PB82-143546

The Flight Service Automation System (FSAS) System Benchmark. Volume II The Model of the Application

MITRE Corp.
McLean, VA

Prepared for

Federal Aviation Administration
Washington, DC

Aug 81

U.S. Department of Commerce
National Technical Information Service
NTIS

The Flight Service Automation System (FSAS) System Benchmark: Volume II, The Model of the Application

PB82-143546

C. R. Spooner

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U. S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

Prepared for the
U.S. Department of Transportation
Federal Aviation Administration

Systems Research and Development Service
Washington, D.C. 20591

by
The MITRE Corporation
MTR-81W191

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO.	2.	3. Recipient's Accession No. PB82 143546

4. Title and Subtitle The Flight Service Automation System (FSAS) System Benchmark: Volume II: The Model Of The Application	5. Report Date August 1981 Publication 6. U363042
7. Author(s) C. R. Spooner	8. Performing Organization Rept. No. MTR 81W131 Vol. II
9. Performing Organization Name and Address The MITRE Corporation Metrek Division 1820 Dolley Madison Blvd. McLean, Virginia 22102	10. Project/Task/Work Unit No. 1471C 11. Contract(C) or Grant(G) No. (C)DT-FA01-81-C-10001 (G)
12. Sponsoring Organization Name and Address Federal Aviation Administration Systems Research & Development Service 400 7th Street, SW Washington, DC 20590	13. Type of Report & Period Covered Technical Report 14.

15. Supplementary Notes

Implementation and application of this System Benchmarking methodology on the Flight Service Station automation program was sponsored by the FAA. The project used an Ultra High Order Language and environment driven translator adapted from other work.

16. Abstract (Limit: 200 words)

In support of the FAA procurement of the Flight Service Automation System (FSAS), MITRE has developed a new system benchmarking technology to meet the requirements of an interactive, dedicated application. This methodology is applicable to procurements in which the hardware/operating system configurations are off-the-shelf and the applications software has not yet been implemented in its eventual form. In this methodology, a model of the application is interfaced with the operating system and is executed, while being driven by a specified load, and performance of the proposed configuration is measured. This report documents the requirements of the FSAS application of this system benchmark methodology and is published in the present form to record and make available the concepts and techniques of the methodology. A previous version of this report had been invoked in the FAA's procurement process. The entire report consists of eight parts, published in four volumes. This volume contains Part IV, Model. This part describes the model of the application and provides requirements and guidelines for mapping the model to the proposed configurations and for selection of options in the model.

17. Document Analysis a. Descriptors

Benchmarks, Synthetic Programs, Interactive Systems, Simulation, Performance Measurement, High Order Languages, System Benchmarks, Modeling

b. Identifiers/Open-Ended Terms

Interactive System Benchmarking, Interactive System Simulation, Data Processing, Acquisition Risk Management, Command & Control System Benchmarks

c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release Unlimited	Unclassified	
	20. Security Class (This Page) Unclassified	22. Price

(See ANSI-Z39.18) See Instructions on Reverse

OPTIONAL FORM 272 (4-77)
(Formerly NTIS-35)
Department of Commerce

The Flight Service Automation System (FSAS) System Benchmark: Volume II, The Model of the Application

C. R. Spooner

August 1981

MTR-81W131

CONTRACT SPONSOR:
Federal Aviation Administration
Systems Research and Development Service
CONTRACT:
DT-FA01-81-C-10001
PROJECT:
1471C
DEPT.:
W-103

This document was prepared for authorized distribution.
It has not been approved for public release.

The MITRE Corporation
Metrek Division
1820 Dolley Madison Boulevard
McLean, Virginia 22102

ABSTRACT

In support of the FAA procurement of the Flight Service Automation System (FSAS), MITRE has developed a new system benchmarking technology to meet the requirements of an interactive, dedicated application. This methodology is applicable to procurements in which the hardware/operating system configurations are off-the-shelf and the applications software has not yet been implemented in its eventual form. In this methodology, a model of the application is interfaced with the operating system and is executed, while being driven by a specified load, and performance of the proposed configuration is measured. This report documents the requirements of the FSAS application of this system benchmark methodology and is published in the present form to record and make available the concepts and techniques of the methodology. A previous version of this report had been invoked in the FAA's procurement process. The entire report consists of eight parts, published in four volumes. This volume contains Part IV, Model. This part describes the model of the application and provides requirements and guidelines for mapping the model to the proposed configurations and for selection of options in the model.

ACKNOWLEDGEMENT

The author gratefully acknowledges the contributions of T. L. Oliver for implementing model options for increased flexibility and for providing liaison with contractors relative to model issues during much of the competitive FSAS benchmarking phase; the contributions of D. L. Bashioum in handling all model calibration efforts and in assuring load tape consistency with the model; the efforts of Claire Lohr in testing many combinations of model options; the efforts of R. Nystrom in generation of two appendices, review of the material and project administration; the work of C. J. Lueder and Coop assistants in the many night and weekend production and test runs for delivery of fifty plus versions of the model during the validation and competitive phases; and last but not least, the patience and support of Mr. J. Johnson, the FAA's FSAS Benchmark Project Leader.

TABLE OF CONTENTS

	Page
1. INTRODUCTION	1
2. OVERVIEW OF THE MODEL	3
2.1 Execution Phases	3
2.2 Functional Areas	3
2.2.1 Top Level	3
2.2.2 Specialist Interface	3
2.2.3 DUAT Interface	4
2.2.4 Radar Management	4
2.2.5 Common Service Input	4
2.2.6 Time Management	4
2.2.7 Graphics Management	4
2.2.8 Flight Plan Management	4
2.2.9 Route Conversion	5
2.2.10 Weather Mangement	5
2.2.11 Message Management	5
2.2.12 Common Service Output	5
2.2.13 Legal & Analysis Recording	5
2.2.14 Data Base Management	5
3. COMPONENTS OF THE MODEL	5
3.1 General	5
3.2 Cells	7
3.3 Natural Flows	7
3.3.1 General	7
3.3.2 Permanent Natural Flows	8
3.3.3 Ancestral Natural Flow	8
3.3.4 Prime Source Natural Flows	9
3.3.4.1 Specialists	9
3.3.4.2 DUATs	10
3.3.4.3 Radar Management	11
3.3.4.4 Common Service Input	11
3.3.4.5 Time Manager	11
3.3.5 Service Natural Flows	12
3.3.5.1 The Scheduler	12
3.3.5.2 The Message Managers	12
3.3.5.3 Common Service Output	12
3.3.5.4 Legal & Analysis Drivers	13
3.3.5.5 Data Base Managers	13
3.3.6 Temporary Natural Flows	13

TABLE OF CONTENTS
CONTINUED

	Page
3.4 Working Space	14
3.4.1 Private Working Space	14
3.4.2 Shared Working Space	15
3.5 Permanent Memory	16
3.5.1 Introduction	16
3.5.2 File Types	17
3.5.3 Bulk Update Files	18
4. STRUCTURE OF A SERVICE FUNCTIONAL AREA	19
4.1 Introduction	19
4.2 Reception Desk Cells	19
4.3 Communication Mechanisms	21
4.3.1 Prods	21
4.3.2 Shared Working Space	22
4.4 Inner Set and Outer Set Cells	22
4.5 Migration Points	23
5. MAPPING THE MODEL	25
5.1 The Mapping Task	25
5.1.1 What Has To Be Done	25
5.1.2 The Rules	25
5.1.3 Considerations	26
5.2 General Approach	27
5.2.1 Procedure	27
5.2.2 The VIP's Task	28
5.2.3 Prod Pairs	28
5.2.4 Temporary Natural Flows	28
5.2.5 Migration	29
5.2.5.1 CPU Balancing	29
5.2.5.2 Access to Files	29
5.2.5.3 Access to Shared Working Space	29
5.3 A Particular Approach	30
5.3.1 Procedure	30
5.3.2 Terminology	31
5.4 A Simple Example	32
5.5 Distributing Functional Areas	35
5.5.1 Distributing Prime Source Functional Areas	35

TABLE OF CONTENTS
CONTINUED

	Page
5.5.1.1 Introdution	35
5.5.1.2 The Specialist Interface Functional Area	36
5.5.1.3 DUAT Interface Functional Area	37
5.5.1.4 Radar Management Functional Area	37
5.5.1.5 Common Service Input Functional Area	37
5.5.1.6 Time Management Functional Area	37
5.5.2 Distributing Service Functional Areas	38
5.5.2.1 Graphics Management Functional Area	38
5.5.2.2 Flight Plan Management Functional Area	38
5.5.2.3 Route Conversion Functional Area	38
5.5.2.4 Weather Management Functional Area	39
5.5.2.5 Message Management Functional Area	40
5.5.2.6 Common Service Output Functional Area	41
5.5.2.7 Legal & Analysis Recording Functional Area	42
5.5.2.8 Data Base Management Functional Area	43
5.5.3 Summary	46
5.5.3.1 The Straight Forward Scheme	47
5.5.3.2 File Access	49
5.5.3.3 Access to Shared Working Space	49
5.5.3.4 Return links to processors	49
5.6 Refinements	50
5.6.1 Pruning Cells	50
5.6.2 Packaging Into Modules	51
5.7 Trial Mapping	51
6. INDIVIDUAL FUNCTIONAL AREAS	52
6.1 General	52
6.2 Top Level	53
6.3 Specialist Interface	54
6.4 DUAT Interface	54
6.5 Radar Management	56
6.6 Common Service Input	56
6.7 Time Management	58
6.8 Graphics Management	59
6.9 Flight Plan Management	60
6.10 Route Conversion	60
6.11 Weather Management	61
6.12 Message Management	62
6.13 Common Service Output	63
6.14 Legal & Ananlysis Recording	64
6.15 Data Base Management Functional Area	64

TABLE OF CONTENTS
CONTINUED

	Page
7. VENDOR OPTIONS	66
7.1 Model Mapping Options	67
7.1.1 Specialists	67
7.1.2 DUATs	68
7.1.3 Message Management	69
7.1.4 Legal & Analysis Recording	70
7.1.5 Data Base Management	71
7.2 Miscellaneous Options	72
7.3 Dynamic Options	76
APPENDIX A. CALCULATING LOADS	A-1
A.1 Code Size	A-1
A.2 CPU	A-1
A.3 Working Space	A-1
A.4 Permanent Memory	A-2
APPENDIX B. SUMMARY GUIDE TO MODEL MAPPING	B-1
B.1 Overview	B-1
B.2 Guide to Migration	B-2
B.3 Guide to Distribution	B-3
APPENDIX C. MODELING LANGUAGE LISTING OF THE MODEL	C-1
APPENDIX D. MODEL CALIBRATION	D-1
D.1 General	D-1
D.2 Cell Target Size	D-1
D.3 Cell Calibration	D-4
APPENDIX E. MODEL VERSIONS	E-1
E.1 General	E-1
E.2 Small Scale Model	E-1

LIST OF TABLES

		Page
Table 1:	Cells by Functional Area	79
Table 2:	Permanent Natural Flows	80
Table 3:	Input Types & Scheduled Action	81
Table 4:	Private Working Spaces	82
Table 5:	Shared Working Spaces	83
Table 6:	Files	84
Table 7:	Weather Files	85
Table 8:	Potential Permanent Memory Ops	86
Table 9:	Data Base Operations on Each File	87
Table 10:	Restriction of Choice of File Type	88
Table 11:	Innerset Cells by Functional Area	89
Table 12:	Outerset Cells by Functional Area	93
Table 13:	Non-Local Variables Used in the Model	94
Table 14:	DataBaseManagement Cells Used; Operations vs. File Type	97

LIST OF FIGURES

		Page
Figure 1:	Terms Used to Describe the Model	98
Figure 2:	Specialist Loop	99
Figure 3:	Specialist: File Flight Plan From Conversation	100
Figure 4:	A Typical Service Natural Flow	101
Figure 5:	Client Using a Reception Desk Cell	102
Figure 6:	Client Signals a Service Natural Flow	103
Figure 7:	Using a Shared Working Space	104
Figure 8:	Inner Set and Outer Set Cells	105
Figure 9:	A Simple Example	106
Figure 10:	The Simple Example Distributed	107
Figure 11:	The Same Example: Prime Source=Specialists Distributed	108
Figure 12:	Simple Example: Service Functional Area=Route Conversion Distributed	109
Figure 13:	Distributing Weather Management--(1) No Temporary Spawning	110
Figure 14:	Distributing Weather Management--(2) With Temporary Spawning	111
Figure 15:	Distributing Message Management	112
Figure 16:	Specialist Interface	113
Figure 17:	Specialist: Simple Input/Page Forward/Page Backward/Request Send Format	114
Figure 18:	Specialist: Flight Log/Start Conversation With New Flight Plan/Resume Conversation With Current Flight Plan/Non Route Data For Flight Plan	115
Figure 19:	Specialist: Execute Link Send/Print Screen	116
Figure 20:	Specialist: Display Message	117
Figure 21:	Specialist: Route Data For Flight Plan	118
Figure 22:	Specialist: Request Weather	119
Figure 23:	Specialist: Start Conversation With Filed Flight Plan	120
Figure 24:	Specialist: File Flight Plan From Conversation	121
Figure 25:	DUAT Interface	122
Figure 26:	DUAT: DUAT Simple Input/DUAT Page Forward/DUAT Page Backward/Request DUAT Send Format	123
Figure 27:	DUAT: DUAT Flight Log/Start DUAT Conversation With New Flight Plan Resume DUAT Conversation With Current Flight Plan/Non Route Data For DUAT Flight Plan	124
Figure 28:	DUAT: Execute DUAT Link Send	125
Figure 29:	DUAT: Route Data For DUAT Flight Plan	126
Figure 30:	DUAT: Request DUAT Weather	127
Figure 31:	DUAT: Start DUAT Conversation With Filed Flight Plan	128
Figure 32:	DUAT: File Flight Plan From DUAT Conversation	129
Figure 33:	Radar Manager	130

LIST OF FIGURES

CONTINUED

		Page
Figure 34:	Common Service Input (Initialization Phase): Initialize Rescue Alerts/Initialize Flight Plan Deletions	131
Figure 35:	Common Service Input: Standard Initialization Of Files	132
Figure 36:	Common Service Input: Schedule End of Scheduling	133
Figure 37:	Common Service Input	134
Figure 38:	Common Service Input	135
Figure 39:	Common Service Input: Acknowledgement From Another Computer	136
Figure 40:	Common Service Input: Individual Update From AWP/Hourly Update From AWP	137
Figure 41:	Common Service Input: Mail Message	138
Figure 42:	Common Service Input: Flight Plan From Another Computer	139
Figure 43:	Common Service Input: AFOS Graphics From AWP	140
Figure 44:	Time Management Functional Area	141
Figure 45:	Time Manager: Delete Flight Plan	142
Figure 46:	Time Manager: Transmit Flight Plan	143
Figure 47:	Time Manager: Start Rescue Operations	144
Figure 48:	Iterated Invocations of Cells (Left) Scheduled in Series, and (right) Scheduled In Parallel	145
Figure 49:	Route Conversion With No Temporary Spawning	146
Figure 50:	Route Conversion With Maximum Temporary Swawning	147
Figure 51:	Weather Management With No Temporary Spawning	148
Figure 52:	Weather Management With Some Temporary Spawning	149
Figure 53:	Weather Management With Maximum Temporary Spawning	150
Figure 54:	Message Management	151
Figure 55:	Common Service Output	152
Figure 56:	Legal & Analysis Recording	153
Figure 57:	Database Management -- (1) Normal Operations & Record Initialization	154
Figure 58:	Database Management -- (2) Directory Initialization & Bulk-Update	155

PART IV: THE MODEL OF THE APPLICATION

1. INTRODUCTION

Part IV assumes the reader has already read Parts II and III. It is addressed to three persons:

(1) To the person responsible for mapping the model to the vendor's configuration (referred to as the model mapper).

(2) To the Vendor Interface Package (VIP) writer who finds a need to understand the model (referred to as the VIP writer).

(3) To the person who is responsible for integration and who finds a need to understand the model (referred to as the benchmark integrator).

Part IV is not directed towards the person responsible for compiling the model as delivered by the Government. Directions for this task will be found in Part VIII, section 4.

The model can be approached in either a top-down or a bottom-up direction. Those who prefer the bottom-up direction are referred to the Modeling Language listings (see Appendix C) because the HOL version is not designed to be readable. However, the Modeling Language listings are only designed to be readable at the level of the individual cell. They do not attempt to describe the higher level structure into which the cells fit. Those who are able to take the top-down approach are well advised to do so. This document describes the model in such a manner. For the top-down reader, the Modeling Language listings merely provide additional detail. Whatever the reason for reading the Modeling Language listing, read Part VIII section 2 first.

Part IV is organized so that if one reads forward through sections 2 to 6 one will start from the simplest overview and descend steadily through increasing detail. The VIP writer, or the benchmark integrator, who wishes to understand the model and who prefers the top-down approach, should read forward until satisfied that enough is understood. If the bottom-up approach is preferred, Part IV will not be as helpful.

The model mappers task is to address a set of questions. The answers to some of the questions are provided to the

Government, and they are reflected in the HOL version of the model that the vendor receives; for example, the settings of vendor options are provided. The answers to the remaining questions are passed to the VIP writer, who arranges to implement them; for example, which processor a particular Natural Flow is to operate in. A summary guide to the model mapper's task is provided in Appendix B. The majority of this document is aimed at giving the model mapper an understanding of the model, so that the answers arrived at do justice to the vendor's configuration.

The model mapper can approach this document in any of three ways:

(1) The model mapper can read forward to gain, in a top-down manner, an understanding of the model. Appendix B can then be used as a check list during the mapping process.

(2) Alternatively, one can start from Appendix B, and read the main sections of this part as the need arises.

(3) As a compromise, the model mapper can read forward part way, first, until comfortable with the major structure of the model and the more important concepts involved. Then the model mapper can proceed as in alternative (2).

The reader who cannot understand or absorb everything on a first reading need not feel discouraged. It is anticipated that generally, several passes will be required. In particular, some readers may find it helpful to make a first pass of section 6 before making a serious study of section 5. To help the reader absorb the material quickly, special symbology has been developed for the figures. To avoid clutter, the legends are not repeated, so the reader should take note of each new symbol as it is introduced. Note that the figures showing the model as a whole have a standard arrangement.

References are made at various points in Part IV to the VIP and services it must provide. Each reference to VIP services is to the VIP and the underlying system (operating system).

2. OVERVIEW OF THE MODEL

Figure 1 shows the interrelationships of various terms that are introduced in the following sections. Some readers may find it helpful to relate each term, as it is introduced, to the chart. It will be seen that at the highest level, there are two alternative ways in which the model can be regarded as divided; timewise into Execution Phases, or functionally into Functional Areas.

2.1 EXECUTION PHASES

The model executes in three consecutive phases:

(1) The Model Initialization Phase
(2) The Measured Period
(3) The Model Termination Phase

Only during the middle phase is the vendor's performance being measured, so naturally this is the phase which attracts the most interest. The division of the total run into three phases is only of interest when describing the overall structure, and when considering starting and stopping. All other discussion applies by default to the Measured Period.

2.2 FUNCTIONAL AREAS

Each Functional Area consists of a set of model components (cells, Natural Flows, Working Space, Permanent Memory) which together perform tasks related to that Functional Area. Each Functional Area is like a smaller model, within the model, that performs a specific set of tasks. The model is divided into 14 Functional Areas. The Functional Areas and their basic functions are identified in the following subparagraphs.

2.2.1 Top Level

The Top Level Functional Area controls the Model Initialization Phase, then initiates the Measured Period. At the end of the benchmark run it takes over control again for the Model Termination Phase.

2.2.2 Specialist Interface

The Specialist Interface Functional Area provides the interface between the human user (specialist) and the various functions provided by the application; that is, it

interprets the user's requests and it initiates the required functions in response to them.

2.2.3 DUAT Interface

The DUAT Interface Functional Area provides an interface similar to the Specialist Interface for the users of the Direct User Access Terminals. It serves both the stand alone terminals and those that are colocated in groups. The groups are referred to in the model as DUAT clusters.

2.2.4 Radar Management

The Radar Management Functional Area provides the function of interrogating the local weather radar sites, and updating the local radar data bases at the AFSSs.

2.2.5 Common Service Input

Though the input from NADIN arrives on one physical port, it originates from several sources, so it is modeled as arriving on several logical ports. This Functional Area administers the arrival of each input and forwards it appropriately.

2.2.6 Time Management

The Time Management Functional Area administers the scheduling of events, such as the issuing of aircraft overdue alerts, for the cases where the schedule of events must not be lost as a result of a system restart.

2.2.7 Graphics Management

The Graphics Management Functional Area administers the arrival of graphic data from AFOS and its distribution to each AFSS and to each DUAT cluster.

2.2.8 Flight Plan Management

The Flight Plan Management Functional Area administers the Flight Plan Data Base, that is, the central record of flight plans, as opposed to copies being held for updating by individual specialists.

2.2.9 Route Conversion

The Route Conversion Functional Area translates a pilot's description of a route of flight into a list of weather reporting points.

2.2.10 Weather Management

The Weather Management Functional Area administers the weather files and the file of NOTAM reports.

2.2.11 Message Management

The Message Management Functional Area administers the pool of messages to be displayed at individual specialist terminals, and arranges their delivery.

2.2.12 Common Service Output

The Common Service Output Functional Area administers spooled output. This includes output to NADIN and to the local printer.

2.2.13 Legal & Analysis Recording

The Legal & Analysis Recording Functional Area administers the output of all audit trails that are required for legal and analysis purposes.

2.2.14 Data Base Management

The Data Base Management Functional Area provides the basic administration of all uses of Permanent Memory.

3. COMPONENTS OF THE MODEL

3.1 GENERAL

Part III introduced the four main components out of which the model is constructed, that is:

Cells
Natural Flows
Working Space
Permanent Memory

To summarize:

- The unit of code is, for modeling purposes, the cell, and for compiling purposes the module. Any number of cells can be contained (at vendor option) in each module, and each cell can be contained (at vendor option) in any number of modules.

- The model is executed in terms of Natural Flows. Each Natural Flow is a succession of cell executions on data from a specific source; external input, hardware timed interrupt, or another Natural Flow. The basic model uses Permanent Natural Flows only, but to increase parallelism the vendor has options at several points to have Temporary Natural Flows spawned.

- Data is held in Working Space, and in Permanent Memory records, both of which are allocated by the VIP, on demand from the model. Whenever a Natural Flow that is executing requires a particular Working Space to be accessible, that Working Space must be directly accessible to the instructions executed by the model. The Working Space need not survive system failures. Permanent Memory on the other hand must survive system failures (and so must the means of identifying its contents), but it need not be directly accessible to executing code, since each record is brought into a Working Space whenever it is used.

Thus, the model is composed of Natural Flows, which execute in cells, and use Working Spaces both for variables and for holding copies of Permanent Memory records.

Each Natural Flow, each cell, each Working Space, and each Permanent Memory Record, is associated with a particular Functional Area, so (with two important exceptions) one can regard the model as divided into 14 mini models operating concurrently, each having its own set of Natural Flows, cells, Working Space, and Permanent Memory records. The two exceptions are:

(1) The Natural Flows of one Functional Area can operate for short periods in the cells of another Functional Area.

(2) Some of the Functional Areas do not have a full complement of components. For example, the Functional Area called Flight Plan Management does not have its own Natural Flows. Its cells are exercised by visiting Natural Flows from another Functional Area.

3.2 CELLS

The model contains 139 cells, 115 of which are used in the Measured Period. Table 1 shows the distribution of cells among the 14 Functional Areas. Since the cells differ considerably in size, one from another, the actual number of cells is of limited significance. Numbers are given in the table only to provide a first overview.

In the Modeling Language listing of the model, the cells are presented by Functional Area, the Functional Areas themselves being presented in the same order as in Table 1.

For much of the work, the model mapper will not need to descend to the level of the individual cell.

3.3 NATURAL FLOWS

3.3.1 General

The FSIS application is modeled by the parallel execution of Natural Flows. Natural Flows are either Permanent or Temporary. The Temporary Natural Flows are optional and are discussed in paragraph 3.3.6. Of the Permanent Natural Flows, there are 4 single Natural Flows and 8 families of Natural Flows. Since the size of three of the families is a vendor option, the total number of Permanent Natural Flows will vary from vendor to vendor. The minimum is 151, see Table 2.

Each Permanent Natural Flow is associated with a Functional Area, in that:

- It is born in a cell belonging to that Functional Area.
- Either it never leaves the cells of that area
  or, it returns to the cells of that area after performing each task*.

---

* Note: In Part IV the word "task" is used in a general sense, for a "unit" of work.

- It performs tasks that are functionally relevant to that Functional Area.

Table 2 shows the distribution of Permanent Natural Flows between Functional Areas. Note that throughout the model the term "...Management" is used for Functional Areas, and the term "...Manager" (among others) for Natural Flows.

3.3.2 Permanent Natural Flows

Aside from being grouped by Functional Area, the Permanent Natural Flows of the model can also be classified according to whether they are:

- The Ancestral Natural Flow.
- Prime Source Natural Flows.
- Service Natural Flows.

A Natural Flow is defined as a sequence of actions driven by data from a specific source. The Ancestral Natural Flow is best regarded as in a class by itself. Prime Source Natural Flows are those that are driven by data from some external source (hardware timer or external I/O). Service Natural Flows are those that are driven by data from other Natural Flows.

Each Fuctional Area can be considered as Prime Source or Service, corresponding to its Natural Flows. The Time Management Functional Area is associated with both, having both a Prime Source and a Service Natural Flow.

3.3.3 Ancestral Natural Flow

The Ancestral Natural Flow is associated with the Top Level Functional Area and is spawned by the VIP to start the Model Initialization Phase. During the Model Initialization Phase it spawns the other Permanent Natural Flows and coordinates their initialization. It plays no part in the Measured Period.

The Ancestral Natural Flow is again activated at the start of the Model Termination Phase. The running of the model terminates as follows. Each of the Prime Source Natural Flows receives a Stop Benchmark directive. For the input driven Natural Flows this directive is a message on the scenario load tape. The form of the directive for the Time Manager Natural Flow is described in paragraph 3.3.4.4. Each Prime Source Natural Flow waits for the death of any Temporary Natural Flows that it has spawned, and for the

completion of any tasks that it has requested from Service Natural Flows. Then it sends a signal to the Ancestral Natural Flow and stops.

The Measured Period ends when the first Stop Benchmark directive arrives at the model. When all Prime Source Natural Flows have signaled "end of benchmark", the Ancestral Natural Flow makes procedure calls to the Common Service Output Functional Area. This procedure call is made once for each of the logical output ports, to signal that no more output will be generated. When each of the output drivers has emptied its spool, it sends a signal to the Ancestral Natural Flow, then stops. The same happens for Legal & Analysis Recording.

Finally, the Ancestral Natural Flow issues a VIP Service Request (VSR) to inform the VIP that the model run is complete.

3.3.4 Prime Source Natural Flows

The Prime Source Natural Flows are:

- The family of Specialists.
- The family of DUATs.
- The Radar Manager.
- The two families for Common Service Input Functional Area (the Input Drivers and the Input Distributors).
- The Time Manager.

3.3.4.1 Specialists

There are 91 Specialist Natural Flows, one per specialist terminal. Each is born and initializes itself among the cells of the Specialist Interface Functional Area, then cycles endlessly. The basic loop, which is contained in the cells of the Specialist Interface Functional Area, is depicted in Figure 2. The functions of the loop are;

- Read from terminal,
- Interpret request,
- "Do it", and
- Output to terminal.

The Specialist Natural Flows respond to 17 input types that are listed in Table 3. For some of the input types, "Do it" involves only the execution of Specialist Interface cells.

For others, it involves procedure calls to cells of other Functional Areas. Thus, the Specialist Interface Natural Flows execute both in their "home" Functional Area and in other Functional Areas.

Figure 2 also introduces some of the standard symbology that is used in many of the benchmark diagrams:

- Shaded area representing the code of a Functional Area.
- Single line arrow representing a one-way path of control.
- Double line arrow representing a two-way path of control (go and return).

Figure 3 shows the pattern of flow in response to one of the input types, FileFlightPlanFromConversation. Figures like this one are intended to show the reader, at a glance, what areas of the model the Natural Flow goes to in response to each input type. This particular figure shows that when a Specialist Natural Flow reads a message of input type FileFlightPlanFromConversation, the Natural Flow leaves its "home" loop within the cells of the Specialist Interface Functional Area, and makes procedure calls to Flight Plan Management. From there it may (depending on parameters in the scenario messages) make further procedure calls to Common Service Output and to TimeManagement. From several places, it makes procedure calls to the Functional Areas Legal&AnalysisRecording and DataBaseMangement. Calls to these two are made from so many places that special symbology is dedicated to them. Note the legend in Figure 3.

The remaining Prime Source Natural Flows have patterns of flow that are broadly similar to this; namely, a basic loop looking for work to do, plus excursions into other Functional Areas to perform the requested actions.

The vendor may not necessarily need to understand each cell; however, looking at selected detail may help to establish the overall picture. This may be accomplished by reading some of the Modeling Language listing at a fairly cursory level. The vendor could start, for example, from the cell InitializeSpecialist and trace a path of a Natural Flow that is born in that cell.

3.3.4.2 DUATs

The 40 DUAT Natural Flows follow a pattern of behavior which is similar to a subset of that for the Specialists Natural Flows. They respond to 14 input types that are listed in Table 3.

3.3.4.3 Radar Management

The Radar Manager Natural Flow follows the same kind of pattern of behavior as the Specialist and DUAT Natural Flows, except that, since there are only two input types, the pattern is simpler.

3.3.4.4 Common Service Input

The Common Service Input Functional Area responds to 12 input types (listed in Table 3) on 4 logical ports:

NAS
Other FSDPSs
AWP
AWP (Hourly)

Unlike the Specialist and DUAT Functional Areas, this Functional Area has two families of Natural Flows. Corresponding to the 4 logical ports is a family of 4 InputDriver Natural Flows, and a family of 4 InputDistributor Natural Flows.

During the Measured Period, the Natural Flows act as follows. For each logical port, separately, an InputDriver Natural Flow and an InputDistributor Natural Flow act as a pair. Taken as a pair, they can be regarded as Prime Source Natural Flows. The InputDriver receives data from an external source, and the InputDistributer executes whatever has to be done in response to the message. A different arrangement is used during Model Initialization, when initialization messages belonging to 5 of the input types are read in on the logical port for AWP. See paragraph 6.6.

3.3.4.5 Time Manager

The Time Manager Natural Flow performs actions that have been previously scheduled by the Scheduler Natural Flow

(described in paragraph 3.3.5.1). Its pattern of flow is similar to that of Specialist and DUAT Natural Flows. It responds to 4 types of requests (listed in Table 3) for scheduled action. The Time Manager is driven once a second by the system timer. Its function is determined by the contents of a file of scheduled events in Permanent Memory.

3.3.5 Service Natural Flows

The remaining Permanent Natural Flows perform a "service" role. Each of them initializes itself, then cycles endlessly. Each time around the loop it waits for a signal from a Natural Flow that requires its services. The form of this signal is described in paragraph 4.3.1. Figure 4 depicts the pattern of flow of a typical Service Natural Flow.

The Service Natural Flows are:

- The Scheduler
- The family of Message Managers
- The family of 5 Common Service Output Drivers
- The family of Legal&Analysis Drivers
- The family of Data Base Managers.

3.3.5.1 The Scheduler

The Scheduler Natural Flow is responsible for updating the schedule of future actions. It accepts new items for insertion into the schedule, and deletes items no longer required (for example, InitiateRescueAlert is no longer required as a scheduled action when a plane is reported as landed).

3.3.5.2 The Message Managers

Each member of the family of Message Managers is responsible for a copy of the Message Pool. It accepts and files messages, and it alerts the addressees at their terminals. The number of Message Managers is a vendor option, see paragraph 7.1.3.

3.3.5.3 Common Service Output

The Common Service Output Functional Area handles spooled output for 5 logical ports; the 4 NADIN logical ports, and

the printer. Each member of the family of 5 Common Service Output Drivers is responsible for driving one of the ports.

3.3.5.4 Legal & Analysis Drivers

Each member of the Legal&AnalysisDriver Natural Flow family drives one of the audit trail output ports. The number of such ports, and the number of drivers, is a vendor option, see paragraph 7.1.4.

3.3.5.5 Data Base Managers

Each member of the family of DataBaseManager Natural Flows is responsible for part of the total data base:

- For initializing the directories during the Model Initialization Phase.
- For the administration of those files that are updated by bulk replacement at regular intervals (referred to as Bulk Update Files, see paragraph 3.5) during the Measured Period.

The number of DataBaseManager Natural Flows is a vendor option, see paragraph 7.1.5.

3.3.6 Temporary Natural Flows

There are six locations in the model where the vendor has the option of having Temporary Natural Flows born (spawned). Three locations are in Route Conversion, two are in Weather Management, and one is in Common Service Input. The first five of the six cases of Temporary Natural Flows are internal to the Functional Area concerned (except for the fact that the Temporary Natural Flows leave the Functional Area when they make calls to Data Base Management). It is expected that most vendors will find options for these three cases of Temporary Natural Flows are advantageous to their configuration, because the options allow the model to make better use of the parallelism that is possible with most disk access software.

The last location where Temporary Natural Flows are spawned is in the Common Service Input Functional Area. This allows each Input Distributor Natural Flow to spawn a Temporary Natural Flow to deal with each message from NADIN. It is expected that, for most vendors, enough parallelism already

exists for dealing with NADIN input, without recourse to this last option. A fuller description of the option is given in paragraph 6.6.

3.4 WORKING SPACE

The third type of component in the model is Working Space. Each allocated Working Space is used as Private Working Space or as Shared Working Space, either, but not as both.

3.4.1 Private Working Space

A Private Working Space is used only by the Natural Flow that acquires it; and it belongs exclusively to this Natural Flow until the Natural Flow releases it. When it is released, its contents are expendable, and the space is available for reallocation.

When a Natural Flow acquires a Private Working Space, this Working Space becomes attached to it in one of 3 states:

- ReadWrite (which means that the model may write to it).
- ReadOnly (which means that the model may read its contents, but will not write to it until further notice).
- DeAccessed (which means that the model will not reference the contents until further notice).

The Working Space remains attached to the Natural Flow as the Natural Flow proceeds from cell to cell. While it remains attached, it has a unique (to that Natural Flow) name by which the Natural Flow can refer to it. There are 8 names available, so a Natural Flow can have up to 8 Private Working Spaces attached simultaneously. The 8 names are:

- DedicatedWorkingSpace
- AdminArea
- SmallInputArea
- WorkArea#1
- WorkArea#2
- RecordArea
- DirectoryArea
- BigIOArea

Details will be found in Table 4.

Each Natural Flow is spawned owning DedicatedWorkingSpace, which it never releases except when it dies, and AdminArea, which is only released during long periods of inactivity. Only the Temporary Natural Flows die before the end of the benchmark run. For the duration of a particular task, a typical Natural Flow will acquire WorkArea#1, and release it when the task is completed. For shorter periods, it may acquire and release WorkArea#2. It will use RecordArea or BigIOArea whenever it is reading from or writing to the Data Base. While it is in the Data Base Management cells, it will also use DirectoryArea. For external input, it uses SmallInputArea or BigIOArea. For external output, it uses Big IOArea.

Whenever the model reads from an external source or from Permanent Memory, it expects the VIP to acquire a Working Space of suitable size in which to deliver the input. In all other cases, the model specifies how much Working Space to acquire. One use of BigIOArea is for accumulating weather reports for a specialist or a DUAT terminal. In this case, BigIOArea grows dynamically as fresh weather records are added to it. BigIOArea is the only Working Space that changes in size once it is acquired.

3.4.2 Shared Working Space

The model uses a limited number of Shared Working Spaces. The minimum is 9, but this can increase as a result of vendor options. Each Shared Working Space is of fixed length, is allocated during model initialization, and exists permanently.

Each Natural Flow can access any Shared Working Space, but only one at a time. The one that is currently being accessed (if any) is referred to, by that Natural Flow, as "SharedWorkingSpace". The access can be either:

- ReadWriteSoleAccess.
- ReadOnlyAllowingReadOnly (implying that any number of Natural Flows can access the same Shared Working Space if they are only reading the contents).

Since they can be accessed by any Natural Flow, the Shared Working Spaces need permanent global names of their own. It is convenient to name each Shared Working Space after a patron Natural Flow; that is, to give it the same run time

IntegerID as a patron Natural Flow. Apart from the fact that its patron is one of its users, this is as far as the association goes. Table 5 shows the Natural Flows that have Shared Working Space named after them. All but one of the patron Natural Flows is a member of a family, so the total number of Shared Working Spaces depends on the various family sizes. For example, there are five OutputDrivers, hence there are five Shared Working Spaces, each named after one of the drivers. If a vendor opts for three DataBaseManagers, there will be three Shared Working Spaces, each named after one of the DataBaseManagers.

It would be misleading to regard the Shared Working Space as owned by the Natural Flow it is named after. On the other hand, the reader is urged to regard the Shared Working Space as associated with the Functional Area that is already associated with the Natural Flow.

3.5 PERMANENT MEMORY

The last of the four types of components in the model is Permanent Memory.

3.5.1 Introduction

The sum of all the modeled uses of Permanent Memory is referred to as the Data Base. The Data Base is modeled as a fixed number of files. The minimum number of files is 45, but the number can increase as a result of vendor options.

Each file is modeled as a fixed number of fixed length records. Records are given integer IDs allocated consecutively either from zero or from one. All the records in any one file are of equal length, except record zero, which when allocated, is usually of a different length.

The request to the VIP, to read or write Permanent Memory, is only issued by Data Base Management cells. For each file, the requests to Data Base Management only come from particular cells, so a correspondence exists naturally between cells and files. The model has been arranged so that each file is only used from the cells of one Functional Area. Tables 6 and 7 list the files by Functional Area.

Nine of the weather files are updated at regular intervals by bulk replacement from the AWP. These files are referred to as the Bulk Update Files. Each is modeled as a pair of files that alternate as the current file.

The Data Base Management in the model assumes only primitive facilities from the vendor's system, and models the strain that an adequate data base management would impose on any underlying system.

3.5.2 File Types

Each file is modeled, mainly at vendor option, as one of four types:

- FixedInformationDirectory
- FixedAddressRecords
- VariableAddressRecords
- DirectoryNotModeled

In the first 3 types, the VIP allocates space for each record individually, on demand from the model. The VIP returns to the model an identifier (referred to as SpaceID) that identifies the allocated space. The model quotes this identifier when it uses the space. Record zero, when used, holds a directory of SpaceIDs for the file.

For FixedInformationDirectory files, record zero is written to only during Model Initialization, or (if it is a Bulk Update file) during a Bulk Update.

For FixedAddressRecords and VariableAddressRecords, two copies of the Directory are used. Both are referred to as record zero, but are distinguished by different SpaceIDs. Each successive update of the directory goes to the other SpaceID. With these files, a user lock on a record can survive system failures.

VariableAddressRecords files have the additional feature that each successive update of any one record goes to a newly acquired allocation of space. These files model the case where the old copy of a record must be available if a system failure occurs during an update.

For some files, the vendor is permitted to handle the record searching entirely within the VIP. If the option is taken, the file is then of type DirectoryNotModeled. In this case,

no SpaceID is returned to the model, or given back to the VIP. The VIP locates the record based on file ID and record number. There is no record zero.

Table 8 shows the potential Permanent Memory transfers for each type of data base operation, for the first three file types. In general, it would seem that FixedInformationDirectoy imposes the least strain, and VariableAddressRecords the most strain, on the system. However, this observation has to be qualified by two factors:

(1) Only the writes to Permanent Memory are obligatory. Reads are unnecessary if an up to date copy of the record is available in main store (this applies equally to record zero). The vendor can hold copies in main store of as many records and directories as desired (space permitting).

(2) Not all operations are applied to all files during the Measured Period. Table 9 lists the operations used on each file.

Table 10 shows the restrictions on the vendor's choice of file type.

3.5.3 Bulk Update Files

For Bulk Update Files, a pair of VSRs has been defined that allows the vendor to make the bulk replacement however desired. As explained in paragraph 3.5.2, each file is modeled as a pair of files. To the VIP, the bulk update is a bulk write to one of the pair. When it receives the DeleteFile VSR, it can either delete the copy, or save the space for the next update.

The model records which of the pair is current, directs weather retrieval to the correct one, and tells the VIP which one to bulk write to and delete. It also handles the following two complications:

(1) Each retrieval is a batch of reads from the same file. This batch of reads may be interrupted by a bulk update, but the batch of reads must be completed on the original data and in the correct sequence.

(2) The vendor may wish to repeat the files in several processors.

4. STRUCTURE OF A SERVICE FUNCTIONAL AREA

4.1 INTRODUCTION

So far, the model has been discussed in relation to 14 Functional Areas. This breakdown is useful for understanding the model, but it is not sufficient for model mapping. The demarcations between Functional Areas are not the best demarcations between processors. For a good distribution, we need to descend to one more level of detail. Knowledge of how the Natural Flows of one Functional Area use the services provided by another Functional Area is required.

This section describes the structure of those Functional Areas that provide a service to other Functional Areas. Thus, it applies to:

Time Management (one half of)
Graphics Management
Flight Plan Management
Route Conversion
Weather Management
Message Management
Common Service Output
Legal & Analysis Management
Data Base Management

It is necessary to introduce 2 new terms.

(1) A "client" of a Service Functional Area is a Natural Flow (usually from another Functional Area) that requests the services of this Service Functional Area.

(2) A "Reception Desk Cell" is a cell that acts as an entry point for procedure calls to a Service Functional Area.

4.2 RECEPTION DESK CELLS

The term Reception Desk Cell has been adopted because of the following analogy, which illustrates the role of this class of cell. In the real world, a person visiting an organization, to obtain a service, may go first to a reception desk. Here, one of several things can happen:

(1) The visitor (client) does the work and;

(a) Transacts the business on hand, on the spot, and leaves, or

(b) Is directed further into the building to transact the business, and then leaves.

or

(2) An internal employee does the work. The client leaves a message at the desk, for people within the organization to act upon, and;

(a) Leaves immediately, or

(b) Waits at the desk until the request has been honored.

In the model, a client Natural Flow that requires the services of another Functional Area always enters, by procedure call, one of the Reception Desk Cells provided with that Functional Area. One of several things can happen:

(3) The client Natural Flow does the work, and;

(a) Obeys code in the cell, then exits, having itself done what had to be done, or

(b) Makes procedure calls to further cells within the Functional Area.

or

(4) The work is performed by a Service Natural Flow. The client Natural Flow, while obeying code in the Reception Desk Cell, signals one of the Natural Flows associated with that Functional Area. The Service Natural Flow will do whatever is required to be done. The client;

(a) Exits immediately, or

(b) Waits for a return signal, to indicate that the job has been done.

Figure 5 shows how the simple case of 3(a) and 3(b) can be depicted in the standard symbology of this document.

To show cases 4(a) and 4(b), knowledge of the mechanism of communication between client and Service Natural Flow is needed.

4.3 COMMUNICATION MECHANISMS

To communicate its requests to a Service Natural Flow, the client uses either the "Prod" or Shared Working Space mechanisms, or both.

4.3.1 Prods

In case 4(a) of paragraph 4.2, the client signals the Service Natural Flow with a VSR known as a Prod. In its simplest form, a Prod is merely a signal to restart a Natural Flow that is waiting for such a signal. The Service Natural Flow loops endlessly. Each time around the loop it uses a VSR known as AwaitProd to wait for and accept the next Prod. This is shown symbolically in Figure 6(a).

The Prod is specified in such a way as to facilitate communication between processors. A Natural Flow in one processor can Prod a Natural Flow in another processor. In the general case the Prod signal does more than merely awaken the target. The Prod can carry information in various forms. More details are given in paragraph 6.11. Full details are in Part V.

The case 4(b) of paragraph 4.2 is implemented by having two Prod-AwaitProd pairs, see Figure 6(b). The client Prods the Service Natural Flow, then waits for a return Prod to indicate the task is completed.

During the Measured Period, the following Prods occur:

- Clients Prod;
  MessageManager (Prod pair),
  Scheduler (Prod pair),
  OutputDrivers,
  Legal&AnalysisDrivers, and
  DataBaseManagers.
- DataBaseManagers Prod each other.
- TimeManager Prods Scheduler (Prod pair).
- CommonServiceInputDrivers Prod corresponding InputDistributors.
- Temporary Natural Flows Prod the Natural Flows that spawned them.

4.3.2 Shared Working Space

In addition to the Prod mechanism, Shared Working Space is also used. A Shared Working Space is named after the Service Natural Flow. The Service Natural Flow and the client Natural Flows access the Shared Working Space. In some cases, Shared Working Space is the more suitable vehicle for communicating between client and Service Natural Flow. For example, in Legal & Analysis Recording, the Shared Working Space models the buffer. Clients add to it while they are executing in the Reception Desk Cell (as in the case of 3(a) of paragraph 4.2). When the buffer becomes full they Prod the Service Natural Flow to deal with it (as in the case 4(a) of paragraph 4.2). The use of Shared Working Space is shown symbolically in Figure 7.

See Table 5 for a list of the Shared Working Spaces.

4.4 INNER SET AND OUTER SET CELLS

Some of the Service Functional Areas are divided into inner set cells and outer set cells.

For the remaining Service Functional Areas, and for the Prime Source Functional Areas, it is useful to regard the whole complement of cells as inner set cells.

Inner set cells are those that use the files and the Shared Working Space associated with the Functional Area. Most vendors will get a better mapping if they place all the inner set cells of a particular Functional Area in the same processor. The demarcation line between inner and outer sets of cells represents the most suitable place to do the interprocessor transfer of control, if a client Natural Flow in another processor needs to use the inner set cells of the Functional Area.

Service Natural Flows executing in their home Functional Area only use the inner set cells. Reception Desk Cells of a Service Functional Area can be either inner set or outer set. All outer set cells are either Reception Desk Cells, or extensions thereof.

For those who find analogies helpful, an outer set cell can be likened to the office that an organization sets up in a foreign country with whose inhabitants it does business: for the purpose of providing an interface locally, and

particularly for taking orders. In the case of the model, the model mapper will want to keep the inner set cells of a Service Functional Area together on the home processor, but will duplicate the outer set cells on each processor that hosts clients of that Functional Area.

4.5 MIGRATION POINTS

When a Natural Flow transfers control to another processor, it is said to migrate. In some cases the VIP is allowed to implement the migration in an alternative way, in which a message to the VIP in the target processor instructs it to set up a Temporary Natural Flow there. The term migrate is used to cover both kinds of implementation.

If the outer set cells of a Service Functional Area are placed in the client's processor, then they can be reached without an interprocessor transfer. On the other hand, the inner set Reception Desk Cells may only be reachable by migrating to the Service Functional Area's processor. Thus, the entry points to the inner set Reception Desk Cells are possible Migration Points. These points represent places in the real application where a separation between processors would be sensible. They have been encapsulated in the model as a set of PossibleMigrationPoints.

It is not suggested that the vendor introduce a separation between processors at every migration point. However, the separation between processors should be restricted to a subset of these points. By basing the mapping on inner set and outer set cells one should be able to achieve a sensible mapping with all migration occuring at such Migration Points. If this is done, then the model can help the VIP in its administrative task.

The use of a Migration Point can be stated as follows. Suppose A and B are two Functional Areas and that a Natural Flow which is executing in the cells of A is about to transfer control to B. Other things being equal, the model mapper will arrange to have A and B in the same processor, so that there is no actual migration. If, however, some other consideration forces the model mapper to place them in different processors, then the Migration Point is recommended as the best place to make the migration.

The Migration Points that have been placed explicitly in the model serve 2 purposes:

(1) As documentation, in the Modeling Language listing, they suggest to the model mapper where processor divisions should be made.

(2) In the corresponding HOL code, they are reflected in the following manner. The normal transfer from cell to cell (whether it be a GoTo or a procedure call at Modeling Language level) becomes, in the HOL code, the VSR GoToCell. A transfer to a Migration Point, however, becomes the VSR BigGoTo, and the corresponding return becomes the VSR ReturnFromBigGoTo. The implications of this concern the VIP writer, but the model mapper should be aware that:

    (a) If the VIP cooperates at the BigGoTo, then the model can tell the VIP where to go back to on the corresponding return.

    (b) The alternative form of migration mentioned above is allowed at a BigGoTo.

In a few cases, a cell contains the Modeling Language statement PossibleMigrationPoint followed by the comment "technical only". The statement is present for code generation purposes, and should be ignored by the model mapper.

Figure 8 show how Migration Points and the inner and outer sets are depicted in the diagrams of this document. Tables 11 and 12 list the cells of each Functional Area according to whether they are inner set or outer set.

5. MAPPING THE MODEL

5.1 THE MAPPING TASK

5.1.1 What Has To Be Done

As was explained in the Introduction (1.), the model mapper has to address a set of questions. These questions divide into:

- Distribution questions related to mapping the model across processors.
- A set of miscellaneous questions related to vendor options, see section 7.

The distribution questions are discussed in this section. Decisions need to be made for the following:

(1) Where each component of the model should be placed at the start of the benchmark run.

(2) For each component, should it move from processor to processor during the run and, if so, on what basis should the decisions to move be made.

5.1.2 The Rules

The model mapper must work within the following rules:

(1) Each Natural Flow shall be unique (that is, there shall not be two simultaneous instances of the same Natural Flow). There are vendor options for having some of the Natural Flows duplicated explicitly in the model, as members of the same family, but each member shall be implemented uniquely.

(2) The authoritative copy of each Permanent Memory record shall be unique, and shall always be in Permanent Memory.

(3) At no time shall there be 2 or more copies of the same Working Space having different contents (in other words, when the model accesses one copy in ReadWrite status, other copies immediately become invalid).

(4) When a Natural Flow is executing, the Working Spaces which are attached to it as ReadWrite or ReadOnly must be present in the main store of the same processor.

There are additional rules that are elaborated in Part V. These limit the extent to which two Natural Flows can use the same module simultaneously, and which module can be selected when a Natural Flow returns to a cell after an interruption (as opposed to a return from VIP servicing). The model mapper is unlikely however, to be concerned with these restrictions.

5.1.3 Considerations

It will be seen that these rules allow the model mapper considerable freedom. For example:

- For each record, separately, in the Data Base, the authoritative copy can be held in the Permanent Memory of any processor; and any number of copies of it can be kept anywhere (provided of course that they are no longer used when they become out-of-date).

- Each Working Space, separately, can be held anywhere when it is not being accessed by a Natural Flow.

- Each cell can be repeated in any number of modules. Each module can be repeated any number of times. Each copy of each module can be anywhere at any time. When a Natural Flow returns to a cell from a VSR it can return to any copy of any module that contains that cell (except, in some cases, if another Natural Flow is using it, see Part V).

- Each Natural Flow can execute in any processor.

However, the benchmark may not fairly reflect the vendor's proposal unless the model mapper has taken some care over the mapping. The aspects that the model mapper should consider are resources, and VIP administration.

Clearly, each processor must have the memory and processing capability for the tasks that are assigned to it. Interprocessor channel capacity should also be considered carefully.

The VIP frequently has to locate components of the model.

The question can take either or both of 2 forms:

- Which instance of the component shall be selected (for example, which copy of a cell, or which member of a family of Service Natural Flows)?

- Where is it?

The model mapper can place an unnecessary burden on the VIP (VIP designer, run time resource requirements, or both), if the mapping is not done properly.

5.2 GENERAL APPROACH

5.2.1 Procedure

For all vendors, the model mapper's general approach should be as follows:

(1) Decide, tentatively at first, on which processor each component of the model should start, and the policy on migration of components from processor to processor.

(2) Assess the resource utilization of each component of the configuration, and identify both over utilization and excess redundancy.

(3) Iterate steps (1) and (2) until a satisfactory arrangement has been found.

(4) Optionally, refine the arrangement as described in paragraph 5.6.

For calculating resource utilization in step 2, the model mapper can use the data presented in Appendix A. If the only concern is to optimize the resource match between model and configuration, then no further advice can be given, except to point out that it may be easier to work at the level of Functional Areas, rather than at the more detailed level of the individual cell. Working with Functional Areas may bring the added benefit of reducing interprocessor traffic.

The rest of section 5 is addressed to the model mapper who is concerned also about the VIP's task.

