING... THE TASK OF FITTING CURVES OR SURFACES (INCLUDING HYPER-SURFACES) TO DATA.
GSP	GRAPHIC SUBROUTINE PACKAGE...IBM TERM FOR THEIR INTERACTIVE GRAPHICS SUPPORT SOFTWARE (FORTRAN CALLABLE).

ICL	IPAD CONTROL LANGUAGE...THE LANGUAGE FOR CONTROLLING IPAD PROCESSES THROUGH THE EXEC OF THE HOST OPERATING SYSTEM, THE OSCL IS A PART OF IPAD'S ICL.
IDEF	SHORT FOR INPUT DEFINITION...THAT PORTION OF THE OM SUBSCHEMA SOURCE THAT DESCRIBES ALL POSSIBLE INPUT REQUIRED TO SUPPORT THAT OM.
IDM	INTERACTIVE DATA MANAGER...A (PRINCIPALLY) FORTRAN REWRITE BY NSRDC OF THE CDC VERSION OF THE DATA HANDLER WITH ADDED IMPROVEMENTS.
IGS	INTERACTIVE GRAPHICS SYSTEM...CDC TERM FOR THEIR INTERACTIVE GRAPHICS SYSTEM SOFTWARE (FORTRAN CALLABLE). CURRENT VERSION IS 2.0.
IGS V.1	VERSION 1.0 OF IGS.
IGS V.2	VERSION 2.0 OF IGS.
IMS	INTERNAL MAINTENANCE SPECIFICATIONS...CDC TERMINOLOGY, THE SOFTWARE DOCUMENTATION CDC PREPARES FOR MAINTAINING THEIR SOFTWARE SYSTEMS.
IMSL	INTERNATIONAL MATHEMATICAL AND STATISTICAL LIBRARIES...IMSL IS A LIBRARY OF ABOUT 200 GENERAL PURPOSE MATHEMATICAL AND STATISTICAL SUBROUTINES CODED IN FORTRAN IV AND AVAILABLE FOR THE IBM 360/370, THE CDC 6000 SERIES AND THE UNIVAC 1100 SERIES COMPUTERS.
INTERCOM	THE INTERACTIVE (TIME-SHARING) SUBSYSTEM OF CDC'S SCOPE 3.0 AND ON.
I/O	INPUT/OUTPUT...REFERS TO BOTH INPUT AND OUTPUT (EG, I/O FILES).
I/ODEF	THE COMBINATION OF THE IDEF AND ODEF OF AN OM.
IOF	I/O FORMATTING...THE TASK OF SUPPLING DATA FOR OM INPUT OR REWORKING, DISPOSITIONING, ETC DATA FROM OM OUTPUT. FORMERLY USED TO DELINEATE THE I/O FORMATTER UTILITY...A CONCEPTUAL UTILITY PRE-DESIGNED TO SOLVE THE IOF FUNCTION PRIOR TO ADOPTING THE CODASYL APPROACH. THE IOF PRE-DESIGN FINALIZED THE REQUIREMENTS, POSTULATED A DESIGN WHICH WOULD MEET THESE REQUIREMENTS AND FOCUS ATTENTION ON THE INHERENT PROBLEMS.
IPAD	INTEGRATED PROGRAM FOR AEROSPACE-VEHICLE DESIGN. USED 2 DISTINCT WAYS 1.FULLY IMPLEMENTED...CONTAINS FULL COMPLEMENT OF OPERATIONAL MODULES. 2.SYSTEM SOFTWARE...JUST SUFFICIENT SYSTEM CODE FOR IPAD (NO OM'S). IPAD IS A SOFTWARE FRAMEWORK INTENDED TO REDUCE THE TIME AND LABOR EXPENDED BY AN (AEROSPACE) ENGINEER IN ACCOMPLISHING HIS ENGINEERING TASK.
ISPONGE	A FILE CONCEPT SUPPORTING THE IOF UTILITY (BEFORE IT WAS REPLACED BY QP). THE ISPONGE WAS A RANDOM FILE INTERMEDIARY BETWEEN THE INPUT FILE AND THE IOF FUNCTION. THE ISPONGE HAS BEEN REPLACED BY THE UF AREA OF THE SCHEMA AND THE IOF BY THE QP AND QPS (QPSS).
LSI	LARGE SCALE INTEGRATION ... AN ELECTRONICS APPROACH TO MICRO-MINIATURE CIRCUITRY.
MACRO	LARGE OR OF THE HIGHEST ORDER.
MAXI	USUALLY REFERRING TO A LARGE SCIENTIFIC COMPUTER SYSTEM (INCLUDING CONSIDERABLE SUPPORTING SYSTEM SOFTWARE).
MDB	MULTIDISCIPLINARY DATA BANK...THE DB AREAS RESERVED FOR 'BLESSED' DATA WHICH IS READ-ONLY FOR THE USER AND UNDER THE STRICT PROJECT CONTROL OF THE DBA AND HIS PEOPLE.
MDBU	MDB UPDATE FILE...THE DB AREA CONTAINING COMBINATIONS OF QPS, TCS, AND DESIGN DATA THAT PERMITS DISPLAY AND REVIEW OF THAT DATA BY THE DBA PRIOR TO INCORPORATION OF THAT DATA INTO THE MDB.
MENU	A TABLEAU OR LIST OF ITEMS ON A GRAPHICS TERMINAL, ONE OR MORE OF WHICH ARE MEANT TO BE SELECTED, GENERALLY BY A TOPOLOGICAL INPUT DEVICE, EG A LIGHT PEN.
MICRO	SMALL OR OF THE LOWEST ORDER.
MIDI	USUALLY REFERRING TO A MEDIUM COMPUTER, LARGER THAN A MINI BUT SMALLER THAN A MAXI. THE DIVISION SEEMS TO BE MORE ON PHYSICAL SIZE AND COST THAN ON COMPUTING CAPACITY.
MINI	USUALLY REFERRING TO A SMALL COMPUTER USED STANDALONE OR AS A PERIPHERAL TO A MAXI
MUJ	MULTI-USER JOB...A TIME-SHARING JOB THAT CAN BE ACCESSED BY MORE THAN ONE USER AT A TIME.
NSRDC	NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER, CARDEROCK, MARYLAND.
ODEF	SHORT FOR OUTPUT DEFINITION...THAT PORTION OF THE OM SUBSCHEMA SOURCE THAT DESCRIBES ALL POSSIBLE OUTPUT REQUIRED TO SUPPORT THAT OM.
OM	OPERATIONAL MODULE...A FULLY FUNCTIONAL PIECE OF CODE WHICH CAN (HAS) RUN STANDALONE IN BATCH MODE OR IS A FULLY CHECKED OUT INTERACTIVE PROGRAM. AN OM IS USUALLY A FULLY OPERATIONAL EXISTING FORTRAN BATCH PROGRAM WHICH REPRESENTS A PORTION OF A DISCIPLINE'S CAPABILITY.
OPTUM	THE IPAD OPTIMIZER DRIVER IS A GENERAL PURPOSE UTILITY (GPU) FOR OPTIMIZATION, SENSIVITY EXTRACTION, PARAMETERIZATION AND CURVE

OR SURFACE (HYPERSURFACE) FITTING.

OSCL — OPERATING SYSTEM CONTROL LANGUAGE...THE LANGUAGE FOR CONTROLLING A COMPUTER'S OPERATING SYSTEM (SEE APPENDIX C FOR DETAILS)....ALSO AN AD HOC COMMITTEE REPORTING TO SPARC (SEE APPENDIX D).

OSPONGE — A FILE CONCEPT SUPPORTING THE IOF UTILITY (BEFORE IT WAS REPLACED BY QP). THE OSPONGE WAS A RANDOM FILE INTERMEDIARY BETWEEN THE OUTPUT FILE AND THE IOF FUNCTION. THE OSPONGE HAS BEEN REPLACED BY THE UF AREA OF THE SCHEMA AND THE IOF BY THE QP AND QPS (QPSS).

PD — PROJECT DIRECTORY...REFERENCE TO ALL THE DATA BASES WITHIN A PROJECT.

PDB — PROJECT DATA BANK...ALL THE DATA (DESIGN DATA, OMS, ETC.) THAT ARE UNIQUE TO A PARTICULAR PROJECT.

PERT — PROGRAM EVALUATION REVIEW TECHNIQUE...A SYSTEM FOR SCHEDULING ACTIONS AND DEFINING TIME-CRITICAL PATHS.

PLC — PROGRAMMING LANGUAGE COMMITTEE...THE CODASYL DEVELOPMENT GROUP WHICH IS RESPONSIBLE FOR THE DEVELOPMENT OF A LANGUAGE WHERE THE OBJECTIVE IS COMPATIBLE, UNIFORM SOURCE PROGRAMS AND OBJECT RESULTS, WHICH REQUIRE MINIMUM CONVERSION FOR PROGRAM AND DATA INTERCHANGE.

PLOT-10 — A PORTION OF THE TEKTRONIX GRAPHICS SOFTWARE.

PP — PERIPHERAL PROCESSOR...CDC TERMINOLOGY FOR A SEPARATE PROCESSOR USED IN CDC COMPUTER SYSTEM DESIGN TO HANDLE ALL I/O ACTIVITIES PERIPHERAL TO THE CENTRAL PROCESSOR.

PRF — PROJECT REVIEW FILE...THE DB AREA CONTAINING COMBINATIONS OF QPS, TCS, AND DESIGN DATA STRUCTURED TO PERMIT DISPLAY AND REVIEW OF DESIGN DATA BY THE ERB OR CORRESPONDING FUNCTION.

PS — PROJECT SCHEMA...THE TOTAL DDL SPECIFICATION FOR A PROJECT DATA BASE.

PSI — A CDC TERMINOLOGY FOR PROGRAMMING SYSTEMS INFORMATION GROUP.

PSR — PROGRAMMING SYSTEM REPORT...A CDC TERM DENOTING A REPORT COVERING ITEMS OF INTEREST TO OPERATING SYSTEM USERS (EG, DIFFICULTIES ENCOUNTERED, RESULTS OF BENCHMARKS, ETC).

QP — QUERY PROCESSOR...A COMPUTER OPERATING SYSTEM UTILITY TO INTERROGATE AND MAINTAIN MASS STORAGE DATA FILES VIA DBMS. THE QP USER REQUIRES MINIMAL COMPUTER ORIENTATION TO SUBMIT DIRECTIVES THAT WILL CAUSE THE PROGRAM TO LIST REQUESTED INFORMATION, COMPARE DATA, REMOVE AND INSERT ENTRIES OR SELECT AND MODIFY PORTIONS OF THE DATA CONTENT. TUTORIALS ARE ALSO PROVIDED FOR THE BEGINNER, OR TO RECALL A PARTICULAR DIRECTIVE OR ITS SYNTAX TO MIND FOR THE EXPERIENCED USER. QP IS REQUIRED TO OPERATE IN EITHER AN INTERACTIVE OR BATCH ENVIRONMENT. QU WAS USED AS A MODEL FOR QP THOUGHOUT THIS REPORT. SEE ALSO QU.

QPL — QUERY PROCESSOR LANGUAGE...THAT PART OF THE OSCL INSTRUCTING THE HOST COMPUTER'S QP SUBSYSTEM.