5.2.2 The VIP's Task

The VIP has to know which instance of each component to select and where this instance is to be found. The model is designed to help, in connection with Prod pairs and Temporary Natural Flows.

5.2.3 Prod Pairs

It was mentioned in paragraph 4.3.1, that there are cases where a client Natural Flow Prods a Service Natural Flow and then waits for a return Prod. In order to save VIP table space, the model keeps and quotes to the VIP a two-part return link; the client's Natural Flow ID and a return Processor ID. The former is automatically kept by the model. The latter requires the cooperation of the VIP on the occasion of the first of the two Prods.

5.2.4 Temporary Natural Flows

When one Natural Flow spawns a second Natural Flow, the two are referred to as "parent" and "offspring" respectively. Temporary Natural Flows Prod their parents, but parents never Prod their temporary offspring (they do not need to, because each offspring is spawned holding sufficient data to define its task). Thus, neither an offspring's ID nor its whereabouts need be known in the processor of the parent, but a return link to the parent has to be held with the offspring.

To save VIP table space, the model keeps and quotes to the VIP the parent's ID and processor. The former is automatic in the model, but the latter requires the cooperation of the VIP when the offspring is spawned. Clearly, this assistance by the model is of less use if the parent's wherabouts cannot be known in advance, but the model is coded such that the parent is unlikely to await Prods from offspring anywhere but on the processor from which it spawned them.

5.2.5 Migration

Three reasons for migrating a Natural Flow are:

- To balance CPU utilization.
- To access files.
- To access Shared Working Space.

5.2.5.1 CPU Balancing

The model mapper must work out the policy on balancing the CPU load across processors. If migrations are arranged at the Migration Points introduced in paragraph 4.4, then the interprocessor traffic will probably be less. In addition, the BigGoTo VSR generated by a transfer to a Migration Point can be used to alert the VIP that a decision is required.

5.2.5.2 Access to Files

Where a Natural Flow in one processor needs to access a Permanent Memory record in another processor, three approaches are possible:

(1) The VIP can avoid a migration by bringing a copy of the record to the processor of the Natural Flow. The simplicity of this approach is attractive, but the interprocessor traffic that would result should be carefully considered. The vendor with very fast interprocessor channels may prefer this approach.

(2) The Natural Flow can be migrated when it first refers (in a VSR) to the record. With this approach it may be difficult to know when, or even where, to return; otherwise it could be a useful approach.

(3) The Natural Flow is migrated at a Migration Point before it first refers to the record. This is the approach that has been assumed in the suggested scheme that is developed in paragraphs 5.3 through 5.5. Instructions to the VIP writer on how to detect where to migrate are in Part V.

5.2.5.3 Access to Shared Working Space

The same three basic approaches apply to accessing Shared

Working Space as have just been described for accessing files. However, the first of the three alternatives is less attractive in the case of Shared Working Space. With Permanent Memory, the model administers the policing of simultaneous access to the same record; with Shared Working Space the VIP does the policing. This might be difficult across processors.

As with files, the suggested approach developed in paragraphs 5.3 through 5.5 follows the policy of migrating at Migration Points before the first request for access.

5.3 A PARTICULAR APPROACH

Migration Points are designed to help the model mapper and the VIP writer in simplification of the administrative task of distributing across processors. However, they can only be helpful if the model mapper stays within the particular approach developed here. This is illustrated by a simple example in paragraph 5.4, and is refined in paragraph 5.5. The general procedure being followed is that described in paragraph 5.2.1. The details of distributing functional areas across processors is discussed throughout this particular approach.

The model mapper is not obliged to follow this approach since it may not lead to the best solution for the proposed configuration. Note that if the vendor has interprocessor channels of very large capacity, then the mapping problem is more straight-forward since accessing of files and working space on another CPU may have minimal impact on performance.

5.3.1 Procedure

The general procedure described in paragraph 5.2.1 is used in the particular approach but with a change to Step (1). In the general approach, each component of the model (that is, each cell, each Natural Flow, each Working Space, and each record) is considered separately, unless the vendor finds a suitable way to consider them in groups. In the particular approach, they are considered in groups based on Functional Areas. The procedure for the particular approach can therefore be stated thus:

(1) Decide (tentatively at first) on which processor(s)

(2) Assess the resource utilization of each component of the configuration, and identify both over utilization and excess redundancy.

(3) Iterate steps (1) and (2) until a satisfactory arrangement has been found.

(4) Optionally, refine the arrangement as described in paragraph 5.6.

In step (1), two topics remain to be discussed:

- The precise meaning of the statement "Place the Functional Area in processor n" (defined in paragraph 5.3.2).

- The possibility of having a Functional Area on more than one processor (developed in paragraph 5.5).

The procedure to be followed here is that defined in this paragraph, as further devloped and refined from here through the end of paragraph 5.5. If the model mapper follows this procedure, and if the following are observed:

- The VIP never migrates a Natural Flow except at a BigGoTo or at the corresponding ReturnFromBigGoTo. See paragraph 5.5.

- At a BigGoTo the VIP always returns, to the model, the processor # of the processor on which the VSR was issued (this VSR is specified to return a processor # of the vendor's choice).

- On each ReturnFromBigGoTo, the VIP always migrates the Natural Flow to the processor quoted by the model if the processor quoted is different from the current processor.

then the following will always be true:

- Whenever files are wanted, they will always be available on the processor on which the request was made.

- Whenever Shared Working Space is required to be accessed, it will be found in the processor on which it is requested.

- During the Measured Period, the VIP need record in each processor only the following ProcessorIDs:

a. That of the Top Level Functional Area.

b. That of each copy of each Service Functional Area.

c. Anything else required for the VIP to make intelligent decisions as discussed in paragraph 5.5.

5.3.2 Terminology

Before describing the particular mapping scheme, two basic operations are defined.

The statement, "Place the Functional Area A in processor P", means:

(1) Load the inner set cells of A in P (or all the cells, if there are no outer set cells); and load the outer set cells, if any, in every processor from which clients of this Functional Area call the outer set Reception Desk Cells.

(2) Have the VIP arrange for the Natural Flows(s) associated with A to be spawned in P.

(3) Have the VIP arrange for the Shared Working Space(s) associated with A to reside in P.

(4) For the files associated with A:
   (a) Make a note that they are to reside in P.
   (b) Have the VIP arrange that Permanent Memory in P is allocated to them.

The statement, "Place a copy of the Functional Area A in processor P", means:

(1) Load copies of the inner set cells of A, in P (or all the cells, if there are no outer set cells); and load the outer set cells, if any, in every processor from which clients of this Functional Area call the outer set Reception Desk Cells.

(2) Have the VIP arrange for one member of each Natural Flow family that is associated with A to be spawned in P.

(3) Have the VIP arrange to place, in P, the Shared Working Spaces named after the Natural Flows of A that are spawned in P.

(4) The files associated with A should be distributed as explained in paragraphs 5.4 and 5.5.

5.4 A SIMPLE EXAMPLE

The mapping scheme is introduced with a simple example depicted in Figure 9. The simplified model has 4 Functional Areas:

- A Top Level Functional Area (which we can ignore for the moment because it is inactive during the Measured Period).
- A Prime Source Functional Area that has several Natural Flows and several files associated with it, but no Shared Working Space.
- A Service Functional Area that has one Natural Flow, a Shared Working Space, and files.
- A DataBaseManagement Functional Area with one optional size family of Natural Flows, one Shared Working Space associated with each Natural Flow, and no files.

This model is to be mapped onto two processors. Proceed as follows (see Figure 10):

(1) Place the Prime Source Functional Area in Processor 1. This will associate the files of that area with Processor 1.

(2) Place the Service Functional Area in Processor 2. This will associate the remaining files with Processor 2. Next, place its outer set cells in Processor 1, because that is where its clients are located.

(3) Request from the Government a version of the Model with 2 Data Base Management Natural Flows, because Data Base Management is to administer files on both processors.

(4) Place a copy of the Data Base Management Functional Area in each processor. This gives a correspondence between files and Data Base Management Natural Flows. Tell the Government which files go with Data Base Management Natural Flow #1, and which go with Data Base Management Natural Flow #2. The HOL model received will reflect this allocation of files. This is a simple case of a more general subject,

distributing Functional Areas, that is dealt with in paragraph 5.5.

If, in this simple example, the VIP observes the following:

(1) Never migrate a Natural Flow except at a BigGoTo to the Service Functional Area, and at the corresponding ReturnFromBigGoTo. The VIP can recognize the relevent BigGoTo's by inspecting the required target cell ID.

(2) At the BigGoTo's, return to the model the processor # of the processor on which the VSR was issued. This VSR is specified to return a processor # of the vendor's choice.

(3) At each ReturnFromBigGoTo, make the Natural Flow migrate to the processor which the model quotes at this point. In all cases, it will be the original processor from which the corresponding BigGoTo came.

then the following will always be true:

(1) Whenever files are wanted, they are available in the processor on which the request is made.

(2) Whenever Shared Working Space is required to be accessed, it will be found in the processor on which the request is made.

(3) The VIP needs only the following information about what is in which processor:

- In both processors, it needs the ProcessorID of the Top Level Functional Area, for end-of-benchmark purposes.

- In processor #1, it needs the ProcessorID of the Service Functional Area, because clients are in processor #1.

Several useful points have been illustrated that can be applied later to the more complicated real case. The major points are:

(1) Mapping is based on Functional Areas.

(2) In general, an n:m correspondence between processors and Functional Areas can be expected:
   (a) There can be several Functional Areas on a processor.
   (b) A copy of a Functional Area can be present on each

of several processors.

(3) The outer set cells of a Service Functional Area reside in the processor of the client. Thus, in a more complicated case, one might want to repeat them in each processor that hosts clients of that Functional Area.

(4) Data Base Management is repeated in every processor that contains files.

5.5 DISTRIBUTING FUNCTIONAL AREAS

The example just described would give the vendor little choice in mapping. Except at the cost of greatly increasing interprocessor traffic, the simplified model only allows mapping to 1 or to 2 processors. In the case of 2 processors, there is no flexibility in the distribution. If the load is distributed unevenly, then the configuration will be unbalanced.

In reality, the vendor's problem is more complicated than in this simple example, but greater flexibility exists for achieving a balanced distribution. This is because there are more Functional Areas (14, not 4) and the real model contains more options for distributing Functional Areas across processors (in the simple example, only Data Base Management could be repeated in each processor). The distribution of Functional Areas is explained in paragraphs 5.5.1 and 5.5.2 by building on the example of Figures 9 and 10.

5.5.1 Distributing Prime Source Functional Areas

5.5.1.1 Introduction

If the Prime Source Functional Area of the example in Figure 9 has no files and no Shared Working Spaces, then there is no problem in distributing it across processors. Repeat the code on each processor. Distribute the Natural Flows, some on each processor; for example, if there are 10 Natural Flows and 3 processors, Natural Flows 1 to 4 could be spawned in processor i, 5 to 7 in processor j, and 8 to 10 in processor k.

None of the Prime Source Functional Areas in the FSIS model

have Shared Working Spaces. Three of them have files; the distribution of files is made easier by vendor options in the model.

5.5.1.2 The Specialist Interface Functional Area

Three files are associated with the Specialist Functional Area:

- TerminalRecordFile
- RouteRecordFile
- CurrentFlightPlanFile

Each of these files has 91 records in the default case, one record per terminal. To make distribution possible, the model is written so that each of the three files is actually a family of files, that degenerates to a one-member family when the Functional Area is to be confined to one processor.

Suppose the vendor opts to distribute this Functional Area across 2 processors. For example:

- Specialists 1 to 45 on processor i
- Specialists 46 to 91 on processor j

Then the vendor would proceed as follows:

(1) Repeat the code of the Specialist Functional Area on both processors.

(2) Tell the Government how many processors the specialists are to be spread over, and which terminals go with which processor. The Government will send a HOL tape that reflects the required arrangement. In the case of this example, the model would have three families of two files each:

TerminalRecordFile Number 1
RouteRecordFile Number 1
CurrentFlghtPlanFile Number 1

having 45 records each, for specialists 1 through 45, and

TerminalRecordFile Number 2
RouteRecordFile Number 2
CurrentFlightPlanFile Number 2

having 46 records each, for specialists 46 through 91.

If the Prime Source Functional Area of Figure 9 were the Specialist Functional Area just discussed, then Figure 11 would depict the resulting distribution. Note how the outer set cells of the Service Functional Area are now repeated with each of its 2 client Functional Areas and that 3 copies of Data Base Management are now required.

If the vendor cannot place the files on the processors assigned to the Specialist Functional Area, then there is no advantage in splitting them into multimember families. See paragraph 5.5.2.8 for what to do when files cannot be where they are used.

5.5.1.3 DUAT Interface Functional Area

The case of distributing the DUAT Functional Area is similar to that of distributing the Specialist Functional Area. The three files, in this case, are;

- DUATTerminalRecordFile,
- DUATRouteRecordFile, and
- DUATCurrentFlightPlanFile.

Each file has 40 records each in the default case.

5.5.1.4 Radar Management Functional Area

There are no provisions in the model for distributing this Functional Area across processors.

5.5.1.5 Common Service Input Functional Area

Whether the vendor would gain by distributing the Common Service Input Functional Area across processors has to be a vendor dependent decision. This Functional Area has no files and no Shared Working Spaces, so it would be easy to distribute. The logic of the model is unaffected by the whereabouts of the 8 Natural Flows.

5.5.1.6 Time Management Functional Area

There are no provisions in the model for distributing this Functional Area across processors. Presumably, the model mapper will place it on the same processor as the Flight Plan Management Functional Area.

5.5.2 Distributing Service Functional Areas

A Service Functional Area can be distributed across processors by repeating its inner set cells on each of those processors, and by repeating its outer set cells on every processor from which calls are made to them. The complications arise when the Functional Area has files, Natural Flows, or Shared Working Space. If the files can be of the type DirectoryNotModeled, then they can be spread across processors, or there can be copies of them on other processors, provided that the vendor's software can do this in a satisfactory manner. The vendor is cautioned against the complication of interprocessor access to Shared Working Space.

5.5.2.1 Graphics Management Functional Area

The Graphics Management Functional Area has one file, the AFOS Graphics File, but no Natural Flows. There is no option to have the file as a multimember family. It is not expected that vendors will want to distribute this Functional Area across processors.

5.5.2.2 Flight Plan Management Functional Area

The Flight Plan Management Functional Area has one file, the Flight Plan File, but no Natural Flows. There is no option to have the file as a multimember family. It is not expected that vendors will distribute this Functional Area across processors.

5.5.2.3 Route Conversion Functional Area

The Route Conversion Functional Area has two files:

- The Airways & Fixes File.

- The Reporting Locations File.

and options for Temporary Natural Flows, but no Permanent Natural Flows. The vendor is permitted to have the files of type DirectoryNotModeled. Since the files are only written to during initialization, it should be simple to have them of this type and to have up-to-date copies on several processors.

To distribute the Functional Area across processors:

- Ask for the files to be of type DirectoryNotModeled.
- Arrange for the VIP to handle this type of file, with copies on several processors.
- Arrange for the VIP to select a copy of the Functional Area, at each BigGoTo to the Functional Area. The choice can be dynamic, based on current loads; or it can be static, based for example on client ID.
- If any of the options for temporary spawning are taken, then the VIP can place each Temporary Natural Flow, separately, on any of the processors.

If the Service Functional Area in the example of paragraph 5.4 were Route Conversion, then Figure 12 would depict the resulting distribution.

5.5.2.4 Weather Management Functional Area

The Weather Management Functional Area has a large number of files, some outer set cells, no Permanent Natural Flows, but options for Temporary Natural Flows. The vendor is permitted to have the files of type DirectoryNotModeled, but in this case many of the files are written to during the Measured Period. Whether it is easy to have them of the type DirectoryNotModeled, and to have up-to-date copies on several processors, will depend on the vendor's own software. To distribute the Functional Area across processors:

- Place the outer set cells wherever there are calls to them.
- Ask for the files to be of the type DirectoryNotModeled.

- Arrange for the VIP to handle this type of file, with up-to-date copies on several processors.

- See paragraph 5.5.2.8 for the Bulk Update files.

- Arrange for the VIP to select a copy of the Functional Area at each BigGoTo to the Functional Area. The choice can be dynamic, based on current load; or it can be static, based for example on client ID.

- If any of the options for temporary spawning is taken, then the VIP can place each Temporary Natural Flow on any of the processors.

Figures 13 and 14 show the Weather Management Functional Area distributed on 2 processors, for the cases with and without temporary spawning, respectively.

5.5.2.5 Message Management Functional Area

The Message Management Functional Area has:

- One optional size family of Natural Flows, each member having a Shared Working Space named after it.

- Two families of files, of the same family size as the Natural Flows.

- One inner set and one outer set Reception Desk Cell.

Thus, if the model mapper elects to have two-member families, a HOL model with 2 Message Management Functional Areas will be provided, each having a Natural Flow, and one member of each of the two file families.

The model is coded so that when a message is delivered to the Message Management Functional Area, it is filed with each copy of the Functional Area. When it is retrieved by an individual specialist it is retrieved from only one of those copies. The default arrangement, which is recommended, is that a copy of the Message Management Functional Area be placed on every processor that contains a copy of the Specialist Functional Area. In this way, there may be interprocessor transfers when the message is filed, but there will be none each time it is viewed by a specialist, nor will there be any migrations. In the example used in paragraph 5.5.1.2, this would mean:

- Message Management inner set cells on processors i and j.

- A family of 2 Message Management Natural Flows, one spawned on processor i, and one on processor j.

- Two Shared Working Spaces, one on processor i and one on processor j.

- Two families of two files each:

    - MessageFile Number 1
    - TerminalMessageListFile Number 1

  on processor i, and

    - MessageFile Number 2
    - TerminalMessageListFile Number 2

  on processor j.

The HOL model will reflect this, based on the options selected for specialists unless the vender opts otherwise. Figure 15 illustrates this example.

If the model mapper does not take the default arrangement suggested, then the VIP will have to detect BigGoTo's to the inner set Reception Desk Cell, and direct the client's migration appropriately. There is a vendor option (which is taken care of automatically in the default case) that associates a member of the Message Management family with each specialist. The VIP should migrate the client (who is a specialist, in this case) to the processor on which the associated copy of Message Management has been placed. Otherwise the model will request interprocessor access to files and to Shared Working Space.

5.5.2.6 Common Service Output Functional Area

The Common Service Output Functional Area has:

- A family of 5 Natural Flows (one per logical port).

- A family of 5 spool files (one per logical port).

- Five Shared Working Spaces (one per logical port).

- No outer set cells.

If this Functional Area is distributed across processors, the model mapper should ensure that, for each logical port separately, the Natural Flow, the spool file, the Shared Working Space and a copy of the code are on the same processor. Otherwise, the model will request interprocessor access to Shared Working Space and to files.

The model mapper has a separate choice for each logical port. A copy of the Functional Area can be placed either where the physical port is, or where its most frequent clients are. Either way, unless client and port are on the same processor, there has to be interprocessor traffic. The first way, a Natural Flow migrates to an inner set Reception Desk Cell. The second way, the VIP has to deliver the output via another processor.

It is important that the client use a Reception Desk Cell on the same processor as the Natural Flow that is associated with the logical port for which it wants output. Otherwise, the model will request interprocessor access to files and to Shared Working Space. Unless the model mapper has been very careful with mapping, the VIP will have to recognize BigGoTo's to Common Service Output, deduce the logical port (see paragraph 7.3.4 of Part V) and migrate the Natural Flow appropriately.

5.5.2.7 Legal & Analysis Recording Functional Area

The Legal & Analysis Recording Functional Area has:

- One optional size family of Natural Flows. Each Natural Flow has a Shared Working Space named after it.
- No outer set cells.
- No files.

It is recommended that the vendor repeat the Functional Area on each processor that has a Legal & Analysis Recording outlet. Tell the Government the family size needed and which member will be on which processor. The HOL version of the model will reflect this choice. It is important to place each copy of the Functional Area on the processor quoted, otherwise the model will request access to a Shared Working Space on another processor.

It would probably be advantageous to have a recording outlet on each processor from which calls to this Functional Area are made. If this is not possible:

(1) There will be an increase in interprocessor traffic.

(2) The VIP will have to make run time decisions. It must recognize BigGoTo's to this Functional Area, and decide to which copy each client will be directed. Instructions for this are in paragraph 7.3.4 of Part V.

Calls to this Functional Area are made from:

- The Specialist Interface Functional Area.
- The DUAT Interface Functional Area.
- The Common Service Input Functional Area.
- The Flight Plan Management Functional Area.
- The Common Service Output Functional Area.

To summarize from a different point of view:

- On any processor which has clients of Legal & Analysis Management, but no copy of the Legal & Analysis Recording Functional Area, the VIP will have to recognize BigGoTo's to the Functional Area, and migrate to a processor that has a copy.

- From any processor on which there is a copy of the Legal & Analysis Recording Functional Area, but no recording outlet, the VIP will have to send each output to a processor that does have an outlet.

- If the family size is greater than 1, then each member must be placed on the processor quoted to the Government or interprocessor access to Shared Working Space will result.

5.5.2.8 Data Base Management Functional Area

The Data Base Management Functional Area contains:

- One optional size family of Natural Flows. Each Natural Flow has a Shared Working Space named after it.

- No files of its own.

- No outer set cells used during the Measured Period.

Each reference to a file of a type other than DirectoryNotModeled involves an access to the Shared Working Space. The vendor is advised to:

(1) Place a copy of the Data Base Management Functional Area on each processor in which files, other than the type DirectoryNotModeled, reside.

(2) Have each file, other than the type DirectoryNotModeled, administered by the local copy of the Functional Area.

Tell the Government how many copies are needed (in other words, the family size for the Natural Flows), and which files are to be administered by which member.

Files of type DirectoryNotModeled do not need Data Base Management except as described for Bulk Update files. However, they do need two small DataBaseManagement cells for reading and two for writing. These should be on the same processor as the client. They are not shown on Figures 11 to 14.

During model initialization, there will be a BigGoTo to the cell StandardInitializationOfFiles. This cell can be anywhere. From this cell there will be many BigGoTo's to the cell WriteNewRecordKeepingWorkingSpace. The VIP should recognize these calls and direct the client (the Input Driver for the AWPLink) to the processor holding the Data Base Management Natural Flow responsible for each file.

If the configuration does not allow a file to be placed where it is to be used, then there are two approaches:

(1) Place a copy of Data Base Management Functional Area in the processor where that file is to be used, and let the VIP fetch and deliver records.

(2) Place a copy of Data Base Management Functional Area on the same processor as the file. Have the VIP detect each BigGoTo call to Data Base Management Functional Area. If the BigGoTo refers to that file, migrate the client to the processor holding that copy of Data Base Management Functional Area. Directions for the VIP are in Part V, paragraph 7.3.4.

The second method would involve fewer interprocessor transfers in all but the simplest cases.

For each Bulk Update file that is repeated on several processors, there is more to consider:

(1) Identify the Bulk Update file as the type DirectoryNotModeled (otherwise, there will be interprocessor accesses to the Shared Working Space of the Data Base Management Natural Flow).

(2) Place a copy of the Data Base Management Functional Area on each of the processors, and identify to the Government the Natural Flow of each as a Local Bulk Update Manager for that file.

(3) Identify a DataBaseManager to the Government as the "Controlling Bulk Update Manager" for that file. It need not be one of the Local Bulk Update Managers for the file.

(4) Have the VIP detect BigGoTo's to the cell BulkUpdate, and direct the client (The Input Distributor for the AWPHourly logical port) as follows. If the file is of type DirectoryNotModeled, then the client should be directed at the convenience of the VIP; otherwise, the client should be directed to the processor containing the Data Base Manager that administers this file.

If there are no constraints on where files reside, then it should be possible to map the model so that the only migrations to the Data Base Managers are during the Model Initialization Phase, and possibly also when Bulk Updates arrive from the AWP.

To summarize for each file that is not of type DirectoryNotModeled:

- If the file and the associated Data Base Manager are not on the same processor, then the VIP will have to fetch and return the records.

- The VIP must detect BigGoTo's as follows, and migrate clients to the processor containing the associated Data Base Manager; otherwise, there will be interprocessor access to Shared Working Space:

  (a) If the associated Data Base Manager is not placed on the client's processor, BigGoTo's to the reception desks for normal operations should be detected, and migration arranged.

  (b) If, for Bulk Update files, the associated Data Base Manager is not placed on the same processor as the

Input Distributor for AWPHourly, BigGoTo's to the cell BulkUpdate should be detected, and migration arranged.

(c) During initialization, BigGoTo's to the cell WriteNewRecordKeepingWorkingSpace should be detected, and migration arranged.

To summarize, for each Bulk Update file that is repeated on more than one processor:

- Type DirectoryNotModeled is recommended.
- The simple approach is to have a Data Base Manager on each processor that holds weather files, and to nominate it as the appropriate Local Bulk Update Manager. Otherwise, consider each processor on which the file is retrieved, but which does not have a copy of Data Base Management nominated as the Local Bulk Update Manager for that file on that processor. On that processor, the VIP must recognize BigGoTo's to StartBatchRead and EndOfBatchRead, and migrate the client to a processor which does have a copy of Data Base Management so nominated; otherwise there will be interprocessor access to Shared Working Space.

5.5.3 Summary

This paragraph summarizes the topic of model mapping from the point of view of various mapping considerations. Paragraph 5.5.3.1 summarizes the straight forward scheme. Paragraphs 5.5.3.2 through 5.5.3.4 summarize the considerations that arise when the straightforward scheme is not adopted.

The VIP is allowed to migrate a Natural Flow at any point. However, it is recommended that migrations only occur at GoTo's, BigGoTo's and ReturnFromBigGoTo's; and it is assumed here (that is, in paragraph 5.5.3) that this recommendation is being followed. Moreover, it will be seen in paragraph 5.5.3.1 that mapping considerations become simpler if migration is restricted even further, to BigGoTo's and the corresponding ReturnFromBigGoTo's.

5.5.3.1 The Straight Forward Scheme

If the model mapper ensures that;

- Copies of all outer set cells are available on processors from which they are called by their clients,

- Each Prime Source Natural Flow is spawned on the processor that holds the files (if any) and the inner set cells of the Functional Area to which it belongs,

- Each Service Natural Flow is spawned on the processor that holds the Shared Working Space (if any), the files (if any), and the inner set cells of the (copy of the) Functional Area to which it belongs,

- If either the Route Conversion or the Weather Management Functional Areas are placed on more than one processor, then the files concerned are of type DirectoryNotModeled,

- A copy of the Message Management Functional Area is placed on each processor on which Specialist Natural Flows are spawned,

- A copy of the Legal&AnalysisRecording Functional Area is placed on each processor from which there are calls to its Reception Desk Cell, and is identified to the Government as the copy for use in that processor,

- Each file, that is not of the type DirectoryNotModeled, resides on the processor on which the DataBaseManager Natural Flow that is responsible for it was spawned,

- On each processor from which there are calls to read a Bulk Update file, there is a DataBaseManager nominated as the Local Bulk Update Manager for that combination of file and processor,

- Wherever a VSR is specified to plant, or to return a ProcessorID, it plants, or returns, the ID of the processor in which the Natural Flow issued the VSR (this and the next two instructions ensure that each Natural Flow operates as if based in the processor on which it was spawned),

- At a ReturnFromBigGoTo, the Natural Flow is routed to the processor quoted by the model,

- No Natural Flow is migrated except at a ReturnFromBigGoTo, or at a BigGoTo as follows;

(a) To the cell AFOSGraphicsFromAWP (no choice of destination),

(b) To any of the Flight Plan Management reception desks (no choice of destination),

(c) To the cell RouteConversion (the VIP selects the choice of cell),

(d) To the Weather Management inner set Reception Desk Cells (the VIP selects the choice of cell),

(e) To the Common Service Output Reception Desk Cells (to the processor containing the copy of Common Service Output that administers the logical port in question),

(f) To the cell WriteNewRecordKeepingWorkingSpace, during model initialization (to the processor containing the DataBaseManager responsible for the file in question), and

(g) To the cell BulkUpdate (to the processor containing the DataBaseManager responsible for the file in question).

Then the following will be true;

- There will be no interprocessor access to Shared Working Spaces or to files of type other than DirectoryNotModeled,

- The VIP need to detect and arrange only the migrations listed in (a) through (g),

- The local VIP in each processor need know only the following information about what is on other processors;

  (a) Where each Service Natural Flow was spawned (synonymous with its home processor),

  (b) Where the Ancestral Natural Flow was spawned,

  (c) Where each Prime Source Natural Flow was spawned (required only during the Model Initialization Phase, and only by the processor holding the Ancestral Natural Flow), and

  (d) Where the target cells for the migrations listed in (a) through (g) reside.

5.5.3.2 File Access

Interprocessor file access will be avoided if the DataBaseManager nominated as responsible for each file is placed on the same processor as the file, and if the DataBaseManager Natural Flows do not migrate. This does not apply to files of the type DirectoryNotModeled.

5.5.3.3 Access to Shared Working Space

The VIP will avoid interprocessor access to Shared Working Space if:

- Each Service Natural Flow awaits Prods on its home processor, with its Shared Working Space, if any, and all the inner set cells present on that processor.

- There is only migration as defined in paragraph 5.5.3.1 and:

  (a) To the 8 Data Base Management Reception Desk Cells for normal operations, with the target cell selected by fileID to cause migration to the processor containing the DataBaseManager responsible for the file to be accessed (does not apply to files of the type DirectoryNotModeled).

  (b) To the cells StartBatchRead and EndOfBatchRead, with the target cell selected by fileID to cause migration to a processor containing a DataBaseManager that is identified as the Local Bulk Update Manager for that file on that processor.

  (c) To the cell MakeLegal&AnalysisRecording, with the target cell selected to cause migration to any processor containing a Legal&AnalysisDriver provided that this Legal&AnalysisDriver was identified to the Government for use on that processor.

  (d) To the cell MessageToScreen, with the target cell selected to cause migration to the processor that contains the copy of Message Management that was identified to the Government as associated with the client in question.

5.5.3.4 Return links to processors

In various situations the model quotes a ProcessorID to the VIP. The processor quoted will be relevant under the following conditions:

- When a Temporary Natural Flow is spawned, the VIP plants a ProcessorID in its DedicatedWorkingSpace, and

  (a) If this is the processor on which its parent issued the VSR,

  (b) the parent never migrates except at a BigGoTo and a ReturnFromBigGoTo, and

  (c) at a ReturnFromBigGoTo the parent always returns to the processor from which it came,

  then the processor quoted with the offspring's Prod VSR will be that on which the parent awaits Prods from its offspring.

- When the VIP conveys a Prod from a prodder to a proddee, it passes a ProcessorID to the proddee. If this is the ID of the processor on which the prodder issued the Prod VSR, then it can be used as the destination for the reflected signal from the proddee to the prodder provided that the prodder behaves as in (b) immediately above (the "reflected signal" feature is introduced in paragraph 6.11).

- When a client Prods a Service Natural Flow, the VIP passes a ProcessorID to the latter. In some cases, the Service Natural Flow answers with a return Prod to the client. If it does, it quotes with the return Prod the ProcessorID that the VIP provided on the first of the two Prods. The original prodder will still be on the processor where it issued the Prod.

- At a BigGoTo, the VIP returns a ProcessorID to the Natural Flow. The Natural Flow will quote this same ID at the ReturnFromBigGoTo.

5.6 REFINEMENTS

5.6.1 Pruning Cells

The mapping described so far maps sets of cells; for some Functional Areas a set of inner set cells only and, for some Functional Areas, a set of outer set and a set of inner set cells.

The mapping can now be refined by looking within each set and deciding which cells of the set are really needed. This is done by a case-by-case study of each Functional Area. Section 6 should help.

An example will illustrate what can be achieved. Suppose Processor #1 contains only the Specialist and DUAT Functional Areas, plus a copy of Data Base Management. During the Measured Period, the latter is present to organize ReadRecord and WriteRecord on 6 files. If these 6 files are of the type FixedInformationDirectory, then the 2 operations between them only require 8 relatively small Data Base Management cells out of a total of 30 cells in Data Base Management.

5.6.2 Packaging Into Modules

The vendor can opt for multicell modules, each with any combination of cells packaged into them. Whether this is useful will depend on the system that is proposed.

To decide what combinations of cells would be useful to have together in the same modules, the model should be studied at the cell level.

5.7 TRIAL MAPPING

The Government translation from Modeling Language to HOL is guided by various preset values, some of which are vendor options. An example is a table that gives the DataBaseManager responsible for each file. Section 7 describes the options.

The vendor's choice of mapping is built into the HOL version of the model that the Government brings for the official benchmark demonstration. It can also be built into versions that are used for testing purposes, but the vendor may not know in advance exactly what mapping is best. For the option settings that are not obvious in advance, the vendor should proceed as follows:

- Family sizes can only be built-in. Therefore, select large enough families to cover all the cases to be tried. If in any one run, an unwanted member of a family remains unused, the only penalty is a slight increase in VIP administration overhead.

- In most cases, the vendor option can be made HOL editable. This works as follows. Select a character string to represent the value, or accept the default character string. In the HOL code this character string will appear wherever the value should have appeared. An automatic editor can change all occurrences of this character string to whatever value is desired next.

- In other cases, it can be arranged that the values are controllable by the VIP at run time. These are the Dynamic Options.

6. INDIVIDUAL FUNCTIONAL AREAS

6.1 GENERAL

The description of the model given so far is sufficient for a gross level of mapping. For some vendors this may indeed be sufficient. For a finer level of mapping, and for integration, one may need to understand the model in more detail. The best way to do this is to study the Modeling Language listing. This section is an introduction to the internal structure of each functional area.

Anyone studying the model in detail may want to know the scope of the variables encountered. All the variables named in the Modeling Language listing of the model are held in Working Spaces. Each variable belongs to one of three types, as follows.

(1) If it is declared to be local at the head of the cell, it is local to the cell and to the Natural Flow. It is held in a dynamic stack in AdminArea.

(2) If it is in a Shared Working Space, it is available to any Natural Flow that is currently accessing that Shared Working Space. All variables of this type are identified in the Modeling Language listing contained in Appendix C.

(3) In all other cases, the variable is available to the cells used by the Natural Flow, but it is local to that Natural Flow. It is only available when the

Working Space that holds it is accessible. These variables are listed in Table 13.

The constants used in the model are either single items or table items. Those tables in each cell that are indexed by run time variables are listed in the IncludeTables directive at the head of the cell.

6.2 TOP LEVEL

The Top Level Functional Area has:

- One Natural Flow (the AncestralNaturalFlow).
- Three cells (one for each phase of the Benchmark).
- No files or Shared Working Space.

In the first cell, the AncestralNaturalFlow spawns the other Permanent Natural Flows, then coordinates the initialization of the model. Initialization follows this sequence:

(1) Each Natural Flow does its private initialization, that is, initialization that does not depend on that of any other Natural Flow. In particular, each Data Base Manager initializes directories for the files under its control.

(2) The Scheduler initializes the scheduling file.

(3) Each Specialist and each DUAT Natural Flow initializes the data base records that describe the condition of its terminal.

(4) The AWPLink Input Driver Natural Flow reads initialization messages from the load tape, executes the various parameter driven initializations, then issues the first output VSR from the model. This output is the signal for VIP and benchmark driver to synchronize the start of the Measured Period.

In the second cell, the AncestralNaturalFlow waits for the Termination Phase.

In the third cell, the AncestralNaturalFlow coordinates termination in the following sequence:

(1) It waits for end of benchmark signals from all Prime Source Natural Flows.

(2) It signals the Output Driver Natural Flows to empty their spools, then waits for them to do so.

(3) Similarly then with the Legal&AnalysisDriver Natural Flows.

(4) It makes the final exit to the VIP.

6.3 SPECIALIST INTERFACE

The Specialist Interface Functional Area has:

- Ninety-one Natural Flows (the family of specialists).
- Twenty-four cells, grouped as follows;

  1 initialization cell,
  4 outer loop cells,
  17 "Do it" cells (1 per Specialist Input Type), and
  2 supporting cells.

- Three families of files, such that each family has one record per terminal;

  TerminalRecordFile, and
  CurrentFlightPlanFile
  RouteRecordFile.

- No Shared Working Space.

A simplified flow of control within the Functional Area, for each Specialist, has already been seen in Figure 2. To this add the following:

- The output is performed by three cells;

  SpecialistFormattedOutput,
  SpecialistFreeTextOutput, and
  SpecialistWeatherDisplay;

  and on any one occasion, one of them is used.

- If the vendor opts for Temporary Natural Flows for weather retrieval, then this retrieval has to be coordinated with subsequent messages arriving from the

terminal. Two outer set cells of Weather Management are available for this purpose, and are called at appropriate times;

(a) EnsureEnoughWeatherToDisplay is called by the specialist cell SpecialistWeatherDisplay.

(b) StopTheRetrieval is called by the specialist cell SpecialistInput.

The resulting flow of control within the Specialist Interface Cells is shown in Figure 16. The overall flow of control for each Input Type is shown in Figures 17 to 24. For what happens when the Specialist Natural Flows enter the cells of another Functional Area, see the description of that Functional Area.

6.4 DUAT INTERFACE

The DUAT Interface Functional Area has:

- Forty Natural Flows (the family of DUATs).
- Twenty cells, grouped as follows:

  1 initialization cell
  3 outer loop cells
  14 "Do it" cells, 1 per DUAT input type
  2 supporting cells.

- Three families of files, such that each member has one record per terminal;

  DUATTerminalRecordFile
  DUATCurrentFlightPlanFile
  DUATRouteRecordFile.

- No Shared Working Space.

The pattern of flow is similar to that for specialists. The functions performed are a copy of a subset of those for specialists. Figure 25 shows the pattern of flow within the DUAT Interface cells, and Figures 26 to 32 show the overall pattern of flow for the individual message types. For what happens when the Natural Flows enter the cells of another Functional Area, see the description of that Functional Area.

6.5 RADAR MANAGEMENT

The Radar Management Functional Area has:

- One Natural Flow; the RadarManager.
- Two cells;

  1 initialization cell
  1 run time cell.

- One file, the RadarFile.
- No Shared Working Space.

In the real application, an output from the computer solicits radar input. This is modeled by reversing the I/O. A scenario input message models the radar input, while the direct response to that input models the request for radar. Secondary responses model the distribution of radar information to the AFSSs.

Figure 33 shows the overall flow of control for radar management.

6.6 COMMON SERVICE INPUT

The Common Service Input Functional Area serves the 4 NADIN logical ports. It has:

- Two families of Natural Flows, each with 4 members, one per logical port;

  the InputDriver Natural Flows and
  the InputDistributor Natural Flows.

- An option for temporary spawning.
- Six cells, grouped as follows;

  2 initialization cells
  1 run time cell
  (used by the InputDriver's)

  1 initialization cell
  2 run time cells
  (used by the InputDistributor's).

- No files.
- No Shared Working Space.

During the Model Initialization Phase, initialization messages arrive on the AWP logical port. The InputDriver for the AWP logical port responds to these messages. Its pattern of flow is similar to that of other Prime Source Natural Flows except that it gives no response to any but the last message. This last message is of the type EndOfInitialization; and the response to it is a signal to the VIP (and hence to the benchmark driver also) to synchronize internal clocks for the Measured Period which now begins.

The following Functional Areas are called by the Input Driver for AWP, from the cell InitializeInputDriving;

Flight Plan Management
Time Management
Data Base Management.

Figures 34 to 36 shows the overall flow of control for the AWPLink Input Driver during the Model Initialization Phase.

During the Measured Period the work of reading the input and of doing what has to be done is apportioned between two Natural Flows per logical port, for the following reason. Unlike input from specialist and DUAT terminals, the input from NADIN is not specified to wait for a response from the previous input. Hence, it can arrive before the previous input has been dealt with. The arrangement that has been modeled allows the Driver to stack up a queue of inputs for the Distributor to attend to.

The following Functional Areas are called by the Input Distributors from the cell CallRecipient;

Legal & Analysis Recording
Flight Plan Management
Weather Management
Message Management
Common Service Output.

Figures 37 and 38 show the flow of control within the cells of the Functional Area, during the Measured Period, for the cases with and without temporary spawning. Figures 39 to 43 show the overall flow of control for the various Input Types that arrive during the Measured Period. For what happens when they are in another Functional Area, see the

description of that Functional Area.

6.7 TIME MANAGEMENT

The Time Management Functional Area is both Prime Source and Service. On the service side it has:

- One Natural Flow, the Scheduler.
- Six cells, grouped as follows;

  1 initialization cell
  2 run time cells
  (used by the Scheduler)

  2 outer set Reception Desk Cells

  1 outer set Reception Desk Cell used during initialization.

- One file, the SchedulingFile.
- One Shared Working Space.

The Scheduler accepts updates to the SchedulingFile, that is, additions to and deletions from the list of scheduled events, and maintains a digest of the file in the Shared Working Space. The content of the Shared Working Space is described in the Modeling Language listing.

The main work is done by the Scheduler Natural Flow, which cycles in the cell RunTimeScheduling, waiting for clients to Prod it from the Reception Desk Cells. The left hand side of Figure 44 depicts the flow of control.

The algorithm for placing and finding entries in the file is as follows. Record number n is calculated as a function of the schedule time for the event. If record n does not contain an empty slot (when scheduling) or the required entry (when cancelling), then records n+1, n+2, ... (with end-around carry at the end of the file) are searched in turn.

On the Prime Source side there are:

- One Natural Flow, the TimeManager.

- Five cells, grouped as follows, all used by the Time Manager;

  1 initialization cell
  3 run time cells
  1 cell used during the Model Termination Phase.

- No files.

- No Shared Working Space.

The TimeManager has a basic cycle activated once a second by the VSR SuspendMe. On each cycle, it inspects the Scheduler's SharedWorkingSpace. If the latter indicates that a scheduled event is due, it reads the Scheduling File and picks from it a directive for action. Like the other Prime Source Natural Flows, it makes procedure calls to other Functional Areas to perform whatever is needed. Finally, it Prods the Scheduler to delete that item from the file.

The right hand side of Figure 44 depicts the flow of control within the Functional Area. Figures 45 to 47 show the overall flow of control for the three scheduled messages during the Measured Period. To find out what happens in each of the other Functional Areas, see the description of that Functional Area.

One of the initialization messages that arrives on the AWP logical port during the Model Initialization Phase is the message ScheduleEndOfScheduling. The response is to schedule the action StopDoingScheduleActions. The scheduled action is to Prod the Ancestral Natural Flow, then stop with an AwaitProd VSR that is never satisfied.

6.8 GRAPHICS MANAGEMENT

The Graphics Management Functional Area has:

- No Natural Flows, or Shared Working Space.

- One file, the AFOSGraphicsFile.

- One inner set Reception Desk Cell.

The cell is called by the Input Distributor for the AWP. The

input models the receipt of updates to the file. The graphics cell models the broadcasting of the update to AFSSs and to each DUAT cluster.

6.9 FLIGHT PLAN MANAGEMENT

The Flight Plan Management Functional Area has:

- No Natural Flow.
- Twelve cells, as follows;

  2 outer set Reception Desk Cells used during initialization

  2 inner set supporting cells used during initialization

  1 inner set Reception Desk Cell used by the Input Distributors during the Measured Period

  3 inner set Reception Desk Cells used by Specialists and DUATs

  3 inner set Reception Desk Cells scheduled by the Scheduler for use by the Time Manager

  1 inner set run time supporting cell.
- One file, the FlightPlanFile.
- No Shared WorkingSpace.

Internally, the pattern of flow is as follows. Either the client does all the work in the Reception Desk Cell, or from that cell it calls the supporting cells. From these cells, however, the client calls other functional areas;

Common Service Output
Time Management
Legal & Analysis Recording
Data Base Management.

6.10 ROUTE CONVERSION

The Route Conversion Functional Area has:

- No Permanent Natural Flow, but three places where the vendor can opt to have the client spawn Temporary Natural Flows.

- Seven cells, all inner set, grouped as follows;

    1 Reception Desk Cell, used by specialists and DUATs

    6 Supporting cells.

- Two files;

    Airways&FixesFile
    ReportingLocationsFile.

- No Shared Working Space.

From three of the cells, the client calls Data Base Management. Figure 48 shows the difference in local flow of control generated by any of the options for temporary spawning; Figures 49-50 show the flow of control generated by election of none (Figure 49) or all three (Figure 50) of the options for temporary spawning within Route Conversion. (The six other possible permutations of the binary options 24, 25 and 29 are not illustrated.) In Figure 48, synchronous processing is diagrammed on the left, and asynchronous on the right. In the latter case, the invoker has two loops, the first executing VSR Spawn, creating Temporary Natural Flows, and the second executing VSR AwaitProd, which fields VSR Prod signaling Temporary Natural Flow completion. In Figure 49, all three loops contain procedure calls, as on the left of Figure 48; in Figure 50, all three contain asynchronous processing as on the right of Figure 48, and the Await Prod loops cascade upward after receiving all the Prods at each level of Temporary Natural Flows.

6.11 WEATHER MANAGEMENT

The Weather Management Functional Area has:

- No Permanent Natural Flows, but two places where the vendor can opt to have the client spawn Temporary Natural Flows.

- Thirteen cells, grouped as follows;

    3 outer set Reception Desk Cells used by

specialist and DUATs
2 outer set Reception Desk Cells used by the AWP InputDistributor
8 inner set supporting cells.

- The Weather Files (listed in Table 7).

- No Shared Working Space.

Parallel retrieval by Temporary Natural Flows is coordinated by using some of the features of the Prod VSR. A full description of Prod and AwaitProd is given in Part V. An overview is given here to help the model mapper understand what is going on.

The model is written so that the parent need not know the Natural Flow ID assigned to its offspring. The offspring Prods its parent with a Working Space (which is the retrieved weather report) and a message key which is meaningful to the parent as an offspring identifier. The parent dictates to the VIP, in the AwaitProd VSR, which key it currently wishes to receive.

AwaitProd is also capable of "reflecting back" a 1-bit message which means to the offspring, or to the VIP on behalf of the offspring; "now you can die".

Figures 51-53 show the flow of control generated by election of none (Figure 51), one (Figure 52) or both (Figure 53) of the options for temporary spawning within Weather Management. (Election of option 30 necessitates election of option 26, so there is no fourth permutation.) In Figure 51, there are two loops, the first by file and the second by record. In Figure 52, the first loop invokes the second by a Spawn, and a third loop, initiated by a procedure call from the client functional area, receives the Prods in order of display position. In Figure 53, the second loop issues a Spawn for each read, and a fourth loop receives the resulting Prods in display order, in turn passing Prods to the third loop.

6.12 MESSAGE MANAGEMENT

The Message Management Functional Area has:

- One family of Natural Flows, the MessageManagers, of

optional family size. By default, the size depends on options taken by the vendor in the Specialist Interface Functional Area.

- Four cells, grouped as follows;

  1 initialization cell
  1 run time cell
  (used by the Message Manager)

  1 inner set Reception Desk Cell
  1 outer set Reception Desk Cell.

- Two families of files with the same family size as for MessageManager Natural Flows;

  MessageFile (holds the messages themselves)
  TerminalMessageListFile (administration file)

- One Shared Working Space for each MessageManager Natural Flow.

The message pool plus administration is repeated with all copies of Message Management. When a new message is added to the pool, it is placed in all copies of the pool. If the vendor follows the suggestions made in paragraph 5.5, then Message Management is repeated on all processors where there are specialists. When a Specialist Natural Flow retrieves a message, the message is retrieved from the copy on its processor.

Messages are placed in the pool by the MessageManagers. The messages are retrieved by the client executing in a Reception Desk Cell. Figure 54 depicts the various patterns of flow. Data Base Management is called from two of the cells.

6.13 COMMON SERVICE OUTPUT

The Common Service Output Functional Area serves 5 logical ports (4 NADIN logical ports, plus the printer). It has:

- One family of 5 Natural Flows, the OutputDrivers, with one member per logical port.

- Five Shared Working Spaces, 1 per logical port.

- Five cells, all inner set, grouped as follows;

1 initialization cell
1 run time cell
(used by the drivers)

2 run time Reception Desk Cells

1 Reception Desk Cell for Model Termination Phase.

- One family of 5 files, the family of "OutputFile", one member per logical port.

The Shared Working Space is used for communication between drivers and clients.

Figure 55 shows the various flows of control. Two of the run time cells call Data Base Management.

6.14 LEGAL & ANALYSIS RECORDING

The Legal & Analysis Recording Functional Area has:

- One optional size family of Natural Flows, the Legal&AnalysisDrivers.
- One Shared Working Space per Natural Flow.
- Three cells, all inner set, grouped as follows;

  1 initialization cell
  1 run time cell
  (used by the drivers)

  1 Reception Desk Cell.

- No files.

The Shared Working Space holds the buffer of output messages, plus communication items. Clients add to the buffer while they are in the Reception Desk Cell. If, in doing so, they fill it, they Prod the Driver to empty the buffer. Figure 56 depicts the flow of control.

6.15 DATA BASE MANAGEMENT FUNCTIONAL AREA

The Data Base Management Functional Area administers both normal file manipulations, that is, operations on individual

records, and the complications arising from Bulk Update files. The code is broken down into a large number of (mostly) small cells, so that only those that are needed for a particular situation need be present. Data Base Management has:

- One optional size family of Natural Flows, the DataBaseManagers, intended as 1 for each processor on which files are held.

- Twenty-eight cells, grouped as follows;

  1 inner set initialization cell
  1 inner set run time cell, for the adminstration of Bulk update files
    (both used by the DataBaseManager Natural Flows)

  12 inner set Reception Desk Cells, grouped as follows;

    1 outer set cell used by the AWP Link driver during initialization
    8 inner set cells used by many clients for normal file manipulations
    3 inner set cells used by clients in Weather Management for Bulk Update files

  7 inner set cells which are GoTo-type extensions of the Reception Desk Cells

  7 inner set supporting cells reached by procedure calls.

- One Shared Working Space per Natural Flow, that holds a Master Directory for the files. It is administered by its patron Natural Flow, and it is also accessed by clients executing in Reception Desk Cells. The contents are described in the Modeling Language listing of the model.

- No files.

Normal file manipulations are handled by the client in the Reception Desk Cells and their extensions. There is one Reception Desk Cell per file operation. Each cell chooses an extension based on file type. Figure 57 shows the pattern of flow, and Table 14 lists the cells used by each combination of operation and file type. Simultaneous clients are coordinated by the VIP through their requests for sole access to the Shared Working Space.

When a client reads individual records from a Bulk Update file, it uses the normal facilities provided by Data Base Management. To find which of the pair of files to read from, it uses a Reception Desk Cell to consult the local Shared Working Space (that is, the Shared Working Space of the DataBaseManager identified as the LocalBulkUpdateManager for that file on that processor). At the end of the batch of reads, it uses a Reception Desk Cell to Prod, if appropriate, the Controlling Bulk Update Manager for that file.

Each DataBaseManager can be the Controlling Bulk Update Manager, or a LocalBulkUpdateManager, or neither, or both, for any given Bulk Update file. Each cycles endlessly waiting for Prods to direct it to perform background maintenance of those files. It expects and deals with four kinds of directive:

- "A bulk update has been completed", received from the AWPLink Distributor executing in the Reception Desk Cell that performs Bulk Updates.
- "How many readers have you currently got for this file?", received from a Controlling Bulk Update Manager.
- "I currently have n readers of this file", received from a Local BulkUpdateManager.
- "I have finished a batch of reads from a recently replaced file (hence possibly it can now be deleted)", received from a client of Weather Management.

To perform the maintenance, it keeps counts and switches in the Shared Working Space. These are detailed in the Modeling Language listing of the model. Figure 58 shows the patterns of flow that control Bulk Update files.

7. VENDOR OPTIONS

Each option has a default value.

Unless otherwise stated, all options can be requested as HOL editable, as described in paragraph 5.7. The phrase "Default

String" refers to the default character string that will be used if the option HOL editable is desired, but the choice of string is not identified.

7.1 MODEL MAPPING OPTIONS

The vendor is responsible for ensuring that the option settings are consistent with each other. Each option is identified with a default value and whether it is HOL editable.

7.1.1 Specialists

The Specialist Functional Area can be placed on n processors by dividing the Specialist Natural Flows into n groups. The 3 files will then become 3 families of files, with n members each;

- TerminalRecordFile
- RouteRecordFile
- CurrentFlightPlanFile

See the example in paragraph 5.5.1.2.

OPTION 1 Number of groups

A. Not HOL editable
B. Default value = 1

OPTION 2 The allocation of specialists to groups

A. The model mapper is encouraged to allocate the specialists in an orderly manner, for example specialists 1 to 30 in group 0, 31 to 60 in Group 1 etc.

B. HOL editable, expressed in 2 tables

(1) The table, TerminalFileIDOffset,

(a) gives the group number (numbering

starting at 0) for each specialist

(b) default value = 0

(c) default string = "$pA", where p is the logical port number

(d) must be in the range 0 to (n-1), inclusive, where n is the number of Specialist groups

(2) The table, "TerminalRecord#",

(a) gives the position (numbering starting at 1) in the group, for each specialist

(b) default value = p

(c) default string = "$pB" (where p is the logical port number)

(d) must be positive, with no 2 Specialists in the same group having the same value.

(e) the maximum value for each Specialist Group determines the file lengths for the group

7.1.2 DUATs

The DUAT Functional Area can be placed on n processors by dividing the DUATs into n groups. The 3 files will then become 3 families of files, with n members each.

- DUATTerminalRecordFile
- DUATRouteRecordFile
- DUATCurrentFlightPlanFile

OPTION 3 Number of groups

A. Not HOL editable

B. Default value = 1

C. Must be in the range 0 to (n-1), inclusive, where n is the number of DUAT groups

OPTION 4 The allocation of DUATs to groups

A. The model mapper is encouraged to allocate the DUATs in an orderly manner, for example DUATs 1 to 10 in group 0, 11 to 20 in group 1, etc.

B. HOL editable, expressed in the same two tables as for specialists:

(1) The table, TerminalFileIDOffset,

(a) gives the group number (numbering starting at 0) for each DUAT
(b) default value = 0
(c) default string = "$pC", (where p is the logical port number)

(2) The table, TerminalRecord#,

(a) gives the position (numbering starting at 1) in the group, for each DUAT
(b) default value = p
(c) default string = "$pD"
(where p is the logical port number)
(d) must be positive, with no two DUATs in the same groups having the same value
(e) the maximum value for each DUAT group determines the file lengths for that group

7.1.3 Message Management

OPTION 5 Family size for Message Management

A. The same family size applies to

MessageManager Natural Flows
MessageFile
TerminalMessageListFile

B. Not HOL editable

C. Default value = the number of groups chosen for the Specialist Functional Area (paragraph 7.1.1)

OPTION 6 Association of each specialist with a copy of Message Management

A. This is reflected in the table SelectedMessageManager, which gives, for each specialist, a membership number (numbering starting at 1) in the Message Management families

B. Default value = 1 + the value chosen for the table TerminalFileIDOffset (paragraph 7.1.1)

C. Default string = "$pE", where p is the logical port

number for the specialist

D. This option governs

(1) which MessageManager Natural Flow alerts each specialist

(2) which Shared Working Space the specialist Natural Flow will attempt to access

(3) which member of each of the two Message Management file families the specialist will refer to

E. If the defaults are not accepted for this option, the VIP will have to handle intelligently the specialist migrations to Message Management (see paragraph 5.5.2.5)

F. Must be in the range 1 to n, inclusive, where n is the family size for Message Management

7.1.4 Legal & Analysis Recording

OPTION 7 Family size for Legal&AnalysisRecording

A. Not HOL editable

B. Default value = 1

Each client of Legal&Analysis Recording adds to a recording buffer, which is the Shared Working Space of a member of the family of Legal&AnalysisDriver's. If it finds that it has filled the buffer, it Prods this same driver to empty it. The choice of driver is governed by the following table, which is indexed by the ID (from 1 to 16) of the processor on which the client fills the buffer. The output of the table is a Natural Flow number. Numbering for the Legal&AnalysisDriver family starts at two.

OPTION 8 The table, RecommendedLegal&AnalysisDriver

A. Default value = 2 (the Natural Flow number of the first member of the family)

B. Default string = "$pF", where p is the processor number

C. The VIP is specified to select the driver

dynamically if the table value is the negated Natural Flow number of the first member of the family (that is, -2)

See the summary at the end of paragraph 5.5.2.7.

7.1.5 Data Base Management

OPTION 9 Family size for Data Base Management

A. Not HOL editable

B. Default value = 1

Each file, unless it is of type DirectoryNotModeled, is managed exclusively, for its normal operations, by one member of the family of Data Base Managers.

OPTION 10 The allocation of files to DataBaseManagers

A. This is governed by the table, which is indexed by File ID, NaturalFlow#OfDataBaseManager

B. Must be zero for files of type DirectoryNotModeled

C. Default value = Natural Flow number of the first member of the family

D. Default string = "$fG", where f is the File ID

E. The numbering for DataBaseManager Natural Flows starts at 2 plus the number of Legal&AnalysisDriver's

F. Must be zero (for files where this is allowed) or in the range (2 + L) to (1 + L + D) inclusive, where

L is the number of Legal&AnalysisDrivers
D is the number of DataBaseManagers

G. This table governs which Shared Working Space the client will attempt to access.

To allow for possible distribution of each Bulk Update file, a Data Base Manager should be identified for each processor on which the file is placed, and one of them must be

identified as the one in control for that file. Values have to be supplied for the following two tables, both of which yield the Natural Flow number of a DataBaseManager. They require values, whatever the file type selected for the files concerned.

OPTION 11 The table, LocalBulkUpdateManager

A. Gives the recommended nearest LocalBulkUpdateManager for each combination of processor number (1 to 16) and pair of Bulk Update files (1 to 9: the fileID of the first of the pair, see Table 7)

B. Default value = Natural Flow number of first member of DataBaseManager family of Natural Flows

C. Default string = "pbH", where

p = processor number (1 to 16)
b = Bulk Update file pair (1 to 9)

D. For each pair of BulkUpdateFiles, considered separately:

(1) For each relevant processor, must yield the ID of a DataBaseManager Natural Flow.
(2) For each processor that is not relevant, must be either zero, or a repeat of an entry for a relevant processor.

Where a processor is "relevant" if either or both of the cells StartBatchRead and/or EndOfBatchRead are executed on that processor for that pair of BulkUpdateFiles.

OPTION 12 The table, ControllingBulkUpdateManager

A. Gives the controlling DataBaseManager for each pair of Bulk Update files

B. Default value = Natural Flow number of first member of DataBaseManager family of Natural Flows

C. Default string = "$fJ", where f is the file ID

D. Must yield a valid ID of a DataBaseManager

See the summary at the end of paragraph 5.5.2.8.

7.2 MISCELLANEOUS OPTIONS

OPTION 13 The constant, Legal&AnalysisBufferSize

A. Sets the buffer size in characters for Legal & Analysis Recording. If this differs from the record size then the VIP or the Operating System will have to make the necessary arrangements

B. Default value = 300 (characters)

C. Not HOL editable

D. Must be at least 300 characters

NOTES ABOUT OPTIONS 14-16

Table 1 of Part V lists Physical Port Numbers which are primarily for documentation purposes. Distinguish these from the corresponding Physical Port IDs, which are vendor-selected, and which the model quotes to the VIP. The Physical Port ID can be any integer whose machine-representation yields a bit pattern which is meaningful in the vendor's system.

AFSS and DUAT Cluster Physical Ports are treated separately because they are used for high-volume low-priority output (namely, AFOS graphics and radar information). It will be seen from Table 1 of Part V that some of them are constrained to be identical with Physical Ports to Specialists and DUATs; but they may be given distinguishing IDs to assist the VIP's scheduling of output.

OPTION 14 The table, PhysicalPortID

A. Gives the ID of the physical port, if any, associated with each Natural Flow.

B. Default value = Physical Port Numbers listed in Table 1 Part V.

C. Default string = "$pK", where p is the default value

OPTION 15 The table, AFSSPhysicalPortID

A. Gives Physical Port ID for each of the 11 AFSSs.

B. Default value = PhysicalPort Number list in Table 1, Part V.

C. Default string = "$aL", where a is the AFSS Number (1 to 11)

D. Must have values that uniquely identify the appropriate physical port for each AFSS

OPTION 16 The table, DUATClusterPhysicalPortID

A. Gives Physical Port ID for each of the five DUAT clusters.

B. Default value = PhysicalPort Number listed in Table 1, Part V.

C. Default string = "$dM", where d is the DUATCluster Number (1 to 5)

D. Must have values that uniquely identify the appropriate PhysicalPort for each DUATCluster

OPTION 17 The table, FileType

A. Gives the file type selected for each file

B. File types have the following numerical representations

(1) DirectoryNotModeled = 1
(2) FixedInformationDirectory = 2
(3) FixedAddressRecords = 3
(4) VariableAddressRecords = 4

C. Default values as follows

(1) FlightPlanFile: FixedAddressRecords
(2) MessageFile: FixedAddressRecords
(3) TerminalMessageListFile: VariableAddressRecords
(4) SchedulingFile: VariableAddressRecords
(5) all others: FixedInformationDirectory

D. Default string = "$fN", where f = fileID

E. See Table 10 for the restrictions on the choice or file type

OPTION 18 The constant, #RecordsInSchedulingFile

A. Default value = 75 times SizeOfWord

B. Not HOL editable

OPTION 19 The constant, SchedulingRecordLength

A. In characters

B. Default value = 80

C. Not HOL editable

D. The total capacity of the SchedulingFile must be at least 3000 words. Exceeding this by a factor of two or more is recommended (see paragraph 6.7). Marginally better performance might be obtained with fewer and longer records than in the default

OPTION 20 The constant, SizeOfWord

A. This gives the number of characters in the word size that is generated by the vendor's HOL compiler, for computational integers. See paragraph 4.1 of Part V.

B. Default value = 2

C. Not HOL editable

OPTION 21 Module packaging (instructions will be provided at a workshop). One specific constraint is that a cell containing a BigGoTo cannot be packaged into a module with any cell which calls it.

OPTION 22 The constant, RadarSliceLength

A. This gives the number of characters of the Radar input data that are processed at one time. Each Radar input is 29700 characters long. It is processed in slices. This option controls the number of characters per slice. Bigger slices mean fewer slices, hence fewer backing store transfers, but the allocation of mainstore and of address space may be harder.

B. DefaultValue = 512

C. Not HOL editable

OPTION 23 The content, length of time slice in milliseconds

A. This controls the amount of CPU time the model will simulate before returning to the VIP. If the time slice expires before the required amount of CPU resource has been simulated, the model returns to the VIP with a TimeSliceOver VSR; when the Natural Flow resumes, it continues with the interrupted simulation

B. Default value: 10000 (effectively switching time slicing off)

C. Not HOL editable

<u>OPTION 24</u> The switch, RadarProcessingByHardware

A. This controls the amount of CPU usage that is simulated when radar data is being processed. If it is requested to be set, the appropriate hardware shall be included in the vendor's system.

B. Default value: no (that is, by software)

C. Not HOL editable

7.3 DYNAMIC OPTIONS

With dynamic options, the model mapper can request a version of the model which tests the option dynamically, by inspecting the debugging area in the Dedicated Working Space. Numbering of words in Dedicated Working Space starts at one. "Dynamic position", as used in the dynamic options, refers to the word in Dedicated Working Space in which the VIP should place the required value, if the dynamic option is selected.

Some of the options are Yes/No switches. At the HOL level, Yes = 1 and No = 0.

OPTION 25 The switch, ConvertingRouteElementsInParallel

A. This governs whether the consecutive elements of a route of flight are to be processed in series, or in parallel by Temporary Natural Flows

B. Default value: No (that is, not in parallel)

C. Dynamic position: word 17

OPTION 26 The switch, ConvertingLegsInParallel

A. This governs whether, within the processing of an individual route element of a flight, the consecutive legs shall be processed in series, or in parallel by Temporary Natural Flows

B. Default value: no (that is, not in parallel)

C. Dynamic position: word 18

OPTION 27 The switch, ParallelWeatherRetrieval

A. This governs whether, during a weather retrieval, the separate types of weather are retrieved in series, or in parallel by Temporary Natural Flows

B. Default value: no (that is, not in parallel)

C. Dynamic position: word 19

OPTION 28 The switch, ParallelDistributionOfNADINInput

A. For each NADIN logical port, separately, there is a member of the family of InputDistributor Natural Flows. This switch governs whether each InputDistributor performs, in series, the action required by each input message on that logical port, or whether the InputDistributor spawns Temporary Natural Flows to perform the actions in parallel

B. Default value: no (that is, not in parallel)

C. Dynamic position: word 20

OPTION 29 The switch, RequestingRouteRecordsSimultaneously

A. This governs whether, within the processing of an

individual leg of a route element of a flight, the scenario directed number of records to be read from permanent memory are to be requested and processed in series (not requesting a record until the previous record has been read and processed) by one Natural Flow, or in parallel (issuing all read requests without requiring that any such request be delayed pending the satisfaction of any other request) by additional Temporary Natural Flows.

B. Default value: no (that is, not in parallel)

C. Dynamic position: word 23

<u>OPTION 30</u> The switch, RequestingWeatherRecordsSimultaneously

A. This governs whether, within the processing of a weather retrieval from an individual file, the scenario directed number of records to be read from permanent memory are to be requested and processed in series (not requesting a record until the previous record has been read and processed) by one Natural Flow, or in parallel (issuing all read requests without requiring that any such request be delayed pending the satisfaction of any other request) by additional Temporary Natural Flows. If this option is chosen, then ParallelWeatherRetrieval (OPTION 26) must also be chosen.

B. Default value: no (that is, not in parallel)

C. Dynamic position: word 24

TABLE 1

CELLS BY FUNCTIONAL AREA

FUNCTIONAL AREA	NUMBER OF CELLS*		
	PHASE A	PHASE B	PHASE C
1. Top Level	1	1	1
2. Specialist Interface	1	22	1
3. DUAT Interface	1	18	1
4. Radar Management	1	1	-
5. Common-Service Input	2	3	1
6. Time Management	3	7	1
7. Graphics	-	1	-
8. Flight Plan Management	4	8	-
9. Route Conversion	-	5	-
10. Weather Management	-	10	-
11. Message Management	1	3	-
12. Common-Service Output	1	3	1
13. Legal & Analysis Management	1	2	-
14. Data Base Management	2	26	-

---

*Phase A = Model Initialization Plan
Phase B = Measured Period
Phase C = Model Termination Phase

TABLE 2

PERMANENT NATURAL FLOWS

FUNCTIONAL AREA	NAME OF NATURAL FLOW	SINGLE OR FAMILY	FAMILY SIZE	PRIME-SOURCE OR SERVICE
1. Top Level	AncestralNaturalFlow	S	1	@
2. Specialist Interface	Specialist	F	91	P
3. DUAT Interface	DUAT	F	40	P
4. Radar Management	RadarManagement	S	1	P
5. Common-Service Input	InputDriver InputDistributor	F F	4 4	P
6. Time Management	TimeManager	S	1	P
	Scheduler	S	1	S
7. Graphics Management	--	--	--	-
8. Flight Plan Management	--	--	--	-
9. Route Conversion	--	--	--	-
10. Weather Management	--	--	--	-
11. Message Management	MessageManager	F	Vendor Option*	S
12. Common-Service Output	OutputDriver	F	5	S
13. Legal&Analysis Management	Legal&AnalysisDriver	F	Vendor Option*	S
14. Data Base Management	DataBaseManager	F	Vendor Option*	S

---

*Minimum of 1 in each case

@special case on its own

TABLE 3

INPUT TYPES & SCHEDULED ACTIONS

FUNCTIONAL AREA	INPUT TYPE OR SCHEDULED ACTION	NUMERICAL ID
Specialist Interface (Input)	SimpleInput	1
	PageForward	2
	PageBackward	3
	DirectionFinding	4
	RequestSendFormat	5
	FlightLog	6
	ExecuteLinkSend	7
	DisplayMessage	8
	PrintScreen	9
	StartConversationWithNewFlightPlan	10
	StartConversationWithFiledFlightPlan	11
	RescueConversationWithCurrentFlightPlan	12
	RouteDataForFlightPlan	13
	NonRouteDataForFlightPlan	14
	FileFlightPlanFromConversation	15
	RequestWeather	16
	StopBenchmark	17
DUAT Interface (Input)	DUATSimpleInput	1
	DUATPageForward	2
	DUATPageBackward	3
	RequestDUATSendFormat	4
	DUATFlightLog	5
	ExecuteDUATLinkSend	6
	StartDUATConversationWithNewFlightPlan	7
	StartDUATConversationWithFiledFlightPlan	8
	ResumeDUATConversationWithCurrentFlightPlan	9
	RouteDataForDUATFlightPlan	10
	NonRouteDataForDUATFlightPlan	11
	FileFlightPlanFromDUATConversation	12
	RequestDUATWeather	13
	DUATStopBenchmark	14
Common Service Input (Input)	InitializeRescueAlerts*	1
	FlightPlanFromAnotherComputer	2
	IndividualUpdateFromAWP	3
	HourlyUpdateFromAWP	4
	AFOSGraphicsFromAWP	5
	AcknowledgementFromAnotherComputer	6
	MailMessage	7
	InitializeFlightPlanDeletions*	8
	ScheduleEndOfScheduling*	9
	StandardInitializationOfFiles*	10
	EndOfInitialization*	11
	EndOfBenchmark	12
Radar Management (Input)	UpdateRadarFiles	1
	RadarStopBenchmark	2
Time Management (Scheduled)	StartRescueOperations	1
	DeleteFlightPlan	2
	TransmitFlightPlan	3
	StopDoingScheduledActions	4

* Initialization messages

TABLE 4

PRIVATE WORKING SPACES

NAME	LENGTH	COMMENT
DedicatedWorkingSpace	14 words*	Held for life of Natural Flow
AdminArea	60 words*	Held for all but dormant periods in life of Natural Flow (e.g., released by Specialist Natural Flow when awaiting input from terminal)
SmallInputArea	Up to 200 characters	Short lived
WorkArea 1	Up to 500 characters	Held for duration of any non-trivial activity
WorkArea 2	Up to 5000 characters	Short periods as required
RecordArea	Up to maximum record length (4096 characters)	For holding records from, or being prepared for, the data base (for as long as record is being used). Not used for Radar File
DirectoryArea	Directory size is n words*, where n is the number of records in the file, excluding directory; up to a maximum directory size of 1374 or #SchedulingRecords words, whichever is greatest (but the VIP specifications allow for accessing only a part of directory during the measured period)	For holding modeled directory of a file (usually very short period)
BigIOArea	29700 characters for Radar Managers; otherwise maximum of 8000 characters	This is the only Work Space that can increase in size after it has been acquired

---

*A word is the size of an integer as compiled by the vendor's HOL compiler, with a minimum size capable of holding numbers in the range -32767 to +32767.

TABLE 5

SHARED WORKING SPACES

FUNCTIONAL AREA	"PATRON" NATURAL FLOW(S)	LENGTH	USED ALSO BY
Message Management	MessageManager	128 Words	%
Common Service Output	OutputDrivers	8* SizeOfWord	%
Time Management	Scheduler	SizeOfWord*(3+2* #Scheduling records (a vendor option))	Time Manager
Legal & Analysis Management	Legal&AnalysisDriver	2* SizeOfWord+ Legal&AnalysisBufferSize	%
Data Base Management	DataBaseManagers	SizeOfWord*(2*#OfFiles≠ +4*#OfBulkUpdateFilePairs)	%

%Used also by clients while executing in the cells of that Functional Area.
*Multiply
≠Function of vendor options, default = 45.

TABLE 6

FILES

Functional Area	File	Family/ Single	Family Size	Numerical ID	Number of Records	Record Length
1. Top Level	--	-	--			
2. Specialist Interface	TerminalRecordFile	F	Vendor option	44	91*	128
	CurrentFlightPlanFile	F	(Discussed in	45	91*	768
	RouteRecordFile	F	4.6.1)	46	91*	600
3. DUAT Interface	DUATTerminalRecordFile	F	Vendor option	47	40	128
	DUATTCurrentFlightPlanFile	F	(discussed in	48	40	768
	DUATRouteRecordFile	F	4.6.1)	49	40	600
4. Radar Management	RadarFile	S		36	26	29700
5. Graphics	AFOSGraphicsFile	S		35	93	4096
6. Flight Plan Management	FlightPlanFile	S		37	1000	768
7. Route Conversion	AirwaysSFixesFile	S		33	832	512
	ReportingLocationsFile	S		34	268	360
8. Weather Management	weather files (listed in Table 7)	all S				
9. Message Management	MessageFile	F	*	50	64	80
	TerminalMessageListFile	F		51	91	64
10. Common Service Input	--	-				
11. Common Service Output	OutputFiles(5)	F	5	39 to 43	100	256
12. Time Management	SchedulingFile	S		38	150*	80*
13. Legal & Analysis Management	--	-				
14. Data Base Management	--	-				

---

*These values apply if default settings are used.
*Same as that chosen by the vendor for the associated family of Natural Flows.

TABLE 7

WEATHER FILES

File Name	ID#	Number of Records**	Record Length***
HourlyWeatherObservationsFile 1*	1	1374	256
HourlyWeatherObservationsFile 2*	2	1374	256
HourlyWeatherObservationsFile 3*	3	1374	256
FGFile*	4	190	512
TWEBRouteForecastFileExpanded*	5	400	512
TWEBRouteForecastFileUnexpanded*	6	400	256
FTFileExpanded*	7	690	256
FTFileUnexpanded*	8	690	128
SDFile*	9	136	200
TWEBRouteSynopsisFileExpanded*	10	23	1480
TWEBRouteSynopsisFileUnexpanded*	11	23	512
NOTAMFileExpanded	12	1300	160
NOTAMFileUnexpanded	13	1300	80
WHFileExpanded	14	6	1200
WHFileUnexpanded	15	6	600
FAFileExpanded	16	11	3600
FAFileUnexpanded	17	11	1800
MilitaryOpsFile	18	650	128
PIREPFileExpanded	19	1300	300
PIREPFileUnexpanded	20	1300	150
ATCCCFile	21	5	450
HourlyWeatherObservationsAlternate 1*	22	1374	256
HourlyWeatherObservationsAlternate 2*	23	1374	256
HourlyWeatherObservationsAlternate 3*	24	1374	256
FGAlternateFile*	25	190	512
TWEBRouteForecastAlternateExpanded*	26	400	512
TWEBRouteForecastAlternateUnexpanded*	27	400	256
FTAlternateFileExpanded*	28	690	256
FTAlternateFileUnexpanded*	29	690	128
SDAlternateFile*	30	136	200
TWEBRouteSynopsisAlternateExpanded*	31	23	1480
TWEBRouteSynopsisAlternateUnexpanded*	32	23	512

#Default value; applies if optional family sizes are set to 1

*Bulk update files

**Excludes the directory

***These lengths do not apply to the directory. For directory length, see private working space #7 in Table 4.

TABLE 8

POTENTIAL PERMANENT MEMORY OPERATIONS

Operation \ File Type	Fixed Information Directory	Fixed Address Records	Variable Address Records
Write New Record	Read D Write R Write D	Read D Write R Write D	Acquire R Write R Read D Write D
Write Record	Read D Write R	Read D Write R	Acquire R Write R Read D Write D Release R
Write Record & Unlock It	---	Read D Write R Write D	
Read Record	Read D Read R	Read D Read R	Read D Read R
Read Record For Updating	--	Read D Write D Read R	Read D Write D Read R
Delete Record	--	Read D Write D Release R (Unless "Pretend Delete")	Read D Write D Release R (Unless "Pretend Delete")

Legend

D = Directory
R = Record

TABLE 9

DATA BASE OPERATIONS ON EACH FILE
(MEASURED PERIOD ONLY)

FILES	OPERATIONS
Weather Files	WR, RR and Bulk Update
TerminalFiles RouteRecordFile CurrentFlightPlanFile TerminalRecordFile DUATRouteRecordFile CurrentDUATFlightPlanFile DUATTerminalRecordFile	RR, WR
RadarFile	WR
SchedulingFile	RR, WR
AFOSGraphics	WR
FlightPlanFile	RRU, WRU, DR, RR, WNR
Airways&Fixes	RR
ReportingLocationsFile	RR
MessageFile	RR, DR, WR
MessageTerminalListFile	RR, RRU, WRU
OutputFile	WR, RR

Key: RR = Read Record
WR = Write Record
RRU = Read Record For Updating
WRU = Write Record & Unlock It
DR = Delete Record
WNR = Write New Record

TABLE 10

RESTRICTION ON CHOICE OF FILE TYPE

FILE NAME*	RESTRICTION
FlightPlanFile MessageFile(s)	Must Be FixedAddressRecords or VariableAddressRecords
TerminalMessageListFile(s) SchedulingFile(s)	Must be VariableAddressRecords

* Restrictions apply to these file types only.

TABLE 11

INNER-SET-CELLS BY FUNCTIONAL AREA

CELL ID	CELL NAME	CODE SIZE (in code units)
	TOP LEVEL FUNCTIONAL AREA	
1	InitializeBenchmark	-
70	BenchmarkRunTime	-
71	WrapUpBenchmark	-
	SPECIAL 157 FUNCTIONAL AREA	
72	InitializeSpecialist	-
73	SpecialistInput	60
74	SpecialistFormattedOutput	40
75	SpecialistFreeTextOutput	40
76	SpecialistWeatherDisplay	40
39	SimpleInput	6
41	PageBackward	6
40	PageForward	6
42	DirectionFinding	6
43	RequestSendFormat	6
44	FlightLog	250
45	ExecuteLinkSend	11
46	DisplayMessage	10
47	PrintScreen	12
48	StartConversationWithNewFlightPlan	50
49	StartConversationWithFiledFlightPlan	15
50	ResumeConversationWithCurrentFlightPlan	10
51	RouteDataForFlightPlan	10
52	NonRouteDataForFlightPlan	70
53	FileFlightPlanFromConversation	20
54	RequestWeather	10
55	StopBenchmark	-
77	OrganizeFlightPlanFormat	20
78	UpdateTerminalRecord	13
	Set Sub Total	713
	DUAT FUNCTIONAL AREA	
79	InitializeDUAT	-
80	DUATInput	60
81	DUATNormalOutput	40
82	DUATWeatherDisplay	40
56	DUATSimpleInput	6
68	DUATPageBackward	5
57	DUATPageForward	6
59	RequestDUATSendFormat	6
60	DUATFlightLog	250
61	ExecuteDUATLinkSend	11
62	StartDUATConversationWithNewFlightPlan	50
63	StartDUATConversationWithFiledFlightPlan	15
64	ResumeDUATConversationWithCurrentFlightPlan	10

TABLE 11 (CONTD)

65	RouteDataForDUATFlightPlan	10
66	NonRouteDataForDUATFlightPlan	70
67	FileFlightPlanFromDUATConversation	20
68	RequestDUATWeather	10
69	DUATStopBenchmark	-
83	OrganizeDUATFlightPlanFormat	20
84	UpdateDUATTerminalRecord	15
	Set Sub Total	645
	RADAR MANAGEMENT FUNCTIONAL AREA	
85	InitializeRadarManagement	-
86	RunTimeRadarManagement	100
	Set Sub Total	100
	COMMON SERVICE INPUT FUNCTIONAL AREA	
99	InitializeInputDriving	-
38	EndOfInitialization	-
100	RunTimeInputDriving	20
101	InitializeInputDistribution	-
102	RunTimeInputDistribution	12
103	CallRecipient	20
	Set Sub Total	62
	TIME MANAGEMENT FUNCTIONAL AREA	
106	InitializeTimeManagement	-
107	RunTimeTimeManagement	12
108	InitiateScheduledEvent	45
114	OneSecondRoutine	15
27	StopDoingScheduledActions	-
109	InitializeScheduling	-
110	RunTimeScheduling	50
111	UpdateSharedWorkingSpace	-
	Set Sub Total	147
	GRAPHICS FUNCTIONAL AREA	
32	APOSGraphicsFromAWP	120
	FLIGHT PLAN MANAGEMENT FUNCTIONAL AREA	
5	InitializeRescueAlerts	-
6	InitializeFlightPlanDeletion	-
26	TransmitFlightPlan	110
24	StartRescueOperations	15
25	DeleteFlightPlan	10
3	GetFlightPlanForUpdating	7
4	FileThisFlightPlan	19
29	FlightPlanFromAnotherComputer	10
87	DealWithFlightPlan	120
5	InitializeRescueAlertsPtII	-
6	InitializeFlightPlanDeletionPtII	-
2	GetFlightPlanReadOnly	7
	Set Sub Total	299

TABLE 11 (CONTD)

	ROUTE CONVERSION FUNCTIONAL AREA	
7	ConvertRoute	245
88	RouteElementsInParallel	30
89	ConvertElement	320
90	LegsInParallel	30
91	ConvertLeg	290
135	RequestRouteRecordSimultaneously	30
136	RequestRouteRecordSerially	30
	Set Sub Total	975
	WEATHER MANAGEMENT FUNCTIONAL AREA	
95	GetIndividualWeatherInParallel	25
96	GetIndividualWeather	230
133	WindRetrieval	205
134	WindComputations	275
8	GetIndividualWeatherInSeries	25
137	DigestWeatherRecord	20
138	RequestWeatherRecordSimultaneously	20
139	RequestWeatherRecordSerially	20
	Set Sub Total	820
	MESSAGE POOL MANAGEMENT FUNCTIONAL AREA	
9	MessageToScreen	20
97	InitializeMessageManagement	-
98	RunTimeMessageManagement	215
	Set Sub Total	235
	COMMON-SERVICE OUTPUT	
10	OrganizeOutput	35
33	AcknowledgementFromAnotherComputer	15
11	DrainSpool	-
104	InitializeOutputDriving	-
105	RunTimeOutputDriving	45
	Set Sub Total	95
	LEGAL & ANALYSIS ACCOUNTING	
12	MakeLegal&AnalysisRecording	30
115	InitializeLegal&AnalysisDriving	-
116	Legal&AnalysisDriving	25
	Set Sub Total	55

TABLE 11 (CONC)

DATA BASE MANAGEMENT

14	WriteNewRecordKeepingWorkingSpace	15
13	WriteNewRecord	15
16	WriteRecordKeepingWorkingSpace	15
15	WriteRecord	15
17	WriteRecord&UnlockIt	15
18	ReadRecord	15
19	ReadRecordForUpdating	15
20	DeleteRecord	20
21	BulkUpdate	100
22	StartBatchRead	20
23	EndOfBatchRead	15
118	WriteRecordDirect	10
119	WriteRecordSimpleCase	10
120	WriteRecord&Directory	20
121	WriteVariableAddressRecord	12
122	ReadRecordDirect	10
123	ReadRecordSimpleCase	10
124	ReadRecordHoldingDirectory	15
117	InitializeDataBaseManagement	-
125	DataBaseRunTime	25
132	TestIfReadyToDelete	20
126	ReverseBulkUpdateSwitch	8
127	GetSpaceIDForRecord	15
128	GetLockedDirectoryUpdateEntry&GetSpaceID	15
129	WriteDirectory&UnlockIt	20
130	Finding&UpdateRecordEntry	10
131	CollectFileStatistics	35
	Set Sub Total	495

TABLE 12

OUTER SET CELLS BY FUNCTIONAL AREA

CELL ID	CELL NAME	CODE SIZE (in Code Units)
	TIME MANAGEMENT FUNCTIONAL AREA (Service Half)	
112	Schedule	8
113	Unschedule	7
36	ScheduleEndOfScheduling	-
	Set Sub Total	15
	WEATHER MANAGEMENT FUNCTIONAL AREA	
92	StartWeatherRetrieval	45
93	EnsureEnoughWeatherToDisplay	20
94	StopTheRetrieval	25
30	IndividualUpdateFromAWP	120
31	HourlyUpdateFromAWP	25
	Set Sub Total	235
	MESSAGE-POOL MANAGEMENT FUNCTIONAL AREA	
34	MailMessage	10
	DATA-BASE MANAGEMENT FUNCTIONAL AREA	
37	StandardInitializationOfFiles	-

Note: Code size shown only for cells used during the Measured Period.

TABLE 13

Non Local Variables Used in the Model

Used by Specialist & DUAT Natural Flows

ReplyLength (exists for short periods only)
WeatherRecordCountDown
NextPageStart
LocalFileIDOffset
LocalFileRecord#
PreviousFlightPlan#
RecordedFlightPlan#
ScenarioDirectedRecipient
ScenarioDirectedMessageLength
UpperHalfOfParameter
ScenarioDirectedPrintLength
ScenarioDirectedMessageRecord#
ScenarioDirectedMessageListRecord#
ScenarioDirectedTimeToDelete?
ScenarioDirected#LegsToLog
ScenarioDirectedWeatherRecordStarter
ScenarioDirectedWeatherRecordIncrement
ScenarioDirected#Records (a run-time array of size 10)
UpperHalfOfParameter
ScenarioDirectedFlightPlan#
FlightPlanToBeUpdated
UpperHalfOfParameter
ScenarioDirectedFlightPlan#
ScenarioDirectedDataBaseStatusOfFlightPlan
ScenarioDirectedFPScheduledAction
ScenarioDirectedFlightPlanScheduleTime
ScenarioDirectedTransmitNow?
ScenarioDirectedFlightPlanDestination
ScenarioDirected#RouteElements
ScenarioDirectedAirways&FixesRecord
ScenarioDirectedAirways&FixesRecord
ScenarioDirectedAirways&FixesOffset
ScenarioDirected#LegsPerElement
ScenarioDirectedFirstLocationsRecord
ScenarioDirectedLocationsOffset
ScenarioDirected#RecordsPerLeg
ScenarioDirected#LocationsPerRecord
ScenarioDirectedLogicalPort#
ScenarioDirectedSequence#
ScenarioDirectedInputType
ScenarioDirectedMessageIdentifier
ScenarioDirectedReplyLength
ScenarioDirectedInputLength

Used by clients of FLIGHT PLAN MANAGEMENT

OriginalPort#
RecordedDistination
RecordedScheduledAction
RecordedScheduleTime

TABLE 13 (cont'd)

Used by MESSAGE MANAGERS

ThisManager
MessageFileID
TerminalMessageListFileID

Used by INPUT DRIVERS

Record#
InitializationOver
#InputsSoFar

Used by INPUT DISTRIBUTORS

Total#OfInputsReceived
BenchmarkOver
#UnfinishedInputs
InputCoordinationCode
OriginatingPort#
CorrespondingOutputRoute
ScenarioDirectedMessageRecord#
UnusedScenarioParameter
ScenarioDirectedMessageListStarter
ScenarioDirectedMessageListIncrement
ScenarioDirectedFirstMessageReceiver
ScenarioDirectedMessageReceiverIncrement
ScenarioDirectedWeatherFileID
ScenarioDirectedWeatherUpdateAction
ScenarioDirectedWeatherRecord#
ScenarioDirectedAFOSRecord#
HOLArray ScenarioDirectedInitialTime size 10
ScenarioDirectedFileID
Verification
SecondUpperHalf
ScenarioDirected#BulkUpdateRecords
ThirdUpperHalf
ScenarioDirectedStartRecord#
UpperHalfOfParameter
ScenarioDirectedTimeToStop
ScenarioDirectedLogicalPort#
ScenarioDirectedSequence#
ScenarioDirectedInputType
ScenarioDirectedMessageIdentifier
ScenarioDirectedReplyLength
ScenarioDirectedInputLength

Used by OUTPUT DRIVERS

File#ForThisSpool

TABLE 14
DATA BASE MANAGEMENT CELLS USED; OPERATIONS VS. FILETYPE

FILE TYPE	OPERATIONS							
	WNRKWS	WNR	WRKWS	WR	WR&U	RR	RR&U	DR
VA VariableAddressRecords	WVAR GL___ F6UE WD&U	WVAR GL___ F6UE WD&U	WVAR GL___ F6UE WD&U	WVAR GL___ F6UE WD&U	WVAR GL___ F6UE WD&U	RRND GL___ F6UE	GL___ WD&U F6UE	GL___ WD&U F6UE
FA FixedAddressRecords	WR&D GLDUEG WD&U F6UE	WR&D GL___ WD&U F6UE	WRSC GSIDfR F6U	WRSC GSIDfR F6UE	WR&D GL___ WD&U F6UE	RRND GL___ F6UE	GL___ WD&U F6UE	GL___ WD&U F6UE
FI FixedInformationDirectory	WR&D GL___ WD&U F6UE	WR&D GL___ WD&U F6UE	WRSC GSIDfR F6UE	WRSC GSIDfR F6UE	X	RRSC GSIDfR	X	GL___ WD&U F6UE
DBM DirectoryNotModeled	WRD.	WRD	WRD	WRD	X	RDD	X	X

Key:
Top Row (Operation)
W[N]R[RWS]= Write[New]Record
[KeepingWorkingSpace]
WR&U=WriteRecord&UnlockIt
RR=ReadRecord
RRfU=ReadRecordForUpdating
DR=DeleteRecord

Inside
WVAR=WriteVariableAddressRecord
GL___=GetLockedDirectory
UpdateEntry&GetSpaceID
WD&U=WriteDirectory&UnlockIt
F6UE =Find&UpdateEntry
RRND=ReadRecordHolding
Directory
WR&D=WriteRecord&Directory
WRSC=WriteRecordSingleCase
GSIDfR=GetSpaceIDforRecord
RRSC=ReadRecordSingleCase
WRD=WriteRecordDirect
RRD=ReadRecordDirect

NOTE that each case also includes:

- the cell of the same name as the operation
- the cell Collect File Statistics

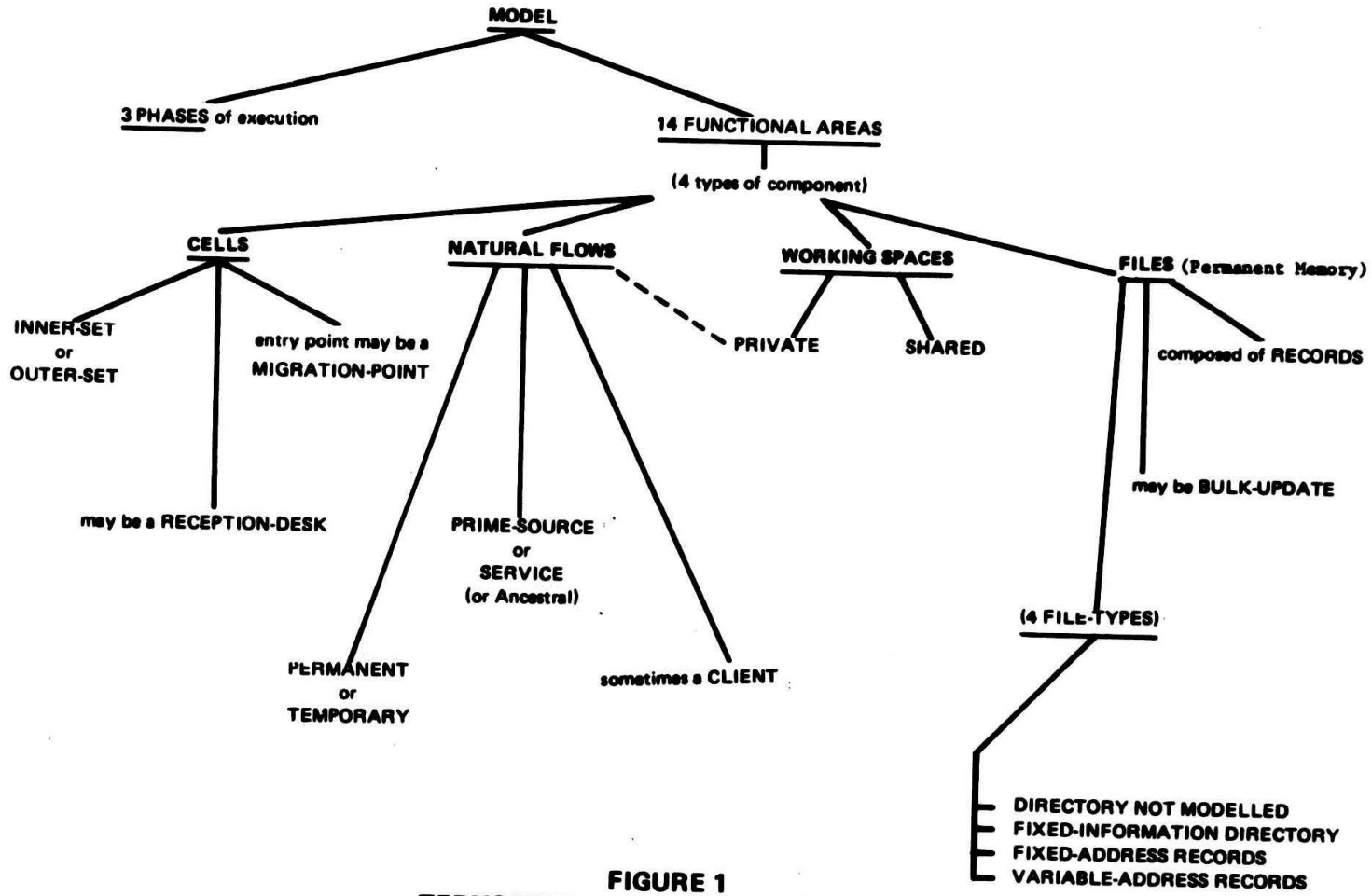
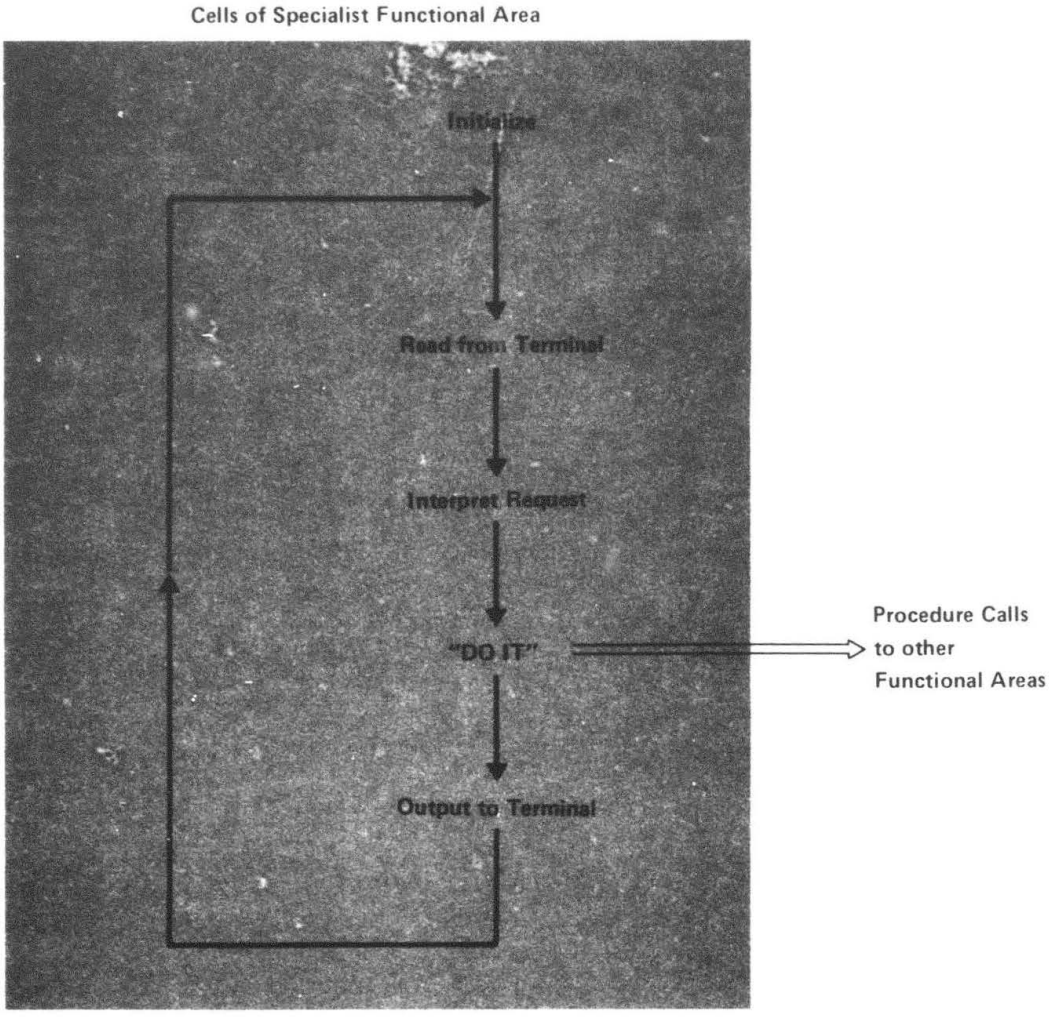


FIGURE 1
TERMS USED TO DESCRIBE THE MODEL



LEGEND

= Code of Functional Area

= One-Way Path of Control ("Go To" Only)

= Two-Way Path of Control ("Go" and "Return")

FIGURE 2
SPECIALIST LOOP

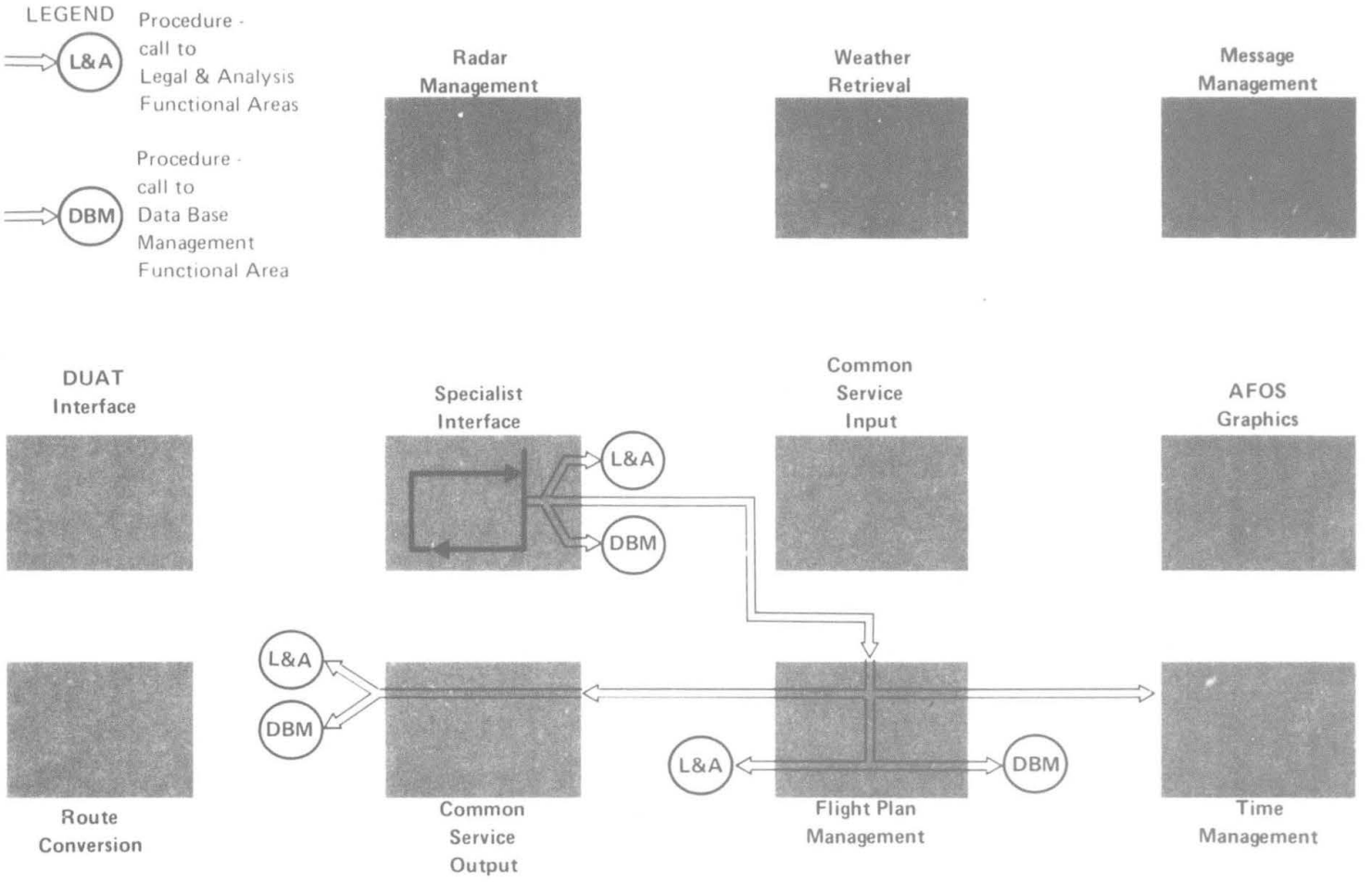


FIGURE 3
SPECIALIST: FILE FLIGHT PLAN FROM CONVERSATION

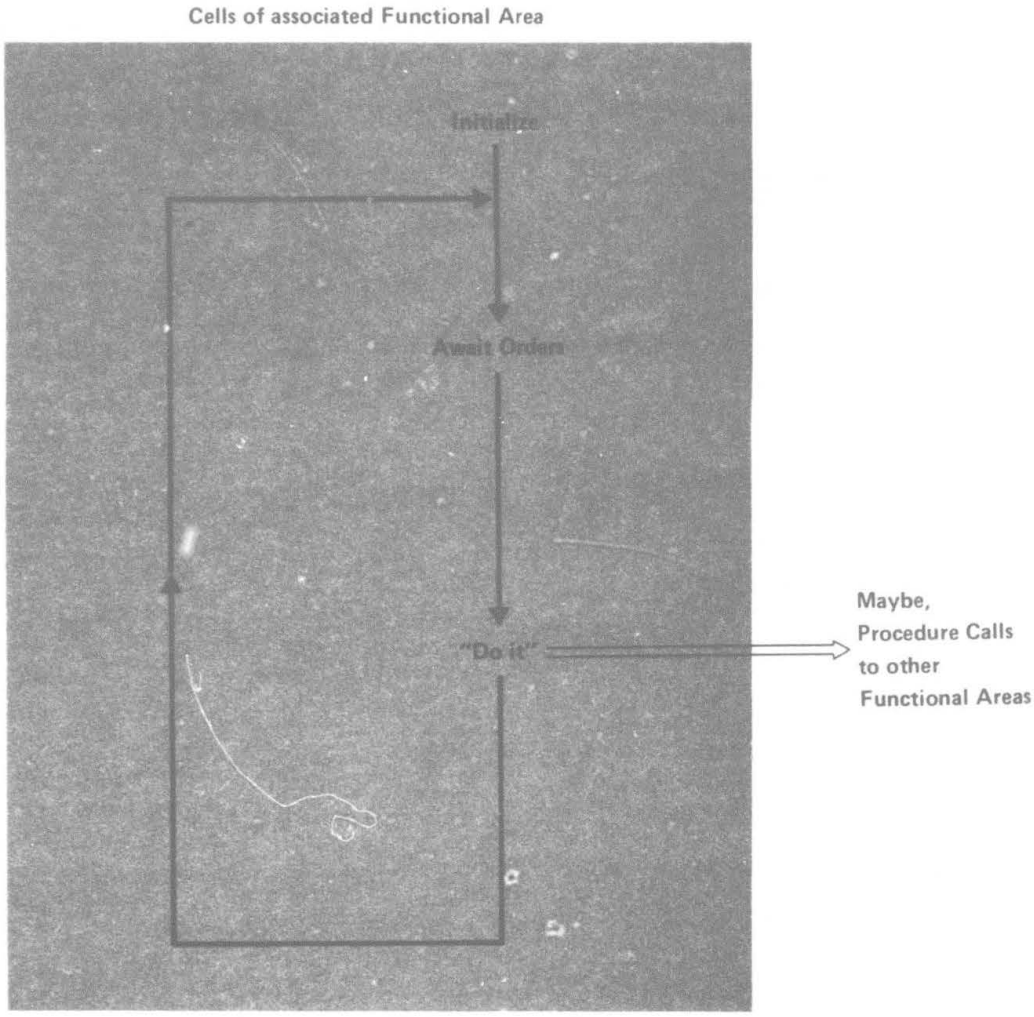


FIGURE 4
A TYPICAL SERVICE NATURAL FLOW

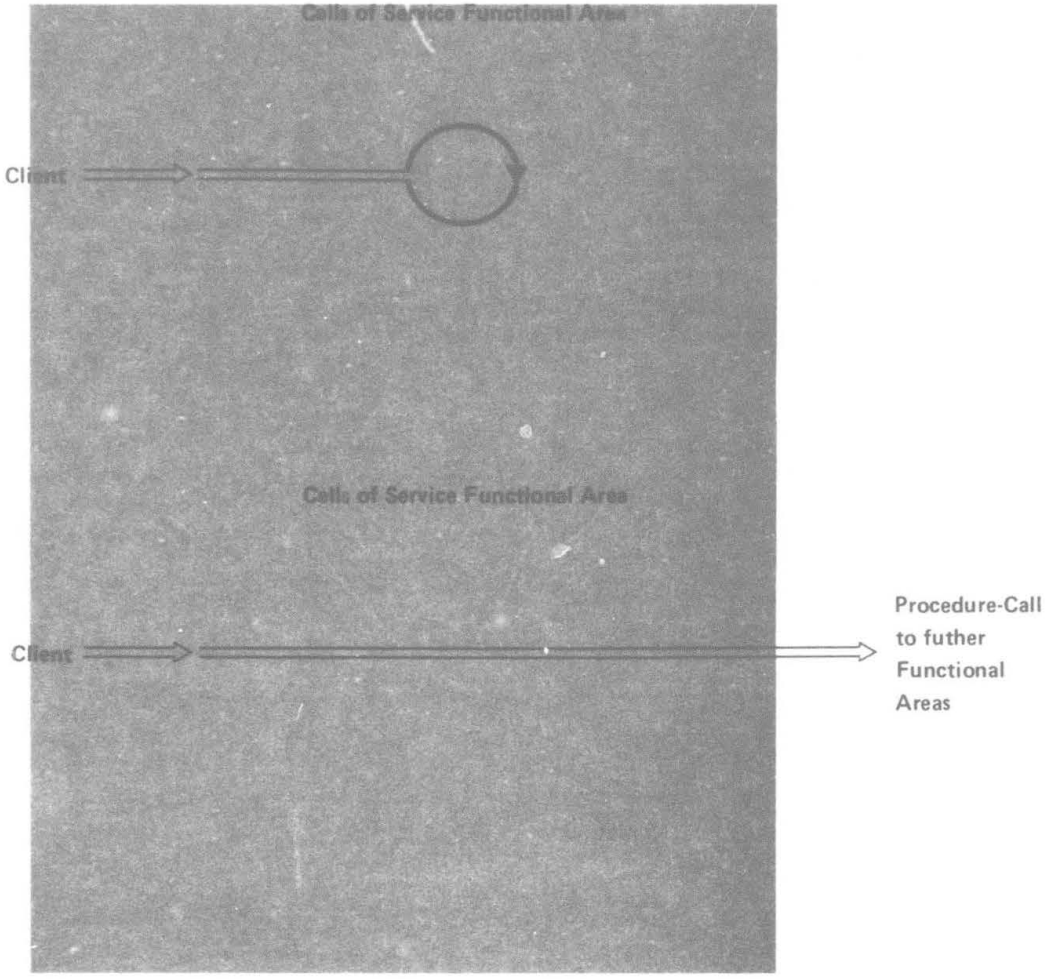
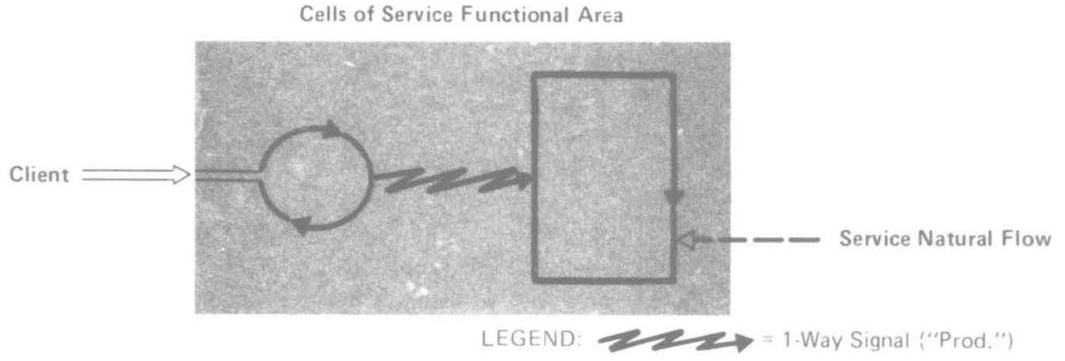
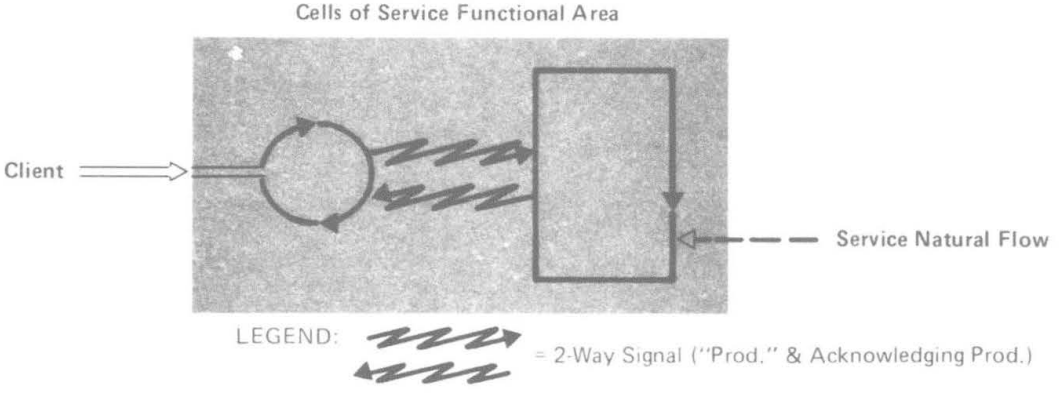