QPS — QUERY PROCESSOR SESSION... THE SERIES OF TRANSMISSIONS SENT BY A USER BETWEEN THE SIGN-ON AND SIGN-OFF MESSAGES SENT BY QP. EACH SESSION OR PORTIONS THEREOF MAY BE RECORDED FOR RE-EXECUTION AT A LATER TIME WITHOUT BEING RECONSTRUCTED. SEE ALSO QP, QPSS, AND EXPANDER. A QPS IS A SPECIAL FORM OF A TCS. SEE TCS AND TCSS.

QPSS — QUERY PROCESSOR SESSION SKELETON...A FILE CONSISTING OF AN INCOMPLETE SESSION, NAMES OF PARAMETERS NEEDED TO COMPLETE THE SESSION, POINTERS TO PLACEMENT OF THESE PARAMETERS TO COMPLETE THE QPS, AND TUTORIALS DIRECTING THE TASK AND DESCRIBING THE GENERAL FUNCTION SUPPLIED BY THE SESSION WHEN CONSTRUCTED. THE EXPANDER UTILITY (SEE) CONSTRUCTS A QPS FROM A QPSS. A QPSS IS A SPECIAL FORM OF A TCSS.

QSS — QUOTE SPECIAL SOFTWARE...CDC TERM FOR A BID ON SPECIAL SOFTWARE TO SUPPORT A SPECIFIC INSTALLATION.

QU — QUERY UPDATE...A CDC SCOPE 6000 SERIES COMPUTER PROGRAM TO INTERROGATE AND MAINTAIN MASS STORAGE DATA FILES. QU/1 (VERSION 1.0) OPERATES UNDER INTERCOM 4.1 AND WAS DELIVERED WITH SCOPE 3.4.0. QU/2 (VERSION 2.0) IS SCHEDULED FOR DELIVERY WITH SCOPE 3.4.1. UNLESS SPECIFICALLY DENOTED, ALL REFERENCE TO QU IS TO QU/2. SEE ALSO QP.

QU/1 — SEE QU.

QU/2 — SEE QU.

QUS — QUERY UPDATE SESSION... THE SERIES OF TRANSMISSIONS SENT BY A USER BETWEEN THE SIGN-ON AND SIGN-OFF MESSAGES SENT BY QU. EACH SESSION OR PORTIONS THEREOF MAY BE RECORDED FOR RE-EXECUTION AT A LATER TIME WITHOUT BEING RECONSTRUCTED. SEE ALSO QU, QP AND QPS.

RECORD — A CODASYL CONCEPT...A NAMED COLLECTION OF ZERO, ONE OR MORE DATA-ITEMS OR DATA-AGGREGATES. THERE MAY BE AN ARBITRARY NUMBER OF OCCURRENCES IN THE DATABASE OF EACH RECORD TYPE SPECIFIED IN THE SCHEMA FOR THAT DATABASE. THE DISTINCTION BETWEEN THE ACTUAL OCCURRENCES OF A RECORD

AND THE TYPE OF THE RECORD IS AN IMPORTANT ONE

REFRESHED SEE CRT, REFRESH.

SAK SCOPE ACTUAL KEY...A CDC FILE ORGANIZATION (TYPE).

SCHEMA A CODASYL CONCEPT...THE DB SCHEMA CONSISTS OF THE FULL DESCRIPTION OF DATA CONTAINED WITHIN THE DB.
MORE SPECIFICALLY, A SCHEMA CONSISTS OF DDL ENTRIES AND IS A COMPLETE DESCRIPTION OF A DATABASE. IT INCLUDES THE NAMES AND DESCRIPTIONS OF ALL OF THE AREAS, SET OCCURRENCES, RECORD OCCURRENCES AND ASSOCIATED DATA-ITEMS AND DATA-AGGREGATES AS THEY EXIST IN THE DATABASE.

SCOPE THE HOST COMPUTER OPERATING SYSTEM FOR THE CDC CYBER 70 SERIES MODELS 72 THRU 74. (ALSO KNOWN AS CDC 6000 SERIES COMPUTERS.)

SDA SCOPE DIRECT ACCESS.....A CDC FILE ORGANIZATION(TYPE).

SET A CODASYL CONCEPT...A NAMED COLLECTION OF RECORD TYPES. AS SUCH, IT ESTABLISHES THE CHARACTERISTICS OF AN ARBITRARY NUMBER OF OCCURRENCES OF THE NAMED SET. EACH SET TYPE SPECIFIED IN THE SCHEMA MUST HAVE ONE RECORD TYPE DECLARED AS ITS OWNER AND 1 OR MORE RECORD TYPES DECLARED AS ITS MEMBER RECORDS. EACH OCCURRENCE OF A SET MUST CONTAIN ONE OCCURRENCE OF ITS OWNER RECORD AND MAY CONTAIN AN ARBITRARY NUMBER OF OCCURRENCES OF EACH OF ITS MEMBER RECORD TYPES.

SIP SCOPE INDEX PROCESSOR....A CDC FILE ORGANIZATION (TYPE).

SIS SCOPE INDEX SEQUENTIAL...A CDC FILE ORGANIZATION (TYPE).

SPARC STANDARDS PLANNING AND REQUIREMENTS COMMITTEE OF ANSC X3 (SEE APPENDIX D FOR THE RELATIONSHIP).

SPU SPECIAL PURPOSE UTILITIES...THOSE UTILITIES OF IPAD WHICH ACCOMPLISH SPECIAL PURPOSES, EG THE DML INSERTION PREPROCESSOR, THE SCHEMA ASSEMBLER, AND THE SUBSCHEMA ASSEMBLER.

SSF SUPPORT SYSTEM FILE...THAT FILE OR COLLECTION OF FILES THAT CONTAINS INFORMATION TO BE MAINTAINED WITHIN THE IPAD SYSTEM THAT IS COMMON OR NECESSARY TO ALL USERS OF IPAD.

STATUM THE IPAD STATISTICAL UTILITY MODULE...A GENERAL PURPOSE, INTERACTIVE STATISTICAL PACKAGE.

SUBSCHEMA A CODASYL CONCEPT...A SUBSCHEMA FOR AN OM CONSISTS OF THE DESCRIPTION OF THAT DATA WITHIN THE DB RELATED TO THE OM, DESCRIBING THE DATA AS THE OM WOULD PREFER TO ACCESS IT (INCLUDING NAMES AND FORMAT CHANGES). MORE SPECIFICALLY, A SUBSCHEMA CONSISTS OF DDL ENTRIES DESCRIBING ONLY THOSE AREAS, SETS, RECORDS, DATA-ITEMS AND DATA-AGGREGATES OF THE DB WHICH ARE KNOWN TO ONE OR MORE SPECIFIC OMS, AND IN THE FORM IN WHICH THOSE OMS EXPECT TO ACCESS OR SUPPLY IT.

SWA SCOPE WORD ADDRESSABLE...A CDC FILE ORGANIZATION (TYPE).

SU A BRAND NAME REFERRING TO THE SOFTWARE FIRM SYSTEMS, SCIENCE AND SOFTWARE, LA JOLLA, CALIFORNIA.

TAS TUTORIAL AIDS SUPPORT...THE GENERAL FUNCTION OF PROVIDING ASSISTANCE (E.G. PROMPTING) TO AN INEXPERIENCED IPAD USER.

TCS TASK CONTROL SEQUENCE...ESSENTIALLY A COMMANDS FILE PROVIDING THE ABILITY TO EXECUTE JOB SEQUENCES AUTOMATICALLY. TCS FILES ARE CODED FILES WHICH CAN BE EDITED BY THE SYSTEMS TEXT EDITOR AND EXECUTED BY THE IPAD EXECUTIVE AS REQUIRED.

TCSS TASK CONTROL SEQUENCE SKELETON...A FILE CONSISTING OF AN INCOMPLETE TCS, NAMES OF PARAMETERS NEEDED TO COMPLETE THE TCS, POINTERS TO PLACEMENT OF THESE PARAMETERS TO COMPLETE THE TCS, AND TUTORIALS DIRECTING THE TASK AND DESCRIBING THE GENERAL FUNCTION SUPPLIED BY THE TCS WHEN CONSTRUCTED. THE EXPANDER UTILITY (SEE) CONSTRUCTS A TCS FROM A TCSS.

TDEF SHORT FOR TUTORIAL DEFINITION..THAT PORTION OF THE OM SUPPORTING SUBSCHEMA CONTAINING TUTORIALS TO ASSIST THE USER--THROUGH QP--TO LOAD HIS UF AREA OF THE DB FROM THE MDB. SEE SECTION 8 OF PART III.

TED TEXT (CONTEXT) EDITOR..THAT TEXT (CONTEXT) EDITING CAPABILITY NORMALLY SUPPLIED BY THE HOST COMPUTING SYSTEM SOFTWARE.

TI A BRAND NAME REFERRING TO THE MANUFACTURER TEXAS INSTRUMENTS.

TIM TOPOLOGICAL INPUT MANIPULATOR ...THAT SUB-CAPABILITY OF THE GPU GGP THAT ENABLES THE DEFINITION OF CERTAIN DISPLAY ITEMS TO BE TOPOLOGICAL INPUT ITEMS AT EXECUTION TIME.

TSA TASK STATUS/ACTION FILE...COLLECTION OF DATA RECORDS PERTAINING TO INDIVIDUAL USERS OR GROUPS OF USERS THAT CONTAIN DIRECTIVES FOR ACTIONS OR MESSAGES OF STATUS.

TTY REFERS TO THE GENERAL TYPE OF INTERACTIVE CONSOLE USING THE STANDARD TELETYPEWRITER CHARACTER TRANSMISSION INTERFACE OR AN EXTENSION OF IT. ALSO, A BRAND NAME REFERRING TO THE WELL KNOWN TELETYPEWRITER.

TREE A FILE POINTER ORGANIZATION LOOKING LIKE A TYPICAL COMPANY ORGANIZATION CHART.

TV SEE CRT, TV

UF	USERS FILE...THOSE AREAS WITHIN THE DB RESERVED FOR A SINGLE USER FOR THE PURPOSE OF CONDUCTING HIS TASK.
UNIGRASP	UNIVAC INTERACTIVE GRAPHICS SUPPORT PACKAGE...UNIVAC TERM FOR THEIR INTERACTIVE GRAPHICS SUPPORT SOFTWARE (FORTRAN CALLABLE). UNIGRASP IS VERY SIMILAR TO GSP.
USER	ANY IPAD USER, GENERALLY CHARACTERIZED AS AN ENGINEER NOT EXPERIENCED WITH COMPUTERS--AT LEAST TO ANY GREAT EXTENT--BUT INTIMATELY FAMILIAR WITH THE PROBLEM HE IS TRYING TO SOLVE.
UTT	USER TASK TRAJECTORY...A SYNOPSIS OF THE PERTINENT JOB STEPS A USER EXECUTES WHILE OPERATING WITHIN IPAD, HENCE HIS TRAJECTORY OR TRACK. ALSO THE AREA OF THE DB WHERE THE UTT IS STORED BY THE IPAD EXEC. THE RECORDING OF A UTT IS OPTIONAL WITH THE INSTALLATION.
VM	VIRTUAL MEMORY...ON A TIME-SHARING SYSTEM, THE STORAGE SPACE EACH USER APPEARS TO HAVE FOR HIS OWN USE. ALSO VIRTUAL MACHINE... AN IBM 370 CONCEPT IN WHICH EACH VM APPEARS TO HAVE ITS OWN COMPLETE MACHINE (INCLUDING A SPECIFIC OPERATING SYSTEM) FOR ITS OWN USE.
2D	TWO-DIMENSIONAL.
3D	THREE-DIMENSIONAL.
6RM	THE SCOPE RECORD MANAGER SYSTEM CODE. 6RM TREATS A WIDE VARIETY OF FILE TYPES (EG SIS,SDA,SWA,SIP, AND SAK).