FIGURE 5
CLIENT USING A RECEPTION DESK CELL



Case (a)



Case (b)

FIGURE 6
CLIENT SIGNALS A SERVICE NATURAL FLOW

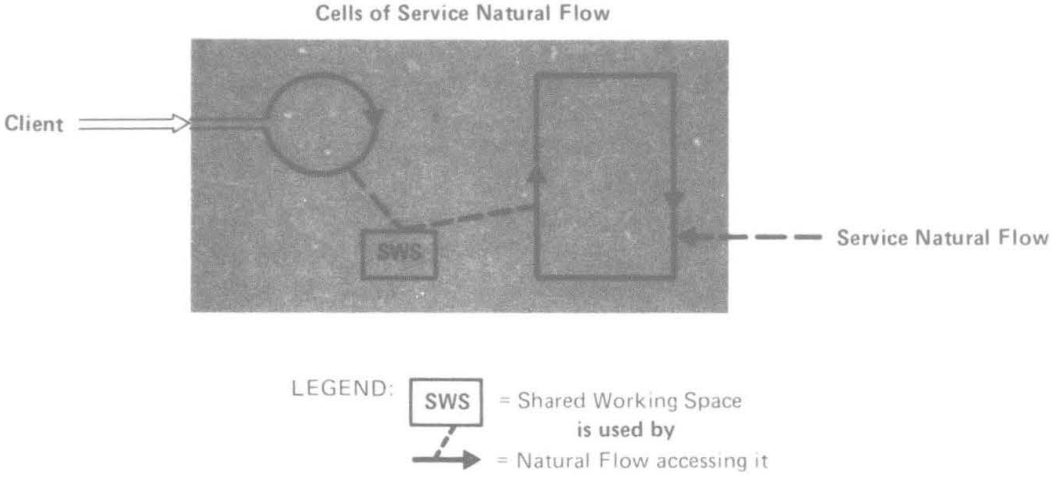


FIGURE 7
USING A SHARED WORKING SPACE

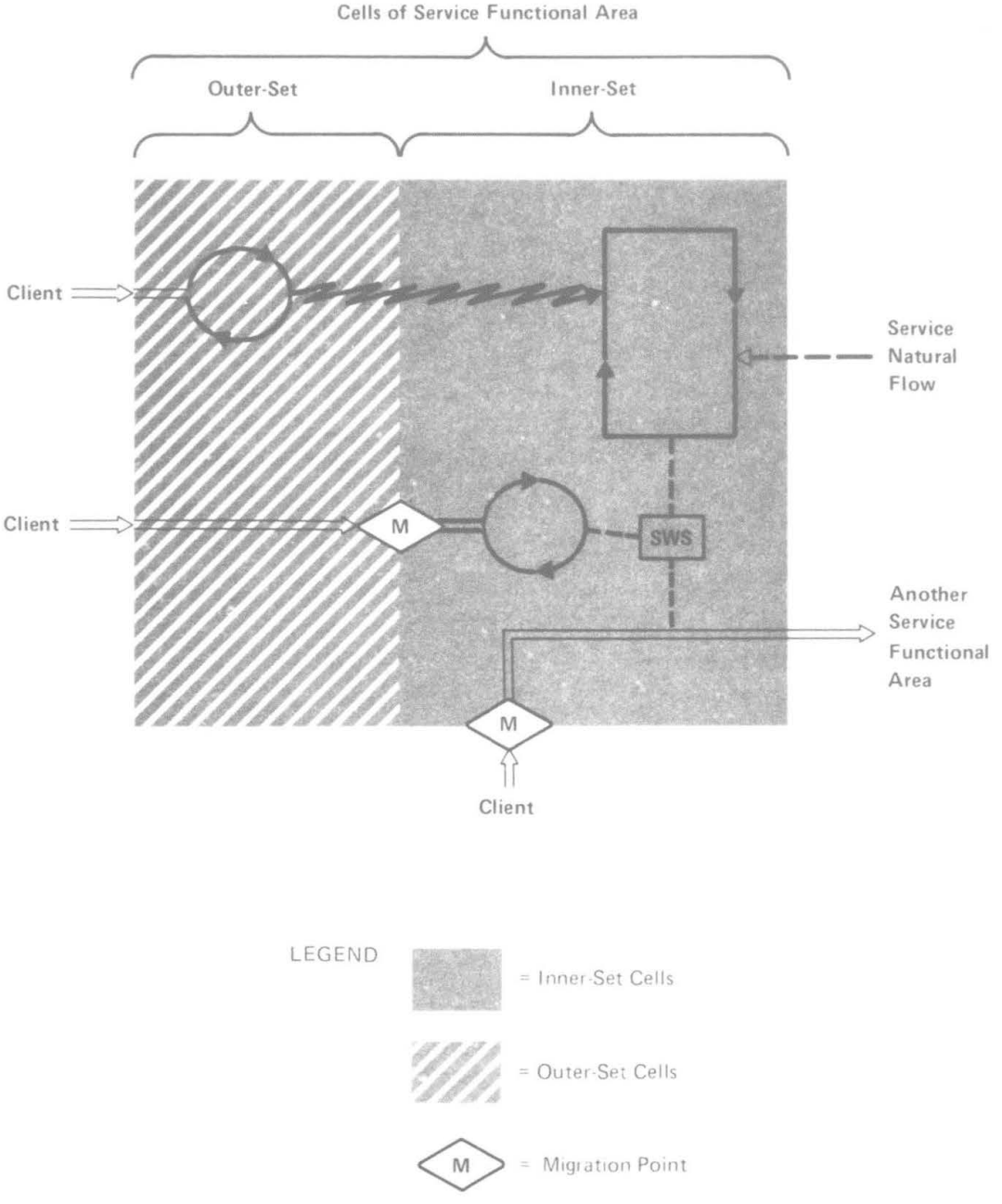


FIGURE 8
INNER SET AND OUTER SET CELLS

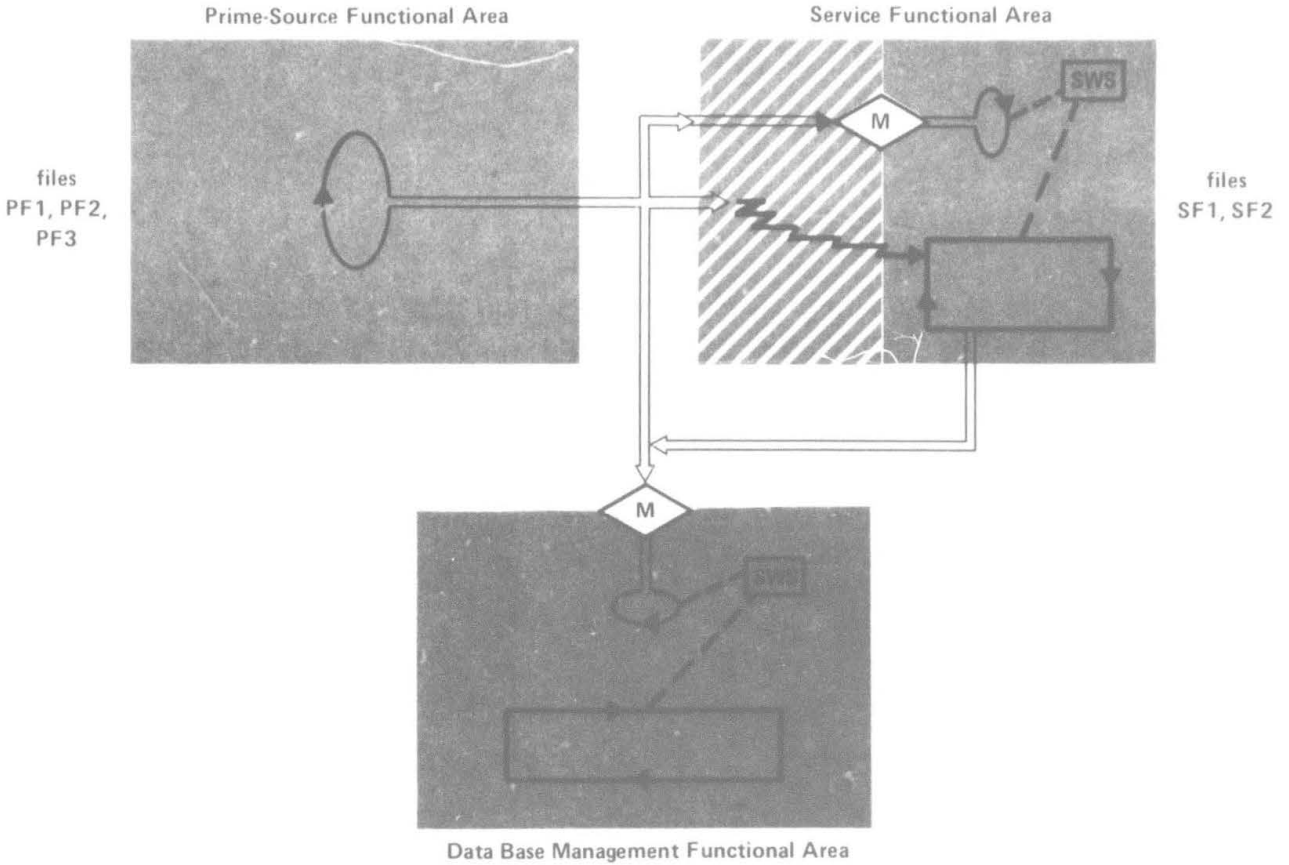


FIGURE 9
A SIMPLE EXAMPLE

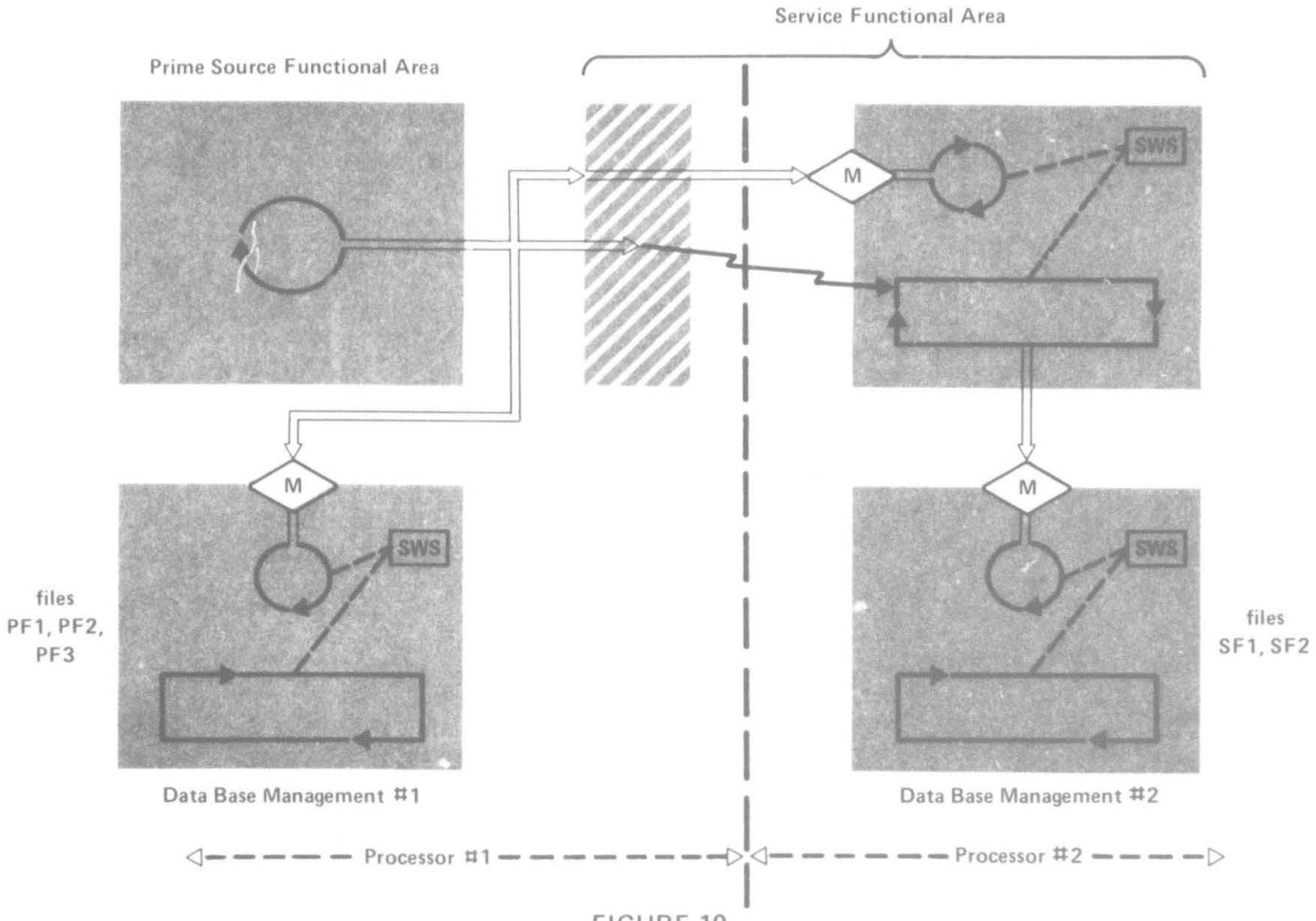


FIGURE 10
THE SIMPLE EXAMPLE DISTRIBUTED

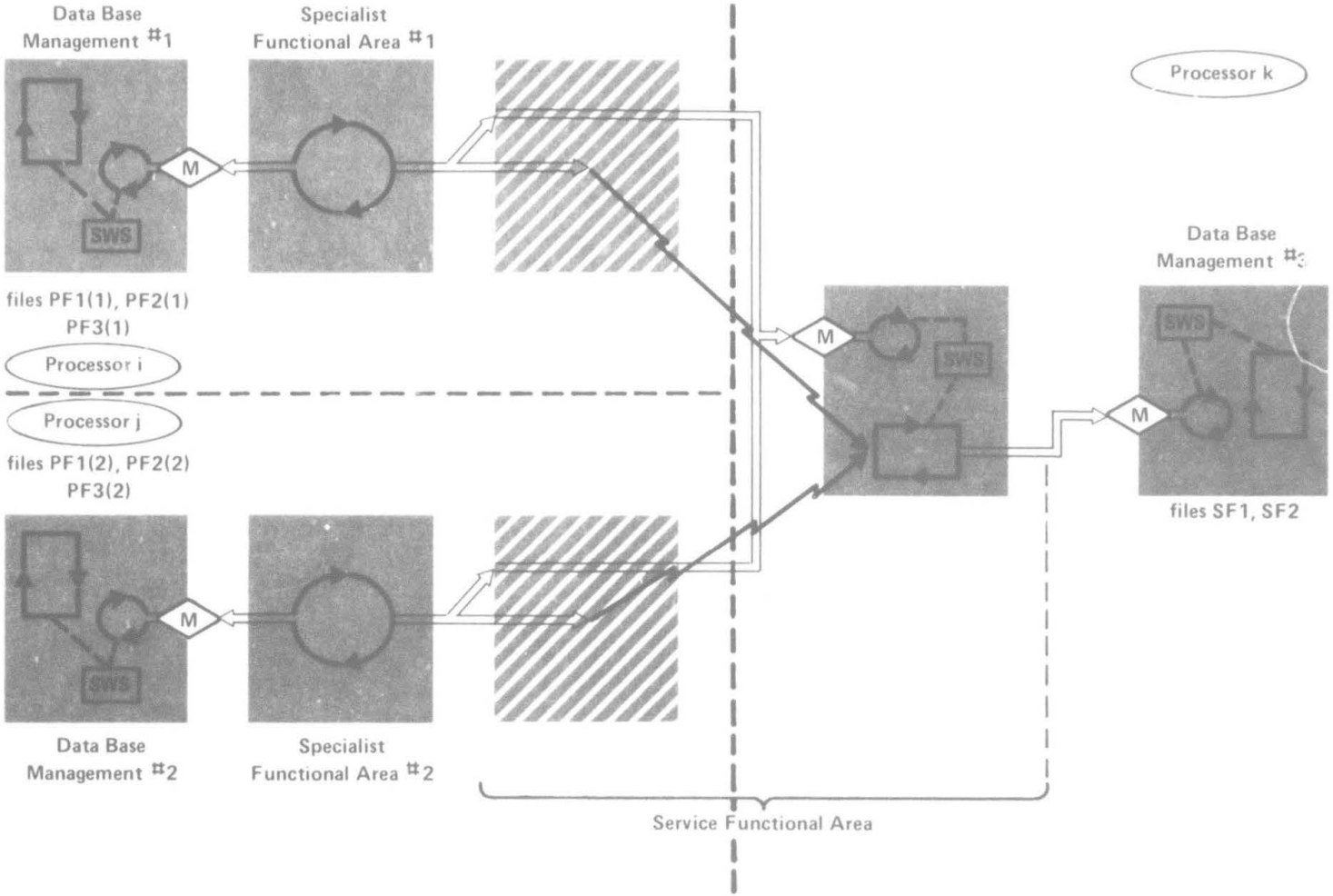


FIGURE 11
THE SAME EXAMPLE: PRIME SOURCE = SPECIALISTS DISTRIBUTED

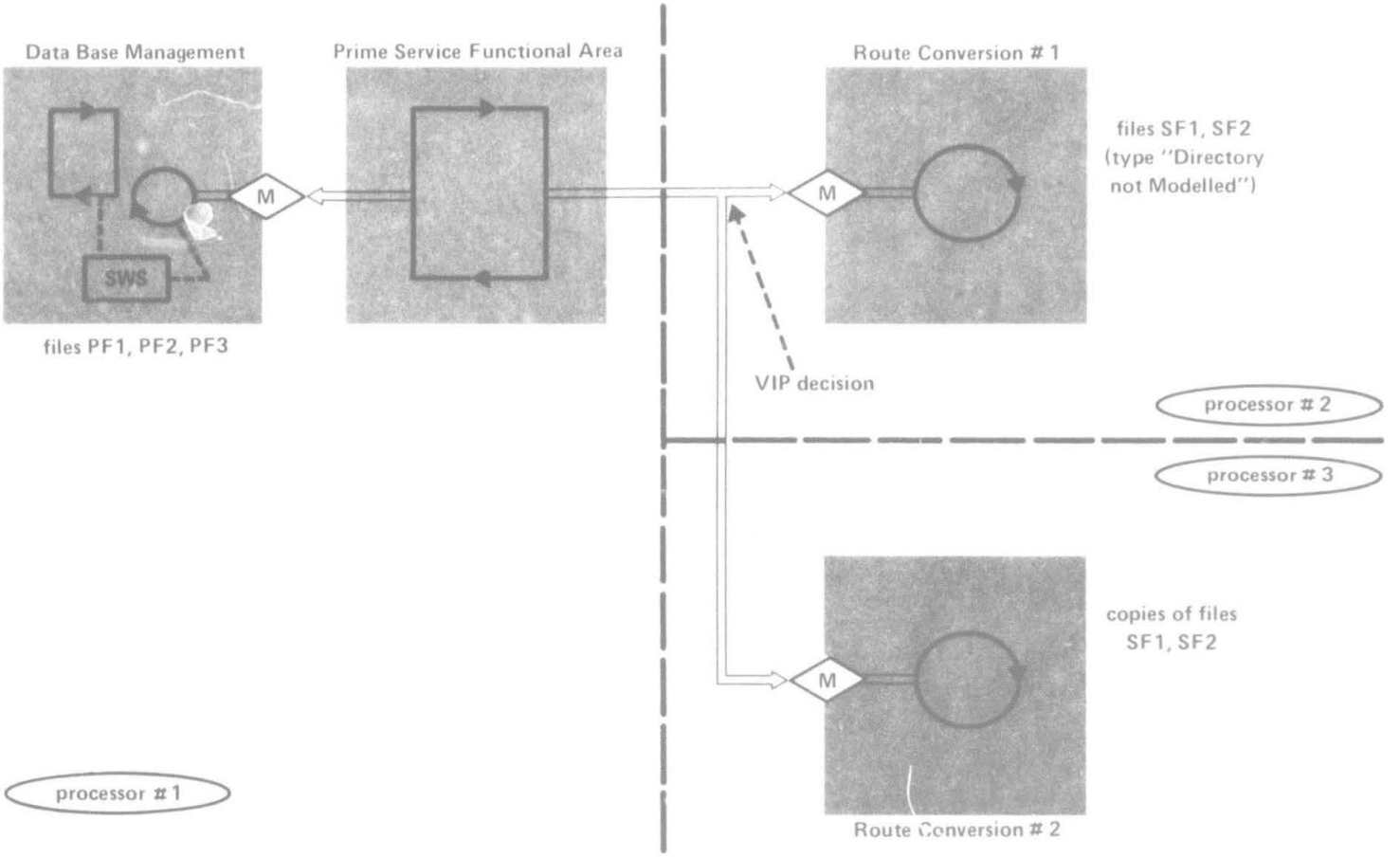


FIGURE 12
SIMPLE EXAMPLE: SERVICE FUNCTIONAL AREA = ROUTE CONVERSION DISTRIBUTED

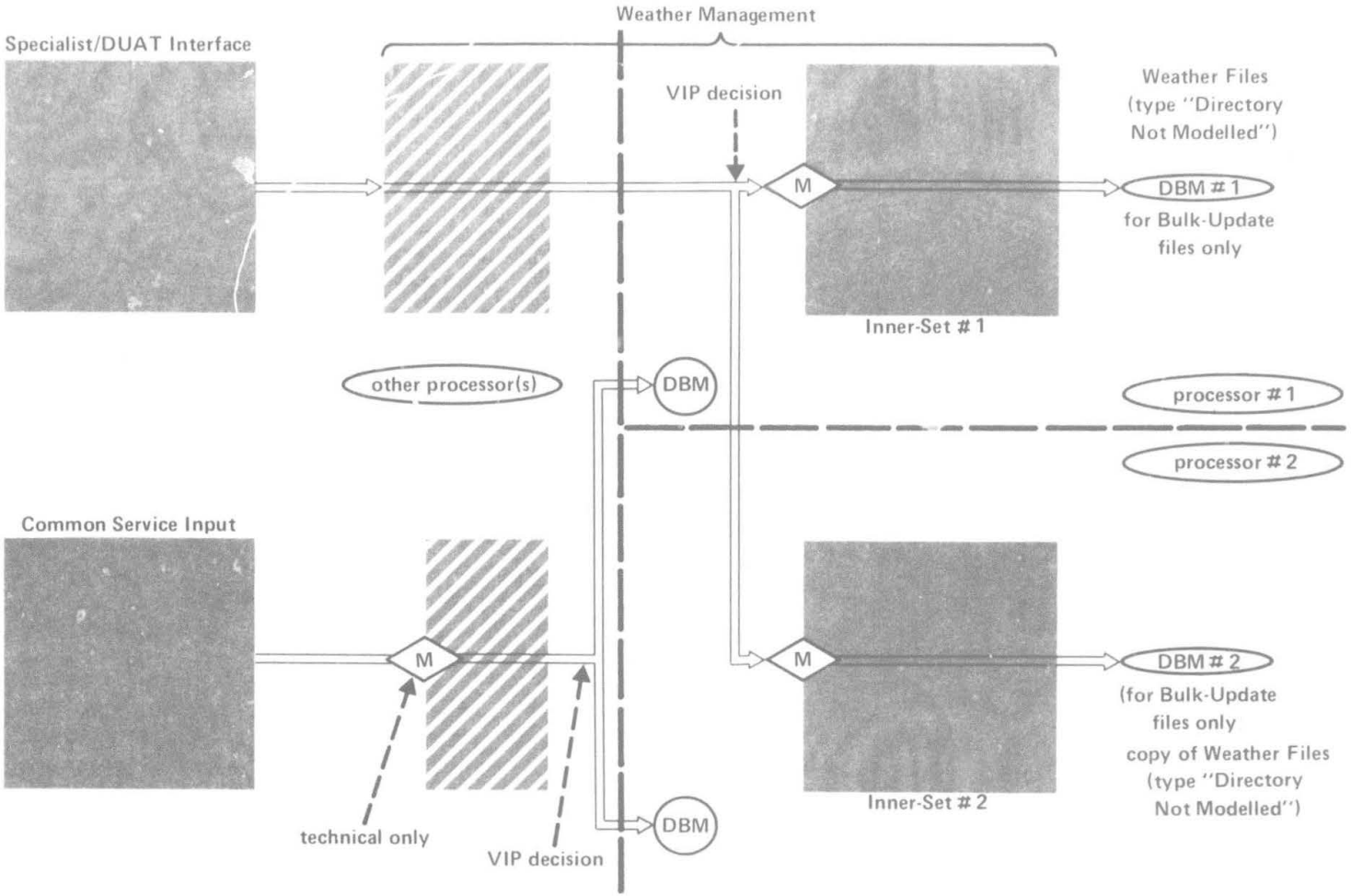


FIGURE 13
DISTRIBUTING WEATHER-MANAGEMENT – (1) NO TEMPORARY SPAWNING

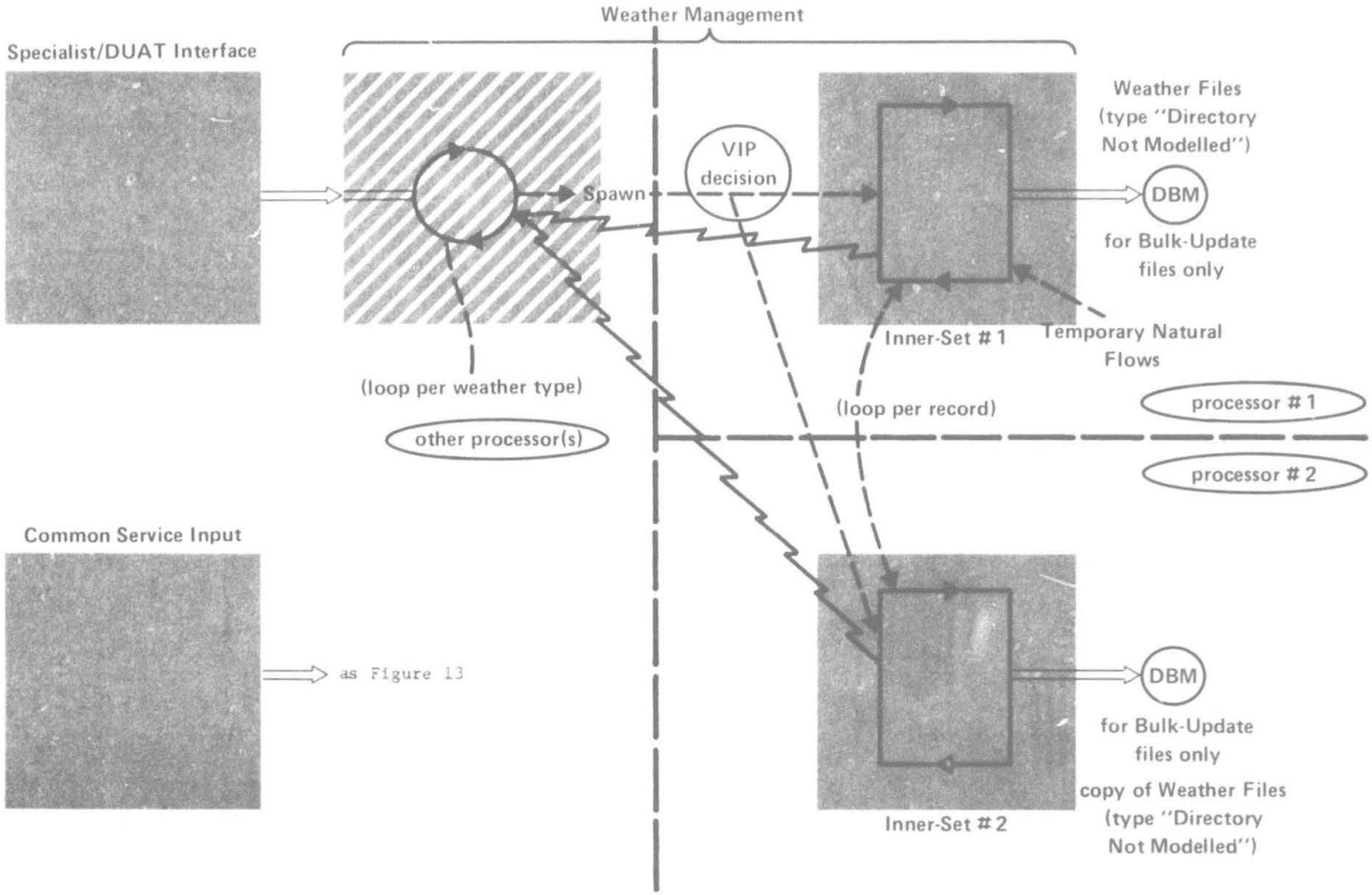


FIGURE 14
DISTRIBUTING WEATHER MANAGEMENT – (2) WITH TEMPORARY SPAWNING

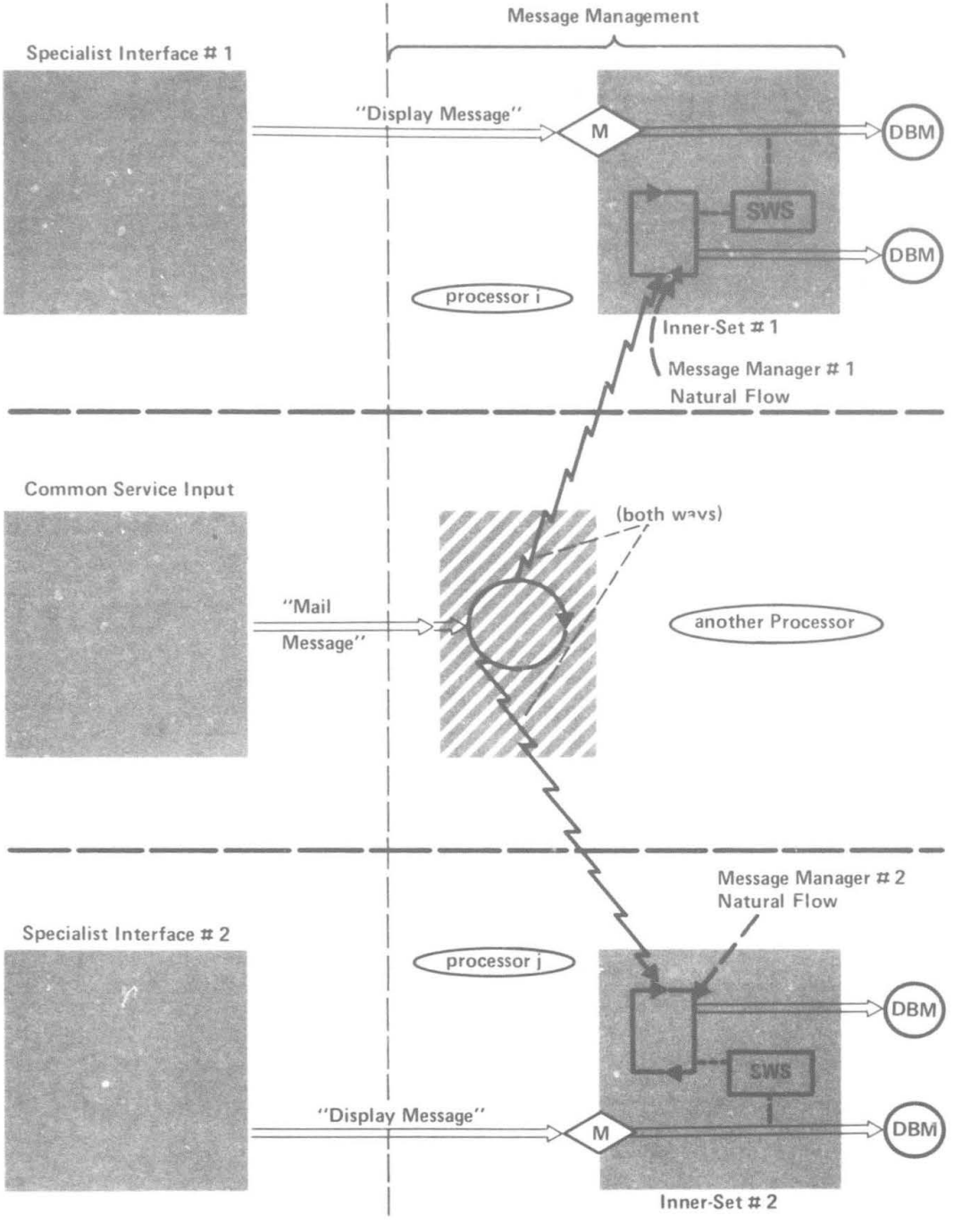


FIGURE 15
DISTRIBUTING MESSAGE MANAGEMENT

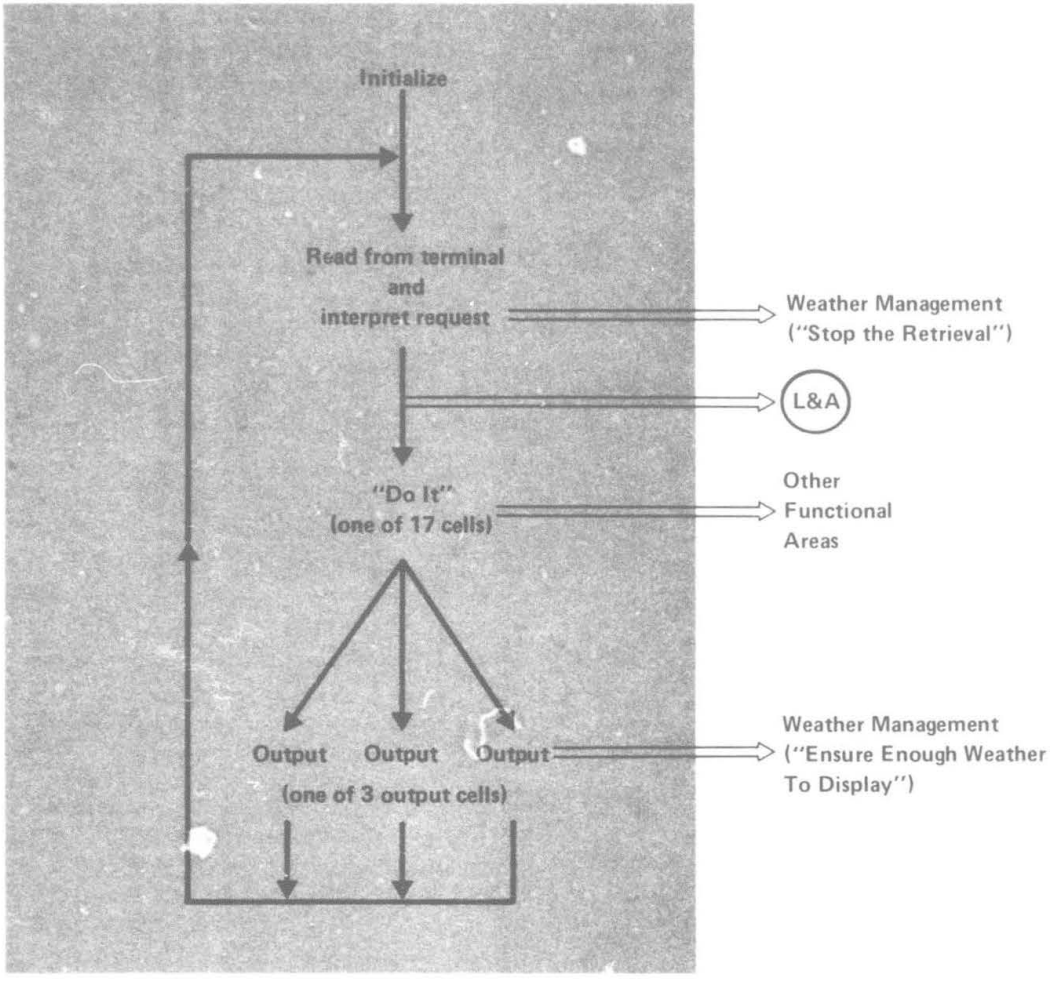


FIGURE 16
SPECIALIST INTERFACE

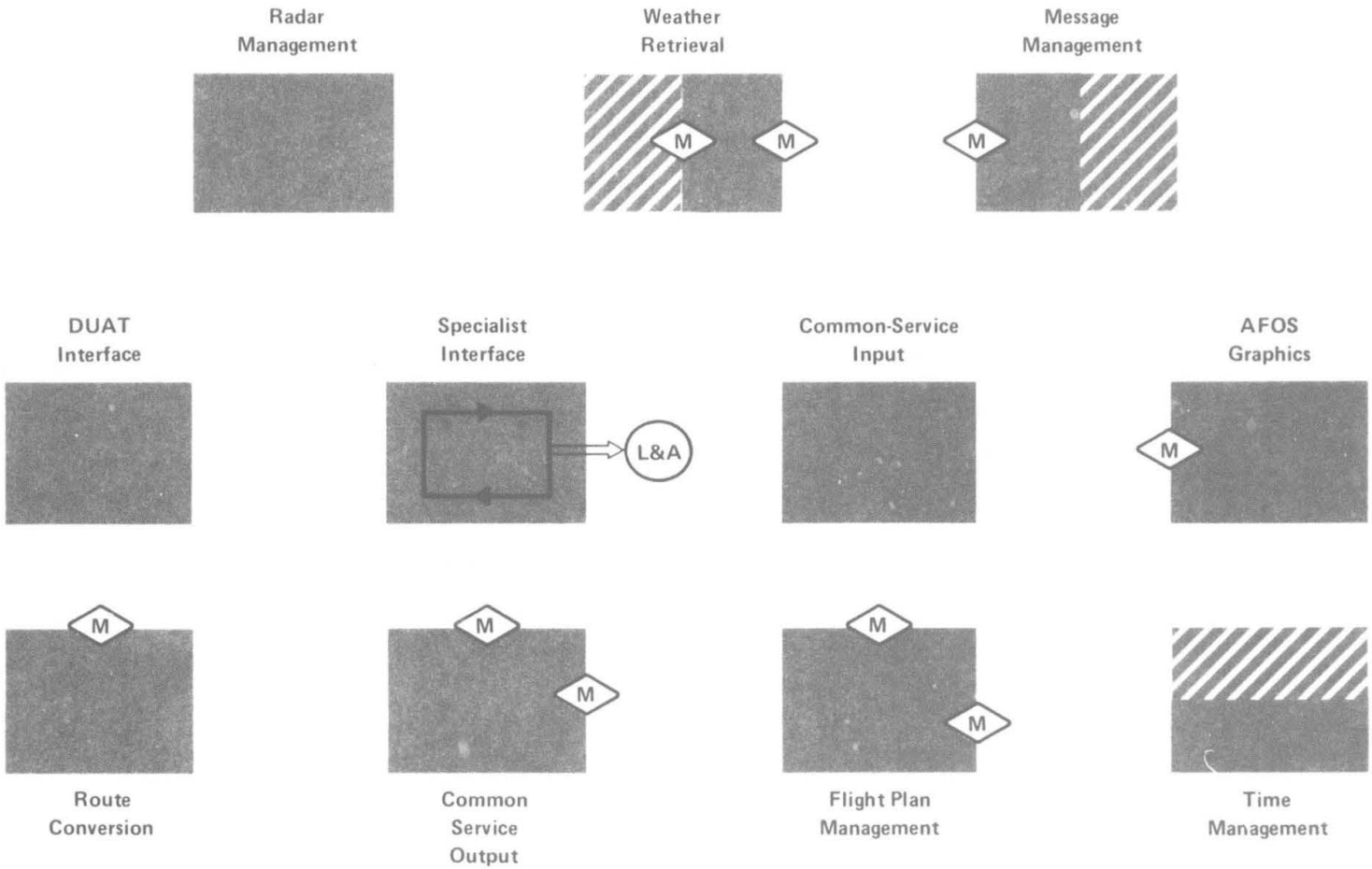


FIGURE 17
SPECIALISTS: SIMPLE INPUT/PAGE FORWARD/PAGE BACKWARD/REQUEST SEND FORMAT

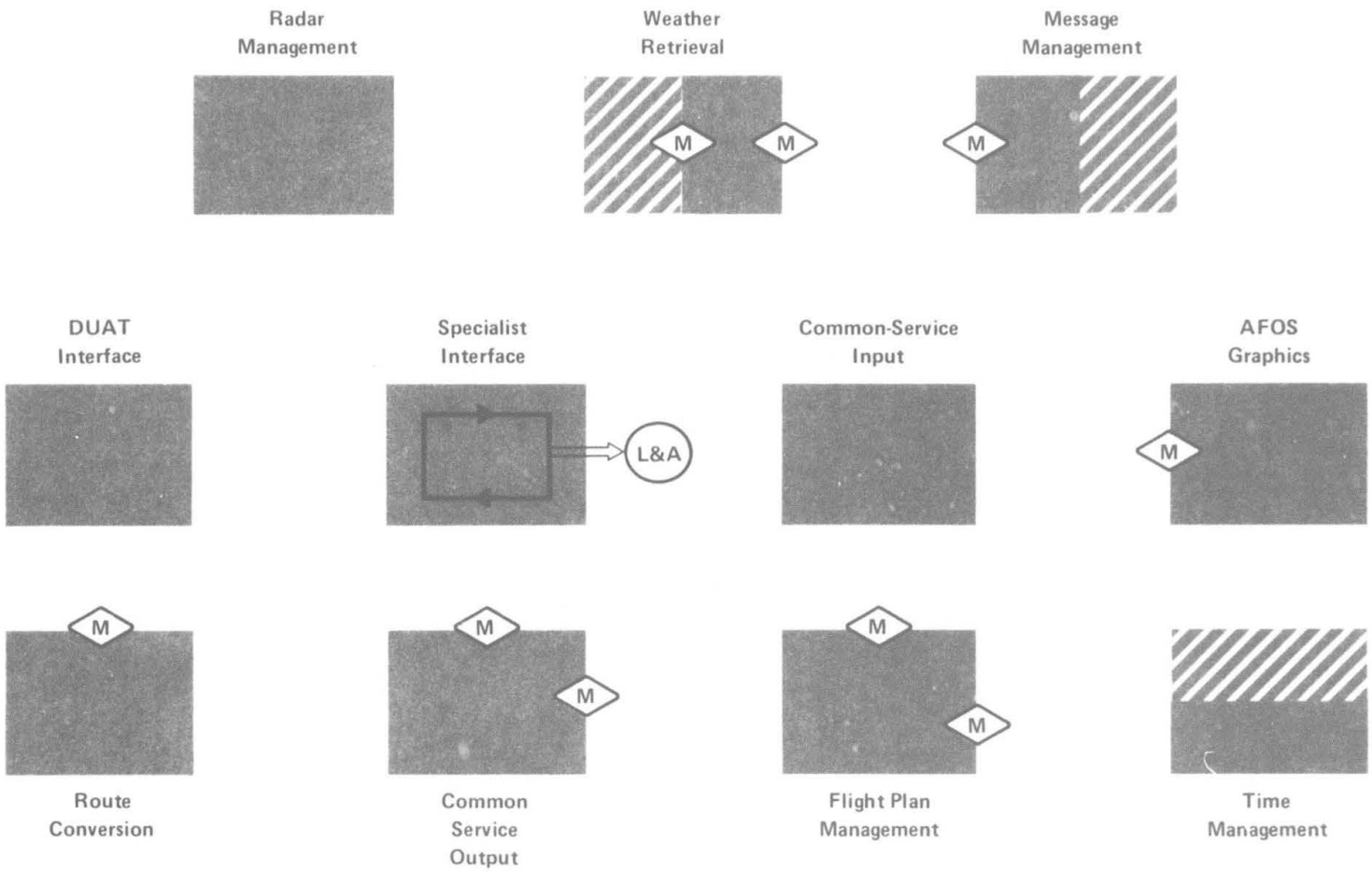


FIGURE 17
SPECIALISTS: SIMPLE INPUT/PAGE FORWARD/PAGE BACKWARD/REQUEST SEND FORMAT

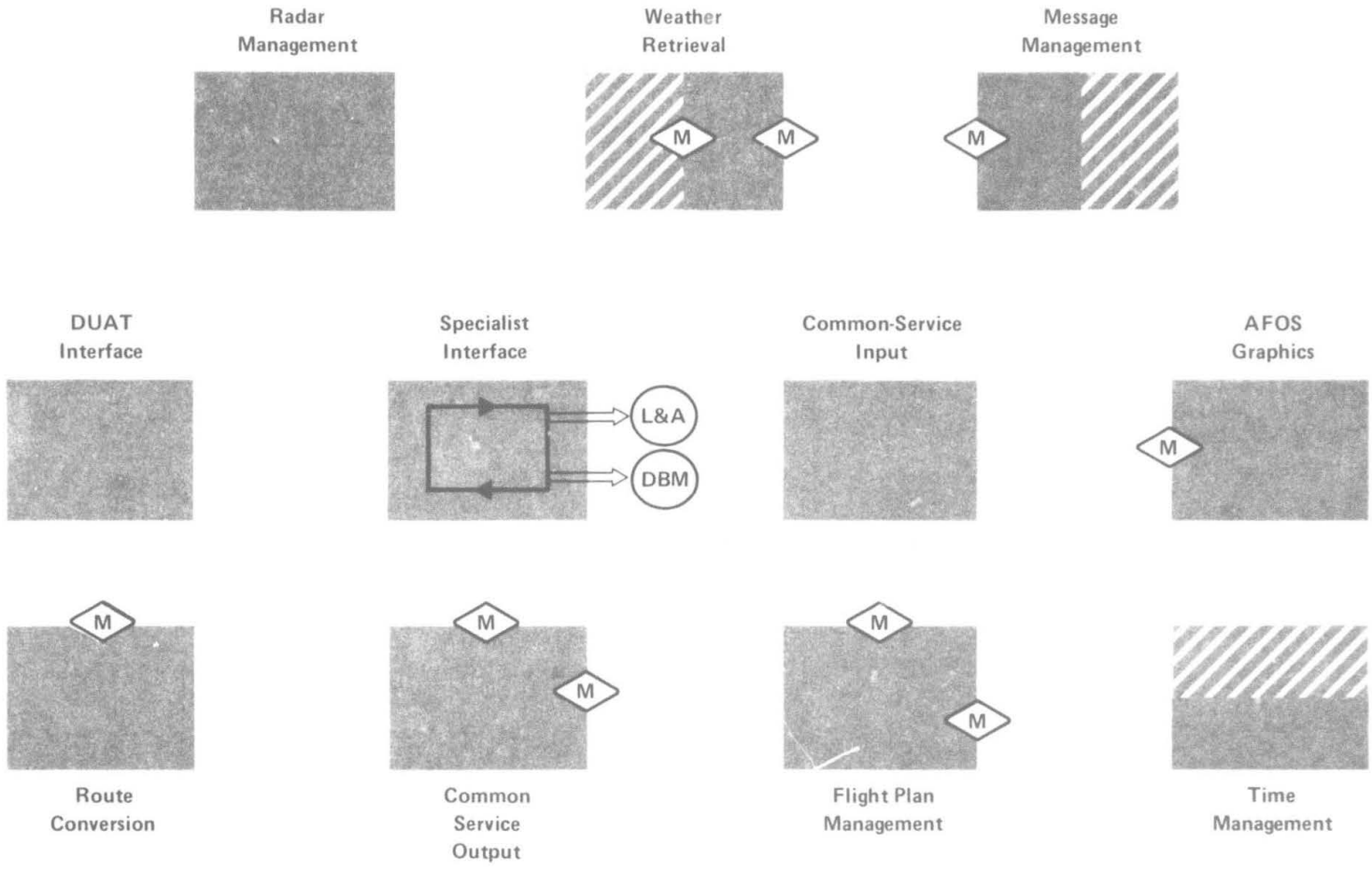


FIGURE 18
SPECIALISTS: FLIGHT LOG/START CONVERSATION WITH NEW FLIGHT PLAN/RESUME CONVERSATION WITH CURRENT FLIGHT PLAN/NON ROUTE DATA FOR FLIGHT PLAN

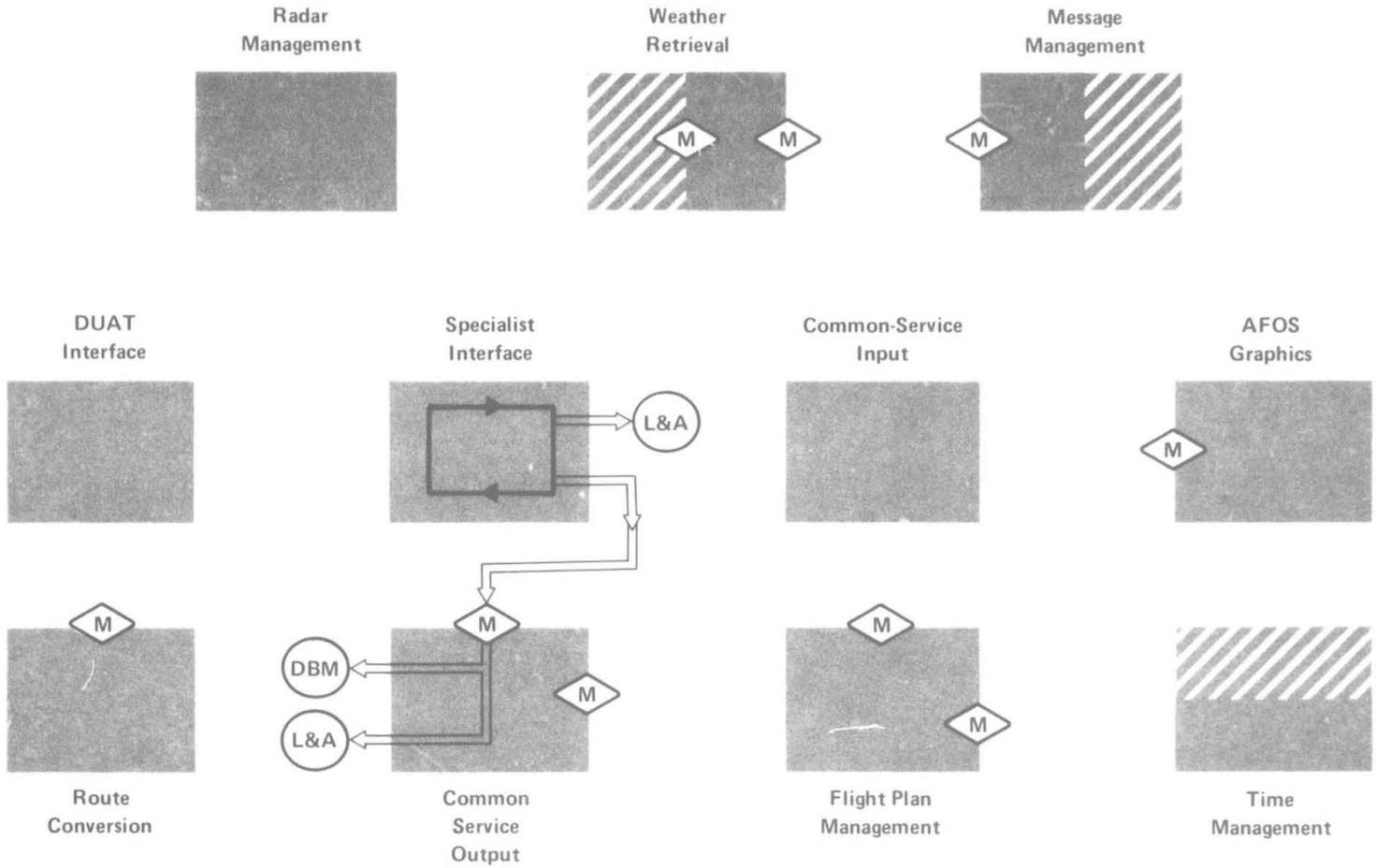


FIGURE 19
SPECIALIST: EXECUTE LINK SEND/PRINT SCREEN

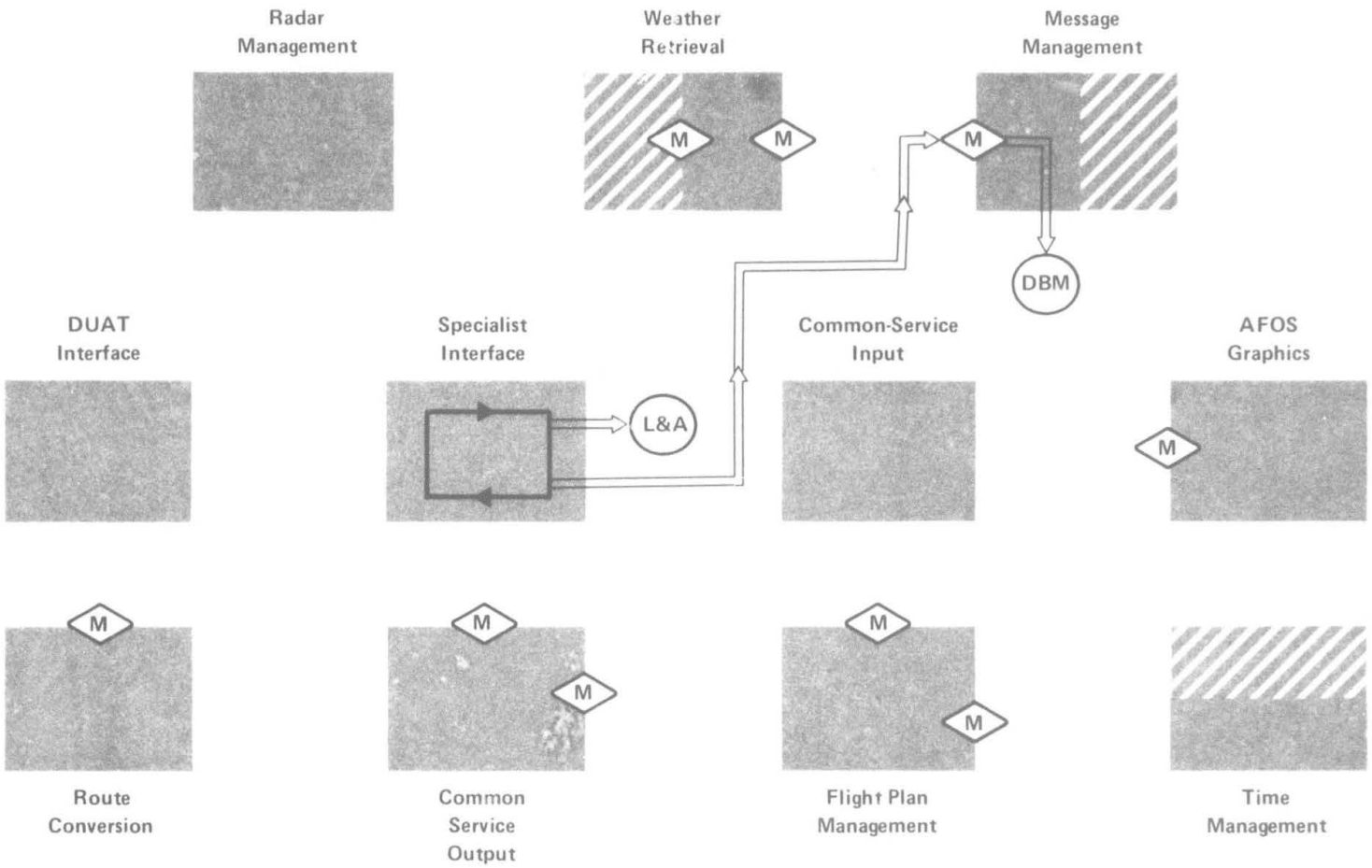


FIGURE 20
SPECIALIST: DISPLAY MESSAGE

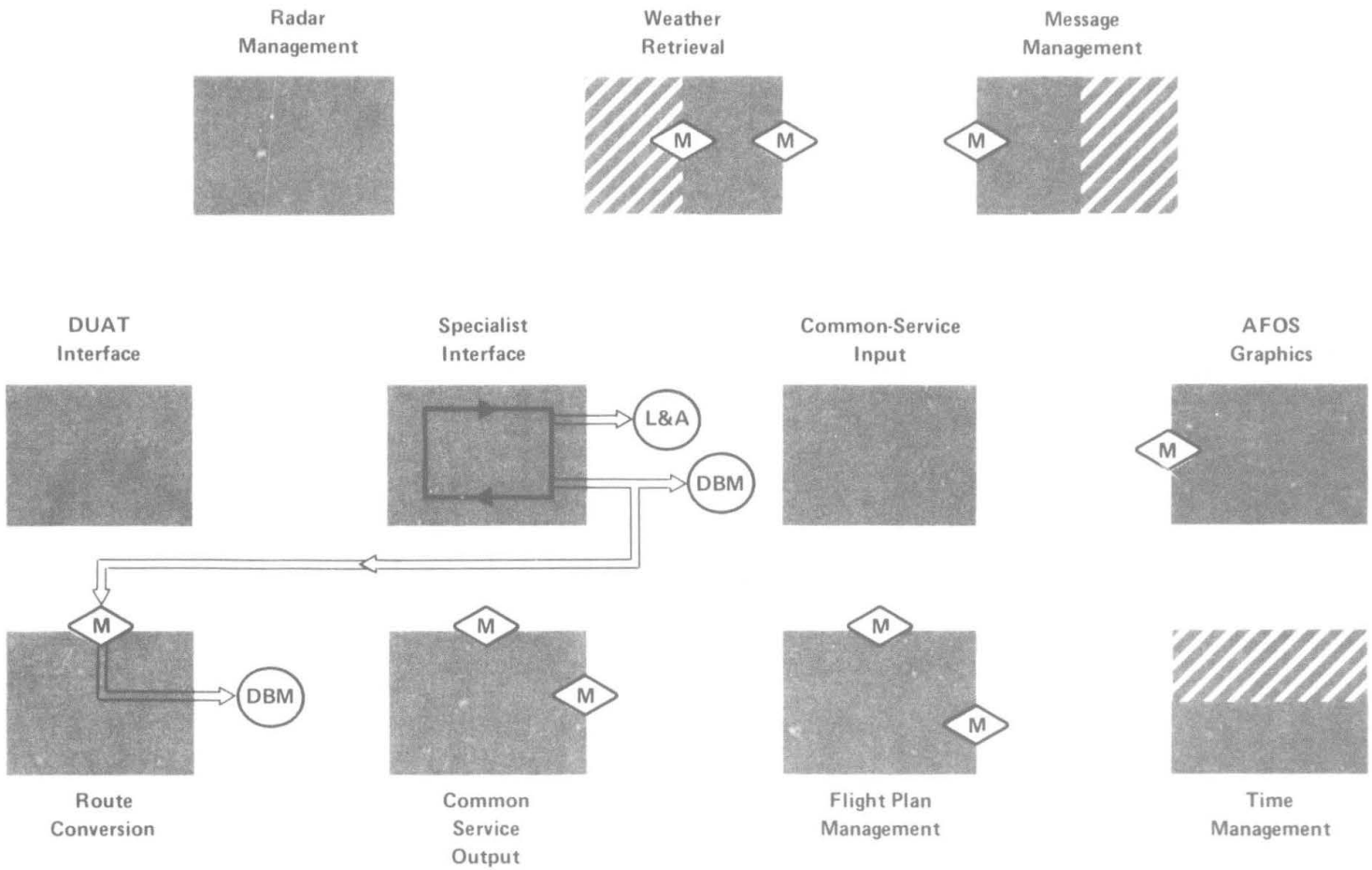


FIGURE 21
SPECIALIST: ROUTE DATA FOR FLIGHT PLAN

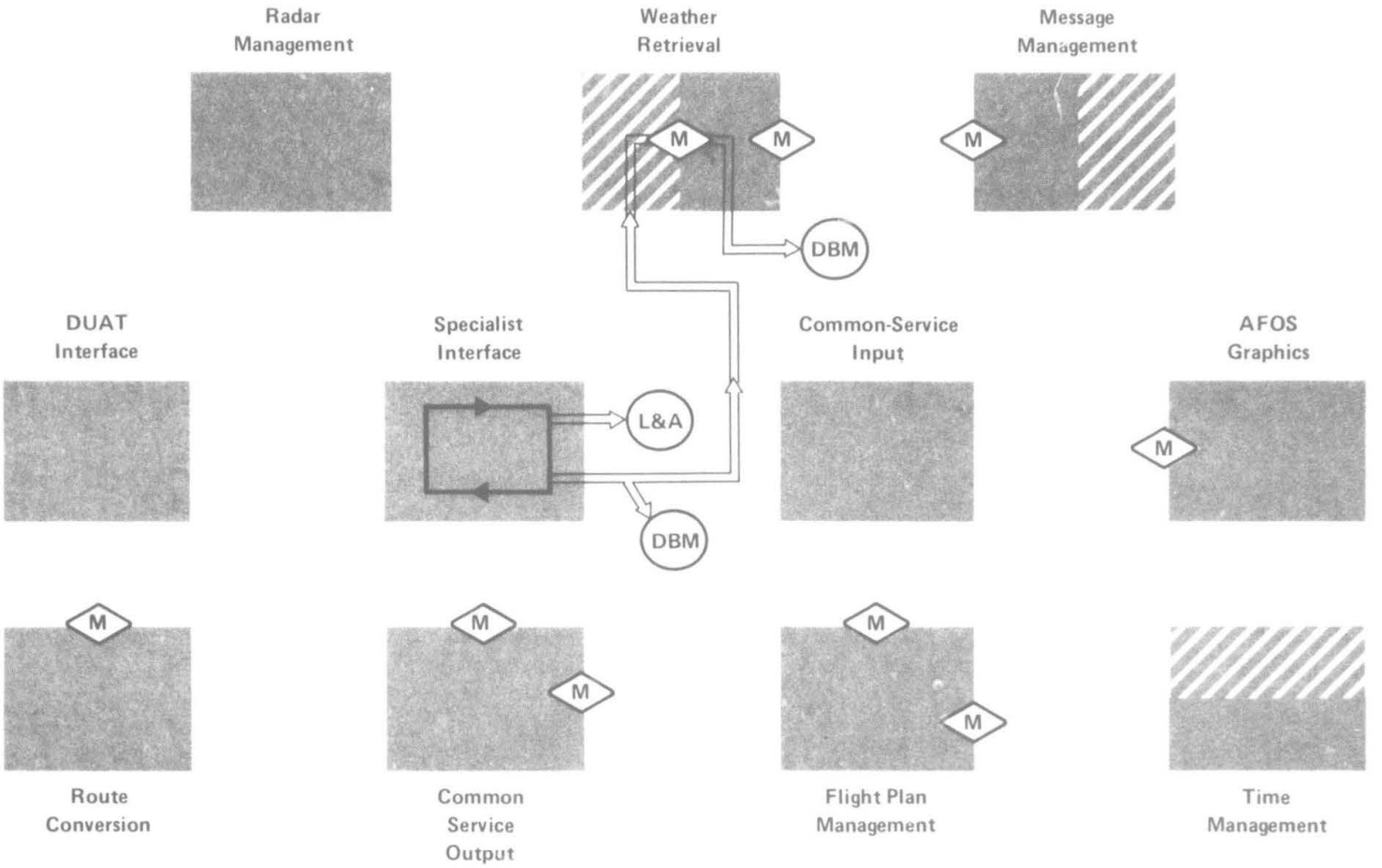


FIGURE 22
SPECIALIST: REQUEST WEATHER

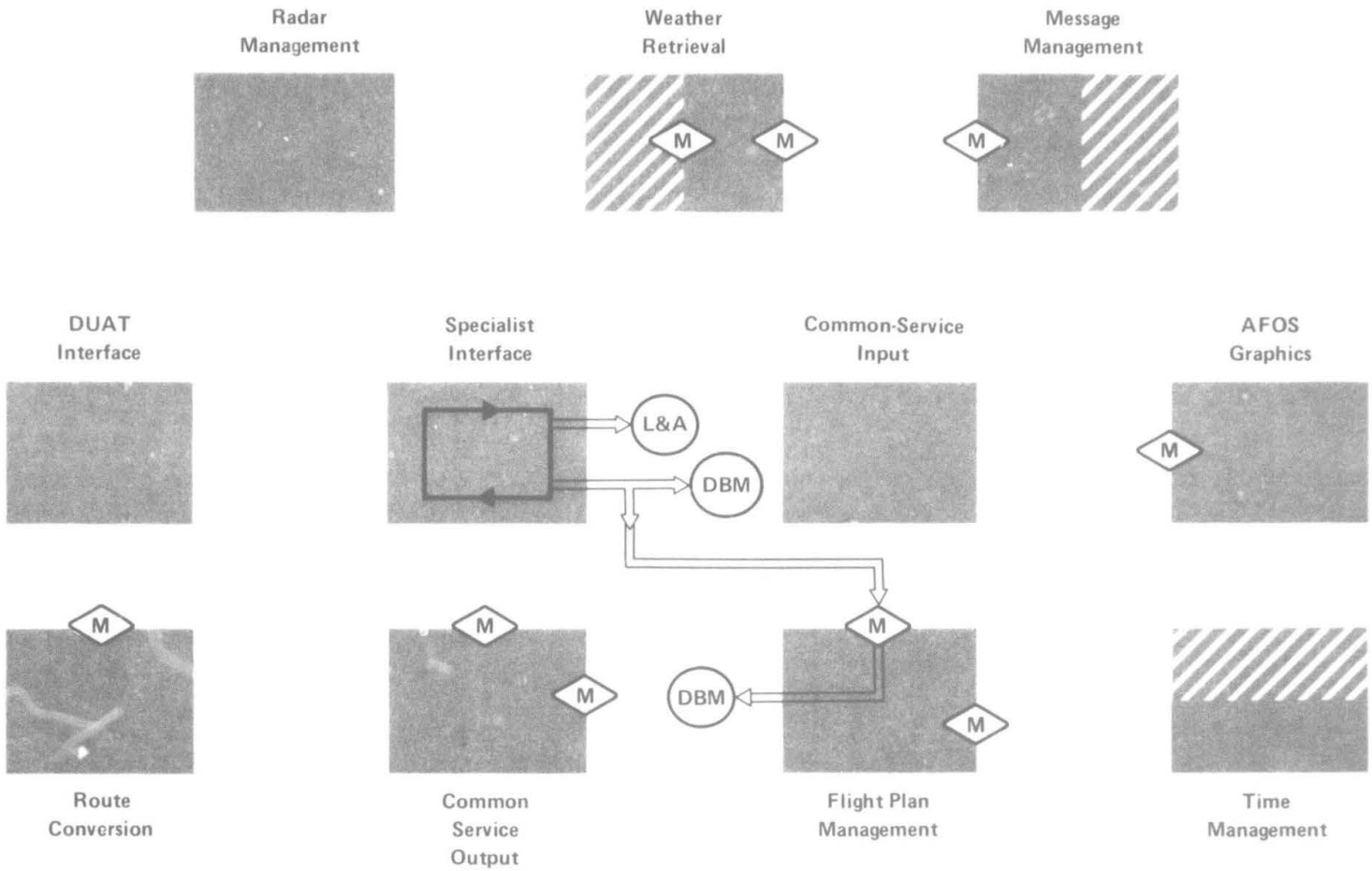


FIGURE 23
SPECIALIST: START CONVERSATION WITH FILED FLIGHT PLAN

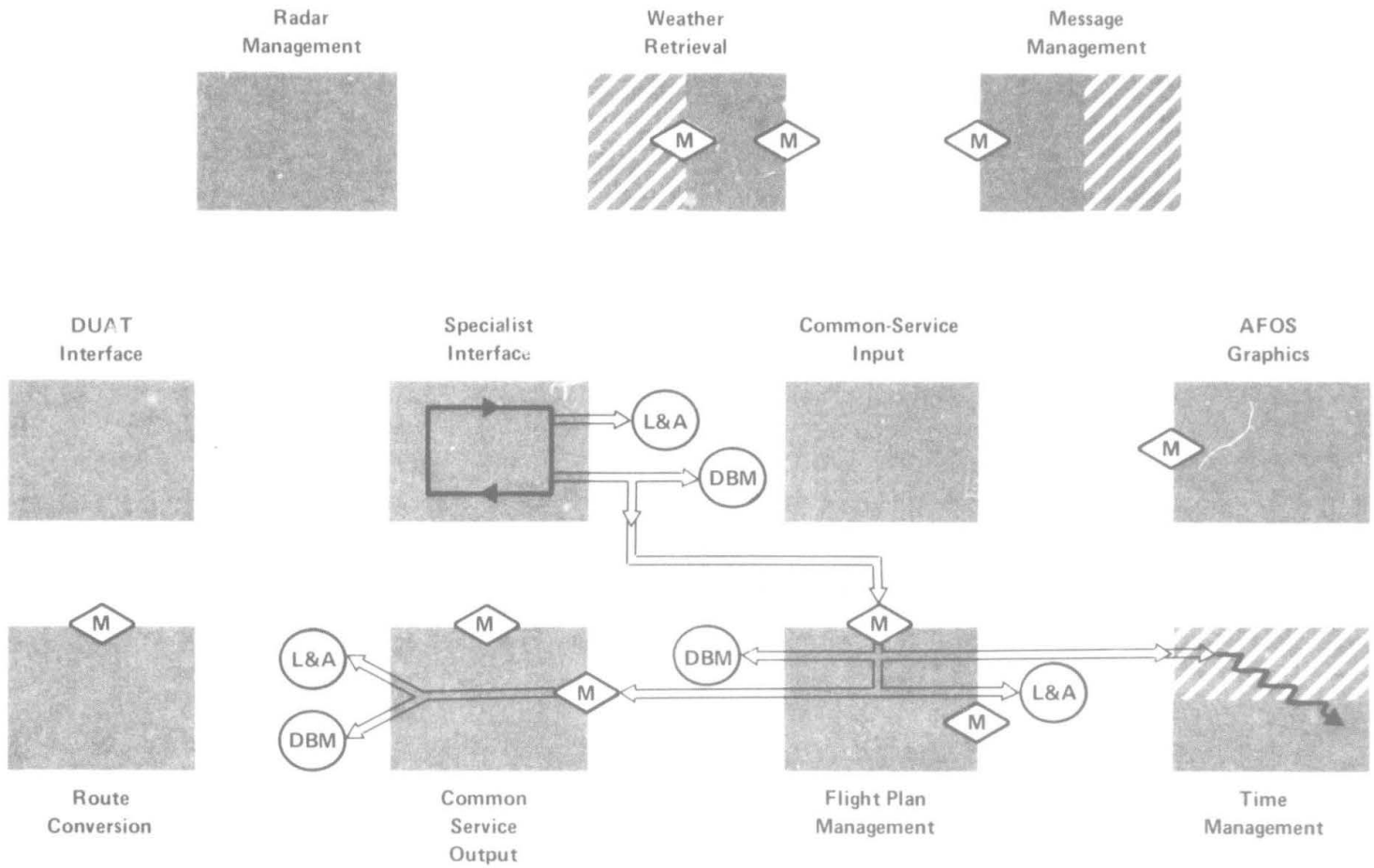


FIGURE 24
SPECIALIST: FILE FLIGHT PLAN FROM CONVERSATION

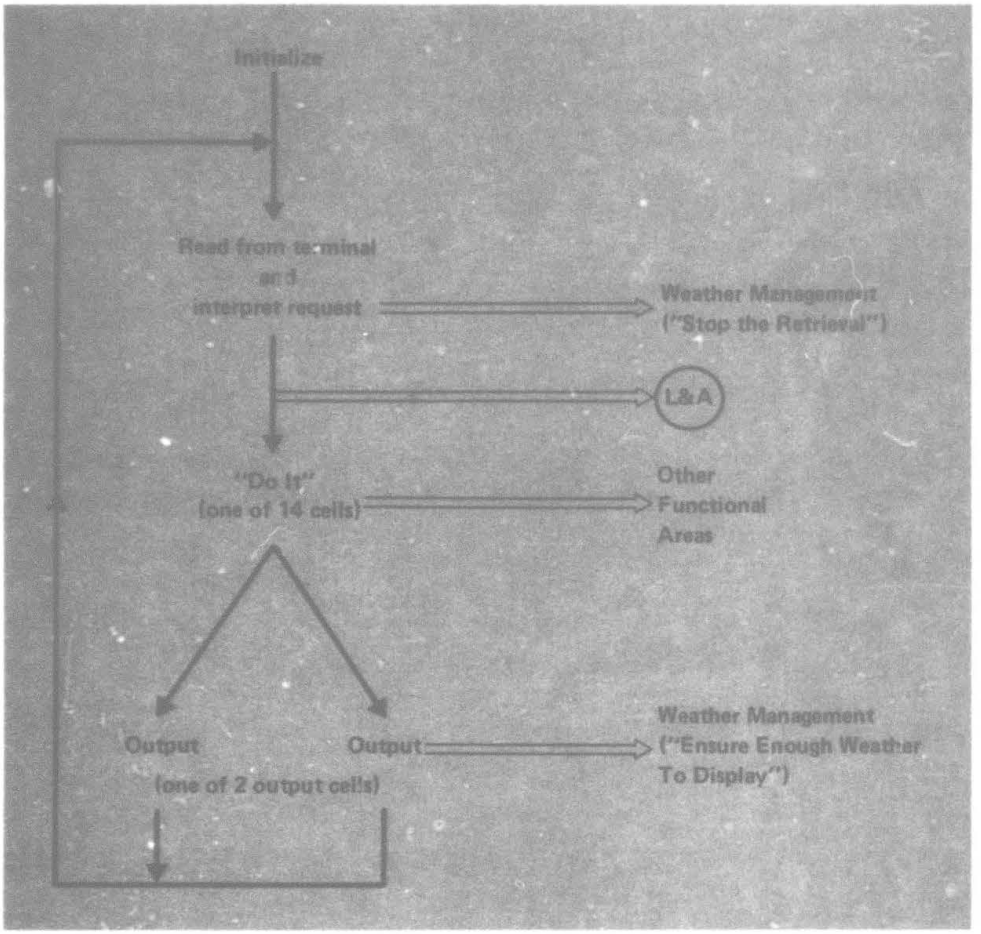


FIGURE 25
DUAT INTERFACE

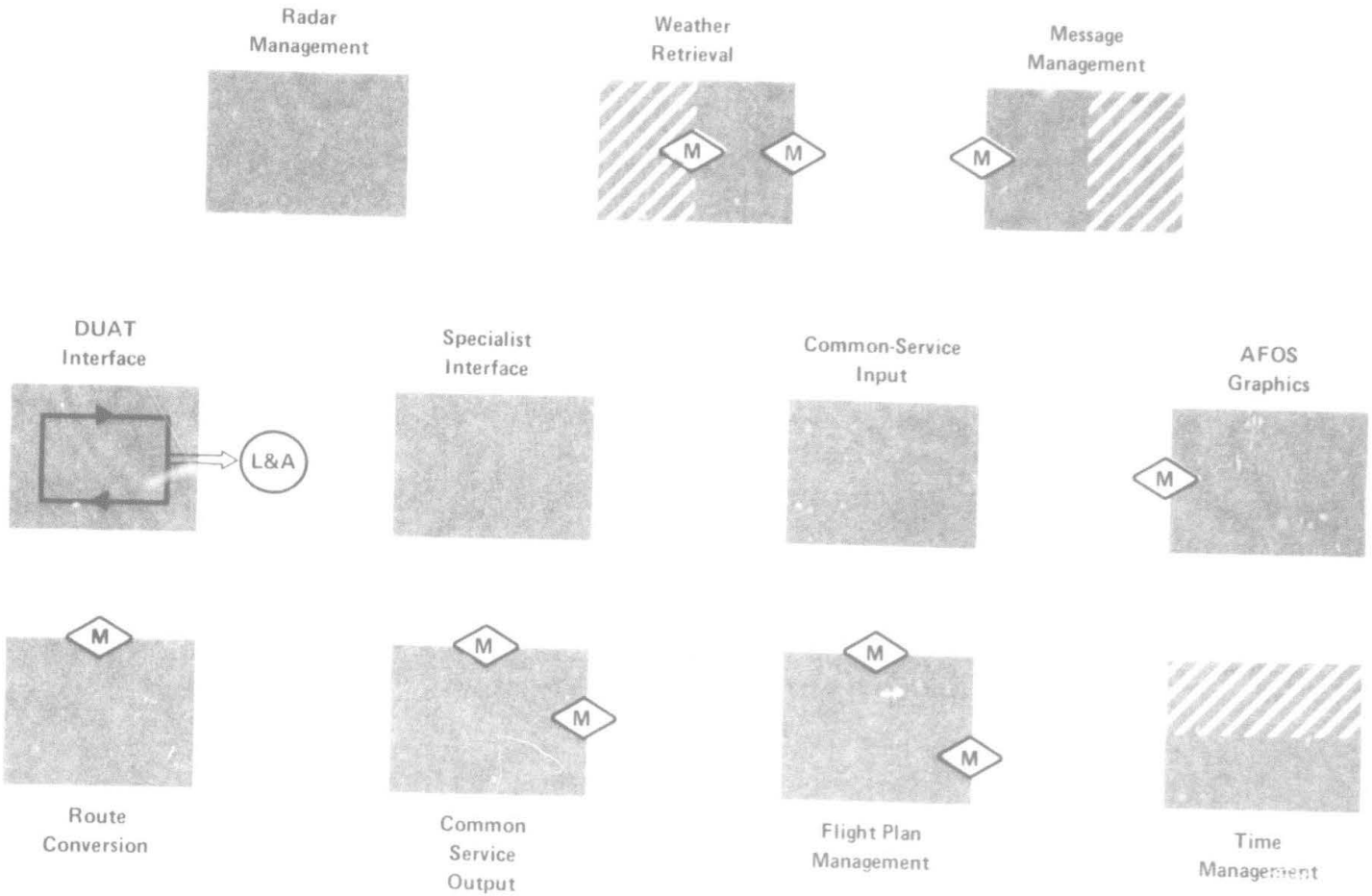


FIGURE 26
DUAT: DUAT SIMPLE INPUT/DUAT PAGE FORWARD/DUAT PAGE BACKWARD/REQUEST DUAT SEND FORMAT

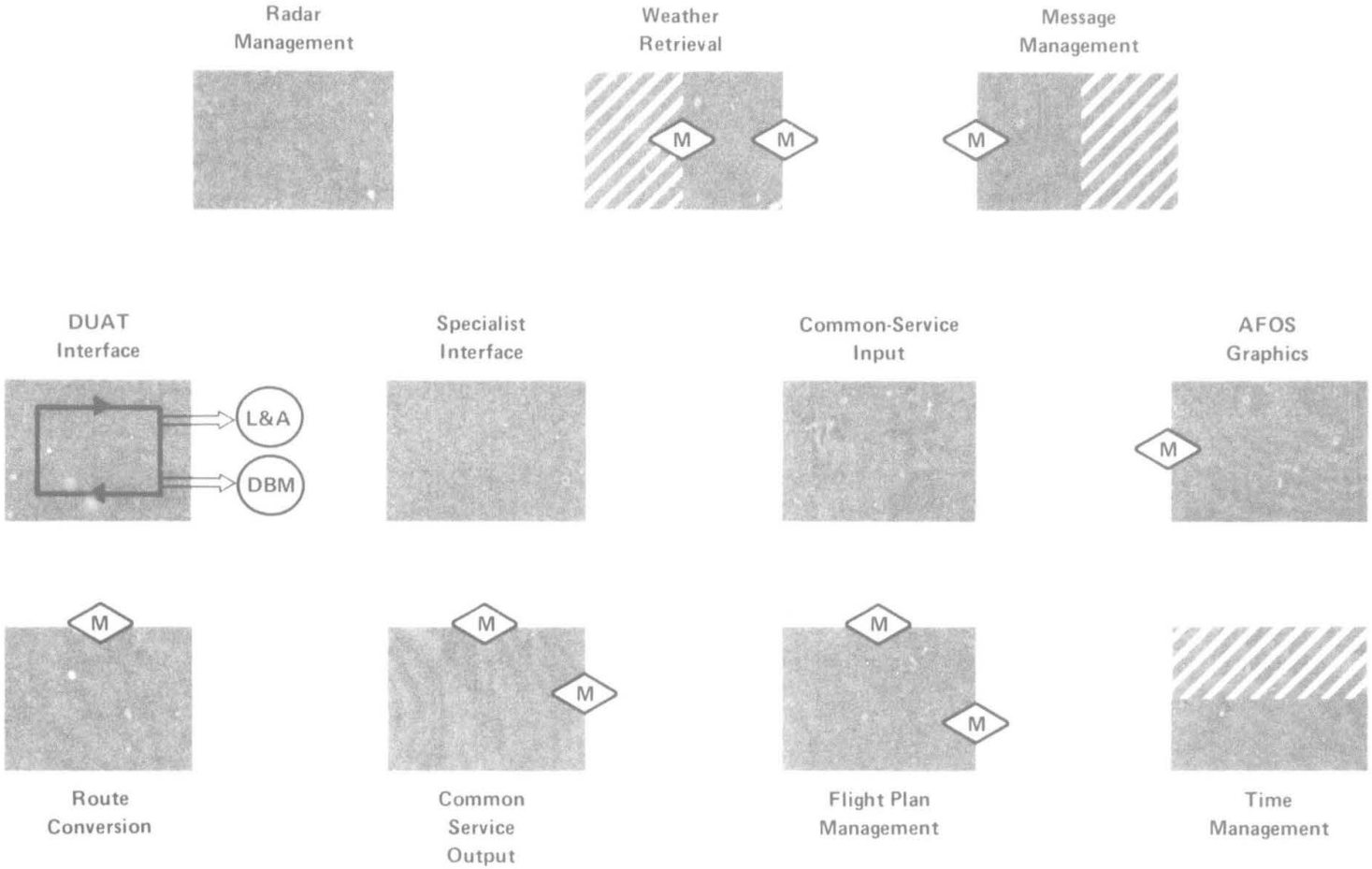


FIGURE 27
DUAT: DUAT FLIGHT LOG/START DUAT CONVERSATION WITH NEW FLIGHT PLAN
RESUME DUAT CONVERSATION WITH CURRENT FLIGHT PLAN
NON ROUTE DATA FOR DUAT FLIGHT PLAN