APPENDIX B
INDUSTRY EXPERIENCE WITH INTERACTIVE GRAPHICS, A LITERATURE SURVEY

(Refer to Volume IV for this Appendix)

APPENDIX C
REPORT TO SPARC FROM AD HOC COMMITTEE ON OPERATING SYSTEM CONTROL LANGUAGE

(Refer for Volume IV for this Appendix)

APPENDIX D
AMERICAN NATIONAL STANDARDS INSTITUTE (ANSI)

(Refer to Volume IV for this Appendix)

APPENDIX E
CONFERENCE ON DATA SYSTEMS LANGUAGES (CODASYL)

APPENDIX E

CONFERENCE ON DATA SYSTEMS LANGUAGES (CODASYL)

The following text is copied from the handbook "The World of EDP Standards", written by Marjorie F. Hill (Control Data Corporation Technical Memo TM 4, September 1972). It is presented as a convenience to the reader with the kind permission of CDC.

E.1 History

Late in May of 1959 a meeting was held in the Pentagon to consider the desirability and the feasibility of establishing a common language for the programming of business-type applications. Present at the meeting were representatives of users, both those in the private sector and those in government, computer manufacturers, and other interested parties.

At this meeting the concept of three committees was agreed upon and the Short Range, Intermediate Range and Long Range Committees were established. The Short Range Committee eventually became the official COBOL branch of CODASYL and the Intermediate and the Long Range Committees evolved into the Systems and Language Structures Committees respectively.

At the initial meeting the Short Range Committee was given the task of developing an immediate language and was instructed to take the best of three existing language compiler systems -- FLOWMATIC, AIMACO, and Commercial Translator -- and to produce a language superior to any of these.

By September 1959 this committee had specified a language and by December 1959 COBOL existed as a language that was not identified with any manufacturer. The initial specification for COBOL as published in April 1960 has since become known as COBOL-60.

In 1961 a portion of the Intermediate Range Committee was combined with the Long Range Committee to form the Development Committee. Out of this group came a Decision Table Structured Language (Detab X) and a nonprocedural approach to problem statement identified as "Information Algebra".

Recognizing the evolving nature of the language, CODASYL, through its working committees, has published several editions of the COBOL language.

.2 Objectives

CODASYL is dedicated to the development of Data Systems Languages independent of any make or model of computer, and provides a forum for the exchange of ideas and knowledge related to those languages. Standardization is the responsibility of the appropriate groups of ANSI/X3.

E.3 Membership

Membership in CODASYL is through membership on a standing committee, which accord membership according to their individual rules. In the case of the Programming Languages Committee (PLC), there is a twenty-five member limit and the added stipulation that not more than two-thirds of the members can be from any one segment of the, industry, i.e., users or implementors.

Membership in one of the subcommittees does not constitute membership in the parent standing committee.

E.4 Organization

CODASYL is organized as five standing committees: The Executive Committee, the Planning Committee, the Systems Committee, the Programming Languages Committee and the recently organized Data Description Language Committee. However, the organization functions as shown in Figure E-1 with the Executive Committee having final authority and the Planning Committee acting in an advisory capacity.

E.4.1 The Executive and Planning Committees. - The Executive Committee is composed of not more than fifteen members, selected as being individuals who have made significant contributions to the advancement of the goals of CODASYL. In addition, the chairman of each standing committee is a member of the Executive Committee. The Chairman is elected by the committee members at the first meeting of each calendar year. The vice chairman and secretary are appointed by the chairman.

The Executive Committee provides policy guidance and direction to the other standing committees, establishes publication policies, approves all formal publications of CODASYL, provides membership policy, appoints chairmen of the other standing committees, and reviews membership in the other standing committees periodically.

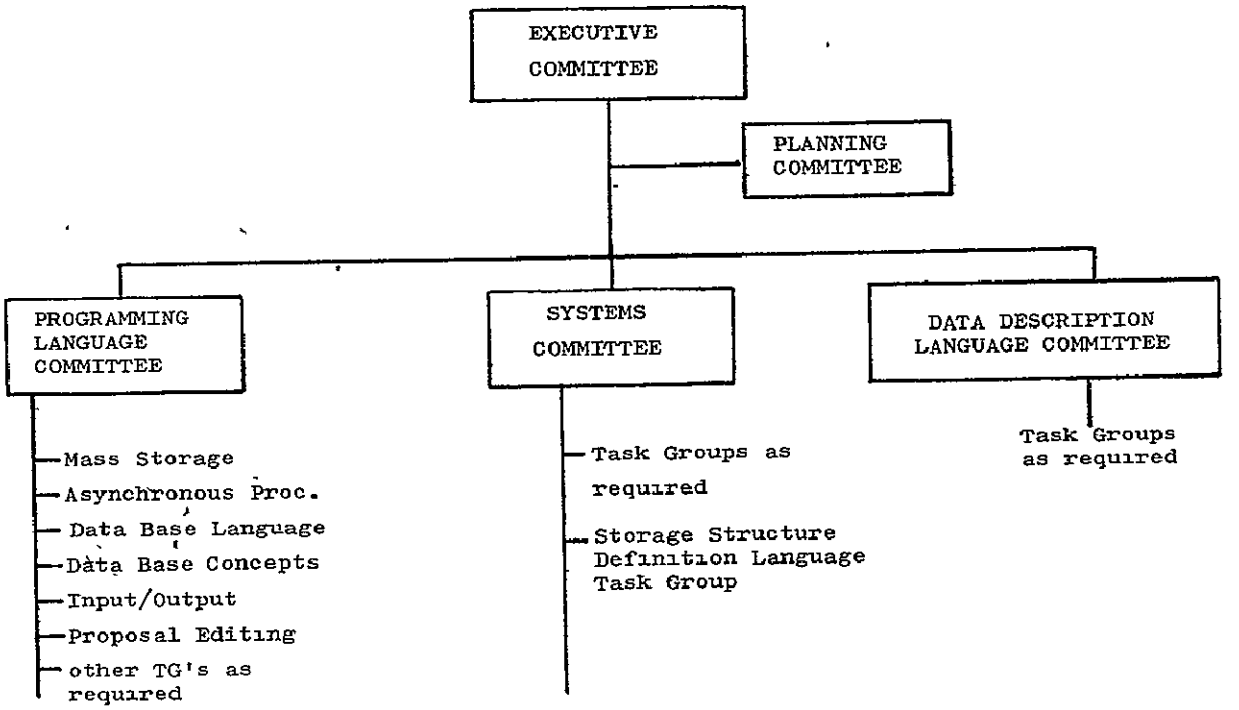


Figure E-1. Functional Organization of CODASYL

The Planning Committee acts in an advisory capacity to the Executive Committee and is responsible for gathering, and disseminating information from implementors and users which is aimed at fulfilling or extending the long range goals of CODASYL. The Chairman is appointed by the Executive committee and the chairman appoints a vice chairman. Mot of the manufacturer's user groups are members, as well as the Association of Independent Software companies and the National Association for State Information Systems.

The Planning Committee may approve release of committee papers. However, final approval as a CODASYL release, is by Executive Committee action.

E.4.2 The Technical Development Committees. - The Programming Language Committee (PLC) is a development group responsible for the development of a language where the objective is compatible, uniform source programs and object results, which require minimum conversion for program and data interchange. The committee concentrates its efforts on the tools and techniques needed by applications programmers.

The Systems Committee's purpose is to build up an expertise in, and to develop, advanced languages and techniques for data processing, with the aim of automating as much as possible of the processes currently thought of as systems analysis, design and implementation.

The recently organized Data Description Language Committee is to establish ways to aid the functions of data administration and systems administration, including specifications required to establish and maintain data base structures.

E.5 Relation to Other Organizations

CODASYL is recognized internationally as the language development body for COBOL. The American National Standards Insitutue (ANSI) standards body uses the CODASYL work as the base for the American National Standard for COBOL. All clarifications, changes or corrections are approved by the Programming Language Committee before incorporation into the American National Standard. The development of the national and international standard is the combined work of CODASYL for the development phase and the appropriate groups of the European Computer Manufacturers Association (ECMA), and ANSI for the standardization phase. CODASYL also maintains liaison with Japanese standards organizations.

E.6 Finance

CODASYL is completely supported by the contributed work of its members. There are no membership dues or other assessments.

E.7 Technical Work

The technical development work is vested in three committees, i.e., Programming Languages, Systems, and Data Description Languages Committees.

The responsibility for the major portion of the development work for the COBOL language resides in the Programming Language Committee. The PLC is the author of the COBOL Journal of Development. Extended capabilities are added to the language as they are developed by the task groups and approved by the PLC.

Task Groups are established as the need is recognized and become responsible for functional segments of the COBOL language. Each task group is responsible for the language extensions and the modifications required for a particular project.

Task Group membership is by organization, but is not restricted to the same organizations as those of the parent committee. For example, a Data Base Language Task Group member may represent an organization which does not choose to be a member of the PLC.

As projects are completed, the task groups are disbanded, so that over a period of time the complement of task groups is continually changing.

APPENDIX F
DATA BASE AND DATE BASE MANAGEMENT, DETAILED REQUIREMENTS

APPENDIX F

DATA BASE AND DATA BASE MANAGEMENT, DETAILED REQUIREMENTS

The function of the IPAD Data Base and Data Base Management systems is basically to provide the structure and facilities so that the various users may construct and use data files according to their project and their individual needs.

Section 4 of Part II presented the overview of the IPAD Data Bases and Data Base Management requirements. It further presented the overall correspondence between the data base management problem and its solutions using the Query Processor of the Data Base Management Systems (QP/DBMS). In this appendix, a more detailed presentation is made of the IPAD data bases and data base management requirements. A detailed set of solutions via QP/DBMS is made to show how QP/DBMS would support the data base function. This appendix further presents the essential overall translation of the data base requirements into the Data Declaration Language (DDL) using Query Processor (QP) procedures. It also discusses the general usage of the SCHEMA DDL required for IPAD.

The notation used in this appendix is the same as that used throughout this report. Words written in all capital letters refer to the terminology presented in Reference F1 (e.g., SCHEMA, WITHIN) or QP directives (actually QU/2 directives) presented in Reference F2.

F.1 Project Data Bank (PDB)

The Project Data Bank (PDB) for an active project is organized out of all the data that is unique to a particular project's design process. There are several major types of data within IPAD that are essentially of use to all types of users. The usage of these data types vary and are dependent on the individual users, therefore, the data types are organized into particular data bases according to the orientation of the users.

To conform with the operational and managerial philosophy of IPAD, the PDB has the requirement of a Project Directory. This Project Directory permits IPAD to coordinate the usage of the facilities (data and programs) of a project or a discipline. The users, in general, are dealing with the IPAD system in different (and sometimes conflicting) time periods and for different purposes. Further, because of the complexities of communications and data flow through the data bases, the individual user must be freed of the responsibility for setting up any data base operations other than those applicable to his problem at hand. The Project Directory contains the necessary reference information for all data files that normally must be directly accessible to a user during the course of his activities.