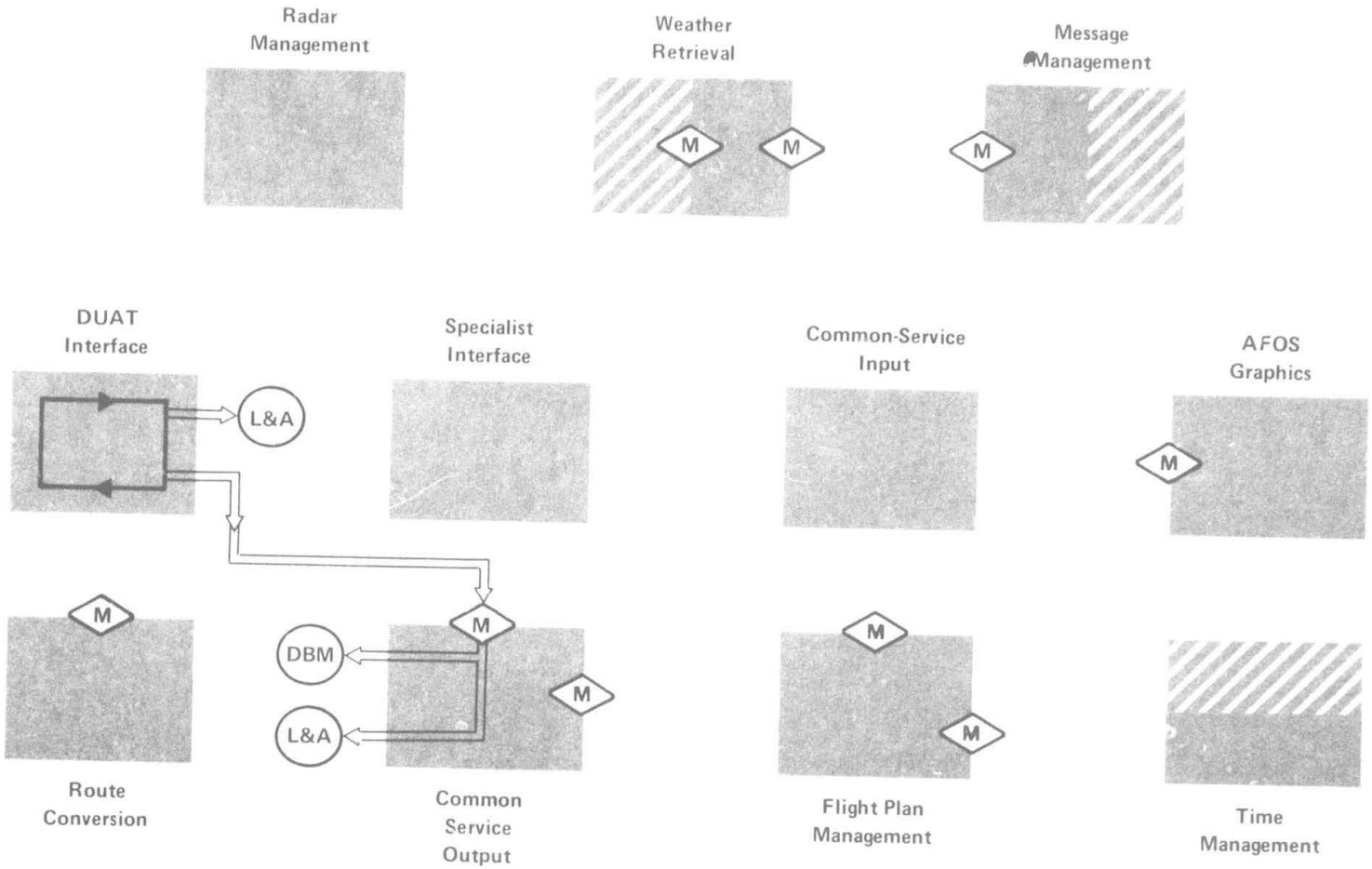


FIGURE 28
DUAT: EXECUTE DUAT LINK SEND

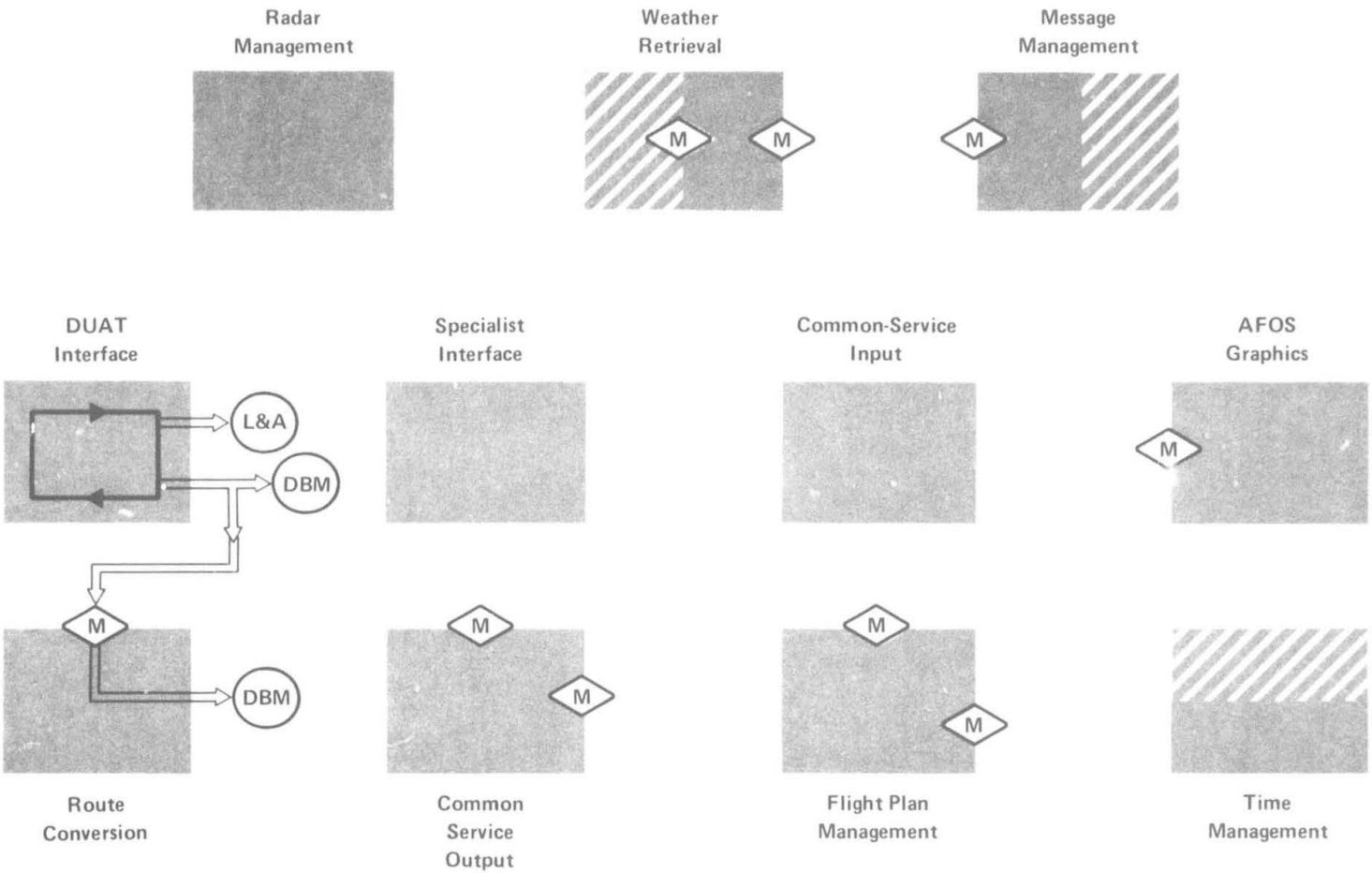


FIGURE 29
DUAT: ROUTE DATA FOR DUAT FLIGHT PLAN

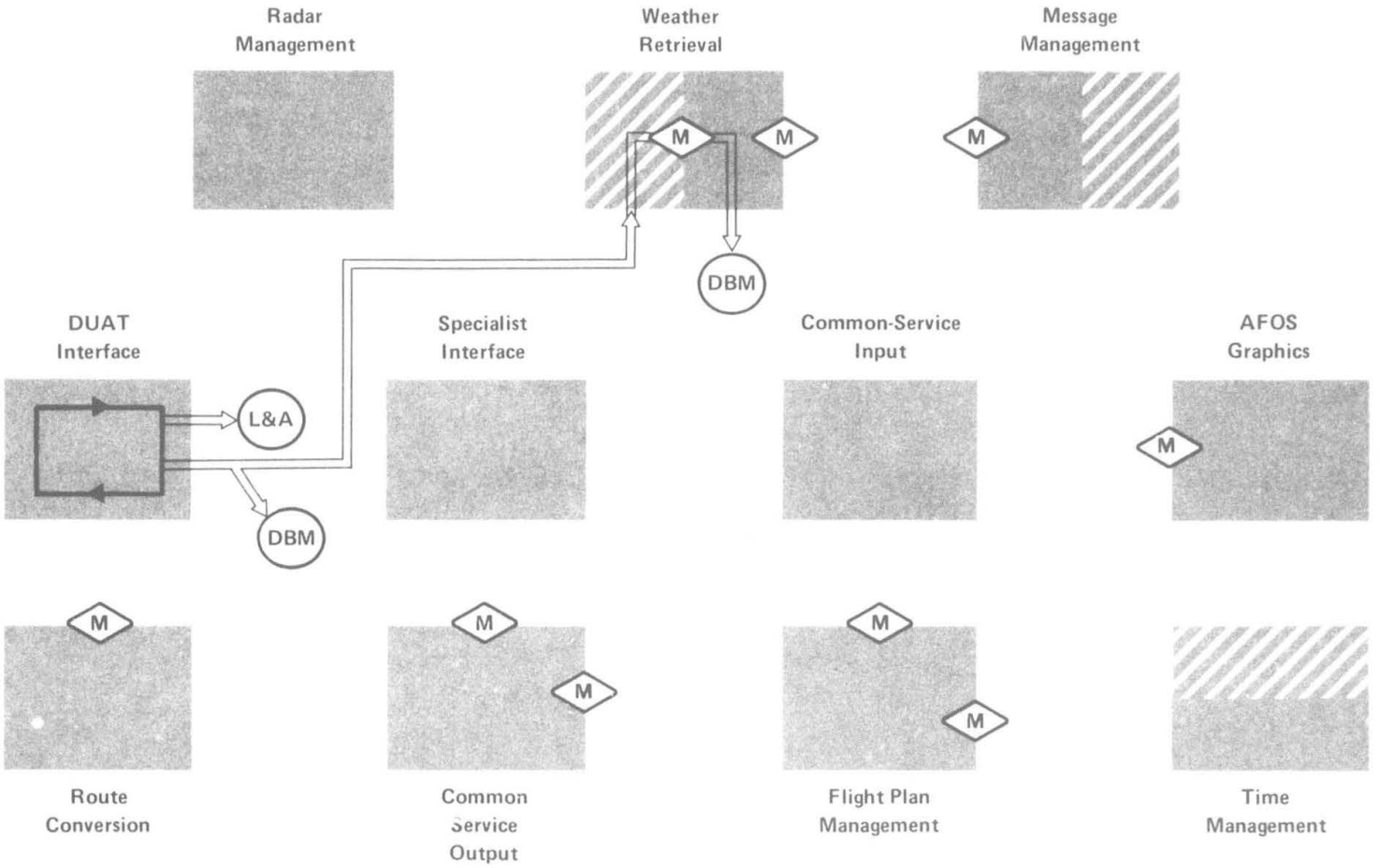


FIGURE 30
DUAT: REQUEST DUAT WEATHER

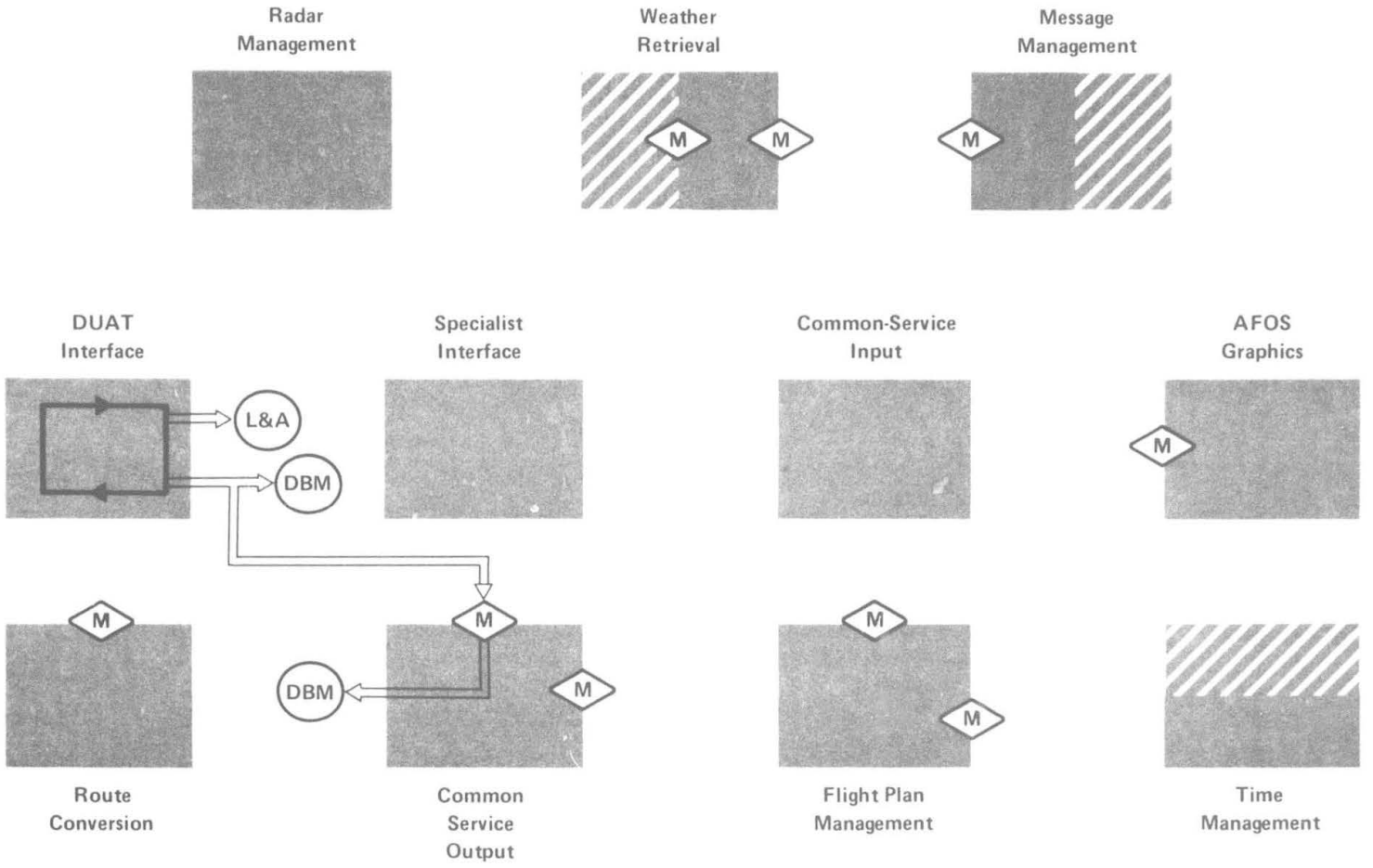


FIGURE 31
DUAT: START DUAT CONVERSATION WITH FILED FLIGHT PLAN

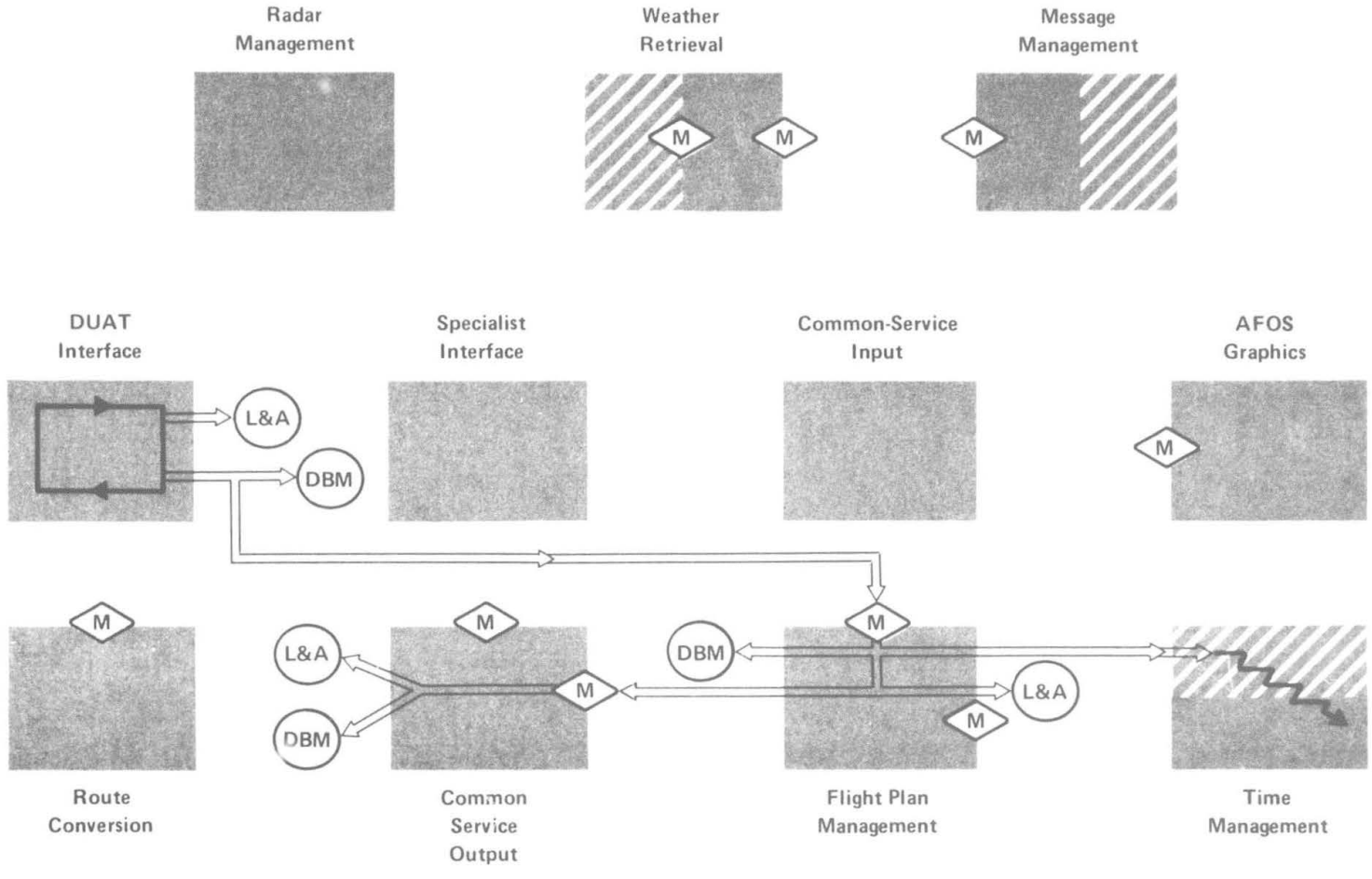


FIGURE 32
DUAT: FILE FLIGHT PLAN FROM DUAT CONVERSATION

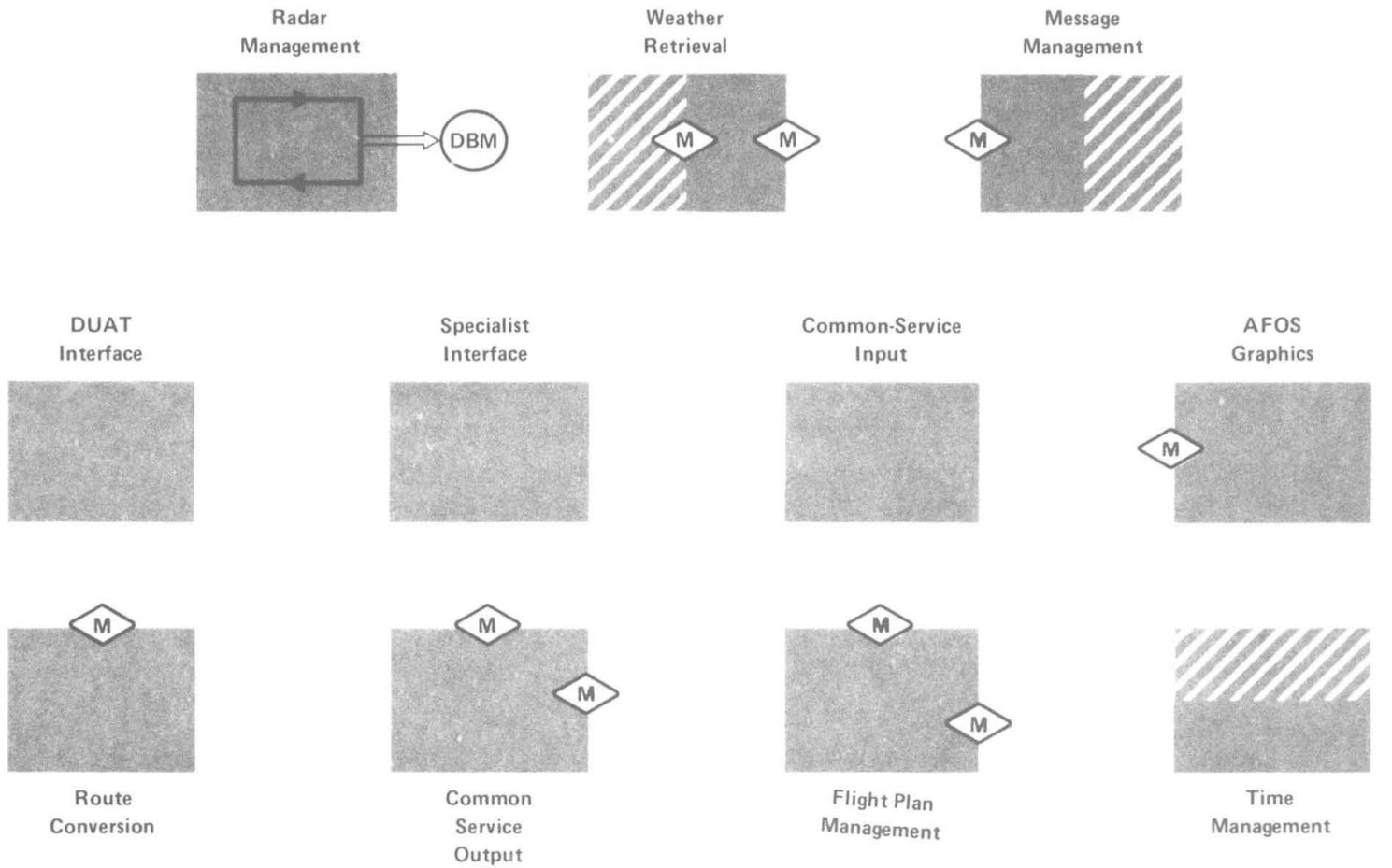


FIGURE 33
RADAR MANAGER

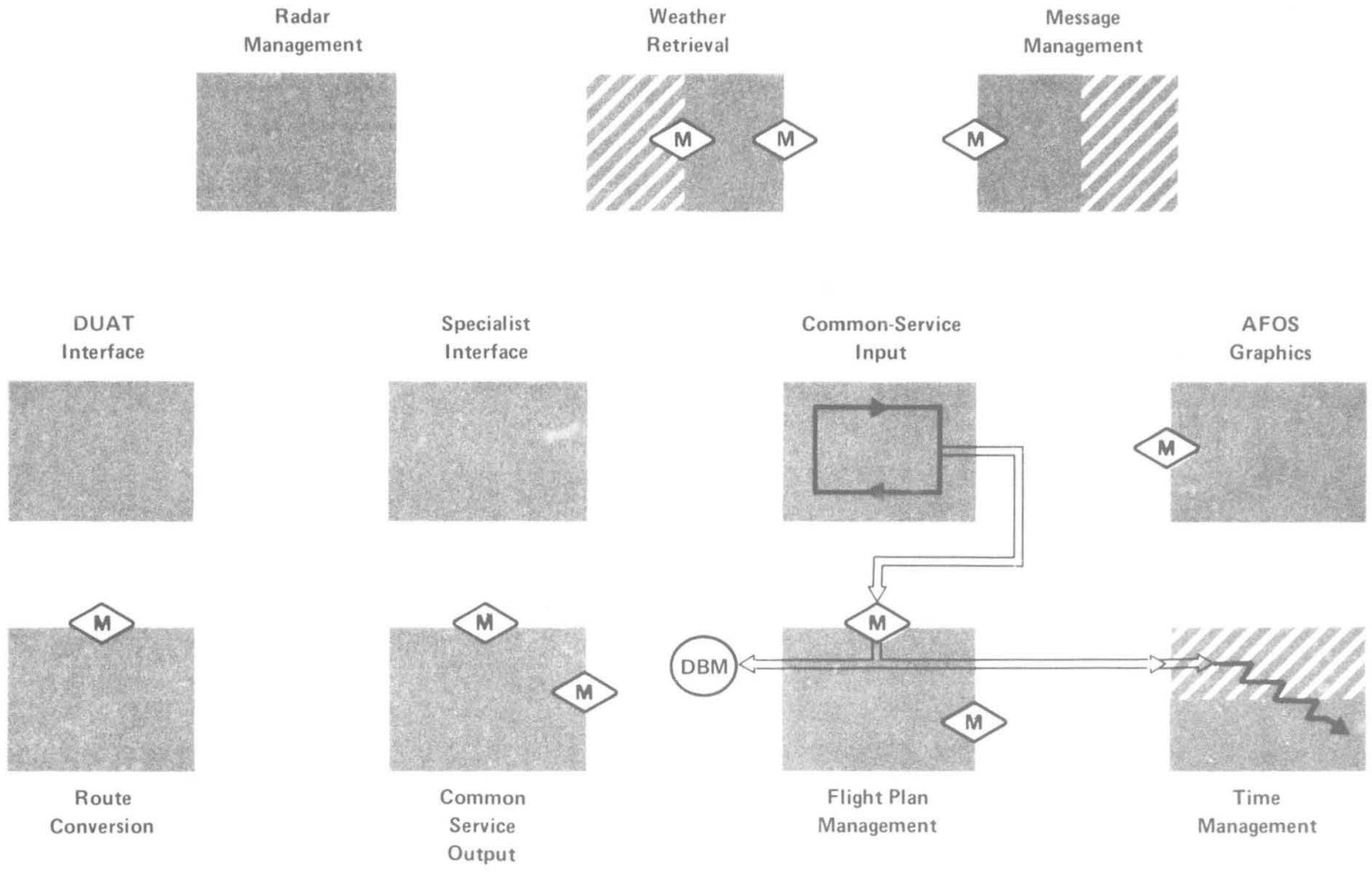


FIGURE 34
COMMON-SERVICE INPUT (INITIALIZATION PHASE): INITIALIZE RESCUE ALERTS
INITIALIZE FLIGHT PLAN DELETIONS

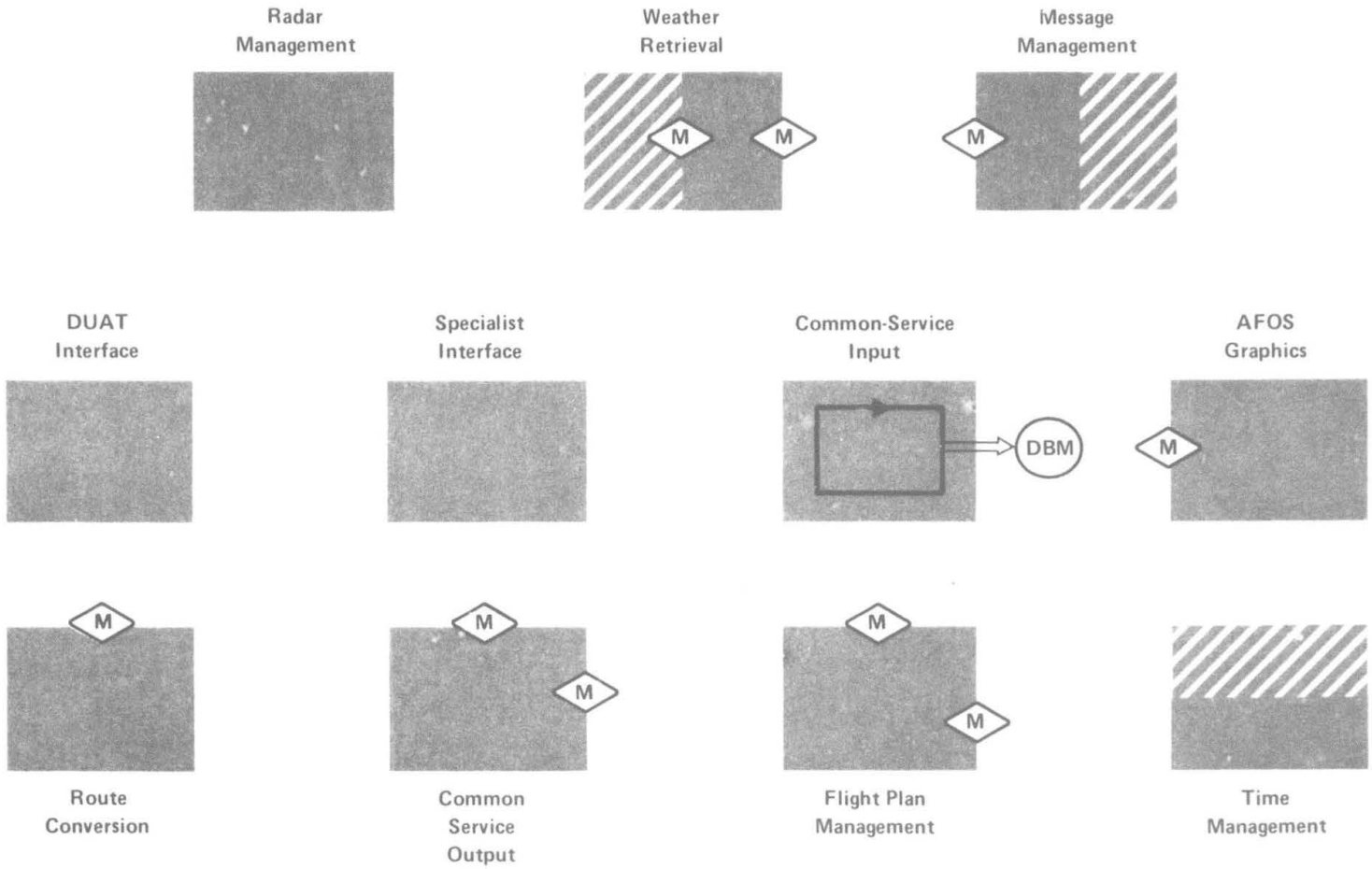


FIGURE 35
COMMON-SERVICE INPUT: STANDARD INITIALIZATION OF FILES

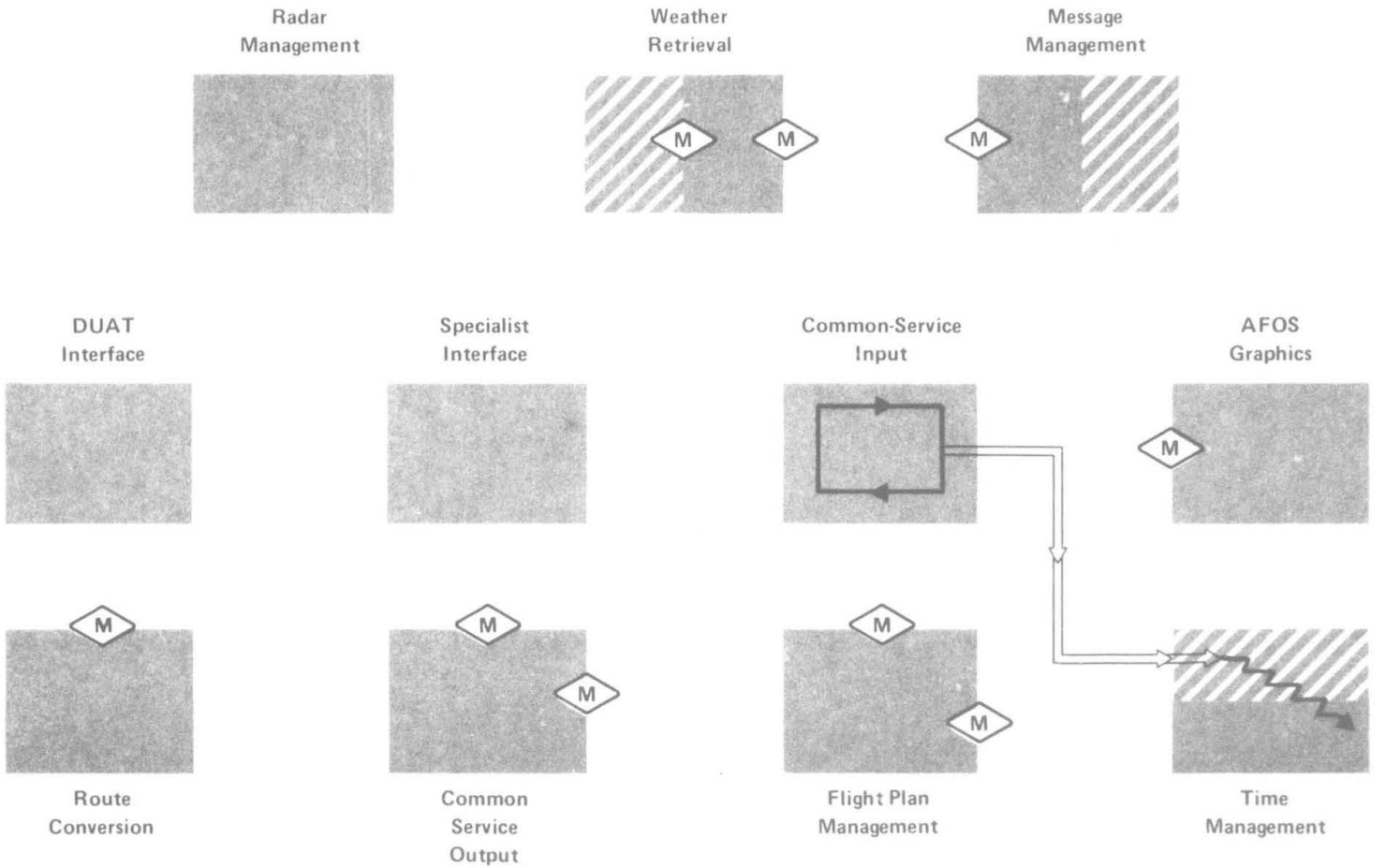
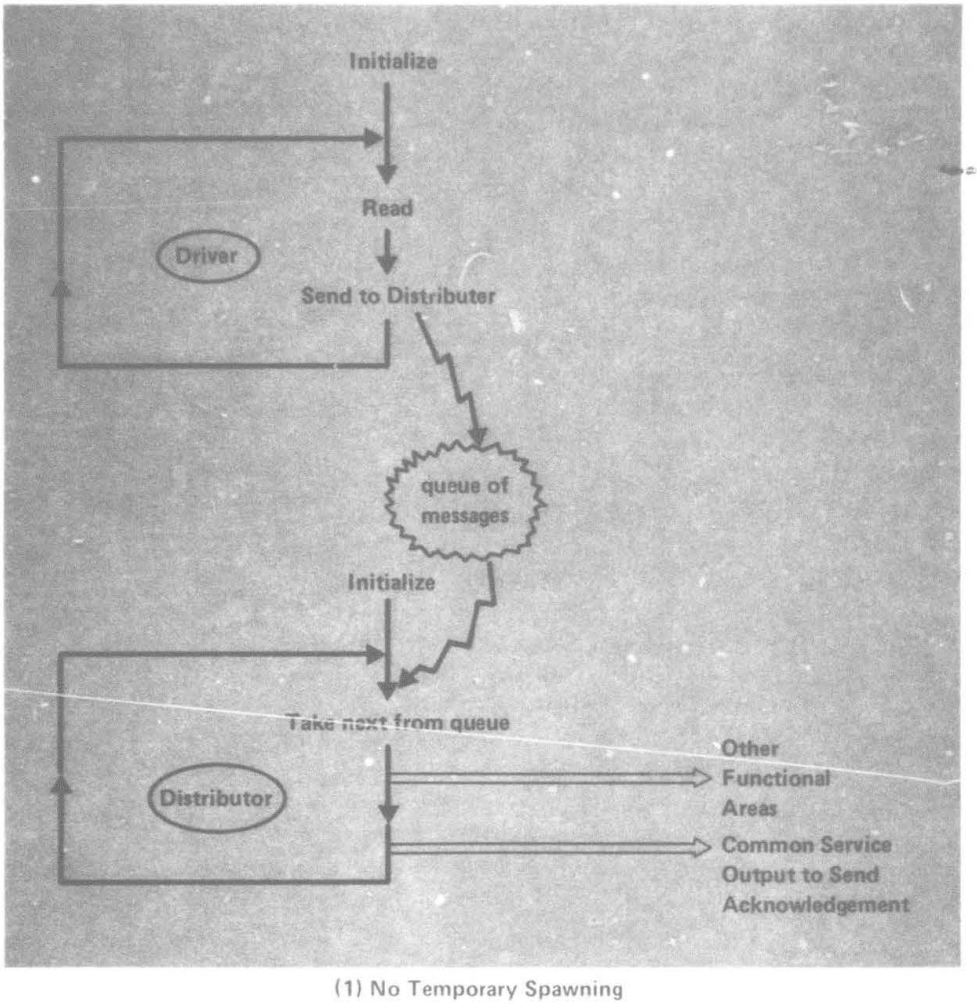
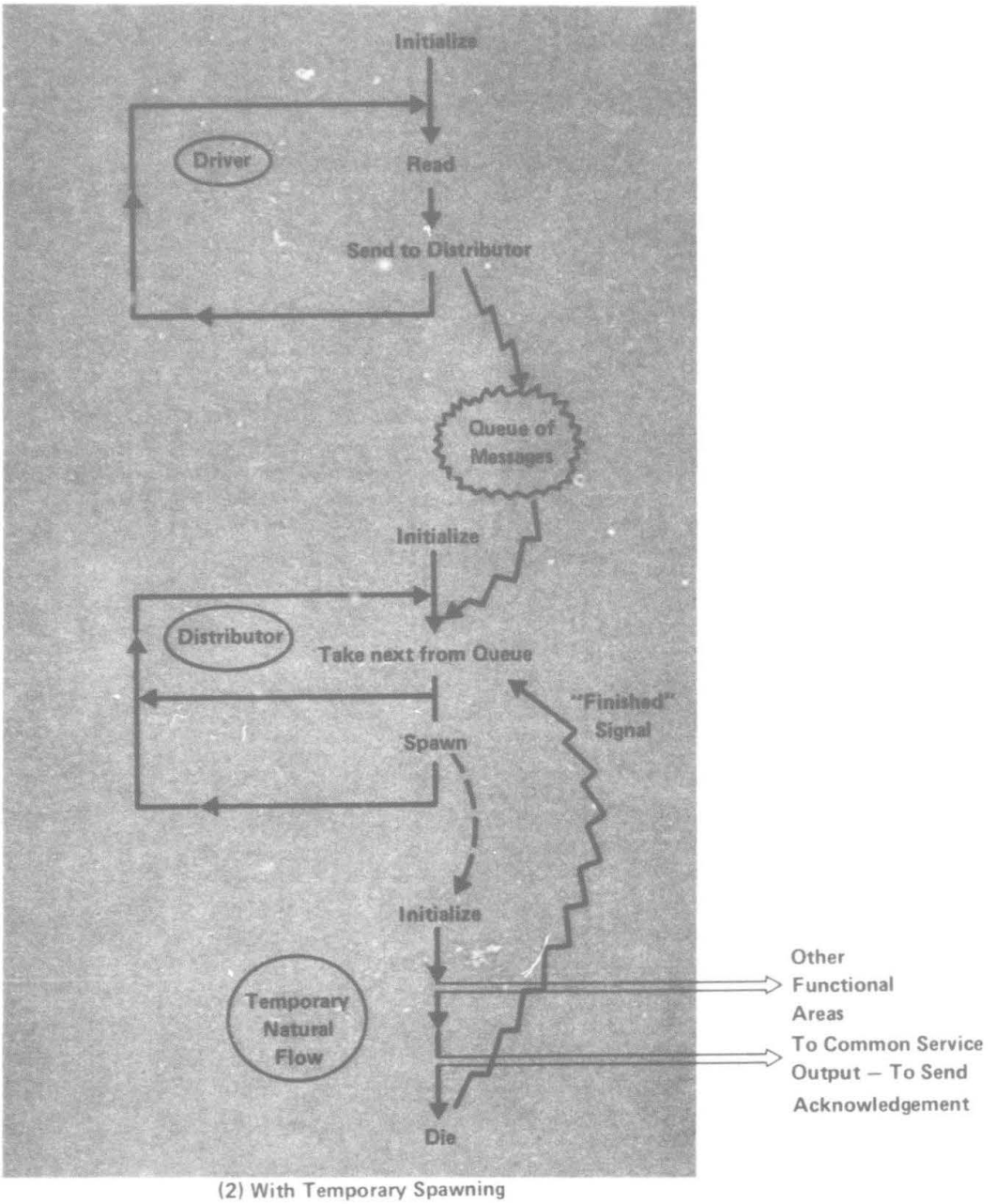