F.1.1 Directories. - Directories are used both to permit cataloguing and linking of the data with the user, and to permit internal structuring and linking of the data. A directory is a collection of data that permits linkage between a symbolic (name) reference to data and the valued data associated with that name. A directory entry consists of two types of information:

1. The entry name.
2. Valued data.

The organization structure of a directory may vary; e.g., consider the alternates:

1. Entry names and data base identifiers for the valued data.
2. Entry names and valued data within the same structure.

The requirements for a directory to the various data files within a project are satisfied by the AREA specifications and are embodied in the object SCHEMA. Likewise, the requirements to manage the project directory are satisfied by the functions that permit management of the Project SCHEMA. A Query Processor (QP) procedure to display the SCHEMA enables the Data Base Administrator (DBA) to obtain the specifications for the total project data base structure at any desired time.

Table F1 summarizes the DDL for the defining Project SCHEMA. The total SCHEMA, of course, is made up of all the DDL specifications pertaining to the data bases. (The terminology used in this and subsequent tables is taken directly from Reference F1):

TABLE F1. PROJECT SCHEMA DDL

CLAUSE	DBA Supplied Information
SCHEMA	Project Identity Name stored in the Project SCHEMA.
PRIVACY	Separate locks (if desired) for each privacy option. A minimum privacy lock is required for the ALTER option.

In addition to the directory references to identify the data files, the Project Directory requires two other pieces of information in order to control access to the data:

1. Project Identity - This identifies to the IPAD system the external access name for the project data.
2. Privacy Data - This data controls access to the project. There are two types of privacy data:

    a. Data access - Identifies the access conditions of the project data bases. At this level only the access to the project data base as a whole is checked. Details of particular data bases are handled within the data base management facilities for the particular data bases themselves.

    b. Project Directory access - This data identifies who has access to modify the Project Directory itself. Access information consists of conditions for access such as display, delete, insert, etc.

Within the DDL, the Project Data Bank (PDB) is represented by a Project SCHEMA (PS), which is designed to encompass all the data that will be associated with the project. Section 4.2 summarized the various data base requirements of an IPAD project and their representation within the PS. The structure of the various data bases are described in the following subsections.

The data base management requirements detailed in the following subsections are concerned with the data base operations on the project level over the lifetime of a project.

F.1.2 Project initialization. - This functional operation is required to identify the project and to set up the initial structure on which the rest of data bases will be built. The operation is satisfied via the construction and compilation of the Project SCHEMA.

The initialization procedure:

1. The DBA generates the source SCHEMA for his project as a permanent file. This includes all AREAs, SETs, RECORDs, etc., for the MDB, DLFs, TSAs, etc.
2. Using the DDL compiler, the DBA compiles the source SCHEMA into a permanent object SCHEMA file.
3. If the DBA so chooses, both source SCHEMA and object SCHEMA can be described via SCHEMA specifications within his Project SCHEMA in AREAs that correspond to the permanent files for processing with QP.

F.1.3 Project archival. - This functional operation permits the user, when engaged in purging a project from IPAD, to maintain data in an archive file. The user can generate separate files by identifying the files and which of the project data banks are to be

part of the separate files. The procedure involved uses the Project SCHEMA and associate SUBSCHEMAs for only those portions of the data base to be saved.

The QP procedure:

1. The DBA extracts the desired project data onto an archival storage medium (tapes, disc packs) via QP functions.
2. The DBA edits the Project SCHEMA as recorded within his data base via QP and incorporates it into the archived file(s). If the SCHEMA is not stored within his data base, other tools such as the Text Editor should be used. The purpose of this step is to have a SCHEMA description that accurately defines the archived portion of the project data.
3. The DBA may then release the permanent files from the IPAD system via the purge procedure.

F.1.4 Project purge. - This functional operation is required to remove the data bases of a project from an IPAD installation. If the user wishes to archive project data, the project archiving operation is used. Otherwise, the permanent files associated with the project are released.

The procedure:

1. Basically the project can be purged from an IPAD installation by releasing all permanent files associated with the project.
2. If the DBA has included within his data base the source SCHEMA, object SCHEMA, source SUBSCHEMAs, and object SUBSCHEMAs, these too are released.

F.1.5 Project privacy data. - This is data which specifies the accessing information necessary for various operations upon the design data. Each entry in the directory consists of three items of information:

1. Access Codes - These identify the user and the permission codes necessary to use the file.
2. Conditions - These specify the conditions for all access forms (fetch, write, delete, replace).
3. Alternate Designs - These specify the actual alternate design to which the above conditions apply.

Conditions of access may vary for different users so that the actual access permission consists of 'and' combinations of the three items.

F.2 Design Data

In the IPAD system much of the data is directly referenceable by the user through appropriate naming and structuring of the hierarchical relationships of the design data. The data base requirements include, therefore, both the externally referenceable identity of the design data (or groups of design data) and the valued data itself.

Design data within IPAD may reside within a number of files (e.g., MDB, DLF, and Support System files). The DDL here pertains to the record entries that the DBA will develop and incorporate into the various files according to their needs. Table F2 is a summary of the various data base requirements of the design data and their general corresponding fulfillment (including options) in the DDL. Table F3 illustrates the corresponding DDL specification for a design data RECORD.

TABLE F2. DESIGN DATA REQUIREMENTS WITH DDL SPECIFICATIONS

Design Data Requirement	DDL Solution
Category Identity	Group name with level number.
Subcategory Identity	Group name with level number subordinate to category to which it belongs.
Category Directory	DATA ITEM with name of category on same level as category subdivision, RECORD occurrences automatically provide the category directory as the data base is built.
Subcategory Directory	DATA ITEM with name of subcategory on the same level as subdivisions or data associated with the subcategory, subcategory directory is automatically updated as subcategories are detailed out.
Design Data Reference	Lowest level of data item specifications with design data RECORD; naming convention automatically produces the referenceable data.
Design DATA ITEM Identity	Optional DATA ITEM on lowest level. To be used if DBA desires actual storage of name for reference purposes.
Privacy Data	The DBA has privacy lock specifications available to him on any level.
Glossary	DATA ITEM at lowest level describable as character with limits set by DBA.

TABLE F2. DESIGN DATA REQUIREMENTS WITH DDL SPECIFICATIONS (contd)

Design Data Requirement	DDL Solution
Units	Same as glossary.
Coordinate Systems	Same as glossary.
Data Type	Specified as part of the DATA ITEM specifications which actually identify the storage requirements for the valued data itself.
Data Structure	Same as data type (e.g., OCCURS clause).
Design Valued Data	Controlled by above two specifications which are repeated for each type of DATA ITEM.
Versions	Each lowest level DATA ITEM has a version identity DATA ITEM which is updated for each version. Searches by QP include Version condition.

TABLE F3. DESIGN DATA DDL

CLAUSE	DBA Supplied Information
RECORD	DBA assigns data name to encompass the total design data to be incorporated in this record.
WITHIN	DBA assigns RECORD to AREA (synonymous with MDB, DLF, and UF).
Data subentry level number	DBA details out as many levels as required to describe the total category to data relationship.
Category name	If required, a DATA ITEM for storage of name for each internally named subcategory.
Data version	ID's for versions of design data, DATA ITEM on same level as design data, or default filler in DDL specs.
Design data identifiers	Described as required by DBA.
Subentry design data	Described as required by DBA.
PRIVACY	Assigns protection down to any level according to options required.

Categories and subcategories are the first major subdivisions in the IPAD system. Category and subcategory definitions are handled by the DBA by establishing each category as a distinctive RECORD entry and then spelling out the relationships by appropriate group level naming until the actual DATA ITEMS are to be detailed.

If the DBA also desires to include an internally stored category (or subcategory) name for retrieval purposes, he can accomplish this by introducing a DATA ITEM on the same level as the subdivision of the category or subcategory for the category desired. He then, by usage of QP, generates the appropriate RECORD occurrence for that name.

The Category Directory is the first subdivision of a design data database. The Category Directory identifies by symbolic name the various design data categories selected by the creator of the data base. Each entry in the Category Directory contains two items of information:

1. Category Identity: The Category Identity is the symbolic name to be associated with the directory for the data base.
2. Subcategory Directory: The Subcategory Directory identifies by symbolic names any further symbolic trail to the actual design data required. A Subcategory Directory can be of two forms:

    a. Hierachical Data Reference - This type has the same construction as a category directory entry and is used to continue symbolic name breakdown if required.

    b. Design Data Reference - This type is used when no further symbolic breakdown is required and the actual data can be referenced. The Design Data Reference consists of three items of information:

        - Design DATA ITEM Identity - The symbolic name of the DATA ITEM.
        - Design Data Structure Information - Information used to identify the structure of the actual DATA ITEM for processing, including descriptive data such as: glossary (textual description of data, if desired), units, coordinate systems, data type (integer, double precision, floating) and data structure (vector, matrices).
        - Alternate Design Directory - When alternate design data exists there is a further subdivision of DATA ITEMs within the data base. (When only one design exists there is only one entry within the directory.) The Alternate Design Directory has two additional items per entry:

- Alternate Design Identification - Identifies the alternate data set.
- Design Data Version Directory - The directory permits access to various versions of data that has been produced during the course of the design.

In situations where the DBA wishes the data to be shared or accessed by different category chains, he either employs, 1) a procedure to copy data into various RECORD types from other RECORD types, 2) maintains the MDB or design data under one standard SCHEMA and provides various SUBSCHEMA for different relationships, or selective usage by design data in other contexts, or 3) replaces the appropriate data subentry level with a fully qualified name that identifies the RECORD occurrence within another category.

F.3 Multidisciplinary Data Bank (MDB)

The MDB is the repository of all approved design data required by a project to represent the design. It is a collection of the design data RECORDs as described in the preceding subsection. Its characteristics which distinguish it from other design data files are:

1. It is a permanent resident of the IPAD data bases for direct reference and access by users.
2. The access controls for operating with it are controlled by the project. Replace, delete, and insert activities are controlled by the DBA. Fetch operations are available to any qualified user.

The Multidisciplinary Data Bank (MDB) is represented within the Project SCHEMA (PS) as an AREA. The AREA is further detailed as a collection of RECORD entries to satisfy the needs of a particular project. The DDL specifications for the individual design data RECORDs is described in Section F.2. This section is concerned with the DDL specification for the MDB as a separate file.

Fundamentally, the updating of the MDB is considered as the exclusive function of the DBA and is performed separately from operations with the data of the various users. For this reason, the privacy controls are selected to permit only one updating activity for the MDB. The MDB consists of the following RECORD entry types:

1. RECORD entries for design data.
2. RECORD entries for history of operations in the MDB.

Table F4 summarizes the DDL for the MDB as a file.

TABLE F4. MDB DDL

CLAUSE	DBA Supplied Information
AREA	MDB name.
PRIVACY	1. DBA selects PROTECTED UPDATE as minimum. If users, in general, always require latest MDB data EXCLUSIVE is the preferable specification. 2. A separate lock for retrieval.

MDB operations are subdivided into two categories of operation:

1. MDB structure definition.
2. MDB design data operation

The MDB Structure definition is accomplished by a set of functional operations that permits a DBA to detail out the data name hierarchy for external reference and operations on the MDB. The MDB Authorization Privacy Specification function is employed to permit authorized personnel to specify the privacy data required to manipulate the MDB. The Category Definition functional operation permits a user to define a category subset of data for the MDB. A function is required either to initialize MDB category structure or to modify it.

The Subcategory Field definition functional operation permits a user to further detail out the hierarchial structures of a file. The function can be used to detail as many levels as may be required. Function is also used to modify existing structure. Finally, the Design Data Structuring function permits authorized personnel to define contents and type of design data.

The activity of AREA definition for the MDB for incorporation into the project SCHEMA satisfies the category definition, subcategory definition, design data structuring and MDB authorization privacy specification functions. The activity is part of design of a project SCHEMA. The procedure is:

1. The DBA details out the entire category/subcategory/design data structure according to design data specifications (see Section F.2).
2. The DBA defines a unique AREA for the MDB according to specifications contained in Table F4.
3. The DBA incorporates this SCHEMA as part of the Project SCHEMA (see Section F.1).

4. Additionally, if choice is made to incorporate category names into RECORDs, then the DBA uses a category naming procedure.

The category (subcategory) naming procedure uses the Project SCHEMA and a SUBSCHEMA that need only specify the category (subcategory) data items for MDB AREA The QP procedure.

1. Insert directive to create a RECORD occurrence for category (subcategories).
2. Move data items into RECORDs that occupy the category (subcategory) name fields to identify the category (subcategory).

Restructuring the MDB or adding more details to it is accomplished by modifications or additions to the MDB DDL specifications. The procedure:

1. DBA generates additional DDL specifications for MDB AREA.
2. The Project SCHEMA is recompiled.

The DBA, however, must insure that existing RECORD occurrences are not violated by changing RECORD definitions.

The MDB display functional requirement is to provide the user with the capability of displaying the contents of the MDB (or a selected portion thereof). The user can declare one of the following options:

1. Organization display - The organization structure of the MDB with the symbolic names will be displayed. This will display each category and the names of data associated with it, including recursion if so structured. The user can specify any name limits (all, a category, a data set of a category, or lower).
2. DATA ITEM display - The actual data values associated with named data will be displayed.
3. Combinations of 1 and 2.

If user is authorized to display contents, the display options specified will be employed to access the structural or data item information and format the contents for display. Both the actual information and its linkage with lower level data will be retrieved. The QP procedure(s) employed uses the Project SCHEMA and SUBSCHEMA designed to limit the view of the MDB to Organization display, DATA ITEM display, or combinations. Alternative QP sequences are:

1. To provide simple listing:

a. A directive specifies the appropriate SUBSCHEMA to control the display options.

b. A QP Display directive provides the contents to display.

2. To obtain a specialized (report type) format:

a. The user must perform a prior construction and store a REPORT format using QP.

b. A directive then prepares the display according to the REPORT specifications.

The following subsections detail the requirements for updating the MDB.

F.3.1 Addition of design data. - A functional operation is required to permit the addition of design data. Basically the MDB is updated from entries within the Multidisciplinary Data Base Update (MDBU) file. The MDBU can contain entries that completely identify the IPAD file and data set that contains the design data or the MDBU can contain the data itself. Alternatively the DBA can incorporate data directly into the MDB from any file provided it satisfies the MDB structure requirements. The QP procedure for updating from the MDBU involves specification of an MDB update SUBSCHEMA which encompasses both the MDBU and MDB AREA of interest. (A SUBSCHEMA that repeats the AREA specification of the MDB and MDBU is adequate.) The procedure involves:

1. Extraction of the desired design data for update into a local file.

2. Insert from local file unto the MDB.

This procedure in combination with the MDBU Data Set Construction procedure (see Subsection F.12) permits flexibility in operation for MDB update. (Alternatively, the user can construct the local file and the DBA can incorporate directly from this file, thereby eliminating step 1 of this procedure. This method, however, necessitates a procedure to pass a message to the DBA to completely identify this file which is not exercised through normal controls.) The requirement to permit new Category Definitions within the operation of actual data incorporation is prohibited by this method; the function is more appropriate to MDB Restructuring.

A procedure is required to permit MDB Data Subset Deletion by the DBA from the MDB. The DBA by reference to a named data substructure can cause deletion of either all or part of the data. The DBA can also place the deleted data into an archive file, if it is so desired. The QP Procedure (actually QU/2 procedure, Reference F2) involved:

1. If archived data is required, the DBA extracts data from files for relocation in off-line storage.

2. The DBA specifies the complete name (for deletion) and exercises a DELETE QP directive.

3. The DBA updates History Record entries in the MDB:

   a. QP INSERT directives create RECORD occurrences for history.

   b. QP MOVE directives enter the history information into the RECORD.

Functional operation is required to permit MDB Design Data Modification, i.e., the changing of existing design data. This operation differs from addition/deletion of data since it is concerned with changes to existing data values. The DBA can either personally replace data values or he can maintain both the original data values and their changes (appropriately identified). The QP procedure uses a SUBSCHEMA specifying the MDB AREA of interest and involves either:

1. Updating individual data items without retention of original data:

   a. A QP UPDATE directive obtains the RECORD occurrence of interest.

   b. A QP MOVE directive (with data values) modifies the data items.

   For history RECORDs:

   c. A QP INSERT directive creates the RECORD occurrence for history RECORDs.

   d. A QP MOVE directive places pertinent information in the RECORD.

2. Maintaining both versions of the data requires:

   a. A QP UPDATE directive to obtain original data RECORDs to modify the version.

   b. A QP MOVE directive to provide the Version Identity field for the Version Identifier.

   c. A QP INSERT directive to create a new version.

   d. A repeat of c and d from procedure 1 to generate the history RECORD.

Functional operations are required to permit the MDB to be subdivided into Alternate Designs. The QP procedure uses a SUBSCHEMA that specifies the MDB portion of interest. The QP Procedure:

1. For each data category that is to belong to an Alternate Design, the DBA introduces Alternate Design identifiers via QP INSERT and MOVE directives into alternate design DATA ITEMs.

2. For DATA ITEM RECORDs that will be different, the DBA generates necessary RECORD occurrences.

F.3.2 Design data copy. - A functional operation is required to permit a user to copy a design data subset. The user specifies a SUBSCHEMA incorporating the MDB AREA of interest and the file onto which the design data is to be copied. The QP procedure:

1. Use the QP EXTRACT directive to copy the MDB data onto a local file.
2. If the local file is already part of a data base DDL, the procedure is complete.
3. If the local data file is not a part of a data base DDL, then it is made a part of the desired data file by a QP INSERT directive.

F.3.3 MDB update history maintenance. - A functional operation is required which permits the maintenance of a history of all update activity applied to the MDB. Any updating reference to the MDB requires a corresponding entry into the MDB history file. The entry recorded corresponds to the activity that is employed against the MDB. The SUBSCHEMA used refers only to that portion of the MDB concerned with occurrence of MDB History RECORDs. The QP procedure:

1. The DBA uses a QP INSERT directive to prepare history RECORD occurrences.
2. The DBA uses a succession of MOVE directives to accomplish the detailing of history RECORDs.

Normally, this procedure is a subprocedure of any MDB update activity. However, it can also be used as a separate procedure to incorporate additional data into the history file for the convenience of the DBA.

F.4 Disciplinary Library File (DLF)

The DLF is the collection of the common body of information required by a functional group or collection of users assigned a related part of the design. A DLF is described as an AREA. For each DLF to be employed by a Disciplinary Group (DG), the DBA constructs a separate AREA to satisfy the requirements of the DG. The DDL description for the DLF can consist of any RECORD entries.

In general operation, several users may update portions of the DLF. The privacy locks in this case are necessary only for PROTECTED update.

The requirements of DLF directories are satisfied by the SCHEMA description itself. Operations on the DLF are those operations possible on any of its data types. The basic DLF structure includes:

1. A subfile directory - The subfile directory identifies all data sets within the DLF that are referenceable by the various users.
2. Privacy data - The privacy data identifies access conditions of the DLF.

The DLF Subfile Directory consists of the following types of information for each subfile:

1. Subfile identity - Identifies the name of the subfile by which the user can externally reference it.
2. Subfile type - Identifies the type of subfile for accessing and management (e.g., OMs, TCS, Local Design Data).
3. Subfile data reference - Locates within the DLF itself the actual subfile.
4. Subfile file cross-reference - Identifies for data management purposes any other file within the DLF that includes this subfile as part of its structure.

Specifics of the subfile entries are presented with the appropriate writeups.

Table F5 summarizes only the DDL pertaining to the DLF as a DLF.

TABLE F5. DLF DDL

CLAUSE	DBA Supplied Information
AREA	Disciplinary Group (DG) name for its DLF.
PRIVACY	Lock for retrieval. Separate lock, if desired, for PROTECTED update.

F.5 Operational Modules (OMs)

The OM is the actual computer process (program) which will perform the design analysis at the user's control. The data requirements associated with the OM include:

1. IDEF (Input DEFinition) - This user provided data defines the input requirements of the OM. The IDEF requirements are described in Section 3 of Part II.
2. ODEF (Output DEFinition) - This data specifies the output requirements of the OM. Its requirements are likewise described in Section 3 of Part II.

3. Executable OM - This is the executable (object) code of the OM.
4. OM Source Code - This permits the user to maintain the OM source code within IPAD.

The storage requirements for OMs are defined by the DBA. An OM and all of its parts are described as SETs. The elements of an OM can also be tied together by SET specifications. The OWNER RECORD for the SET is a Micro Menu (see Subsection F6.3.1). The MEMBER RECORD types (assigned to various AREAs as required by DBAs) are as follows:

1. Object Code - RECORD description for residency of object code (summarized in Table F6).
2. Source Code - Same for object code (summarized in Table F7).
3. IDEFs, ODEFs - RECORD descriptions that control the I/O requirements for the OM according to DDL specifications.

The source code RECORD can be treated separately from the OM SET, if so desired.

TABLE F6. OM SOURCE CODE DDL

CLAUSE	DBA Supplied Information
RECORD	DBA (project or system) assigns a generic name for the OM source code.
WITHIN	DBA designates file within which source code will reside.
PRIVACY	Separate codes for. a) Update. b) Retrieval (optional).

F.5.1 OM management. - The following functions are to explain the requirements for the data base management functions for the operations to be performed on the OMs.

F.5.1.1 OM access: A functional operation is required to permit a user to specify an OM for execution. The OM is selected when the user specifies the OM identity. When the user chooses a Macro/Micro Menu sequence (see Section F.6) he specifies the successive entries including the multiple entries of Micro Menu as is necessary. At the completion of the functional sequence, the executable OM, its input map, IDEFs and ODEFs are available for direct reference by the user.

TABLE F7. OM OBJECT CODE DDL

CLAUSE	DBA Supplied Information
RECORD NAME	DBA (project or system) assigns a generic name for OMs. Each class of OMs (i.e., controlled by a Micro Menu) has a separate RECORD description.
WITHIN	DBA (project or system) assigns the object code to a particular AREA (file).
PRIVACY	DBA provides separate locks for: a) Update. b) Retrieval (optional).
Data Subentry	OM subentry for object code should be defined to correspond to host system requirements for manipulation of object files.

The user employs the SUBSCHEMA identifying the appropriate files (e.g., DLF or common data base) on which the OM resides. The QP procedure(s):

1. For a direct reference to an OM, the user specifies directly the fully qualified name for the IPAD EXEC.
2. For reference using the Macro/Micro menu:
   a. The user performs a Macro/Micro access procedure to determine the OM data base residency name from the IPAD data bases for ACCESS.
   b. This name is then used for other directives and procedures to utilize the OM.