FIGURE 36
COMMON-SERVICE INPUT: SCHEDULE END OF SCHEDULING



(1) No Temporary Spawning

FIGURE 37
COMMON SERVICE INPUT



(2) With Temporary Spawning

FIGURE 38
COMMON-SERVICE INPUT

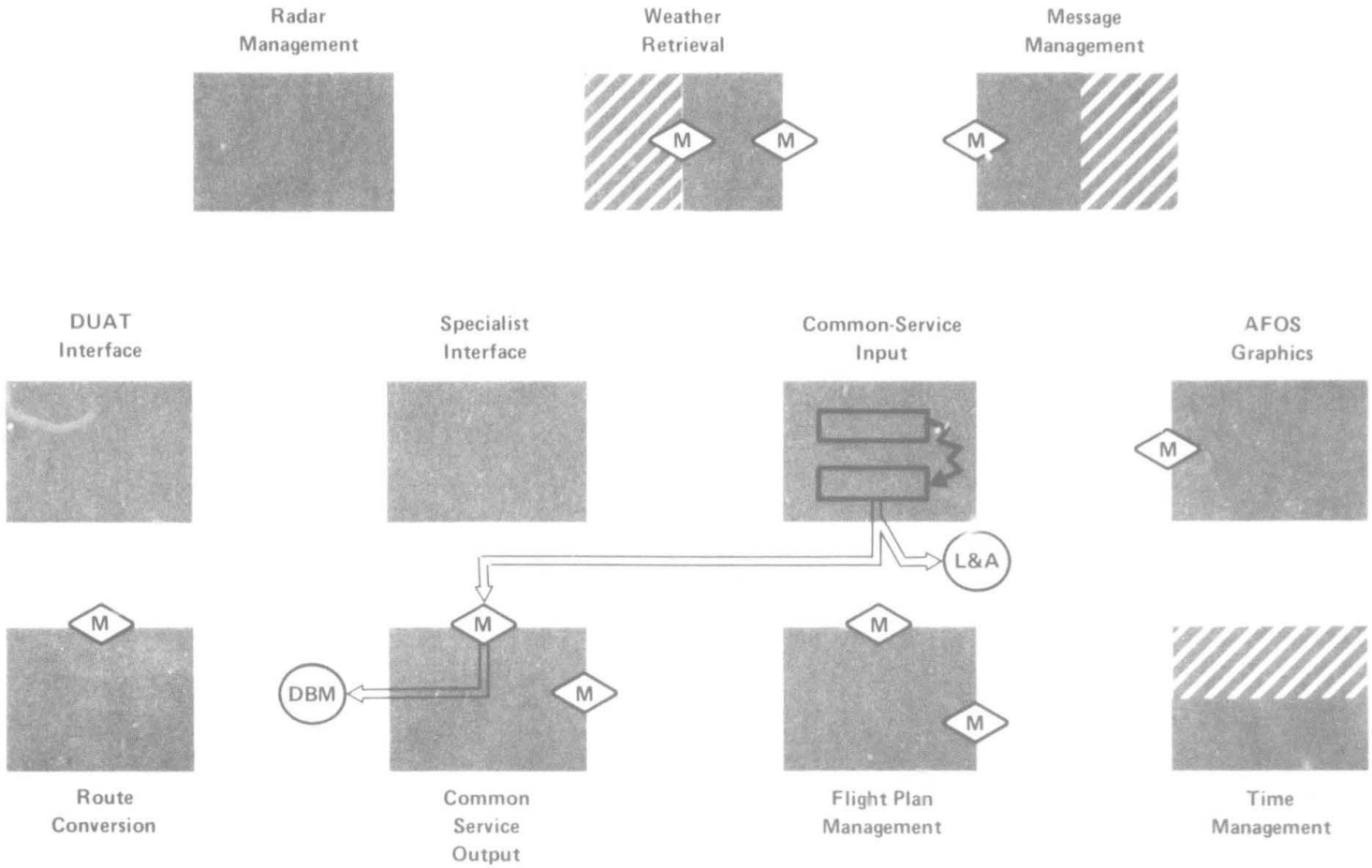


FIGURE 39
COMMON-SERVICE INPUT: ACKNOWLEDGEMENT FROM ANOTHER COMPUTER

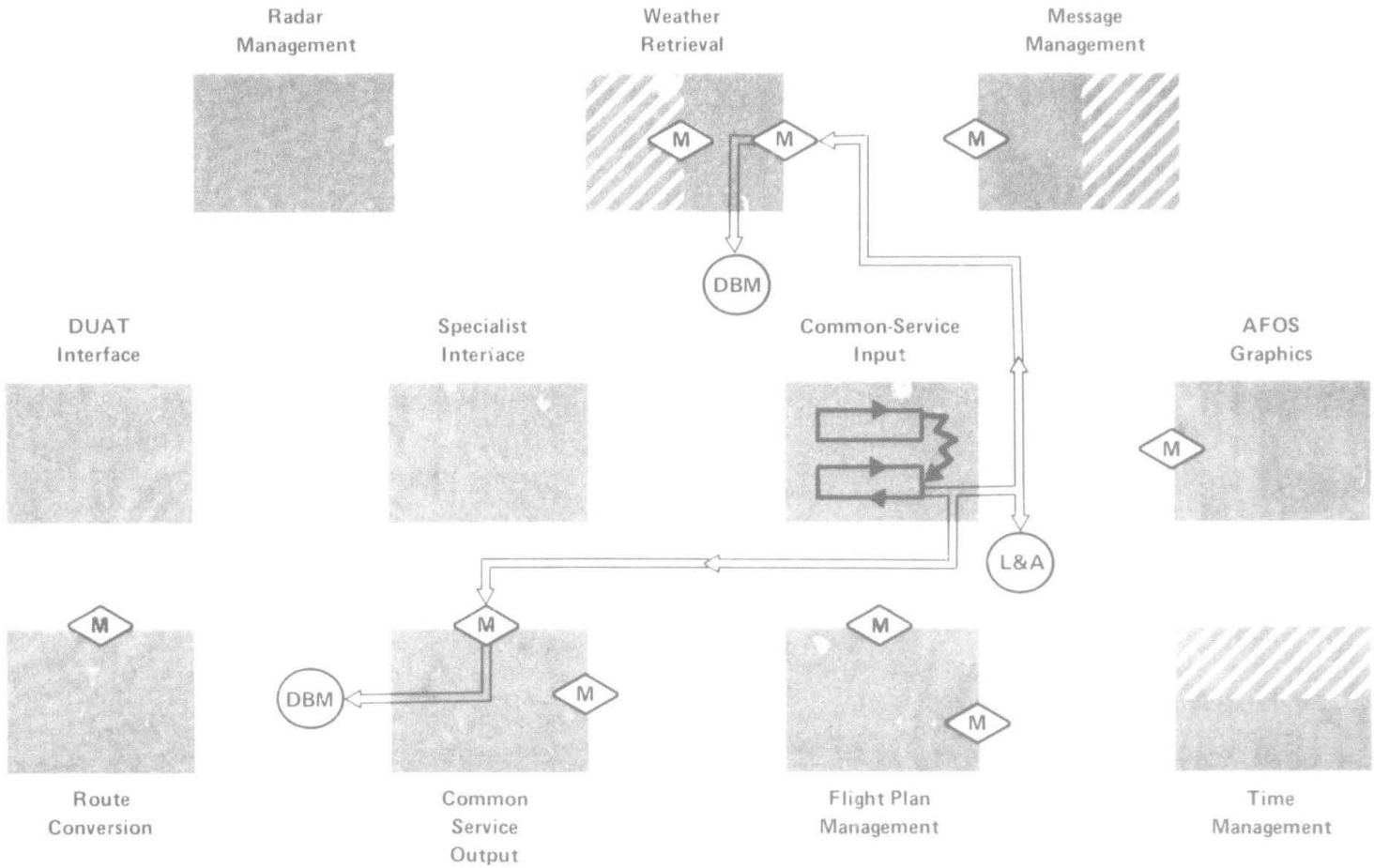


FIGURE 40
COMMON-SERVICE INPUT: INDIVIDUAL UPDATE FROM AWP
HOURLY UPDATE FROM AWP

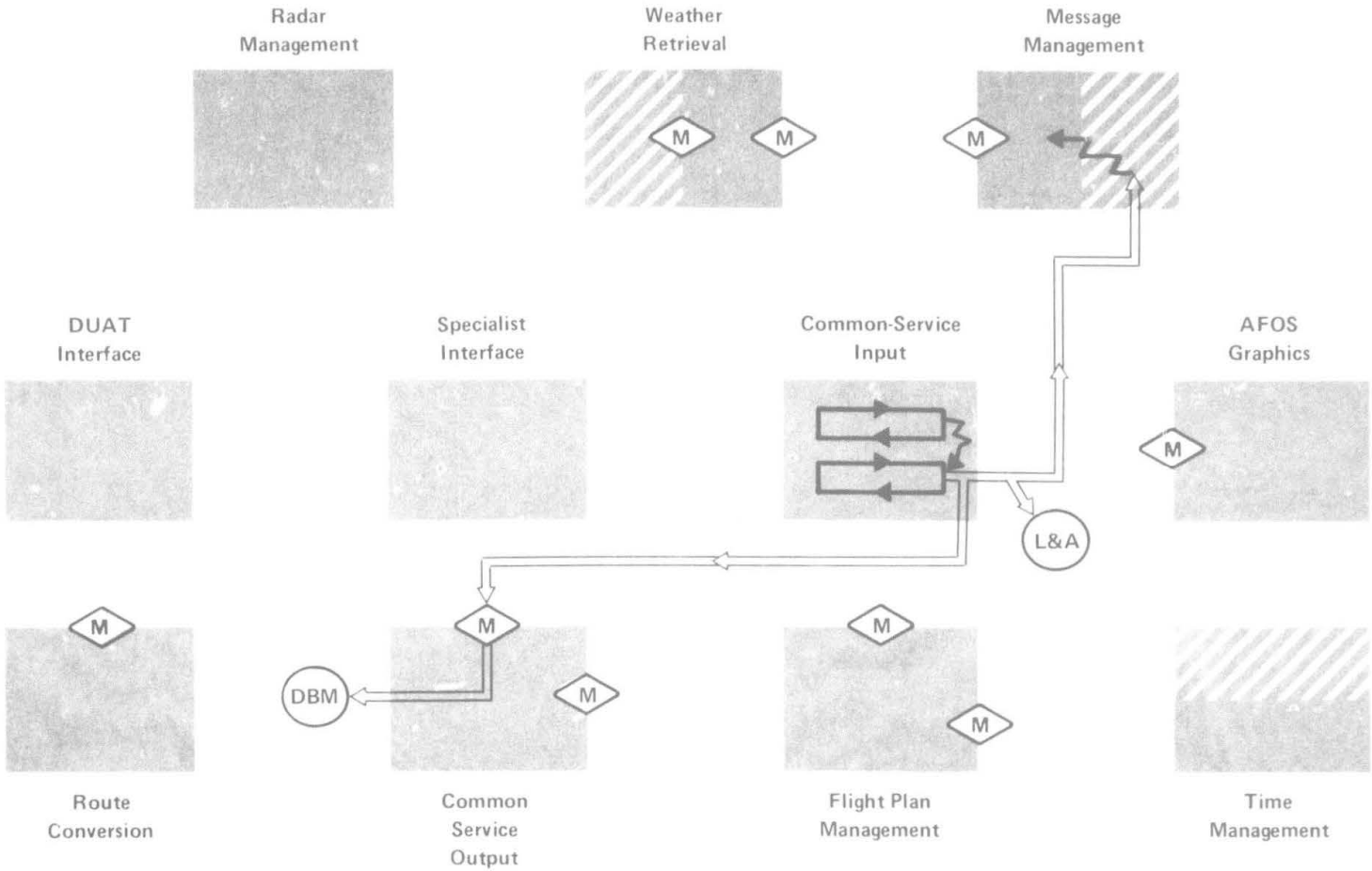


FIGURE 41
COMMON-SERVICE INPUT: MAIL MESSAGE

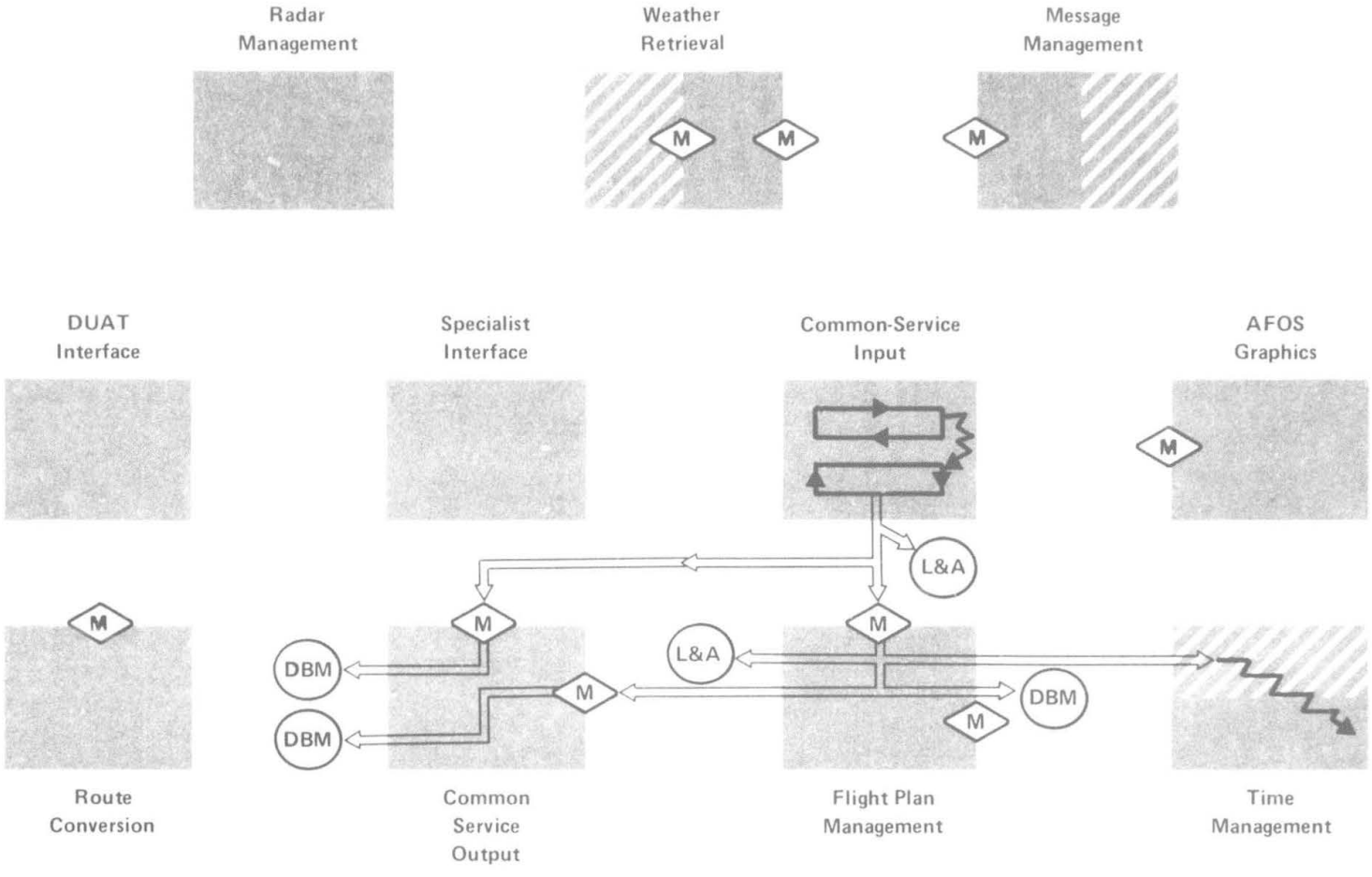


FIGURE 42
COMMON-SERVICE INPUT: FLIGHT PLAN FROM ANOTHER COMPUTER

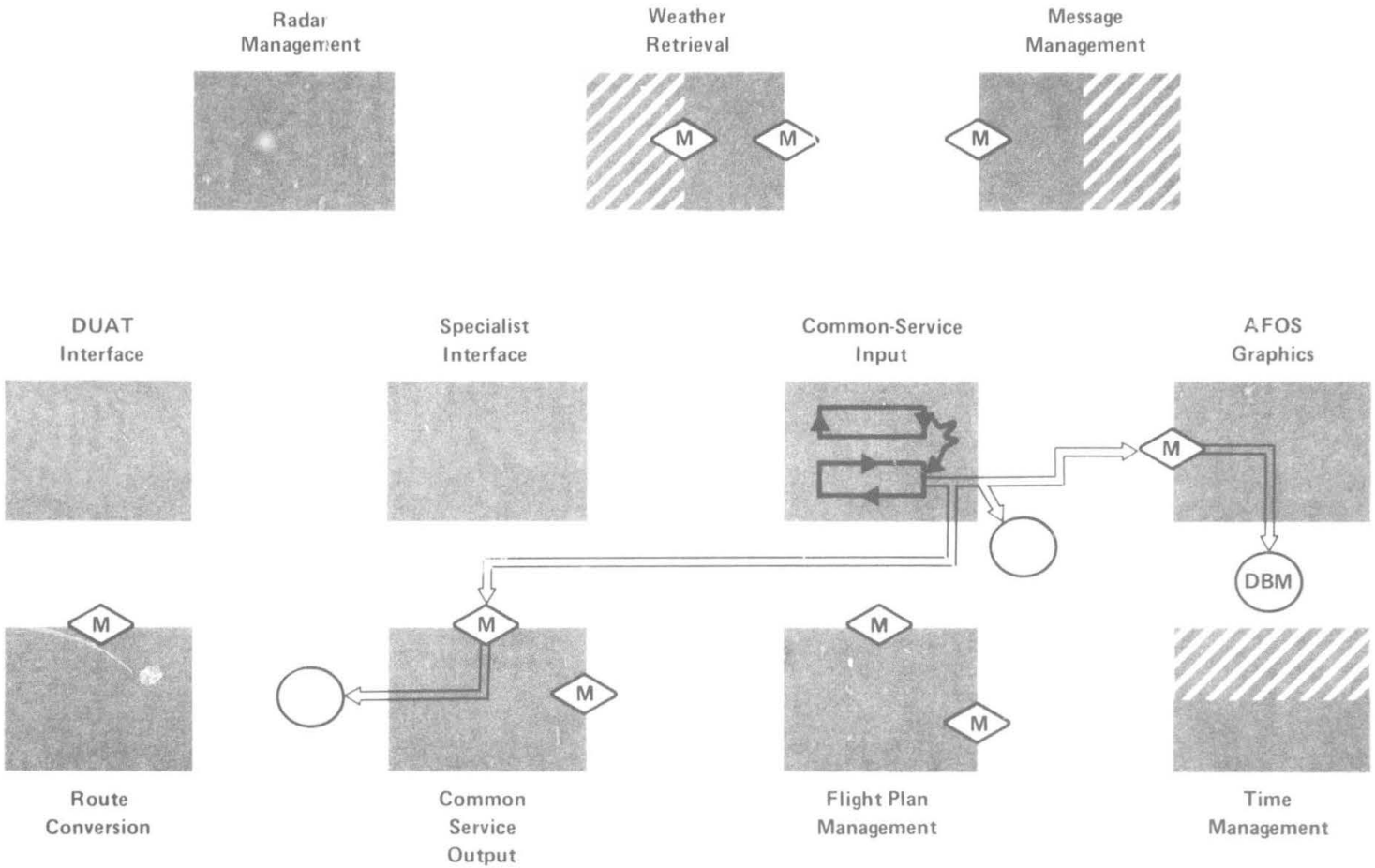


FIGURE 43
COMMON-SERVICE INPUT: AFOS GRAPHICS FROM AWP

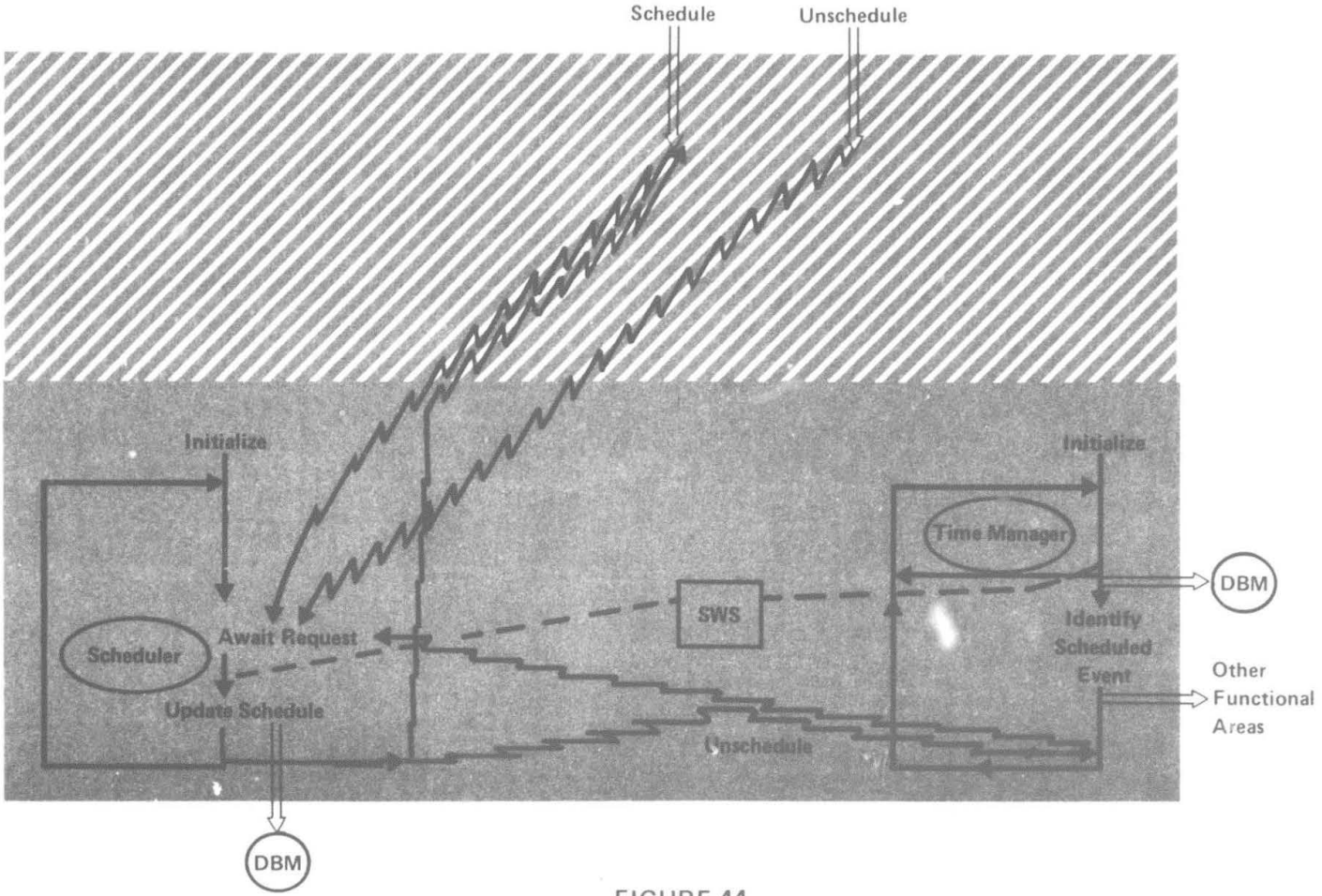


FIGURE 44
TIME MANAGEMENT FUNCTIONAL AREA

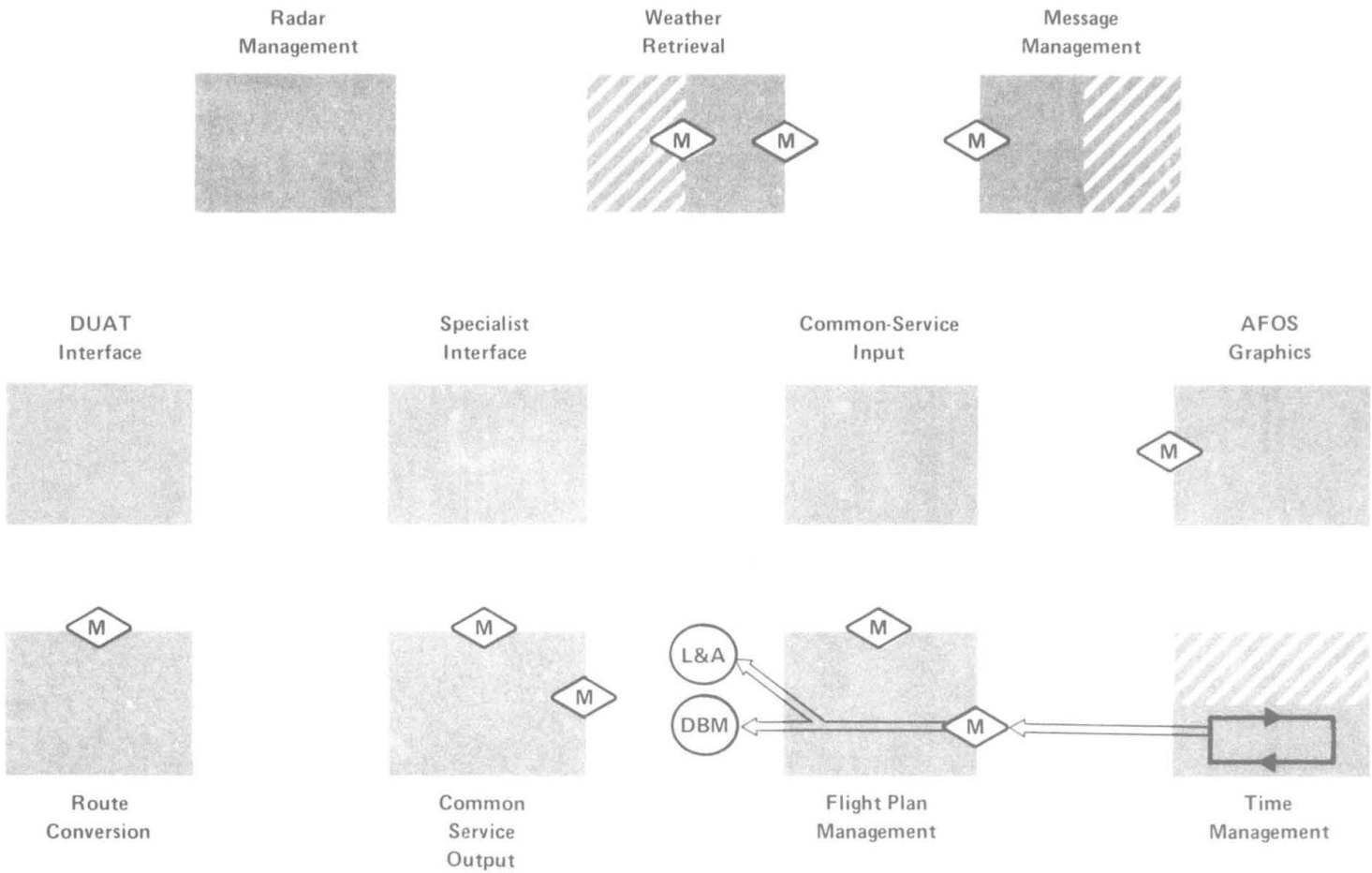


FIGURE 45
TIME MANAGER: DELETE FLIGHT PLAN

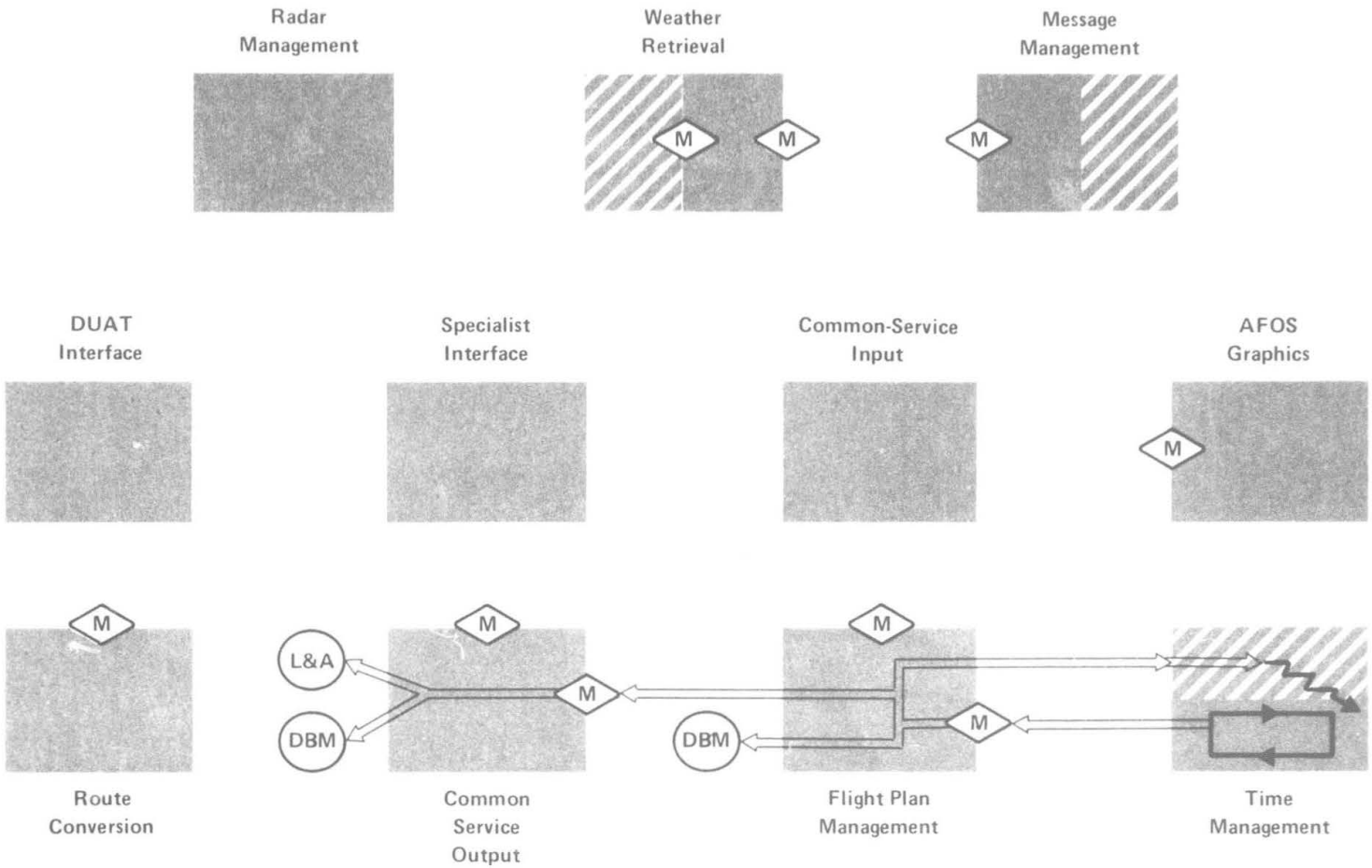


FIGURE 46
TIME MANAGER: TRANSMIT FLIGHT PLAN

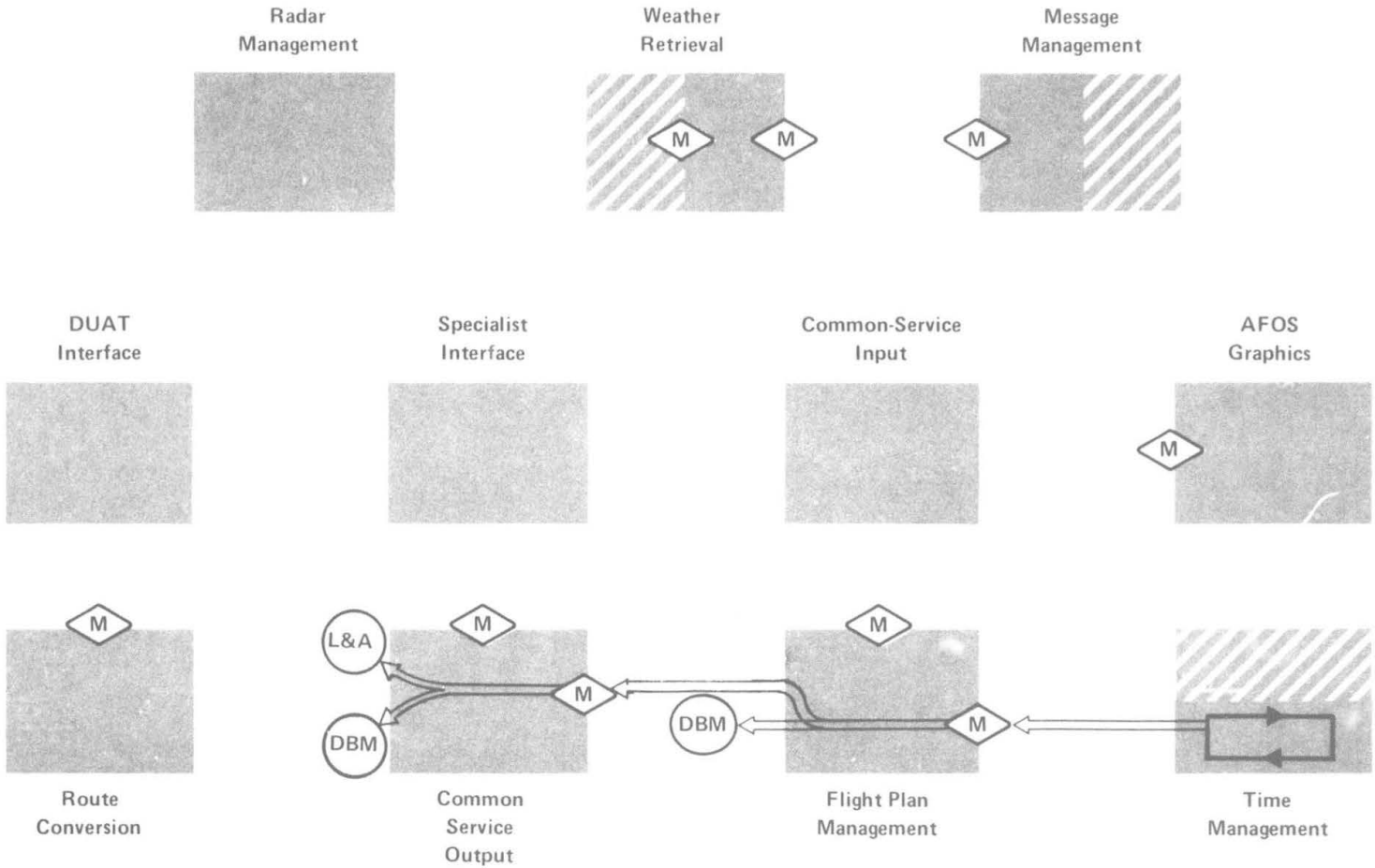


FIGURE 47
TIME MANAGER: START RESCUE OPERATIONS

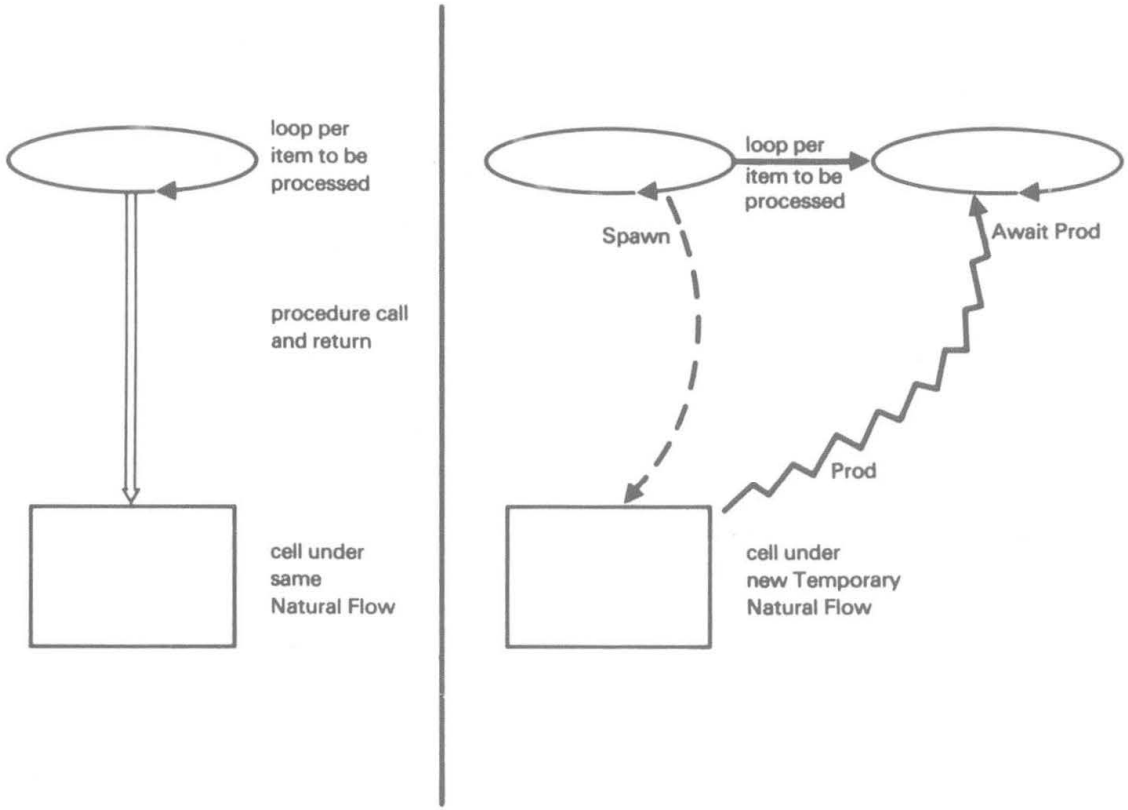


FIGURE 48
ITERATED INVOCATIONS OF CELLS (LEFT) SCHEDULED IN SERIES, AND (RIGHT) SCHEDULED IN PARALLEL

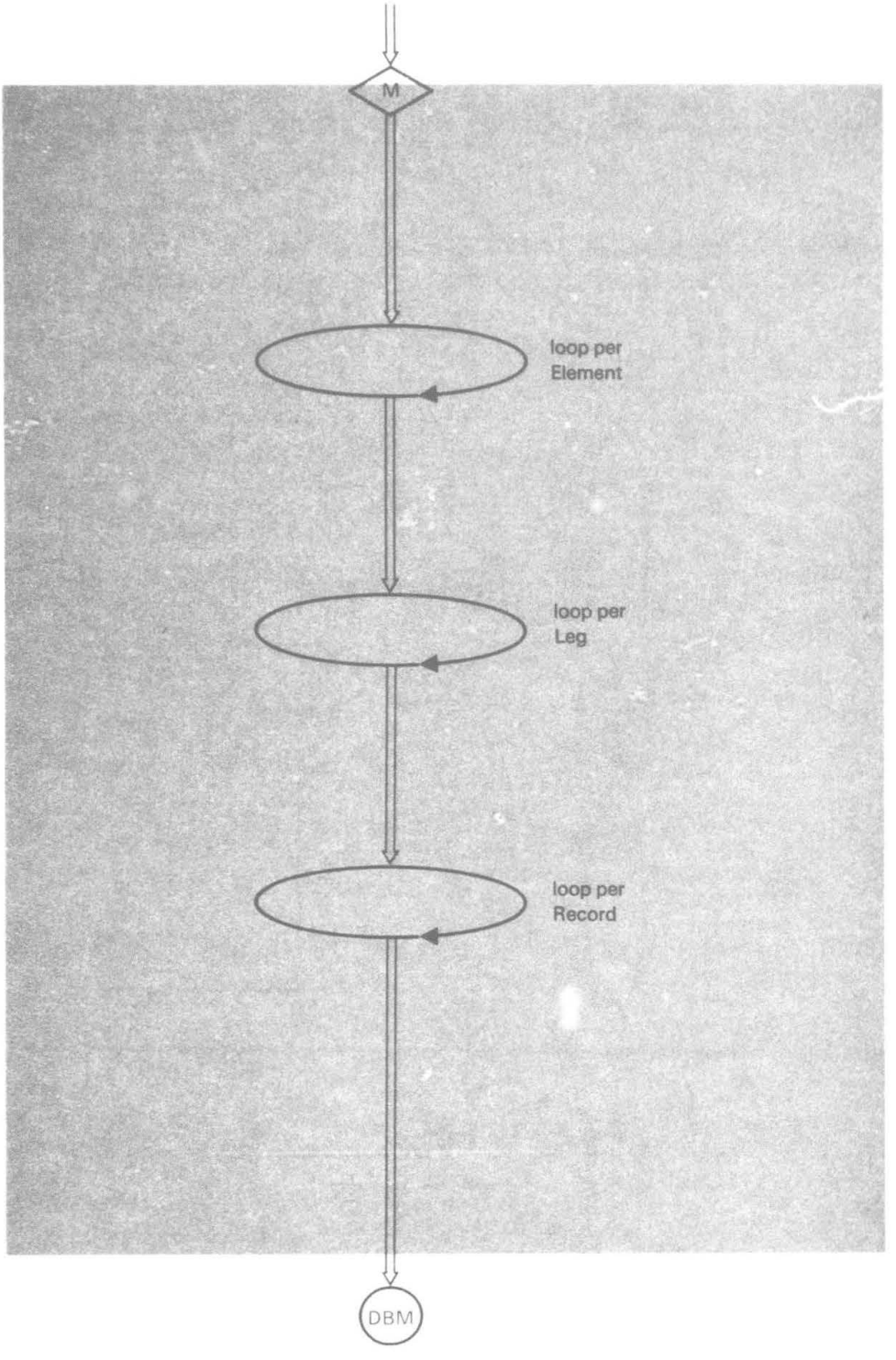


FIGURE 49
ROUTE CONVERSION WITH
NO TEMPORARY SPAWNING

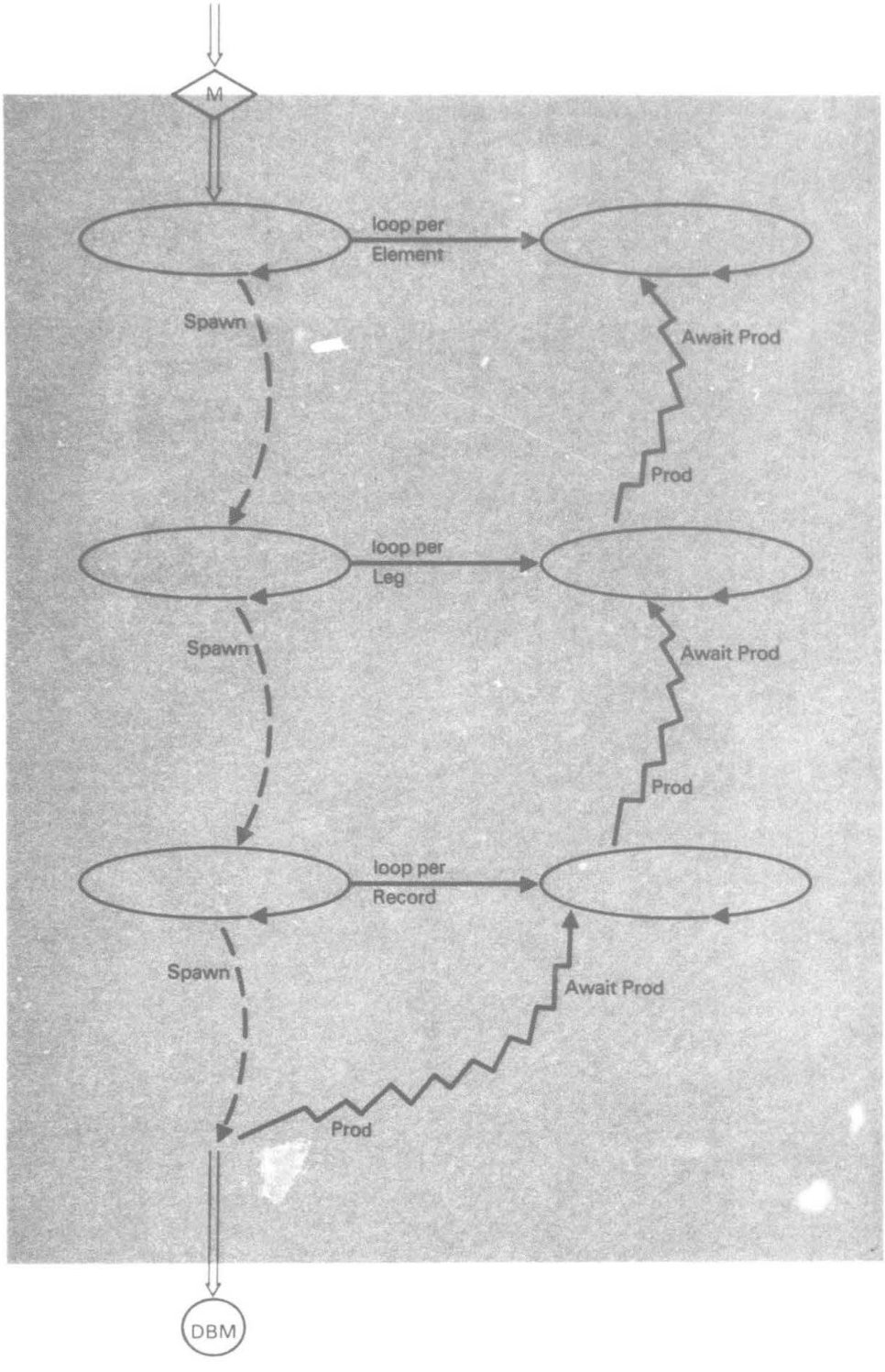


FIGURE 50
ROUTE CONVERSION WITH
MAXIMUM TEMPORARY SPAWNING

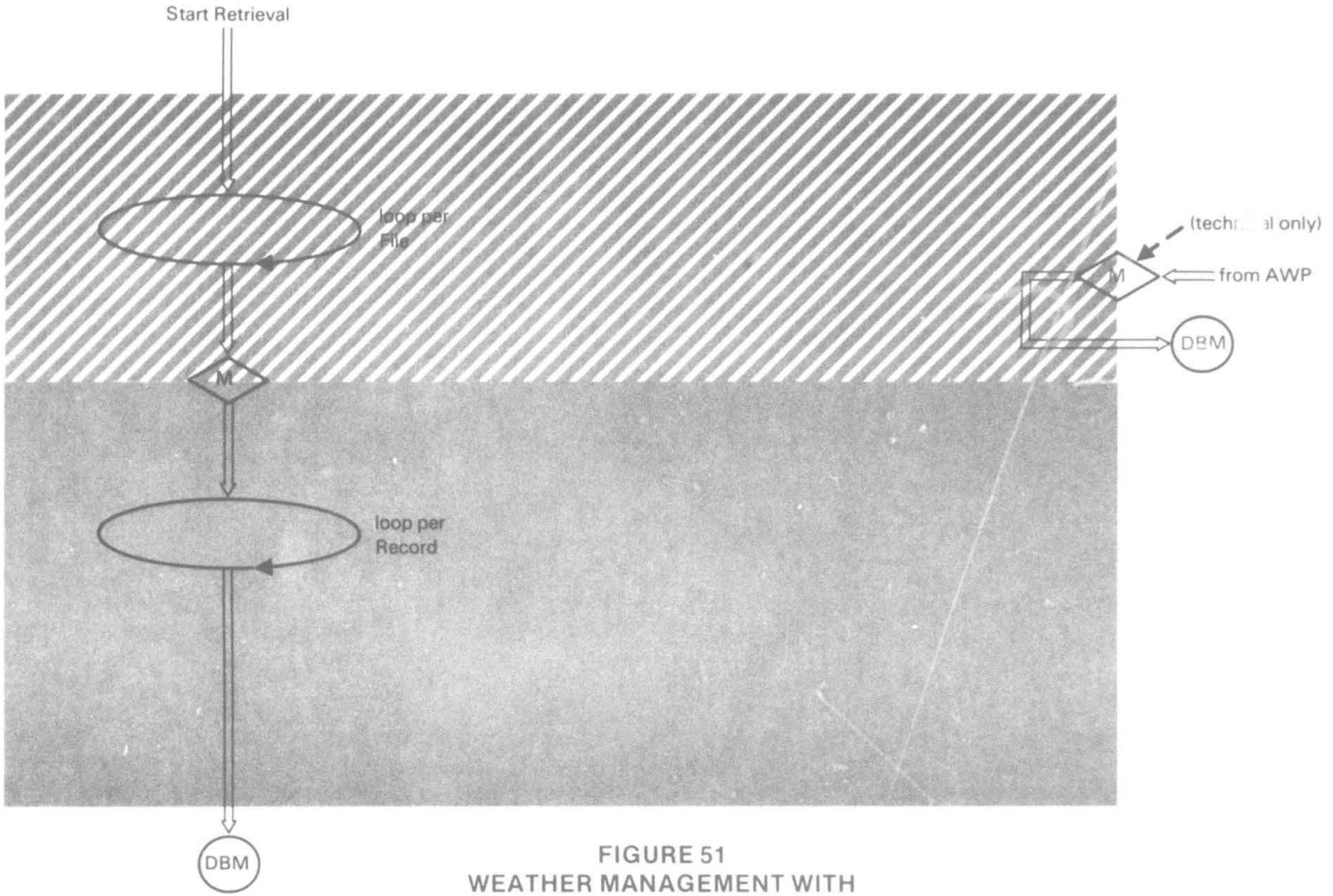


FIGURE 51
WEATHER MANAGEMENT WITH
NO TEMPORARY SPAWNING

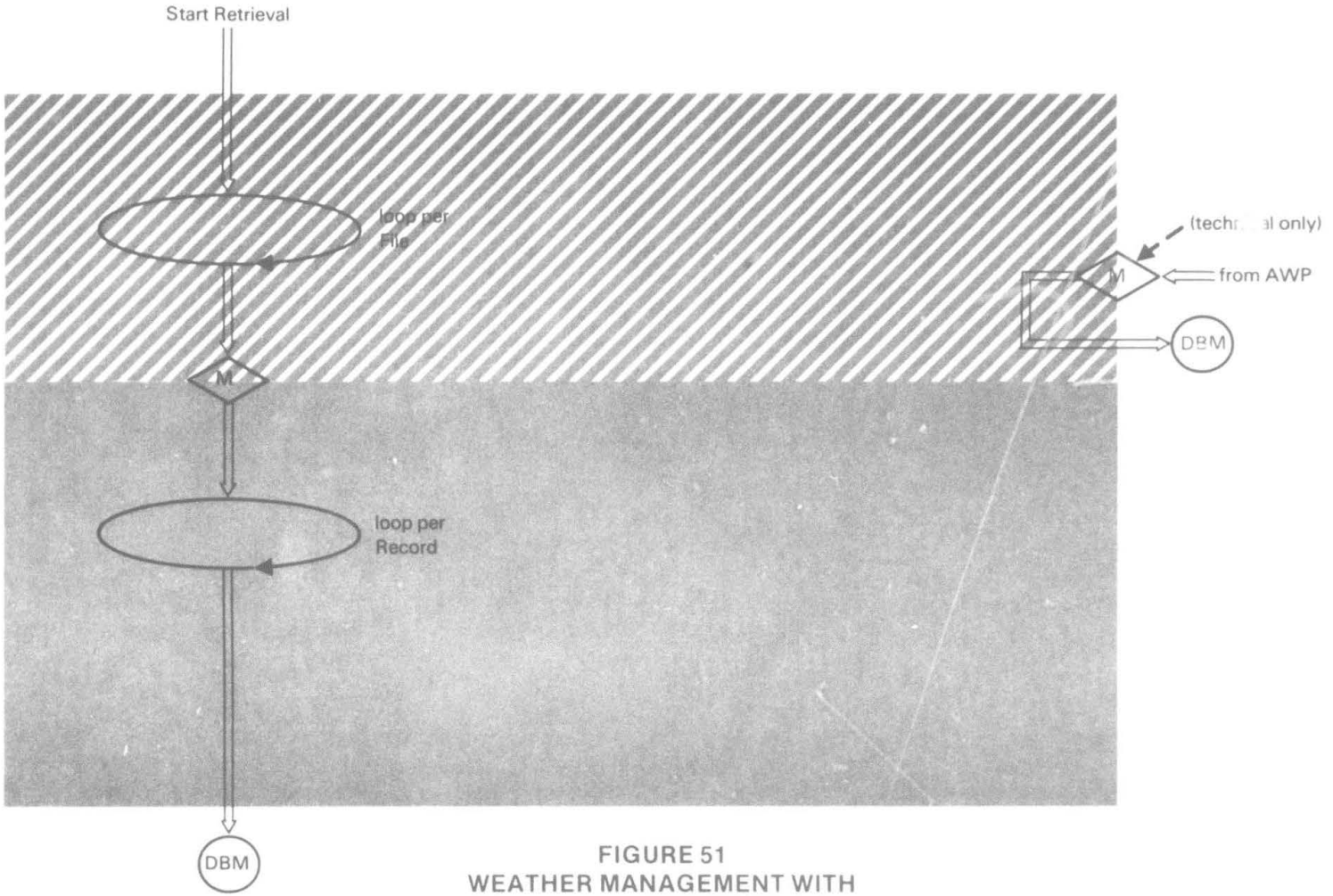


FIGURE 51
WEATHER MANAGEMENT WITH
NO TEMPORARY SPAWNING

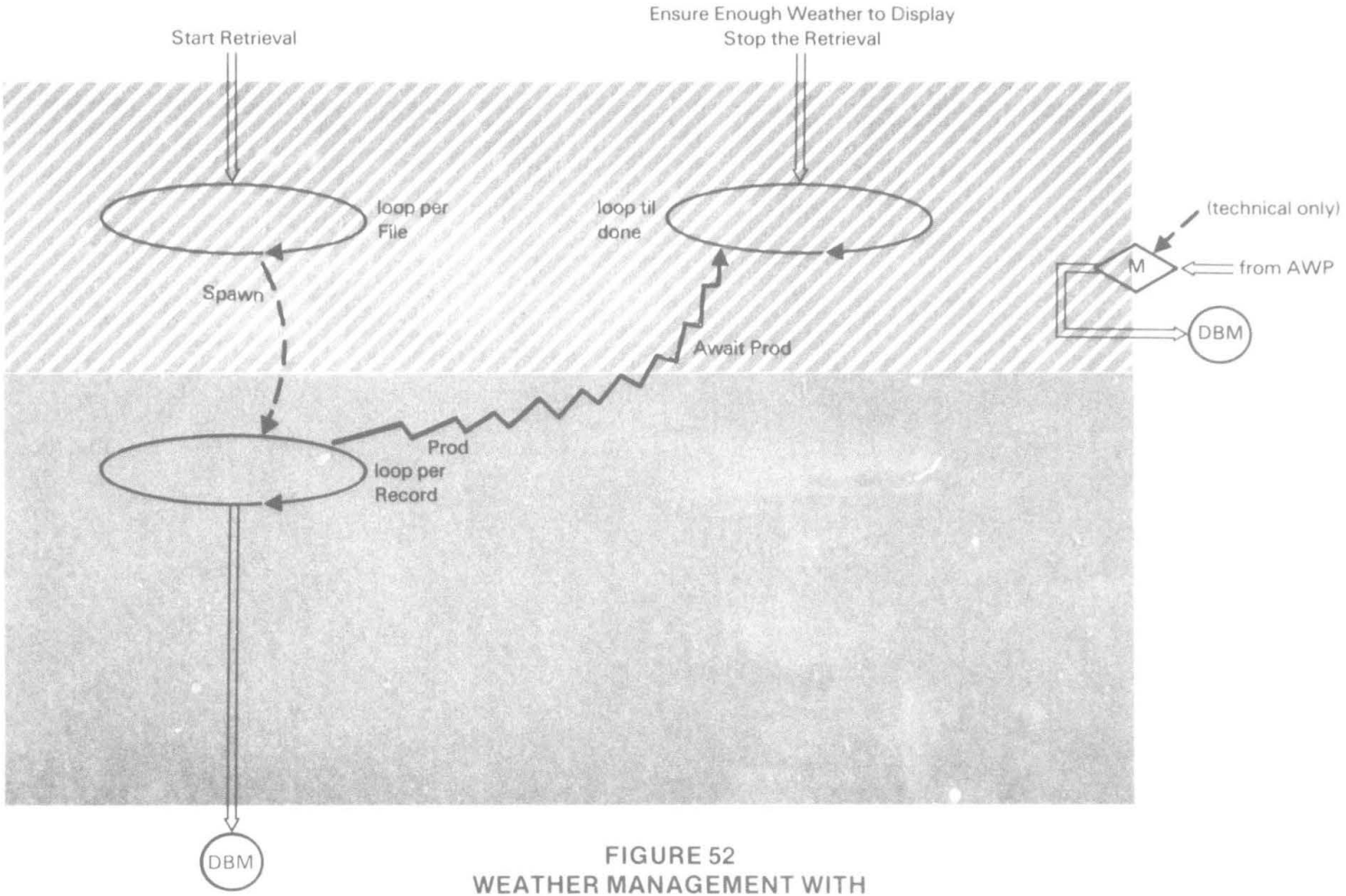


FIGURE 52
WEATHER MANAGEMENT WITH
SOME TEMPORARY SPAWNING

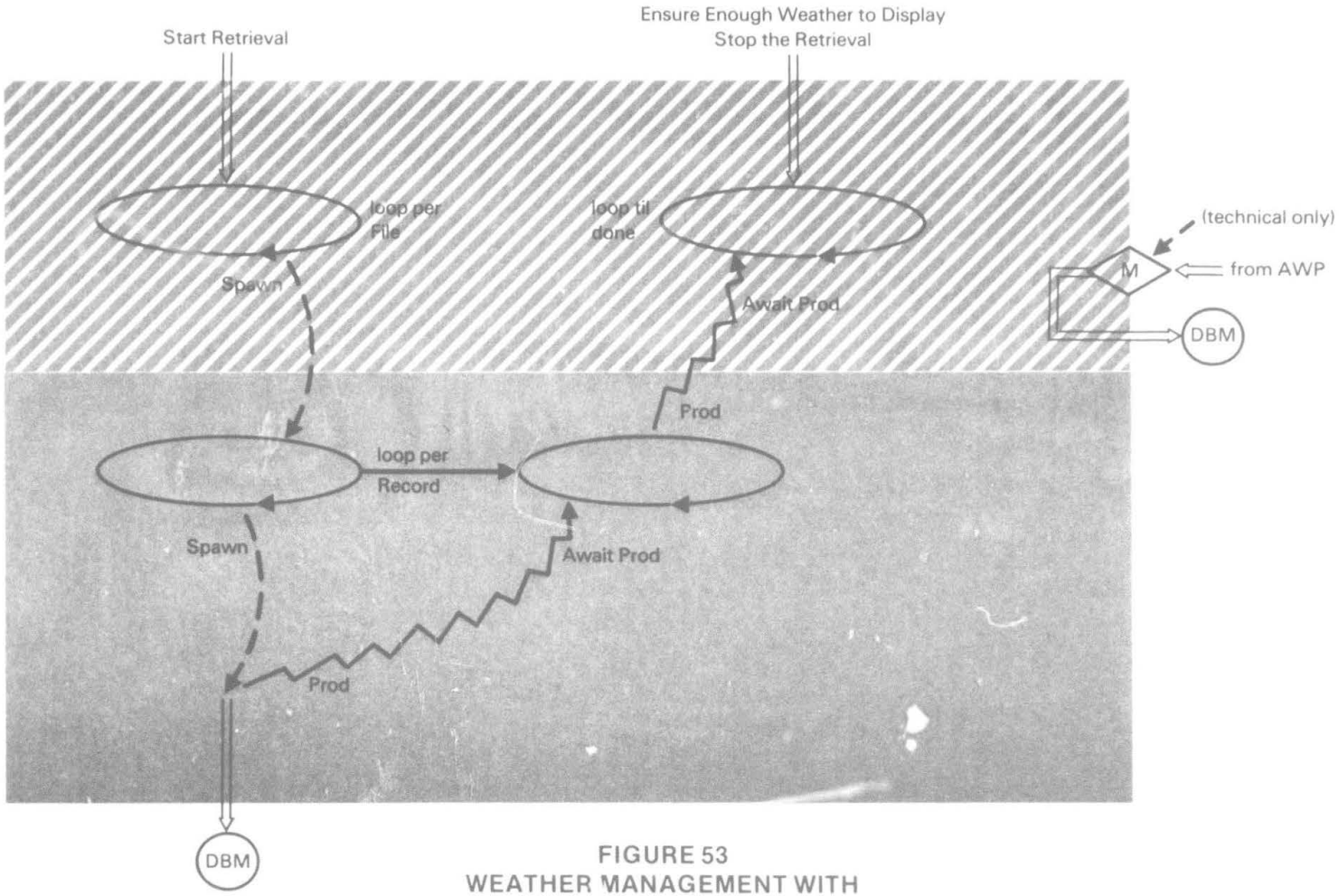


FIGURE 53
WEATHER MANAGEMENT WITH
MAXIMUM TEMPORARY SPAWNING

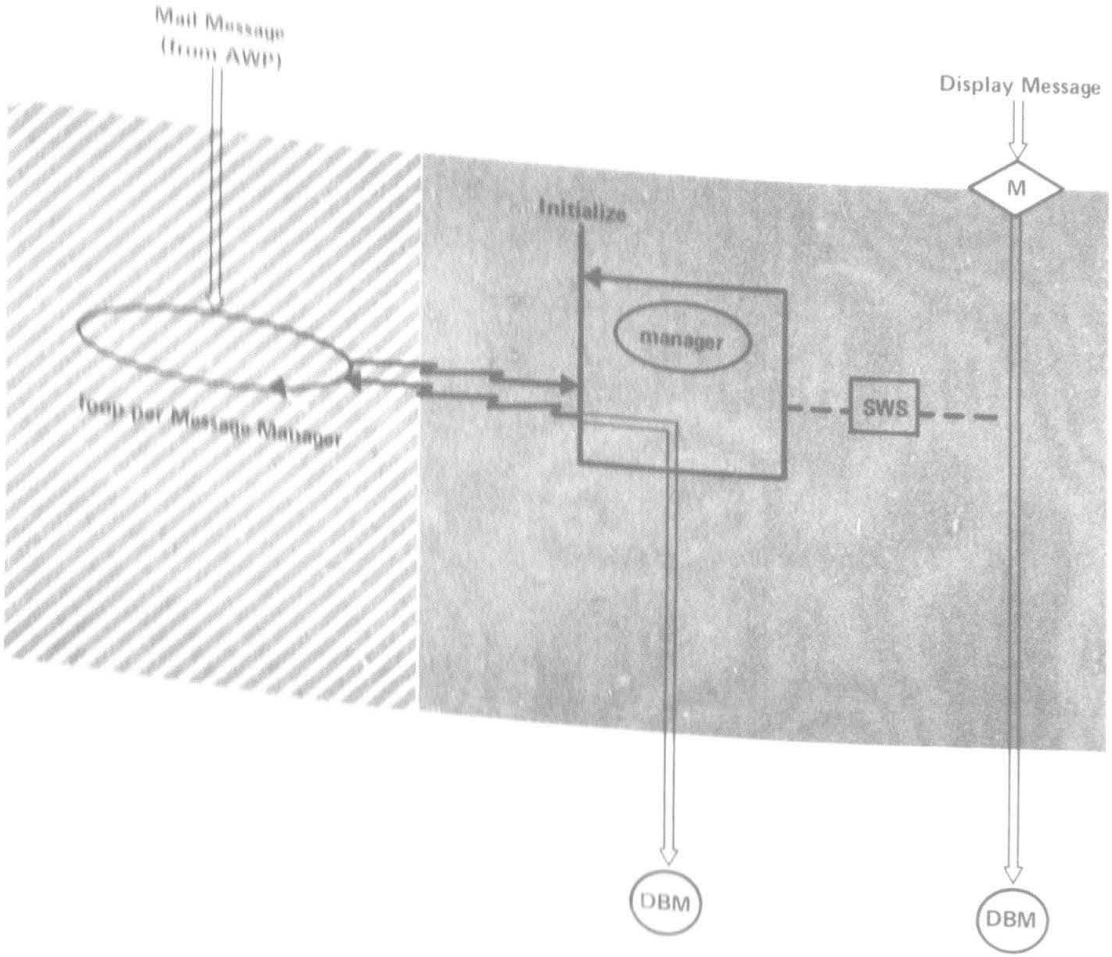


FIGURE 54
MESSAGE MANAGEMENT

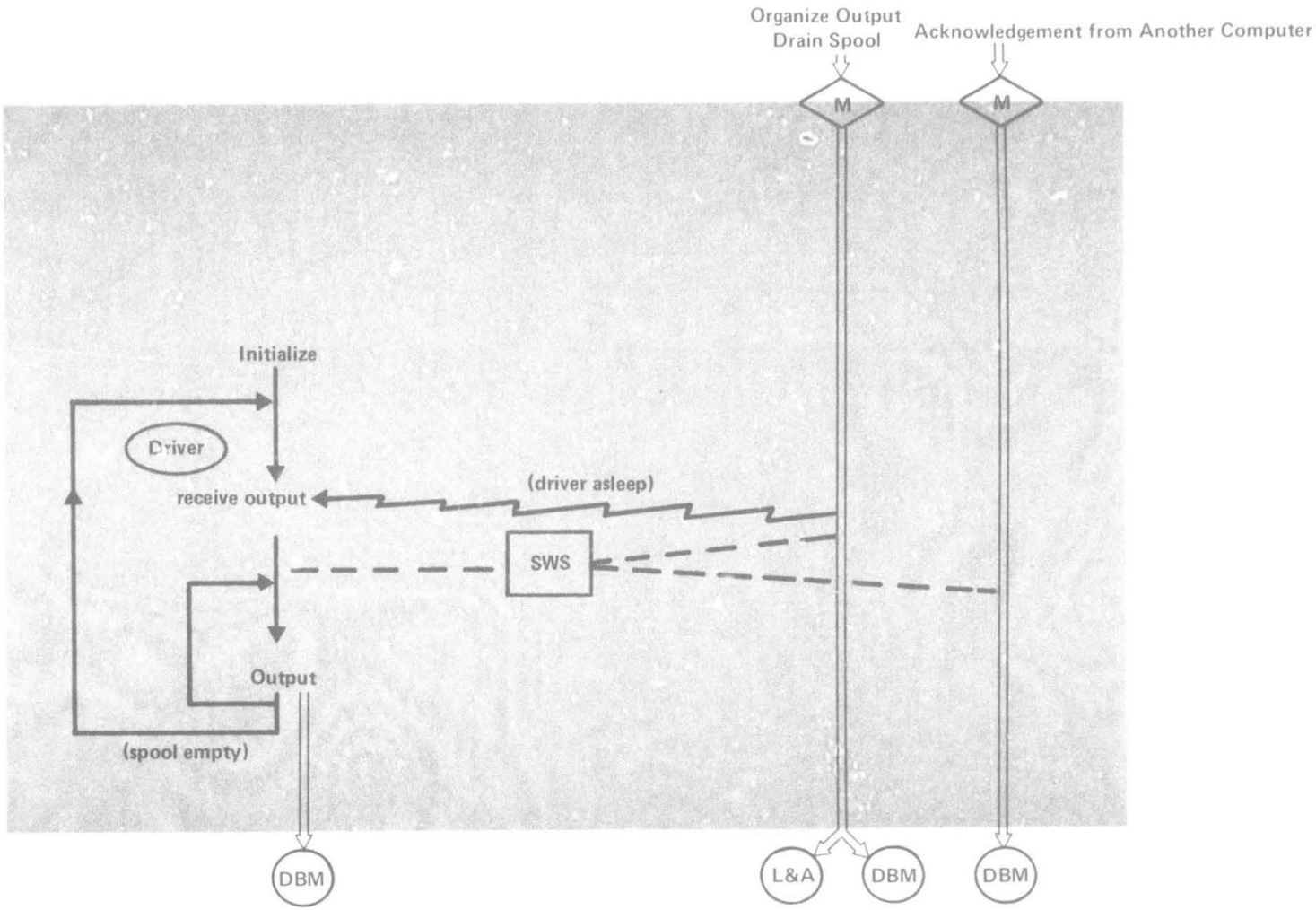


FIGURE 55
COMMON-SERVICE OUTPUT

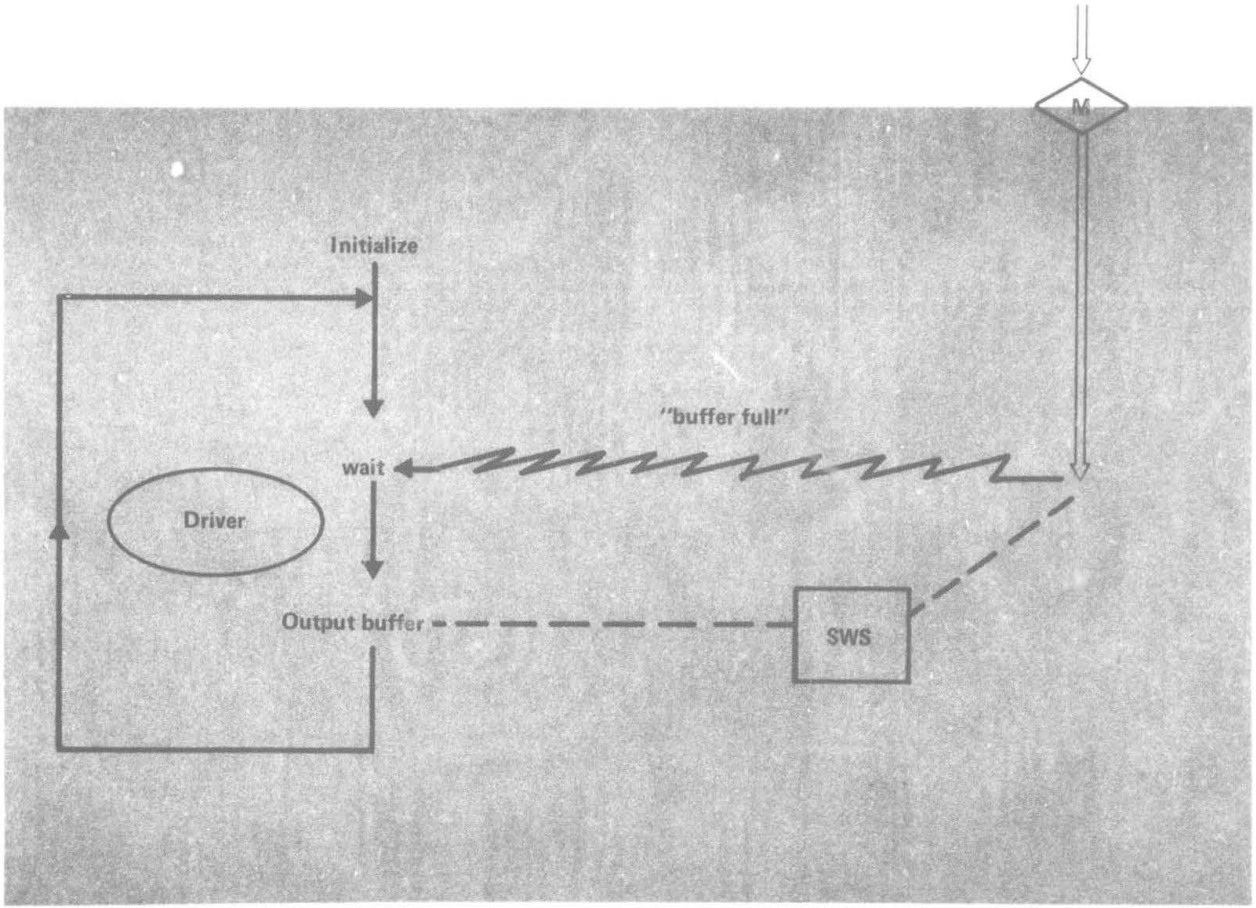


FIGURE 56
LEGAL & ANALYSIS RECORDING

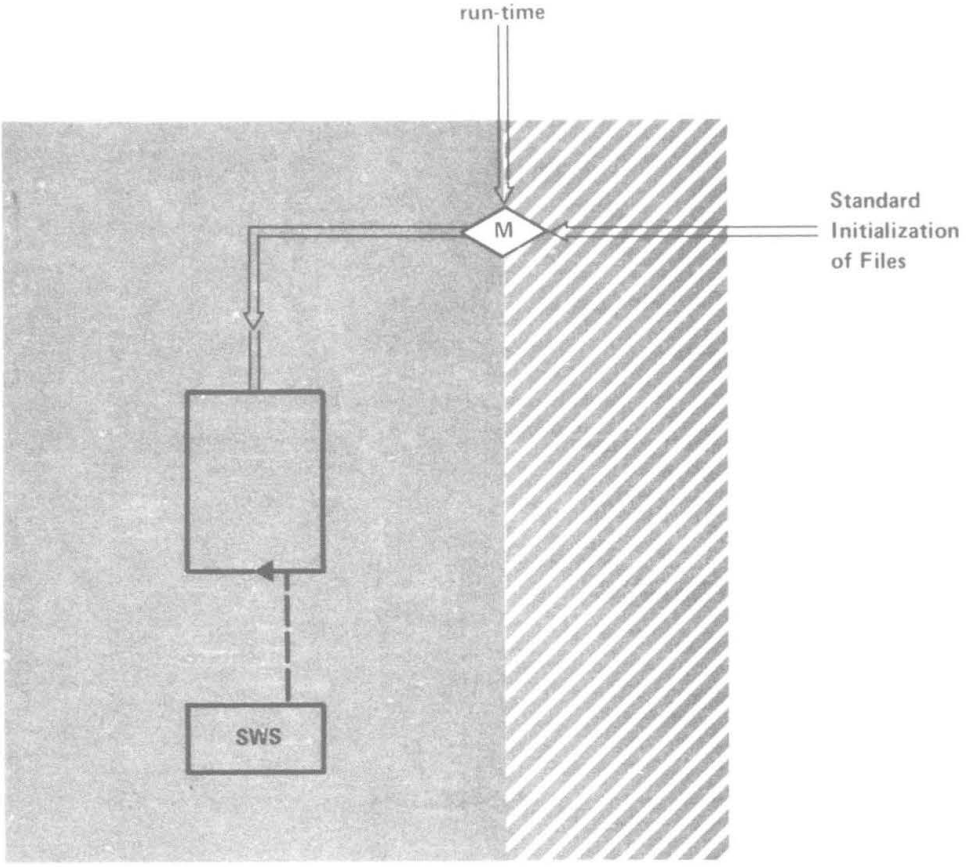


FIGURE 57
DATABASE MANAGEMENT – (1) NORMAL OPERATIONS &
RECORD INITIALIZATION

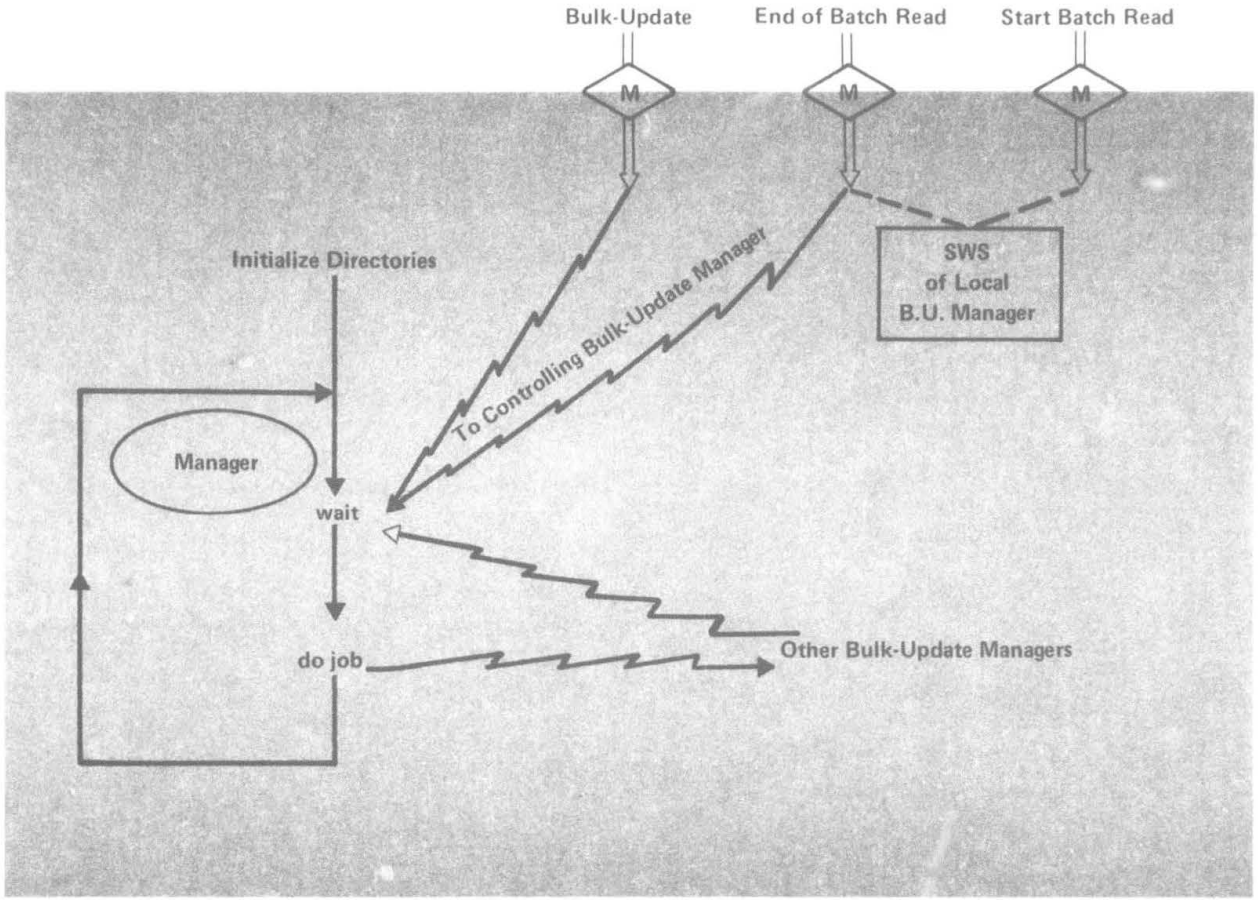


FIGURE 58
DATABASE MANAGEMENT – (2) DIRECTORY INITIALIZATION & BULK-UPDATE

APPENDIX A

CALCULATING LOADS

The data in the tables of this Appendix were obtained by taking measurements with the model running in simulated real time against loads which are typical of the first two hours of the load tape. The run was made against 30 minutes of 1986 load (see 3.6 of FAA-E-2683) and the resulting data was normalized to one hour. The model mapper should allow for the fact that there can be no guarantee that these loads represent the worst possible conditions. Further, more comprehensive and up to date data will be supplied at the workshop.

In Tables A-2, A-4 and A-6, the statistics are broken down by input type. However, the work of the Service Natural Flows cannot readily be associated with particular messages, and is shown as classified by Natural Flow.

For an interpretation of units for data presented in this Appendix, see paragraph 6.3 of Part II and Appendix D of Part IV.

A.1 CODE SIZE

Total code size for a processor can be deduced approximately from the data given in Table 11 and 12, or, for finer detail, the Modeling Language listing of the model in Appendix C. For a finer evaluation, or for the distinction between residence and non-residence (main memory vs. disk), study the Modeling Language listing and relate to the mapping and scheduling strategies of the proposed design.

A.2 CPU

Table A-1 shows by Functional Area, taking inner and outer set cells separately, the simulated CPU requirements. Table A-2 shows these figures broken down by input type.

A.3 WORKING SPACE

Table A-3 was derived by measurements with typical loads. It shows the average amount of Working Space at each Migration Point and at each return. The statistics reflect the alternative method of migration, that is, the method that uses temporary Spawning. Table A-4 shows these figures broken down by input type.