F.5.1.2 Operational Module (OM) update: A functional operation is required to permit an authorized user to update an OM file. The OM file may be within a DLF, a general OM, or a project common OM. The user must supply identity of the file within which the OM resides.

For addition or modification to an OM, the user supplies the identity of the OM, and the identity of the file containing all pertinent OM data (e.g., IDEFs, ODEFs, OM code data to modify). For modification, the user supplies the identity of the OM and identity of the OM associated data to modify (IDEF, ODEF, code). For deletion, the additional information required is the identity of the OM and whether the option to retain backup file is to be exercised.

The OM file is processed according to user specifications. The user (if authorized) is permitted to change either a part or the entirety of an OM file. The QP Procedure (using appropriate SUBSCHEMA for file concerned):

1. The DBA completely identifies the RECORDs he wishes to work with (including object, source, interface, etc.) to determine types of OM data.
2. For addition:
    a. A QP INSERT directive is given for RECORD entering into the file containing the OM data (object or source).
    b. MOVE directives are given for special types of DATA ITEMS (identifiers, descriptions).
3. For deletion:
    a. Use the QP DELETE directive with the specifications developed in step 1.
4. For modification:
    a. Use the QP UPDATE directive with specifications developed in step 1.
    b. Use the QP MOVE directives to edit appropriate DATA ITEMs, or use file name in the UPDATE directive to handle large volumes of data (such as object code).

F.6 Macro/Micro Menus

Within IPAD the selection and operation of the OMs is assisted by the usage of Macro/Micro Menus, IDEF/ODEF, and display option data subfiles.

F.6.1 Macro Menu entry. - A Macro Menu is a collection of OM category names from which a user chooses the type of design analysis he intends to engage in. The names in turn are bound to successively more definitive names that enable specification of the portion of design for which analysis would be made. The data requirements of a Macro Menu are:

1. Identity of Macro Menu names.
2. Macro Menu directory which consists of two elements per entry:
    a. Name associated with the entry.
    b. Identity of next Macro/Micro Menu directory that corresponds to the selected entry. The next reference may in turn be a Macro Menu. Eventually the user will obtain the Micro Menu directory.

F.6.2 Micro Menu directory. - The Micro Menu directory is a collection of names and names of options that permits the user to select an OM and the conditions under which it will run. For each Micro Menu the data requirements structure is as follows:

1. Micro Menu identity which is related back to its entry in the selecting Macro Menu.
2. Micro Menu directory, a collection of names from which the user selects his desired analysis.
3. Micro Menu selection masks, a collection of combinations of selections of entries from the directory. Its data requirements are for each entry:
    a. Micro Menu entry combination.
    b. Operational Module (OM) that corresponds to the combination.

After choosing from the Micro Menu, the user will have the selected OM with which he will operate.

Macro Menus are satisfied by RECORD entry descriptions which contain:

1. Specifications which identify the Macro Menus.
2. Identifications of the strings of Micro Menu (or Macro Menus) identifiers associated with a Macro Menu.

This arrangement permits the user to access a particular Macro Menu by its identity, and display the data base names of its subdivisions. This then enables the user to directly access any of the programs or subdivisions. Table F8 summarizes the DDL for the Macro Menu.

The Micro Menu requirements are satisfied within DDL by usage of RECORD entry requirements that account for the following characteristics:

1. A unique identifier for each Micro Menu.
2. For each Micro Menu, there are as many entries as required to contain:
    a. The conditions for selecting an OM.
    b. The identity of the OM and its residency.

This arrangement permits both the selection of a particular Micro Menu and the display of conditions necessary for direct selection of OM from whatever file it is recorded on. Table F9 summarizes the DDL for Micro Menus.

TABLE F8. MACRO MENU DDL

CLAUSE	DBA Supplied Information
RECORD Name	Generic name for all Macro Menus.
WITHIN	AREA residency for RECORDs (normally one for each DLF).
PRIVACY	Unique privacy locks for each DG (optionally the same as DLFs).
Data Subentries Identifier	Identity of Macro Menu character type.
Macro/Micro Subdivisions	Two entries per Micro/Macro reference: 1. Identity of Micro/Macro Menu. 2. Data base name to use to locate the subsidiary menu.

TABLE F9. MICRO MENU DDL

CLAUSE	DBA Supplied Information
RECORD	Generic name for all Micro Menus.
WITHIN	Assignment to particular file (normally DLF).
PRIVACY	Unique privacy locks for DG.
Data Subentry Identifier	Identifies Micro Menu, character type, length determined by DBA.
Conditions	1. Character types that specify the combination of conditions that determine OM. 2. OM complete identifier - the character type that uniquely identifies an OM which may reside anywhere in the system.

F.6.3 Macro/Micro Menu operations. - The following data base management functional requirements apply to Macro/Micro Menus associated with Operational Modules (OMs).

F.6.3.1 Macro/Micro Menu construction: A functional operation is required to permit the construction of a Macro/Micro Menu. The QP Procedure using the SUB-SCHEMA for the appropriate file (DLF, UF):

1. A QP INSERT directive to create RECORD occurrence.

2. Successive QP MOVE directives to build the Macro (Micro) Menus. The DATA ITEMs to be generated are part of specifications summarized in previous subsections.

F.6.3.2 Micro Menu selection mask construction: These are the functional requirements which permit the user to specify the combinations of entries within the Micro Menu which will in turn select an OM. The QP Procedure:

1. The function is accomplished as part of the Micro Menu construction.
2. If Micro Menu is to be updated to reflect additional options, then:
    a. A QP UPDATE directive is used.
    b. Necessary QP MOVE directives are used to complete the updating.

F.6.3.3 Macro/Micro Menu display: A functional operation is required to permit the user to display menus and successively track down to an OM. The QP Procedure:

1. The user gives the qualified name for menu (with the identity of menu to display) and specifies the subdivision data items and their IPAD names (their menu locations) for the QP DISPLAY directive.
2. The user successively repeats step 1 until the desired OM level is achieved.

F.6.3.4 Micro/Macro update: A functional operation is required to permit a Macro/Micro Menu to be updated. This operation is similar to both the Macro and Micro Menu file construction functions but is used to update an existing program file. The QP Procedure:

1. A QP UPDATE directive is given with an adequate identity of the menus to isolate the RECORD occurrence desired.
2. Necessary QP MOVE directives are utilized to update the menu.
3. The user repeats steps 1 and 2 for as many subdivisions of menus as required, using the Macro/Micro access procedure if necessary.

F.7 Task Control Sequence (TCS) Strings

The Task Control Sequence strings are special groupings of TCSs. These TCS strings are stored within various data bases either for direct usage by the IPAD executive (on reference by the user) or for reference by the user to construct the command sequences he desires.

The separate TCS strings differ from the TCSs within the utilities by the fact that these strings may combine operations from many different file sources (i.e., system files, DLFs, User Files, project, etc.). Indeed, TCSs from the utilities file may be extracted. However the control strings supporting utilities are, in general, of a Task Control Sequence Skeleton (TCSS) nature.

A specific TCS string is directly executable by the user without further specification of data. This string is normally constructed for repetitive usage on similar data files. TCS data requirements are:

1. TCS string identity, the name by which the TCS will be invoked.
2. TCS Expansion, the data required to identify all the components of the string:
   a. Textual data which identify for the user the function of the string.
   b. The TCS string, each entry of which is a command in the sequence:
      - TCS command, to identify the actual TCS.
      - Data base command which identifies the specific data bases utilized by the TCS commands.
      - Absolute code reference, which identifies the object code (if required) that satisfies the TCS.

   An alternate type of entry can occur which will specify that the TCS is to be yet another TCS string. Its data requirements are:

      - TCS command, the TCS string identity.
      - The TCS string reference, identity of the TCS strings which will satisfy the TCS.
3. TCS cross reference which identifies all TCS strings which contain a particular TCS as one part of their string.

The TCSS string requires specification by the user of some of the parameters before execution can take place. The data requirements are:

1. TCSS string identity, the name by which the TCSS will be invoked.
2. TCSS data subfile, which contains the data associated with the TCSS. Data requirements are:
   a. Textual data identifying the function of the TCS to the user.
   b. Substitutable parameters which specify the data bases within the TCS data fields for which specific data bases must be supplied.

c. TCS string - Data that identifies each element of the string. For each entry, Type 1:

- TCS command which identifies an existing TCS that satisfies the component of the string.
- Specific data references, i. e., non-substitutable data references in the data fields.

For Type 2, where reference is again to another TCS string:

- TCS command - Identity of the TCS with the Macro string.
- Reference - Identity of the TCS string that corresponds to TCS within the string.

3. TCS cross references - Collection of identities of all TCSs that include this TCS as part of their expansions.

The TCSs are handled by simple RECORD entry descriptions. These RECORD descriptions are attached to any AREAs required by the DBA. Since TCSs are free-field, data descriptions are, likewise, relatively simple. Where TCS refer to other TCSs which may reside in other files, the data name of the TCS storage is provided. Table F10 summarizes the DDL specifications.

TABLE F10: TCS DDL

CLAUSE	DBA Supplied information
RECORD	Generic name for TCS RECORDs.
WITHIN	Generates one RECORD type for each file. The TCS will reside in (MDB, MDBU, PRF, DLF, UF).
PRIVACY	Provide separate privacy locks or permit the AREA locks to suffice.
DATA ITEM	
Identifier	Identity of TCS, character type.
TCS	One per each TCS in string, implementation dependent.
TCS Pointer	If TCS in above data field is expanded in another file, the DATA ITEM provides the identity of the file in which it is located.
Tutorial	DBA may provide a textual tutorial field for a TCS.

The following functions provide for the management of TCSs within IPAD.

F. 7. 1 New TCS incorporation. - A functional operation is required to permit the adc tion of new TCSs into data files. The TCS supplied by the user is assembled into a TCS record and attached to the designated file. The QP Procedure:

1. User generates the TCS source image either as a series of RECORD occu rences using QP INSERT directives or generates the TCS as a local file.
2. The INSERT directive is then used to move the TCS into the appropriate d base.

   Data base names must include the identifiers necessary to uniquely locate the IPAD data bases required (the names employed within DDL specificati or the data identifying other DATA ITEMS.

F. 7. 2 TCS string construction. - Functional operation to permit a user to build a T string from other TCSs.

In the user assembled TCSs, specific information is stored in the TCS string subfile. If the user has referenced other TCS strings, the strings are incorporated totally into the subfile if the referenced strings were originally stored in nonperman files, otherwise, only references to the permanent file are stored in the subfile. Op tionally, the user can include the referenced TCS within his own file (particularly if is not a permanent resident). The QP Procedure:

1. User constructs a file to correspond to the TCS Expander requirements f storage.
2. User then employs QP INSERT directive to store the TCS into appropriat file.

F. 7. 3 TCS file update. - The functional operation to permit the updating of a TCS, TCS file, or TCS RECORD within a file.

This operation requires the location of the TCS data files and the appropriate TCS. For complete TCS operation, the function deletes, adds, or replaces. For editing within the TCS, the user specifies the field and the modification which the function then incorporates into file. The QP Procedure:

1. DATA ITEM modification:
   a. The user specifies the qualified name and sets up a QP directive for the particular TCS.
   b. A QP UPDATE directive is then employed to obtain the TCS.
   c. A QP MOVE directive is used to edit the particular DATA ITEMs.

2. Addition of TCS:

   a. The new TCS incorporation procedure (Subsection F.7.1) is used.

3. Deletion of TCS:

   a. A QP IF directive identifies the TCS(s) to delete.

   b. QP DELETE directive accomplishes the deletion.

F.7.4 TCS file display. - A functional operation to permit the user to display a TCS string. The function may be directed to display down to the executable utility level. The function locates and displays the TCS data (within the appropriate file). The operation can be directed by the user to display all expansions of a TCS string which is made by other strings. The QP Procedure:

1. Specify a QP DISPLAY directive for all TCS occurrences within the file:

   a. For TCS only specify identifier items.

   b. For TCS and all commands specify commands and identifier items.

2. For recursive or cross referenced TCSs:

   a. The TCS names are used to continue DISPLAY directives.

F.7.5 TCS file operational access. - A functional operation to permit the user to designate a TCS file for execution. The identity must be as complete as necessary to identify the file. The procedure is:

1. Give name of TCS to IPAD Executive.

2. The TCS Executive interfaces with TCS records via DBMS by DML operatio to fetch the TCS string.

Alternately, a user can:

1. Extract TCS from data base via QP.

2. Give the IPAD Executive the TCS file name.

F.8 IPAD Utilities

The IPAD utilities, as executable code, have the same requirements for storage and operation as IPAD OMs. The same DDL specifications and QP procedures apply for handling them within the data bases. The RECORD descriptions are made part of any AREA which will contain utilities.

F.9 User Files (UFs)

User Files are associated with a specific user and contain data of primary concern only to the individual user in the performance of his particular design task; information on these files, however, may be of value to other users. The basic data requirements of this file are to permit the user to appropriately organize the files to be of maximal use to himself and to make maximum use of IPAD data base management facilities to support them.

User File (UF) data requirements consist of three basic types:

1. File identity - The file identity is the name by which the user will refer to a file.

2. Privacy Data - The user specifies the privacy data to control the access to his file. The data requirements of access authorization are:

   a. Privacy data for manipulating the contents of the file.

   b. Control data to permit operation on the privacy data itself.

3. Subfile directory - The subfile directory identifies the various data files which the user may wish to have contained within this file. Each entry in the subfile directory contains three types of information:

   a. Identity of subfile - Identity by which the user will refer to the information within the subfile.

   b. Types of subfile - Identifier to indicate the type of data in the subfile:

      - Design data
      - TCS data
      - Utilities
      - User's own I/O data.

   c. Subfile data - The subfile data is stored according to the file type and is entered into the subfile according to the data type rules.

User Files are described by the DBA as separate AREAs. As many AREAs are allowed as necessary to support a user. User Files are exclusively oriented to a single user and are provided for his convenience in utilizing the capabilities of the IPAD system for operations that are not satisfied within the scope of the other files (e.g., DLFs, MDB). The DBA, in general, at project initialization or when a new User File requirement arises during the course of the project, will construct a generalized AREA that includes all types of data that the user may require. A UF may include the same data types as the DLFs.

TABLE.F11: USER FILE DDL

CLAUSE	DBA Supplied Information
AREA	Unique name to identify User File.
PRIVACY	In general, lock for PROTECTED. LOCK FOR UPDATE and RETRIEVAL.

Table F11 summarizes the UF DDL for the UF. Controls and operations over the file are exclusively the responsibility of the user.

Operations on the UF are those necessary to manipulate the data RECORD types that the user has included in his AREA.

F.10 User Task Trajectory (UTT)

The UTT files are used to automatically record the design activities of individual users within a project. The recording of the information of the activity of the user serves several functions:

1. Permits the user's activity to be monitored by authorized personnel.
2. Permits the user in non-continuous sessions to review his status in the activities he has been performing.
3. Permits an historical record of how an activity was actually accomplished.

In addition to the UTT files themselves, there are data requirements to permit specifications for controlling the content and size of the UTTs. Data requirements for specifying limits are:

1. Length of retention for entry - a specification of how long the record should be maintained after the activity is performed.
2. Activities to record:
   a. Data bases used - only activity in specified data bases will be recorded.
   b. OMs or utilities used - only data of specific OMs used will be recorded.
3. User File to record - only specified individual User Files will have their activities recorded.

Data requirements for UTT RECORD:

1. User identity.
2. Activities entry:

a. Process used - identifies the OM or utility used.

b. Data sources - identifies the data bases processed.

c. Data disposition - identifies the data bases generated as a result of the process.

d. Time of activity - record of date/time activity was performed.

All UTTs are combined into a single AREA with a unique RECORD entry per UF. The DDL requirements are an AREA specification for all UTTs (Table F12). For each individual UF, there is a RECORD entry (Table F13). To control size and conditions of entry into UTT there is another type RECORD entry (Table F14).

TABLE F12: USER TASK TRAJECTORY DDL

CLAUSE	DBA Supplied Information
AREA	Unique file name for UTT.
PRIVACY	Privacy locks are best handled on individual RECORD entry.

TABLE F13: INDIVIDUAL USER's FILE UTT DDL

CLAUSE	DBA Supplied Information
RECORD	Unique name for each user.
WITHIN	UTT.
PRIVACY	Privacy locks to permit: a. IPAD executive to enter information b. DBA/DG leader to delete/modify. c. User to review.
DATA	a. TCS b. Data Bases used. c. Time of record.

TABLE 14: UTT ENTRY CONDITIONS DDL

CLAUSE	DBA Supplied Information
RECORD	Generic name.
WITHIN	UTT.
PRIVACY	Privacy locks for: DBA to define conditions for entry into UTT.
DATA	a. Condition description needed by IPAD Executive utility for recording. b. Condition description needed by IPAD Executive utility for editing.

F.10.1 User registration. - A functional operation to permit the user's identity and association with project to be registered within IPAD. The procedure is:

1. The DBA generates the appropriate RECORD entry for the user's TSA. This may be generated as part of project initialization.
2. The SCHEMA is compiled or recompiled.

F.10.2 Task trajectory initialization. - Functional operation to permit an authorized user to specify the conditions under which task trajectories will be recorded. The QP Procedure (using SUBSCHEMA corresponding to UTT):

1. QP INSERT directive is used to create RECORD occurrence for UTT identities.
2. QP MOVE directive(s) are used to permit identifying information.

F.10.3 Task trajectory display. - A functional operation to permit a user to display a UTT can be displayed within specified limits, and its contents formatted for identifying OMs, utilities, data bases. The QP Procedure:

1. a. The user uses the DISPLAY directive specifying the RECORD entry type associated with the particular user.

   b. If RECORD selection limits the DISPLAY are required, the user sets up an IF directive to specify the DATA ITEM and conditions to check for.

2. Alternatively, if a previous operation of report formatting had been developed via QP REPORT directives, then a QP PREPARE directive is issued.

F.10.4 Task trajectory entry. - This is an automatic function to record user activity. The function tests to determine if the conditions for recording are met. If conditions are met, a UTT entry RECORD is assembled and the UTT updated. An executive utility gathers the data for incorporation into the UTT using the conditions described by UTT entry conditions RECORD. It interfaces with the IPAD executive by usage of a DML STORE operation.

F.11 Task Status/Action Files (TSAs)

The Task Status/Action files refer to the collection of communication files whereby various members of the project communicate action and information requests amongst themselves. The first part of this paragraph details the general requirements for the files. The general data requirements of this file are:

1. File Identity - Identifies the owner of file (e.g., ERB, DBA) and personnel associated with it.

2. Messages - a textual communication.

3. Privacy - Identifies access conditions for:

   a. Interrogation of file.

   b. Insertion of messages.

   c. Deletion of messages.

The general requirements of a message are:

1. Message type - Identifies the message type as one of the following:

   a. Action request - Identifies requirements for action and person generating the requirements.

   b. Action response - Identifies the response to the request now entered into another task action file (e.g., DBA, ERB).

   c. Information request - Identifies a request for information from a user but is not regarded as of the same priority as an action request.

   d. Non-action messages - Identifies information with no requirement for response or action.

2. Message ID - Identification appended to message whereby users can refer to a message.

3. Message - Textual content of message.

4. Sender/Receiver ID - Identifies the source or destination of message.

5. Data file references - For ERB/ERBC and DBA files, this gives the references to additional data files necessary for processing.

6. Access statistics - This data is automatically recorded by the system for action request messages. It is to record when the owner of the file encountered the message.

All TSAs for a project are described within one AREA which includes separate RECORD descriptions for each TSA required. Table F15 summarizes the DDL requirements for the entry. The choice of a single AREA is made to minimize the number of files necessary for a user to do his job. A separate RECORD entry type for each distinct TSA is made to make the access and display of pertinent data more efficient (summarized in Table F16). Messages that are to be addressed to all members of a project are recorded with the designation of ALL.

TABLE F15: TSA DDL

CLAUSE	DBA Supplied Information
AREA	DBA assigns unique file name for all TSAs.
PRIVACY	Privacy locks are handled on RECORD entry level.

TABLE F16: INDIVIDUAL TSA

CLAUSE	DBA Supplied Information
RECORD	DBA assigns unique name for each DG, User, etc. that has a TSA. He also creates the ALL set.
WITHIN	Each is assigned to a TSA AREA.
PRIVACY Lock	Separate locks for DISPLAY, UPDATE, and DELETE.
DATA Subentry Message	Entire message, so that total contents can be displayed.
Message Components (subdivision of above)	Data storage specifications.
Message Type	Integer field.
Message ID	Integer field.
Message	Character type field.
Sender ID	Character field.
Access Statistics	Integer type field.
File REF	Identifies by name, location of any referenceable file within IPAD (usually MDBU or PRF).

The functional requirements detailed in subsequent paragraphs pertain to operations that generally can be performed on Status/Action Files. The following subparagraphs qualify, where necessary, the requirements for specific Status/Action Files.

F.11.1 TSA definition. - A functional operation to permit authorized personnel to identify the conditions of access and manipulation on a TSA. The procedure is:

1. DBA generates DDL RECORD entry specification for each TSA.
2. DBA generates DDL AREA specs for all TSAs.
3. DBA, if required, generates an ALL TSA for project common messages.

For an additional TSA once data has been entered, DBA must modify the AREA associated with the TSAs and recompile the SCHEMA.

F.11.2 TSA interrogation. - A functional operation to permit the interrogation of a TSA. The user may employ optional specifications of types of messages to display such as:

1. All messages in file.
2. Messages added since last interrogation (owner of file request).
3. Requests for information.
4. Non-action messages.
5. Historical message records.

The operation results in the display of messages as appropriately specified by user. If the user owns the file, the access statistics for messages requested will be updated. The procedure is:

1. An IF directive is used to set the conditions for display of TSA messages as summarized in Table F17.
2. A DISPLAY directive is then given.

The requirement to update access statistics is satisfied by incorporating (at the end of the procedure) a QP that:

1. Updates RECORD occurrences for designated TSA.
2. Moves by an updating expression the new access statistic into DATA ITEM field.

TABLE F17: SUMMARY OF CONDITIONS FOR TSA DISPLAY

Display Type	Conditions
All messages	Specific TSA record and the common TSA by ALL.
Unprocessed (Additional since last access)	Specific access statistics field zero.
Specific message types	Specific identifier for message type in message type data item.

F.11.3 TSA message purge. - A functional operation to permit user to delete an entry from a TSA. User must provide identify of TSA file, and identity of message to delete. The procedure is:

1. If the message has no corresponding source, the message simply may be deleted.
2. If it has a source, a message is placed on the source's TSA file to indicate that the action has been deleted but information is retained pending the corresponding approval by the source.
3. If the deletion refers to a deletion response on another file, the deletion is performed for all files.
4. If the deleter of the message is the originator, the message is deleted.

The following QP Procedures will satisfy the requirements:

1. Straightforward delete – QP DELETE directive is used which identifies the message by its message ID.
2. Deletion of mutual message.
   a. QP DELETE directive for user's own TSA.
   b. QP DELETE directive for corresponding message.
3. Proposed deletion of mutual message.
   a. QP UPDATE directive for RECORD occurrence.
   b. QP MOVE directive to set flags pending deletion.

F.11.4 TSA entry (new message). - A functional operation to permit the addition of a message into a TSA file. The user supplies the identity of the file into which message will be placed and the message type action request (required), and the associated file references, if required. The message is appended to originator's file (and to receiver's file if copy is desired). The message and its associated information is given an identity for further reference and the appropriate message fields detailed. The procedure is:

1. User designates RECORD type, (e.g., ERB, DBA, ALL) for INSERT.
2. User develops message via QP MOVE directives for individual DATA ITEMS.
3. For entry into more than one file, procedure is repeated with new RECORD type.
4. If copy is desired, steps 1 and 2 are performed for user's own TSA.

Steps 3 and 4 can be repeated by saving all associated directives of step 2 and executing them after changing the RECORD designation of step 1.

F.11.5 TSA entry (response). - A functional operation to permit the user to tie a response to the originator's file message. The user supplies the identity of file into which the message will be placed, the message type (same as new message entry), the message ID to which this message is associated, and associated file references, if required. The message is appended to requesting message. The QP Procedure is:

1. User performs a QP INSERT directive to record the response for the requesting TSA.
2. User performs a succession of QP MOVE directives to completely specify response.

For copy of response:

1. User extracts created message onto temporary file.
2. INSERT directive is used to record message on temporary file into own TSA.

F. 12 Presentation File/PRF/MDBU

Both the Project Review File (PRF) and the Multidisciplinary Data Base Update file (MDBU) belong to classes of presentation files which have the same general structure and requirements.

The MDBU file. - The data requirements of the MDBU file are:

1. MDBU file directory - References to various data to be inserted into the MDB. For each entry:
   a. Identity of design data.
   b. File reference - Two types are possible for file reference:
      - Design data can be within MDBU file.
      - Reference can be made to appropriate user file.
2. MDB Update Data - The form of MDB update data corresponds to any of the specifications of Section F. 2.
3. TCS subfiles - TCS data that can be executed by the DBA to process and display the design data. References to TCS within other file of the system can be included or the TCS itself can be included.
4. Executable code (Utility) subfiles - Additional processing code required to process the design data is included either by reference or actual incorporation in the MDBU entry.

F.12.1 The Project Review File (PRF). - A collection of data and instructions whereby the ERB/ERBC can evaluate a design (or portion of a design) from the specially prepared data file.

The data requirements for a Project Review File are:

1. Project Review File directory - Collection of references to all design data and associated TCS strings that have been prepared for the ERB/ERBC.
2. Project Review data - The Project Review data can reside either in the Project Review file or be referenced to a User File. The data requirements of the Project Review File are:
   - a. TCS string - The specific TCS string subfile required to process the data for ERB review.
   - b. Design data - Data prepared according to requirements of the utilities incorporated in the corresponding TCS string.
   - c. Utilities - Executable utility provided by creator of data file to support its processing (if required).

The Project Review File (PRF) is composed of several record types bound together in SETs. Table F18 summarizes the PRF as a file.

TABLE F18: PRF DDL

CLAUSE	DBA Supplied Information
AREA	Unique project name for PRF.
PRIVACY	Separate locks to permit: 1. ERBC to review and update. 2. User to add (but not delete).

The RECORD entry types consist of:

1. TCS RECORD entry.
2. Design data RECORD entry.
3. Utilities RECORD entry.

The SET entry (Summarized in Table F19) consists of:

1. The previously mentioned three RECORD entry types as MEMBERs.
2. An additional OWNER RECORD (which belongs to the PRF) that for each occurrence has recorded the occurrences of the previously mentioned types (Table F20).

TABLE F19: PRF ENTRY DDL

CLAUSE	DBA Supplied Information
SET	Generic name for PRF entry.
PRIVACY	1. Permit user to add. 2. Only ERBC to delete.
OWNER RECORD	PRF entry identifier RECORD.
MEMBER RECORD	1. TCS. 2. Design Data. 3. Utilities.

The method selected here assumes all data to be in the PRF AREA. However, a DBA can utilize the SET specifications to make only the PRF entry identifying RECORD a part of the PRF, whereas the associated data may belong to other AREAs.

TABLE F20: PRF ENTRY IDENTIFIER DDL

CLAUSE	DBA Supplied Information
RECORD	Generic name.
WITHIN	PRF.
PRIVACY	Same as PRF entry.
Data Entry	Definition for identifying PRF SET.

The MDBU structure has the same DDL specifications as PRF. These are summarized in Table F21 for the AREA, Table F22 for MDBU entry, and Table F23 for the MDBU entry identifier.

TABLE F21: MDBU DDL

CLAUSE	DBA Supplied Information
AREA	Unique project name for MDBU.
PRIVACY	Separate privacy lock to: 1. Permit Non-DBA user to add. 2. DBA to delete.

TABLE F22: MDBU ENTRY DDL

CLAUSE	DBA Supplied Information
AREA	Generic name for MDBU entry.
PRIVACY	Privacy locks to permit: 1. User to add 2. Permit only DBA to modify or delete.
OWNER RECORD	MDBU entry identifier RECORD.
MEMBER RECORD	1. TCS. 2. Design Data. 3. Utilities.

TABLE F23: MDBU ENTRY IDENTIFIER

CLAUSE	DBA Supplied Information
RECORD	Generic name.
WITHIN	MDBU.
PRIVACY	Same as MDBU entry.
Data Entry	Definition for identifier of MDBU entry.

F.12.2 Presentation file operations. - The following functions support the construction and processing of files that can be accessed.

F.12.2.1 Presentation file definition: A functional operation to permit definition of a file for presentation. The procedure is:

1. Define AREA for file.
2. Incorporate SET and RECORD entries necessary.
3. Incorporate into project SCHEMA.

This procedure can be used as a subprocedure of project initialization.

F.12.2.2 Presentation file, data set construction: A functional operation to permit a user to construct a data set for a presentation file. The QP Procedure (using appropriate SUBSCHEMA to handle the data source file and presentation file):

1. User employs a QP EXTRACT directive to obtain data from his files for the presentation file):

2. If the procedure involves incorporating TCS, QPS, or utilities to display the data for review, then the user performs the appropriate operations to incorporate this data.

3. Then the user:

   a. Uses a QP INSERT directive to move data from the local data file (created in previous two procedures) into the presentation file.

   b. Generates OWNER RECORD with entry identification.

This operation creates either the MDBU entry for a DBA to incorporate into the MDB, or an entry into PRF which can be used to review the data.

F.12.2.3 Presentation file usage: A functional operation to permit a user to execute the presentation stored on the presentation data file. The QP Procedure:

1. The user uses a DISPLAY directive to scan the entries available for display.

2. Then the user selects an identifying name of a SET occurrence to process the data from the display.

3. The user then instructs the IPAD EXEC to execute the TCS member of the SET.

F.13 Project Data Base Definition

This functional operation provides the user with the ability to define data bases of the various classes of information that are to be made directly available to all members of a project. Any selection of basic data types may be combined including project general utilities, design data TCS strings, and OMs. The DBA may declare any such collection as an AREA. For the DDL specs, the DBA provides, via RECORDs, an identity by which the file is known, and the type of data it is to contain. The general procedure is:

1. The designated collection of data is encompassed in a single AREA for the project. The AREA description is produced at project initialization and includes all RECORD descriptions necessary for the common project file.

2. The incorporation of the data into the common data bases is handled according to their type in the subsequent paragraphs.

Once the project data base has been initialized, the DBA can extend the description of the data file, if required, by editing the SCHEMA and recompiling. It is recommended, however, that the single file be described to include all common data types for the project since the corresponding object SCHEMA will occupy very little space and will reduce the necessity of recompilation of the SCHEMA.

F.14 IPAD Support System Data Bases

The support system data bases are the responsibility of special system DBAs who build and maintain the IPAD system itself (rather then project data bases within IPAD). It is the responsibility of these system DBAs to construct the various types of common data files that are needed by projects and their project DBAs. Table F24 lists the types of support system data bases available to all users of IPAD, and the types of access authorized. Section F.15 discusses the method of integrating these data bases with the project data bases via DDL specifications. The common data bases are contained in a single AREA. The other data bases are separate AREAs.

The Multi-Project Directory requires the DBAs to enter pertinent information into a Support System Data Base for that purpose.

TABLE F24: IPAD SUPPORT SYSTEM DATA BASES

Data Base Type	Availability	SCHEMA Type
Multi-Project Directory	Users and system	AREA belonging to Support System (SS) SCHEMA.
Common OMs	Users - retrieval System - retrieval/update	RECORD entries per SS AREA.
General Utilities	User - retrieval System - retrieval/update	RECORD entries per SS AREA.
System Message Files	User - retrieval System - retrieval/update	SS AREA.
SCHEMA/SUBSCHEMA/ QPSS Support	User - retrieval System - retrieval/update	SS AREA.

F.15 DBA Support Facilities

Additional data base and data base management functions can be introduced to facilitate construction of the required interface to employ the capabilities of DBMS and QP. These functions include:

1. QPS data - This type of data represents specific sequences of QP directives which can be cataloged and stored. Their purpose is to permit data base activity to take place by the single usage of a PERFORM directive

rather than the actual repetition of the step-by-step procedure required by the DBA. The QPS data includes both directives and identification of specific data bases.

2. QPSS data - This type of data is the more general form of the QPS. In general, the directive sequence is fixed and is applicable to many data bases or types of data bases. The DBA employs QP or a QPS to place the QPSS within the data bases required, where they will be available for expansion into specific QPSs.

   The format and operations performed in these data types are similar to those performed with a TCS, including their incorporation into various files. However, their actual requirements and conditions for usage are dependent upon the requirements of vendor supplied QPs.

3. SCHEMA, SUBSCHEMA - This type of data base can be used to incorporate the SCHEMA and SUBSCHEMA (source DDL) within IPAD data bases so that the operations on them can be performed via QP and DBMS. This arrangement also gives the DBA greater privacy control over various portions of SCHEMA and SUBSCHEMAs. Not doing this provides protection only on a permanent file within the host system.

The DBA can integrate support system data bases into his project data bases by copying the portions of the support system SCHEMA into his project SCHEMA. This permits the contents to be used in conjunction with other project data bases. However, the DBA would have no authority to modify the contents of these files.

F.16 References

F1. Jones, J. L: CODASYL Data Base Task Group Report, (no report number), Conference on Data Systems Languages, April 1971.

F2. Semegran, S. D.: Query-Update Version 2. External Reference Specification T038:2.0 - E013*3.4.1, Control Data Corporation, August 4, 1971.