A.4 PERMANENT MEMORY

Table A-5 was derived by taking measurements with typical loads. It shows the number of Permanent Memory Reads and Writes, for records and for directories separately, for each file. Table A-6 shows this broken down by input type. The default file types were used, and the statistics do not distinguish the case where a read is unnecessary because a copy already exists in main store.

TABLE A-1

SUMMARY OF CPU UNITS BY FUNCTIONAL AREA

FUNCTIONAL AREA	CPU UNITS*
SpecialistInterface	2,522,932
DUATInterface	366,990
CommonServiceInput	141,670
TimeManagementInnerSet	6,385,484
TimeManagementOuterSet	23,600
GraphicsManagement	2,760
FlightPlanManagement	243,082
RouteConversion	112,652,000
WeatherManagementInnterSet	91,019,750
WeatherManagementOuterSet	92,208
MessagePoolManagementInnerSet	70,440
MessagePoolManagementOuterSet	480
CommonServiceOutput	275,790
Legal&AnalysisRecording	1,245,780
DataBaseManagement	45,224,098
RadarManagement	435,000

* Run time of approximately 30 minutes, normalized to one hour, against the 1986 load.

TABLE A-2

CPU UNITS BY INPUT TYPE

FUNCTIONAL AREA/INPUT TYPE	NUMBER OF CPU UNITS*
SpecialistInterface	
SimpleInput	134930
PageForward	812522
PageBackward	110976
DirectionFinding	12500
RequestSendFormat	6750
FlightLog	267200
ExecuteLinkSend	7020
DisplayMessage	198380
PrintScreen	81480
StartConversationWithNewFlightPlan	84320
StartConversationWithFiledFlightPlan	114270
ResumeConversationWithCurrentFlightPlan	36400
RouteDataForFlightPlan	187938
NonRouteDataForFlightPlan	134136
FileFlightPlanFromConversation	181650
RequestWeather	139720
StopBenchmark	12740
DUATInterface	
DUATSimpleInput	21484
DUATPageForward	68382
DUATPageBackward	9520
RequestDUATSendFormat	1250
DUATFlightLog	94480
ExecuteDUATLinkSend	1350
StartDUATConversationWithNewFlightPlan	11220
StartDUATConversationWithFiledFlightPlan	8970
ResumeDUATConversationWithCurrentFlightPlan	8400
RouteDataForDUATFlightPlan	22458
NonRouteDataForDUATFlightPlan	83916
FileFlightPlanFromDUATConversation	18200
RequestDUATWeather	11760
DUATStopBenchmark	5600
RadarManagement	
RadarManager	435000

* Statistic from a 30 minute run, normalized to one hour.

TABLE A-2 (CONTD)

CommonServiceInput	
InputDrivers (1)	29890
InputDrivers (2)	21310
InputDrivers (3)	63220
InputDrivers (4)	27250
TimeManagementInnerSet	
scheduler	6269924
TimeManager	115560
TimeManagementOuterSet	
FileFlightPlanFromConversation	20950
FileFlightPlanFromDUATConversation	1940
InputDrivers (2)	710
GraphicsManagement	
InputDrivers (3)	2760
FlightPlanManagement	
StartConversationWithFiledFlightPlan	6446
FileFlightPlanFromConversation	202410
StartDUATConversationWithFiledFlightPlan	506
FileFlightPlanFromDUATConversation	20280
InputDrivers (2)	6240
TimeManager	7200
RouteConversion	
RouteDataForFlightPlan	100484000
RouteDataForDUATFlightPlan	12168000
WeatherManagementInnerSet	
RequestWeather	83724750
RequestDUATWeather	7295000
WeatherManagementOuterSet	
RequestWeather	48528
RequestDUATWeather	4160
InputDrivers (3)	35420
InputDrivers (4)	4100
MessagePoolManagementInnerSet	
DisplayMessage	68670
MessageManagers	1770

TABLE A-2 (CONTD)

MessagePoolManagementOuterSet	
InputDrivers (3)	480
CommonServiceOutput	
ExecuteLinkSend	3120
PrintScreen	34920
FileFlightPlanFromConversation	31800
ExecuteDUATLinkSend	600
FileFlightPlanFromDUATConversation	3120
AncestralNaturalFlow	600
OutputDrivers (1)	21600
OutputDrivers (2)	39750
OutputDrivers (3)	31500
OutputDrivers (4)	12450
OutputDrivers (5)	43800
InputDrivers (1)	4500
InputDrivers (2)	6580
InputDrivers (3)	22010
InputDrivers (4)	9840
TimeManager	9600
Legal&AnalysisRecording	
SimpleInput	56650
PageForward	341000
PageBackward	44880
DirectionFinding	5500
RequestSendFormat	2970
FlightLog	2200
ExecuteLinkSend	5720
DisplayMessage	83930
PrintScreen	64020
StartConversationWithNewFlightPlan	27280
StartConversationWithFiledFlightPlan	32230
ResumeConversationWithCurrentFlightPlan	14300
RouteDataForFlightPlan	52470
NonRouteDataForFlightPlan	45540
FileFlightPlanFromConversation	86240
RequestWeather	54890
StopBenchmark	10010
DUATSimpleInput	9020
DUATPageForward	28710
DUATPageBackward	3850
RequestDUATSendFormat	550

TABLE A-2 (CONTD)

DUATFlightLog	770
ExecuteDUATLinkSend	1100
StartDUATConversationWithNewFlightPlan	3630
StartDUATConversationWithFiledFlightPlan	2530
ResumeDUATConversationWithCurrentFlightPlan	3300
RouteDataForDUATFlightPlan	6270
NonRouteDataForDUATFlightPlan	28490
FileFlightPlanFromDUATConversation	8580
RequestDUATWeather	4620
DUATStopBenchmark	4400
AncestralNaturalFlow	550
Legal&AnalysisDrivers	119490
InputDrivers (1)	9900
InputDrivers (2)	12760
InputDrivers (3)	40590
InputDrivers (4)	18040
TimeManager	8800
DataBaseManagement	
FlightLog	8400
ExecuteLinkSend	12480
DisplayMessage	2047965
PrintScreen	139680
StartConversationWithNewFlightPlan	342240
StartConversationWithFiledFlightPlan	877976
ResumeConversationWithCurrentFlightPlan	54600
RouteDataForFlightPlan	7519684
NonRouteDataForFlightPlan	372600
FileFlightPlanFromConversation	1653060
RequestWeather	3789103
DUATFlightLog	2940
ExecuteDUATLinkSend	2400
StartDUATConversationWithNewFlightPlan	45540
StartDUATConversationWithFiledFlightPlan	69000
ResumeDUATConversationWithCurrentFlightPlan	12600
RouteDataForDUATFlightPlan	917450
NonRouteDataForDUATFlightPlan	233100
FileFlightPlanFromDUATConversation	165360
RequestDUATWeather	335900
AncestralNaturalFlow	2400
scheduler	25506360
MessageManagers	36000
OutputDrivers (1)	69120
OutputDrivers (2)	129900
OutputDrivers (3)	100800
OutputDrivers (4)	39840

TABLE A-2 (CONC)

OutputDrivers (5)	140160
InputDrivers (1)	64800
InputDrivers (2)	76320
InputDrivers (3)	174480
InputDrivers (4)	90200
TimeManager	189240
RadarManager	2400

TABLE A-3
WORKING SPACE TRANSFERS
LISTED BY CELL

WORKING SPACE TAKEN AT MIGRATION Recipient Cell	BYTES TRANSFERRED (AVERAGE)	NUMBER OF TRANSFERS*
GetFlightPlanReadOnly	360	10
GetFlightPlanForUpdating	360	622
FileThisFlightPlan	1018	1142
ConvertRoute	250	1068
GetIndividualWeatherInSeries	250	6586
MessageToScreen	250	1526
OrganizeOutput	667	1994
DrainSpool	610	10
MakeLegal&AnalysisRecording	342	20478
WriteNewRecord	1018	552
WriteRecord	659	14116
WriteRecordKeepingWorkingSpace	378	2028
WriteRecord&UnlockIt	547	2344
ReadRecord	252	178122
ReadRecordForUpdating	278	2382
DeleteRecord	250	1074
StartBatchRead	250	4418
EndOfBatchRead	250	4418
StartRescueOperations	250	160
StopDoingScheduledActions	250	2
FlightPlanFromAnotherComputer	515	52
IndividualUpdateFromAWP	340	308
AFOSGraphicsFromAWP	3150	24
AcknowledgementFromAnotherComputer	270	282
MailMessage	302	24
HourlyUpdateFromAWP	3950	164
BulkUpdate	3950	164

WORKING SPACE RETURNED FROM MIGRATION Sending Cell	BYTES TRANSFERRED (AVERAGE)
GetFlightPlanReadOnly	768
GetFlightPlanForUpdating	768
ConvertRoute	600
GetIndividualWeatherInSeries	4964
MessageToScreen	80
ReadRecord	163
ReadRecordForUpdating	304

* 30 minute run time normalized to one hour.

TABLE A-4
WORKING SPACE TRANSFERS
BY INPUT TYPE

WorkingSpaceTakenAtMigration

RECIPIENT CELL/INPUT TYPE	BYTES*
GetFlightPlanReadOnly	
StartConversationWithFiledFlightPlan	360
GetFlightPlanForUpdating	
StartConversationWithFiledFlightPlan	360
StartDUATConversationWithFiledFlightPlan	360
FileThisFlightPlan	
FileFlightPlanFromConversation	1018
FileFlightPlanFromDUATConversation	1018
ConvertRoute	
RouteDataForFlightPlan	250
RouteDataForDUATFlightPlan	250
GetIndividualWeatherInSeries	
RequestWeather	250
RequestDUATWeather	250
MessageToScreen	
DisplayMessage	250
OrganizeOutput	
ExecuteLinkSend	580
PrintScreen	1250
FileFlightPlanFromConversation	610
ExecuteDUATLinkSend	520
FileFlightPlanFromDUATConversation	610
AncestralNaturalFlow	20
InputDrivers (2)	380
InputDrivers (3)	266
InputDrivers (4)	266
TimeManager	280
MakeLegal&AnalysisRecording	
SimpleInput	412
PageForward	412
PageBackward	412
DirectionFinding	250

* Average working space size transferred at a BigGoTo.

TABLE A-4 (CONTD)

RequestSendFormat	250
FlightLog	250
ExecuteLinkSend	500
DisplayMessage	250
PrintScreen	250
StartConversationWithNewFlightPlan	250
StartConversationWithFiledFlightPlan	360
ResumeConversationWithCurrentFlightPlan	360
RouteDataForFlightPlan	250
NonRouteDataForFlightPlan	500
FileFlightPlanFromConversation	250
RequestWeather	250
StopBenchmark	250
DUATSimpleInput	412
DUATPageForward	412
DUATPageBackward	412
RequestDUATSendFormat	250
DUATFlightLog	250
ExecuteDUATLinkSend	500
StartDUATConversationWithNewFlightPlan	250
StartDUATConversationWithFiledFlightPlan	360
ResumeDUATConversationWithCurrentFlightPlan	360
RouteDataForDUATFlightPlan	250
NonRouteDataForDUATFlightPlan	500
FileFlightPlanFromDUATConversation	250
RequestDUATWeather	250
DUATStopBenchmark	250
InputDrivers (1)	250
InputDrivers (2)	250
InputDrivers (3)	250
InputDrivers (4)	250
TimeManager	250
WriteNewRecord	
FileFlightPlanFromConversation	1018
FileFlightPlanFromDUATConversation	1018
InputDrivers (2)	1018
WriteRecord	
StartConversationWithNewFlightPlan	698
StartConversationWithFiledFlightPlan	701
RouteDataForFlightPlan	748
NonRouteDataForFlightPlan	1268
FileFlightPlanFromConversation	378
StartDUATConversationWithNewFlightPlan	698
StartDUATConversationWithFiledFlightPlan	701

TABLE A-4 (CONTD)

RouteDataForDUATFlightPlan	748
NonRouteDataForDUATFlightPlan	1268
FileFlightPlanFromDUATConversation	378
scheduler	274
MessageManagers	330
OutputDrivers (1)	609
OutputDrivers (2)	568
OutputDrivers (3)	506
OutputDrivers (4)	506
OutputDrivers (5)	1247
InputDrivers (3)	615
WriteRecordKeepingWorkingSpace	
ExecuteLinkSend	620
PrintScreen	370
FileFlightPlanFromConversation	370
ExecuteDUATLinkSend	620
FileFlightPlanFromDUATConversation	370
InputDrivers (2)	370
InputDrivers (3)	370
InputDrivers (4)	370
TimeManager	370
RadarManager	620
WriteRecord&UnlockIt	
DisplayMessage	314
FileFlightPlanFromConversation	1018
FileFlightPlanFromDUATConversation	1018
MessageManagers	314
InputDrivers (2)	1018
TimeManager	1018
ReadRecord	
FlightLog	250
DisplayMessage	250
StartConversationWithNewFlightPlan	250
StartConversationWithFiledFlightPlan	360
ResumeConversationWithCurrentFlightPlan	360
RouteDataForFlightPlan	250
NonRouteDataForFlightPlan	500
FileFlightPlanFromConversation	250
RequestWeather	250
DUATFlightLog	250
StartDUATConversationWithNewFlightPlan	250
StartDUATConversationWithFiledFlightPlan	360

TABLE A-4 (CONTD)

ResumeDUATConversationWithCurrentFlightPlan	360
RouteDataForDUATFlightPlan	250
NonRouteDataForDUATFlightPlan	500
FileFlightPlanFromDUATConversation	250
RequestDUATWeather	250
scheduler	250
OutputDrivers (2)	250
TimeManager	250
ReadRecordForUpdating	
DisplayMessage	250
StartConversationWithFiledFlightPlan	360
StartDUATConversationWithFiledFlightPlan	360
MessageManagers	250
InputDrivers (2)	250
TimeManager	250
DeleteRecord	
DisplayMessage	250
InputDrivers (1)	250
InputDrivers (2)	250
InputDrivers (3)	250
BulkUpdate	
InputDrivers (4)	3950
StartBatchRead	
RequestWeather	250
RequestDUATWeather	250
EndOfBatchRead	
RequestWeather	250
RequestDUATWeather	250
StartRescueOperations	
TimeManager	250
StopDoingScheduledActions	
TimeManager	250
FlightPlanFromAnotherComputer	
InputDrivers (2)	515

TABLE A-4 (CONC)

IndividualUpdateFromAWP	
InputDrivers (3)	340
HourlyUpdateFromAWP	
InputDrivers (4)	3950
AFOSGraphicsFromAWP	
InputDrivers (3)	3150
AcknowledgementFromAnotherComputer	
InputDrivers (1)	270
InputDrivers (2)	270
InputDrivers (3)	270
MailMessage	
InputDrivers (3)	302

TABLE A-5
SUMMARY OF PERMANENT MEMORY READS AND WRITES

DIRECTORY READS File	NUMER OF READS*
HourlyWeatherObservationsFile	4846
FGFile	3584
NOTAMFileExpanded	56
NOTAMFileUnexpanded	406
FTFileExpanded	162
FTFileUnexpanded	1912
SDFile	902
TWEBRouteSynopsisFileExpanded	36
TWEBRouteSynopsisFileUnexpanded	486
WHFileExpanded	22
WHFileUnexpanded	388
FAFileExpanded	352
FAFileUnexpanded	3716
MilitaryOpsFile	350
PIREPFileExpanded	128
PIREPFileUnexpanded	648
HourlyWeatherObservationsAlternate	60
FTAlternateFileExpanded	64
FTAlternativeFileUnexpanded	32
SDAlternateFile	8
Airways&FixesFile	4138
ReportingLocationsFile	30108
AFOSGraphicsFile	24
FlightPlanFile	2178
SchedulingFile	114868
OutputFiles (1)	756
OutputFiles (2)	1154
OutputFiles (3)	866
OutputFiles (4)	332
OutputFiles (5)	1168
TerminalRecordFiles	9396
CurrentFlightPlanFiles	5944
RouteRecordFiles	1992
DUATTerminalRecordFiles	960
DUATCurrentFlightPlanFiles	1540
DUATRouteRecordFiles	212
MessageFiles	2316
TerminalMessageListFiles	4662
RadarFile	14

DIRECTORY WRITES File	NUMBER OF WRITES*
HourlyWeatherObservationsAlternate	60
FTAlternateFileExpanded	64
FTAlternateFileUnexpanded	32
SDAlternateFile	8
FlightPlanFile	2168
SchedulingFile	1936

*Run time normalized to one hour.

TABLE A-5 (CONTD)

DIRECTORY WRITES File	NUMBER OF WRITES*
OutputFiles (1)	180
OutputFiles (2)	76
OutputFiles (3)	26
MessageFiles	766
TerminalMessageListFiles	3136

RECORD READS File	NUMBER OF RECORDS*
HourlyWeatherObservationsFile	4740
FGFile	3584
NOTAMFileExpanded	30
NOTAMFileUnexpanded	378
FTFileExpanded	162
FTFileUnexpanded	1912
SDFile	902
TWEBRouteSynopsisFileExpanded	36
TWEBRouteSynopsisFileUnexpanded	486
WHFileExpanded	20
WHFileUnexpanded	386
FAFileExpanded	352
FAFileUnexpanded	3716
MilitaryOpsFile	350
PIREPFileExpanded	56
PIREPFileUnexpanded	576
Airways&FixesFile	4138
ReportingLocationsFile	30108
FlightPlanFile	824
SchedulingFile	112932
OutputFiles	6
TerminalRecordFiles	4698
CurrentFlightPlanFiles	3080
RouteRecordFiles	1038
DUATTerminalRecordFiles	480
DUATCurrentFlightPlanFiles	796
DUATRouteRecordFiles	98
MessageFiles	1526
TerminalMessageListFiles	3094

* Run time normalized to one hour.

TABLE A-5 (CONC)

RECORD WRITES File	NUMBER OF RECORDS
HourlyWeatherObservationsFile	106
NOTAMFileExpanded	28
NOTAMFileUnexpanded	28
PIREPFileExpanded	72
PIREPFileUnexpanded	72
HourlyWeatherObservationsAlternate	2160
FTAlternateFileExpanded	1440
FTAlternateFileUnexpanded	1440
SDAlternateFile	200
AFOSGraphicsFile	24
FlightPlanFile	1328
SchedulingFile	1936
OutputFiles (1)	576
OutputFiles (2)	1072
OutputFiles (3)	840
OutputFiles (4)	332
OutputFiles (5)	1168
TerminalRecordFiles	4698
CurrentFlightPlanFiles	2864
RouteRecordFiles	954
DUATTerminalRecordFiles	480
DUATCurrentFlightPlanFiles	744
DUATRouteRecordFiles	114
MessageFiles	24
TerminalMessageListFiles	1568
RadarFile	14

TABLE A-6
PERMANENT MEMORY READS AND WRITES
BY INPUT TYPE

#DirectoryReads

FILE/INPUT TYPE	NUMBER OF READS
HourlyWeatherObservationsFile	
RequestWeather	4378
RequestDUATWeather	362
input from AWP	106
FGFile	
RequestWeather	3274
RequestDUATWeather	310
NOTAMFileExpanded	
RequestDUATWeather	30
input from AWP	26
NOTAMFileUnexpanded	
RequestWeather	378
input from AWP	28
FTFileExpanded	
RequestDUATWeather	162
FTFileUnexpanded	
RequestWeather	1912
SDFile	
RequestWeather	816
RequestDUATWeather	86

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
TWEBRouteSynopsisFileExpanded	
RequestDUATWeather	36
TWEBRouteSynopsisFileUnexpanded	
RequestWeather	486
WHFileExpanded	
RequestDUATWeather	20
input from AWP	2
WHFileUnexpanded	
RequestWeather	386
input from AWP	2
FAFileExpanded	
RequestDUATWeather	352
FAFileUnexpanded	
RequestWeather	3716
MilitaryOpsFile	
RequestWeather	314
RequestDUATWeather	36
PIREPFileExpanded	
RequestDUATWeather	56
input from AWP	72

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
PIREPFileUnexpanded	
RequestWeather	576
input from AWP	72
HourlyWeatherObservationsAlternate	
input from AWP	60
FTAlternateFileExpanded	
input from AWP	64
FTAlternateFileUnexpanded	
input from AWP	32
SDAlternateFile	
input from AWP	8
Airways&FixesFile	
RouteDataForFlightPlan	3696
RouteDataForDUATFlightPlan	442
ReportingLocationsFile	
RouteDataForFlightPlan	26802
RouteDataForDUATFlightPlan	3306
AFOSGraphicsFile	
input from AWP	24
FlightPlanFile	
StartConversationWithFiledFlightPlan	586
FileFlightPlanFromConversation	1038
StartDUATConversationWithFiledFlightPlan	46
FileFlightPlanFromDUATConversation	104
input from Other FSDPSs	84
TimeManager	320
SchedulingFile	
scheduler	114706
TimeManager	162

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
OutputFiles (1)	
FileFlightPlanFromConversation	252
FileFlightPlanFromDUATConversation	20
AncestralNaturalFlow	2
output to NAS	288
input from NAS	180
input from Other FSDPS	14
OutputFiles (2)	
FileFlightPlanFromConversation	278
FileFlightPlanFromDUATConversation	20
AncestralNaturalFlow	2
output to Other FSDPSs	542
input from Other FSDPSs	140
TimeManager	160
OutputFiles (3)	
ExecuteLinkSend	52
ExecuteDUATLinkSend	10
AncestralNaturalFlow	2
output to AWP	420
input from AWP	382
OutputFiles (4)	
AncestralNaturalFlow	2
output to AWP	166
input from AWP	164
OutputFiles (5)	
PrintScreen	582
AncestralNaturalFlow	2
output to Printer	584
TerminalRecordFiles	
StartConversationWithNewFlightPlan	992
StartConversationWithFiledFlightPlan	2344
RouteDataForFlightPlan	1908
FileFlightPlanFromConversation	4152

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
CurrentFlightPlanFiles	
StartConversationWithNewFlightPlan	496
StartConversationWithFiledFlightPlan	586
ResumeConversationWithCurrentFlightPlan	260
RouteDataForFlightPlan	1908
NonRouteDataForFlightPlan	1656
FileFlightPlanFromConversation	1038
RouteRecordFiles	
FlightLog	40
RouteDataForFlightPlan	954
RequestWeather	998
DUATTerminalRecordFiles	
StartDUATConversationWithNewFlightPlan	132
StartDUATConversationWithFiledFlightPlan	184
RouteDataForDUATFlightPlan	228
FileFlightPlanFromDUATConversation	416
DUATCurrentFlightPlanFiles	
StartDUATConversationWithNewFlightPlan	66
StartDUATConversationWithFiledFlightPlan	46
ResumeDUATConversationWithCurrentFlightPlan	60
RouteDataForDUATFlightPlan	228
NonRouteDataForDUATFlightPlan	1036
FileFlightPlanFromDUATConversation	104
DUATRouteRecordFiles	
DUATFlightLog	14
RouteDataForDUATFlightPlan	114
RequestDUATWeather	84
MessageFiles	
DisplayMessage	2292
MessageManagers	24
TerminalMessageListFiles	
DisplayMessage	4578
MessageManagers	84
RadarFile	
UpdateRadarFiles	14

TABLE A-6 (CONTD)

#DirectoryWrites

FILE/INPUT TYPE	NUMBER OF READS
HourlyWeatherObservationsAlternate	
input from AWP	60
FTAlternateFileExpanded	
input from AWP	64
FTAlternateFileUnexpanded	
input from AWP	32
SDAlternateFile	
input from AWP	8
FlightPlanFile	
StartConversationWithFiledFlightPlan	576
FileFlightPlanFromConversation	1038
StartDUATConversationWithFiledFlightPlan	46
FileFlightPlanFromDUATConversation	104
input from Other FSDPSs	84
TimeManager	320
SchedulingFile	
scheduler	1936
OutputFiles (1)	
input from NAS	180
OutputFiles (2)	
input from Other FSDPSs	76
OutputFiles (3)	
input from AWP	26
MessageFiles	
DisplayMessage	766
TerminalMessageListFiles	
DisplayMessage	3052
MessageManagers	84

TABLE A-6 (CONTD)

#Records Reads

FILE/INPUT TYPE	NUMBER OF READS
HourlyWeatherObservationsFile	
RequestWeather	4378
RequestDUATWeather	362
FGFile	
RequestWeather	3274
RequestDUATWeather	310
NOTAMFileExpanded	
RequestDUATWeather	30
NOTAMFileUnexpanded	
RequestWeather	378
FTFileExpanded	
RequestDUATWeather	162
FTFileUnexpanded	
RequestWeather	1912
SDFile	
RequestWeather	816
RequestDUATWeather	86
TWEBRouteSynopsisFileExpanded	
RequestDUATWeather	36
TWEBRouteSynopsisFileUnexpanded	
RequestWeather	486
WHFileExpanded	
RequestDUATWeather	20
WHFileUnexpanded	
RequestWeather	386

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
FAFileExpanded	
RequestDUATWeather	352
FAFileUnexpanded	
RequestWeather	3716
MilitaryOpsFile	
RequestWeather	314
RequestDUATWeather	36
PIREPFileExpanded	
RequestDUATWeather	56
PIREPFileUnexpanded	
RequestWeather	576
Airways&FixesFile	
RouteDataForFlightPlan	3696
RouteDataForDUATFlightPlan	442
ReportingLocationsFile	
RouteDataForFlightPlan	26802
RouteDataForDUATFlightPlan	3306
FlightPlanFile	
StartConversationWithFiledFlightPlan	586
StartDUATConversationWithFiledFlightPlan	46
TimeManager	160
input from Other FSDPSs	32
SchedulingFile	
scheduler	112770
TimeManager	162
OutputFiles (2)	
output to Other FSDPSs	2
TerminalRecordFiles	
StartConversationWithNewFlightPlan	496
StartConversationWithFiledFlightPlan	1172
RouteDataForFlightPlan	954
FileFlightPlanFromConversation	2076
CurrentFlightPlanFiles	
ResumeConversationWithCurrentFlightPlan	260
RouteDataForFlightPlan	954
NonRouteDataForFlightPlan	828
FileFlightPlanFromConversation	1038

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
RouteRecordFiles	
FlightLog	40
RequestWeather	998
DUATTerminalRecordFiles	
StartDUATConversationWithNewFlightPlan	66
StartDUATConversationWithFiledFlightPlan	92
RouteDataForDUATFlightPlan	114
FileFlightPlanFromDUATConversation	208
DUATCurrentFlightPlanFiles	
ResumeDUATConversationWithCurrentFlightPlan	60
RouteDataForDUATFlightPlan	114
NonRouteDataForDUATFlightPlan	518
FileFlightPlanFromDUATConversation	104
DUATRouteRecordFiles	
DUATFlightLog	14
RequestDUATWeather	84
MessageFiles	
DisplayMessage	1526
TerminalMessageListFiles	
DisplayMessage	3052
MessageManagers	42
	#RecordWrites
HourlyWeatherObservationsFile	
input from AWP	106
NOTAMFileExpanded	
input from AWP	28
NOTAMFileUnexpanded	
input from AWP	28
WHFileExpanded	
input from AWP	2
WHFileUnexpanded	
input from AWP	2
PIREPFileExpanded	
input from AWP	72
PIREPFileUnexpanded	
input from AWP	72

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
HourlyWeatherObservationsAlternate	
input from AWP	2160
FTAlternateFileExpanded	
input from AWP	1408
FTAlternateFileUnexpanded	
input from AWP	1440
SDAlternateFile	
input from AWP	200
AFOSGraphicsFile	
input from AWP	24
RadarFile	
RadarManager	14
FlightPlanFile	
FileFlightPlanFromConversation	1038
FileFlightPlanFromDUATConversation	104
input from Other FSDPSs	26
TimeManager	160
SchedulingFile	
scheduler	1936
OutputFiles (1)	
FileFlightPlanFromConversation	252
FileFlightPlanFromDUATConversation	20
output to NAS	288
AncestralNaturalFlow	2
input from Other FSDPSs	14

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
OutputFiles (2)	
FileFlightPlanFromConversation	278
FileFlightPlanFromDUATConversation	32
AncestralNaturalFlow	2
output to Other FSDPSs	536
input from Other FSDPSs	64
TimeManager	160
OutputFiles (3)	
ExecuteLinkSend	52
ExecuteDUATLinkSend	10
AncestralNaturalFlow	2
output to AWP	420
input from AWP	356
OutputFiles (4)	
output to AWP	166
AncestralNaturalFlow	2
input from AWP	164
OutputFiles (5)	
PrintScreen	582
output to Printer	584
AncestralNaturalFlow	2
TerminalRecordFiles	
StartConversationWithNewFlightPlan	496
StartConversationWithFiledFlightPlan	1172
RouteDataForFlightPlan	954
FileFlightPlanFromConversation	2076
CurrentFlightPlanFiles	
StartConversationWithNewFlightPlan	496
StartConversationWithFiledFlightPlan	586
RouteDataForFlightPlan	954
NonRouteDataForFlightPlan	828

TABLE A-6 (CONTD)

FILE/INPUT TYPE	NUMBER OF READS
RouteRecordFiles	
RouteDataForFlightPlan	954
DUATTerminalRecordFiles	
StartDUATConversationWithNewFlightPlan	66
StartDUATConversationWithFiledFlightPlan	92
RouteDataForDUATFlightPlan	114
FileFlightPlanFromDUATConversation	208
DUATCurrentFlightPlanFiles	
StartDUATConversationWithNewFlightPlan	66
StartDUATConversationWithFiledFlightPlan	46
RouteDataForDUATFlightPlan	114
NonRouteDataForDUATFlightPlan	518
DUATRouteRecordFiles	
RouteDataForDUATFlightPlan	114
MessageFiles	
MessageManagers	24
TerminalMessageListFiles	
DisplayMessage	1526
MessageManagers	42

APPENDIX B

SUMMARY GUIDE TO MODEL MAPPING

In the following text, "read" means "read if you have not already done so".

This Appendix assumes that the reader has read and understands Parts II and III, and section 2 and paragraphs 3.1, 3.4.2 and 3.5 of Part IV.

B.1 OVERVIEW

The model mapper has to:

(1) Select values for the options summarized in section 7, and forward the selected choices to the Government (or, by omission, accept the options at their default values).

(2) Tell the VIP writer;

- On which processor(s) each module is to be available,
- On which processor each Natural Flow is to start, and on what basis the VIP should decide when and where to migrate it, and
- Where each file and each SharedWorkingSpace should be held. Also whether each should move from processor to processor, and if so on what basis.

(3) Preferably, assist the VIP writer in arriving at optimum scheduling strategies.

The Government provided documentation does not address the third item. The first two are summarized here. They divide into:

(a) Miscellaneous options.

(b) Strategy on migrating Natural Flows from one processor to another.

(c) Other distribution decisions (such as, what is placed on which processor).

For item (a), go through the list in paragraphs 7.2 and 7.3.

For items (b) and (c), the options concerned are listed in paragraph 7.1. The model mapper can adopt any desired scheme, or can follow the suggestions offered in section 5.

The model mapper should approach the options in the model as follows:

(1) Decide which options need further experimenting.

(2) For those options that do require further experimenting;

- If they are HOL editable, ask for HOL editable settings in the first trial full scale model
- If they are dynamic, ask for dynamic settings
- If they are neither;

  (a) in the case of the options in paragraph 7.1, name the upper bound of the possible values you might be interested in

  (b) for any in paragraph 7.2 that is of critical importance, you will have to ask for several copies of the model, each having a different setting.

(3) Analyze performance with varied settings of the option, and select final values for the demonstration run.

Paragraphs B.2 and B.3, present a summary guide to the suggestions offered in this part for tackling migration and other distribution decisions. The recommended scheme is the scheme described in section 5. It is not necessarily the optimum approach for a particular vendor.

B.2 GUIDE TO MIGRATION

Question 1: Where a Natural Flow is about to access a file or a Shared Working Space that is on another processor, should the Natural Flow migrate to that processor (answer separately for files and Shared Working Spaces)?

If you answer No:

- With Shared Working Space, the VIP may have administration problems.
- With files, there may be excessive interprocessor traffic.
- This is not the recommended scheme.

If you answer Yes: read section 4, then go to question 2.

Question 2: When should a Natural Flow migrate?

(a) when the file or the Shared Working Space is first referenced?

(b) when a Migration Point is reached?

(c) neither, necessarily

If you answer (a), then you may have a problem knowing when to migrate back.

If you answer (a) or (c), this is not the recommended scheme.

If you answer (b), go to B.3.

B.3 GUIDE TO DISTRIBUTION

Read paragraphs 5.1, 5.3, and follow the procedure in paragraph 5.3.1.

Read paragraph 5.5 as necessary to decide how to distribute each Functional Area.

APPENDIX C

MODELING LANGUAGE LISTING OF THE MODEL

This Appendix provides a listing of the model in Modeling Language.

Page references of the form "p15]" refer to the page numbers that appear in the top-right corner of the page as the listings are read.

References of the form "Vn]" indicate the issuing of VSR#n.

```
[ ******************************************************************************
[ ******************************************************************************
[ ******************************************************************************

[                M O D E L     o f    F. S. I. S.

[ ******************************************************************************
[ ******************************************************************************
[ ******************************************************************************
```

StartOfModel

Section 1 Top Level . . . . . p. 3

PRIME-SOURCE

Section 2 Specialist Interface
top level . . . . p. 7
input messages . . p. 13
supporting cells . p. 20
Section 3 DUAT Interface
top level . . . . p. 21
input messages . . p. 26
supporting cells . p. 32
Section 4 Radar Management . . . p. 33
Section 5 Common-Service Input . p. 35
Section 6 Time Management Pt 1 . p. 41

SERVICES

Time Management Pt 2 . p. 45
Section 7 AFOS Graphics . . . p. 51
Section 8 Flight-Plan Management . p. 53
Section 9 Route Conversion . . . p. 63
Section 10 Weather Management . . p. 70
Section 11 Message-Pool Management p. 79
Section 12 Common-Service Output . p. 85
Section 13 Legal&Analysis Recording p. 93
Section 14 Data-Base Management
reception-desk cells p. 98
extensions to
reception desks . p. 111
supporting cells . . p. 115
cell for DB Managers p. 119

```
[ ***************************************************************************
[ ***************************************************************************

[                        S E C T I O N    1

[                       T O P     L E V E L
[ ***************************************************************************
[ ***************************************************************************


cell InitializeBenchmark
                        [Ancestral Natural Flow is born here


      for each NaturalFlow, unless NaturalFlow = AncestralNaturalFlow
v1]      Spawn  NaturalFlow, StartingAt StartingCell for NaturalFlow,
                                with InheritedSharedWorkingSpace for NaturalFlow

                        [each Natural Flow is doing its private initialization
                        [ -- includes initialization of data base management

v13]     AwaitProd

v14]  prod scheduler
v13]  AwaitProd
                        [ Scheduling file is being initialized
```

```
        for each IndividualSpecialist
v14]       prod NaturalFlow#OfSpecialist CorrespondingTo IndividualSpecialist
v13]       AwaitProd

        for each IndividualDUAT
v14]       prod NaturalFlow#OfDUAT CorrespondingTo IndividualDUAT
v13]       AwaitProd
                        [that was specialists and DUATs initializing terminal files

v14]    prod AWPInputDriver
v13]    AwaitProd
                              [ scenario-driven initialization is now taking place

v14]    prod TimeManager

        for each IndividualSpecialist,
v14]            prod NaturalFlow#OfSpecialist CorrespondingTo IndividualSpecialist

        for each IndividualDUAT,
v14]                         prod NaturalFlow#OfDUAT CorrespondingTo IndividualDUAT

v14]    prod RadarManager

        for each CommonServiceIORoute, unless CommonServiceIORoute = printer,
v14]           prod NaturalFlow#OfInputDriver CorrespondingTo CommonServiceIORoute

p5]     go to BenchmarkRunTime
```

```
cell BenchmarkRunTime

v13]  AwaitProd                    [from Time Manager or one of the input sources
                                   [the benchmark is now in session
p5]   go to WrapUpBenchmark



cell WrapUpBenchmark

v13]  loop  #OfInputDrivenNaturalFlows times, AwaitProd
                                     [ waiting for external input sources to stop

      AcquireWorkingSpace WorkArea#1, MinimumLength For WorkArea#1 words

      for each CommonServiceIORoute
p89]     DrainSpool for CommonServiceIORoute
         AwaitProd                                [remaining output is being spooled

      for each IndividualLegal&AnalysisDriver
v14]     prod NaturalFlow#OfL&ADriver for IndividualLegal&AnalysisDriver,
                                                          WithDataWord  FlushNow
         AwaitProd                             [legal & analysis output being spooled

      AdministrativeTidyUp
v28]  FinalExitToVIP
```

[this page is left blank intentionally

```
[ ***************************************************************************
[ ***************************************************************************

[                 S E C T I O N     2

[              S P E C I A L I S T   I N T E R F A C E

[                           (ALL  CELLS  ARE  INNER-SET)

[ ***************************************************************************
[ ***************************************************************************



[ ************************** SPECIALIST  TOP-LEVEL  ************************
```

```
cell InitializeSpecialist
                                        [Specialist Natural Flows are born here

      clear WeatherRecordCountDown
      set NextPageStart, to 1

v14]  prod AncestralNaturalFlow
v13]  AwaitProd
                              [waiting for the data base to be ready to write to

v2]   AcquireWorkingSpace  WorkArea#1, MinimumLengthForWorkArea#1 words

v2]   AcquireWorkingSpace  RecordArea, TerminalRecordLength characters
      clear RecordedFlightPlan#
p96] WriteNewRecord  to TerminalRecord# for OwnNaturalFlow#,
                         of file TerminalRecordFile ...
                               ... + TerminalFileIDOffset for OwnNaturalFlow#,
                                                   OutOf&Releasing RecordArea
v3]   ReleaseWorkingSpace WorkArea#1
v3]   ReleaseWorkingSpace AdminArea

v14]  prod AncestralNaturalFlow
v13]  AwaitProd
                              [waiting for the Measured Period to start

      FinalizeSpecialistInitialization

p9]   go to SpecialistInput
```

```
cell SpecialistInput

      LocalIntegers MessageType, WorkLength
      CodeLength 60 CodeUnits

v9]   Acquire&Input into SmallInputArea, ReadOnly
v2]   AcquireWorkingSpace  AdminArea, AdminAreaLength words

      RecordInputLength

      set LocalFileIDOffset, to TerminalFileIDOffset for OwnNaturalFlow#
      set LocalFileRecord#, to TerminalRecord# for OwnNaturalFlow#

      Translate2Digits  FromCharacter 9, of SmallInputArea, setting MessageType
v2]   AcquireWorkingSpace  WorkArea#1,
                                 SpecialistWorkLength for MessageType  characters

      TranslateScenarioParametersToWorkArea#1  from SmallInputArea
      UseCPU 70 CPUUnits

v3]   ReleaseWorkingSpace SmallInputArea
v3]   ReleaseWorkingSpace WorkArea#2

      if LoseBigIOArea?  with ScenarioDirectedInputType
         if ParallelWeatherRetrieval, if WeatherRecordCountDown > 0,
p68]                                                                StopTheRetrieval
v3]      if OwnedLength of (BigIOArea) > 0, ReleaseWorkingSpace BigIOArea

p94]  MakeLegal&AnalysisRecording
      go to  cell DealWithSpecialist for ScenarioDirectedInputType
```

```
                    [the cells referred to above are those in the next section
                    [They all come back eventually with a 'go to' to one of
                    [the next three cells, except in the
p19]                [case of StopBenchmark, which does not come back


cell SpecialistFormattedOutput

     CodeLength 40 CodeUnits

v2]  if OwnedLength of (BigIOArea) = 0, AcquireWorkingSpace BigIOArea,
                                        ScenarioDirectedReplyLength characters
v2]  AcquireWorkingSpace  WorkArea#2, ScenarioDirectedReplyLength/4 characters

     set ReplyLength, to ScenarioDirectedReplyLength
     FormatDirectResponse
     UseCPU  50  CPUUnits

v5]  DeAccessWorkingSpace  WorkArea#2
v3]  ReleaseWorkingSpace  WorkArea#1
v3]  ReleaseWorkingSpace  AdminArea

v10] Output  ReplyLength characters, OutOf&Releasing BigIOArea

p9]  go to SpecialistInput
```

```
cell SpecialistFreeTextOutput

      CodeLength 40 CodeUnits

      if OwnedLength of (BigIOArea) < ScenarioDirectedReplyLength,
v2]                 ExtendWorkingSpace  BigIOArea, to ScenarioDirectedReplyLength

      set ReplyLength, to ScenarioDirectedReplyLength
      FormatDirectResponse

      UseCPU  50  CPUUnits

v3]   ReleaseWorkingSpace WorkArea#1
v3]   ReleaseWorkingSpace AdminArea

v10]  Output ReplyLength characters, OutOf&DeAccessing BigIOArea

p9]   go to SpecialistInput
```

```
cell SpecialistWeatherDisplay

      CodeLength 40 CodeUnits

p68]  if ParallelWeatherRetrieval, EnsureEnoughWeatherToDisplay

      loop
         unless OwnedLength of (BigIOArea) < NextPageStart +
                                   ScenarioDirectedReplyLength - 1, escape loop
         if NextPageStart = 1, escape loop
         subtract AverageScreenful, from NextPageStart

      RecordLengthOfBigIOArea

      if OwnedLength of (BigIOArea) < ScenarioDirectedReplyLength,
v2]                ExtendWorkingSpace BigIOArea, to ScenerioDirectedReplyLength,
                                                                      DeAccessed

v5]   AccessPartOfWorkingSpace BigIOArea,
                         ScenerioDirectedReplyLength characters, at NextPageStart
      FormatDirectResponse at NextPageStart
      set ReplyLength, to ScenarioDirectedReplyLength
      UseCPU  50 CPUUnits

v3]   ReleaseWorkingSpace  WorkArea#1
v3]   ReleaseWorkingSpace  AdminArea

v10]  Output ReplyLength characters, OutOf&DeAccessing BigIOArea,
                                                                 at NextPageStart
p9]   go  to SpecialistInput
```

```
[ ***************************  SPECIALIST  INPUT-MESSAGES  ***************

cell SimpleInput

      CodeLength 6 CodeUnits
      UseCPU 11 CPUUnits
p10]  go to SpecialistFormattedOutput


cell PageBackward

      CodeLength 6 CodeUnits
      UseCPU 5 CPUUnits
      unless NextPageStart = 1,
                                subtract  2 * AverageScreenful, from NextPageStart
p13]  go to PageForward


cell PageForward

      CodeLength 6 CodeUnits
      UseCPU 11 CPUUnits
      add AverageScreenful, to NextPageStart
p12]  go to SpecialistWeatherDisplay
```

```
cell DirectionFinding

      CodeLength 6 CodeUnits
      UseCPU  5 CPUUnits
p10]  go to SpecialistFormattedOutput



cell RequestSendFormat

      CodeLength 6 CodeUnits
      UseCPU  5 CPUUnits                [to prepare the format
p10]  Go  to SpecialistFormattedOutput



cell ExecuteLinkSend

      CodeLength 11 CodeUnits

v2]   AcquireWorkingSpace  BigIOArea,
                     AverageOutputLength for MessageFromSpecialist  characters
      UseCPU 15 CPUUnits
      FormatSecondOrderResponseFromOwnPort  for MessageFromSpecialist

p83]  OrganizeOutput to ScenarioDirectedRecipient

p10]  Go to SpecialistFormattedOutput
```

```
cell PrintScreen

      CodeLength 12 CodeUnits

v2]   AcquireWorkingSpace  BigIOArea,
                            AverageOutputLength for PrintedScreenful  characters
      FormatSecondOrderResponseFromOwnPort  for PrintedScreenful
      UseCPU  20 CPUUnits

p83]  OrganizeOutput  to printer
v3]   ReleaseWorkingSpace  BigIOArea

p10]  go to SpecialistFormattedOutput


cell NonRouteDataForFlightPlan

      CodeLength  70 CodeUnits

p104] ReadRecord  LocalFileRecord#,
                  of file CurrentFlightPlanFile + LocalFileIDOffset,
                                                  IntoNewWorkingSpace RecordArea
      UseCPU  30 CPUUnits              [editing local flight plan record

p98] WriteRecord  to LocalFileRecord#,
                   of file CurrentFlightPlanFile + LocalFileIDOffset,
                                                       OutOf&Releasing RecordArea
      UseCPU  12 CPUUnits              [ updating screen format

p10]  Go to SpecialistFormattedOutput
```

```
cell StartConversationWithNewFlightPlan

      CodeLength 50 CodeUnits
v2]   AcquireWorkingSpace  BigIOArea, ScenarioDirectedReplyLength characters
      UseCPU  15 CPUUnits
p20]  Go  to OrganizeFlightPlanFormat



cell RouteDataForFlightPlan

      CodeLength 10 CodeUnits
p56]  ConvertRoute  [into RecordArea
      UseCPU  20 CPUUnits                [ to update the screen format
p98] WriteRecord  to LocalFileRecord#,
                   of file  RouteRecordFile + LocalFileIDOffset,
                                                    OutOf&Releasing RecordArea
p20]  UpdateTerminalRecord
p15]  Go to NonRouteDataForFlightPlan


cell FlightLog

      CodeLength 250 CodeUnits
p104] ReadRecord  LocalFileRecord#, of file RouteRecordFile + LocalFileIDOffset,
                                     IntoNewWorkingSpace RecordArea, ReadOnly
v2]   AcquireWorkingSpace  WorkArea#2, FlightLogWorkArea characters
v2]   AcquireWorkingSpace  BigIOArea, ScenarioDirectedReplyLength characters
      UseCPU  ScenarioDirected#LegsToLog * 1600  CPUUnits

v3]   ReleaseWorkingSpace  RecordArea
v3]   ReleaseWorkingSpace WorkArea#2

p11]  go to SpecialistFreeTextOutput
```

```
cell StartConversationWithFiledFlightPlan

      CodeLength 15 CodeUnits

p20]  UpdateTerminalRecord
      add  100 * UpperHalfOfParameter, to ScenarioDirectedFlightPlan#

      if FlightPlanToBeUpdated
p55]     GetFlightPlanForUpdating  ScenarioDirectedFlightPlan#  [into RecordArea
         UseCPU  25 CPUUnits
p20]     Go  to OrganizeFlightPlanFormat  for ScenarioDirectedFlightPlan#
      else
p55]     GetFlightPlanReadOnly  ScenarioDirectedFlightPlan#     [into RecordArea
         UseCPU  25 CPUUnits
p20]     go to OrganizeFlightPlanFormat


cell ResumeConversationWithCurrentFlightPlan

      CodeLength 10 CodeUnits

p104] ReadRecord  LocalFileRecord#,
                  of file CurrentFlightPlanFile + LocalFileIDOffset,
                                                IntoNewWorkingSpace  RecordArea
v2]   AcquireWorkingSpace  BigIOArea, ScenarioDirectedReplyLength characters
      UseCPU  20 CPUUnits
v3]   ReleaseWorkingSpace  RecordArea

p10]  Go to SpecialistFormattedOutput
```

```
cell FileFlightPlanFromConversation

      CodeLength 20 CodeUnits

      add 100 * UpperHalfOfParameter, to ScenarioDirectedFlightPlan#

p20]  UpdateTerminalRecord  [setting PreviousFlightPlan#
      if PreviousFlightPlan# = 0, if ScenarioDirectedFlightPlan# = 0,
v29]                                                  CryFault NoFlightPlan#Given
      if ScenarioDirectedDataBaseStatusOfFlightPlan = NewFlightPlan
v29]     if PreviousFlightPlan# > 0, CryFault NotANewFlightPlan
p104]    ReadRecord  LocalFileRecord#,
                                 of file CurrentFlightPlanFile + LocalFileIDOffset,
                                                     IntoNewWorkingSpace RecordArea
      else
         if PreviousFlightPlan# = 0,
p55]                          GetFlightPlanForUpdating  ScenarioDirectedFlightPlan#
         else
            set ScenarioDirectedFlightPlan#, to PreviousFlightPlan#
p104]       ReadRecord  LocalFileRecord#,
                                 of file CurrentFlightPlanFile + LocalFileIDOffset,
                                                   IntoNewWorkingSpace  RecordArea

      UseCPU  25 CPUUnits
      set OriginalPort#, to AssociatedLogicalPort# for OwnNaturalFlow#

p56]  FileThisFlightPlan

p20]  UpdateTerminalRecord  recording NoCurrentFlightPlan
p10]  Go to SpecialistFormattedOutput
```

```
cell DisplayMessage

      CodeLength 10 CodeUnits
      UseCPU 10 CPUUnits

p78]  MessageToScreen  from SelectedMessageManager for OwnNaturalFlow#
p11]  Go to SpecialistFreeTextOutput



cell RequestWeather

      CodeLength 10 CodeUnits

      set NextPageStart, to 1
p104] ReadRecord  LocalFileRecord#, of file RouteRecordFile + LocalFileIDOffset,
                                                   IntoNewWorkingSpace  RecordArea
      UseCPU  20 CPUUnits
v3]   ReleaseWorkingSpace  RecordArea

p68]  StartWeatherRetrieval of SpecialistWeatherMenu
p12]  Go  to SpecialistWeatherDisplay



cell StopBenchmark

      if ParallelWeatherRetrieval,
p69]                          if WeatherRecordCountDown > 0, StopTheRetrieval
v14]  prod AncestralNaturalFlow
v13]  AwaitProd             [which never comes
```

```
[        ****************   SPECIALIST  SUPPORTING  CELLS  ************


cell OrganizeFlightPlanFormat  for integer FlightPlanRecord# (default none)

      CodeLength 20 CodeUnits

v2]   if OwnedLength of (RecordArea) = 0, AcquireWorkingSpace  RecordArea,
                                                 FlightPlanRecordLength characters
      UseCPU  20 CPUUnits     [to organize a format for the screen
p98] WriteRecord  to LocalFileRecord#,
                              of file CurrentFlightPlanFile + LocalFileIDOffset,
                                                      OutOf&Releasing  RecordArea

p20]  UpdateTerminalRecord  recording FlightPlanRecord#
p10]  Go to SpecialistFormattedOutput


cell UpdateTerminalRecord  recording integer FlightPlan# (default NoChange)

      CodeLength 15 CodeUnits

p104] ReadRecord  LocalFileRecord#,
                                 of file TerminalRecordFile + LocalFileIDOffset,
                                                 IntoNewWorkingSpace RecordArea
      set PreviousFlightPlan#, to RecordedFlightPlan#
      UseCPU  15 CPUUnits     [updating the contents
      unless FlightPlan# = NoChange, set RecordedFlightPlan#, to FlightPlan#
p98] WriteRecord  to LocalFileRecord#,
                                 of file TerminalRecordFile + LocalFileIDOffset,
                                                      OutOf&Releasing  RecordArea
```

```
[ ******************************************************************************
[ ******************************************************************************

[                        S E C T I O N    3

[                      D  U  A  T    I N T E R F A C E

[                         (ALL CELLS ARE INNER-SET)

[ ******************************************************************************
[ ******************************************************************************
```



```
[          ******************  D U A T   TOP-LEVEL  *********************
```

```
cell InitializeDUAT
                                [DUAT Natural Flows are born here

      clear WeatherRecordCountDown
      set NextPageStart, to 1

v14]  prod AncestralNaturalFlow
v13]  AwaitProd
                                [waiting for the data base to be ready to write to

v2]   AcquireWorkingSpace WorkArea#1, MinimumLengthForWorkArea#1 words

v2]   AcquireWorkingSpace  RecordArea, TerminalRecordLength characters
      clear RecordedFlightPlan#
p96] WriteNewRecord  to TerminalRecord# for OwnNaturalFlow#,
                      of file DUATTerminalRecordFile ...
                                 ... + TerminalFileIDOffset for OwnNaturalFlow#,
                                                    OutOf&Releasing RecordArea
v3]   ReleaseWorkingSpace  WorkArea#1
v3]   ReleaseWorkingSpace  AdminArea

v14]  prod AncestralNaturalFlow
v13]  AwaitProd
                                [waiting for the MeasuredPeriod to start
      FinalizeDUATInitialization
p23]  go to DUATInput
```

```
cell DUATInput

      LocalIntegers MessageType, WorkLength

      CodeLength 60 CodeUnits

v9]   Acquire&Input into SmallInputArea, ReadOnly

v2]   AcquireWorkingSpace  AdminArea, AdminAreaLength words
      RecordInputLength

      set LocalFileIDOffset, to TerminalFileIDOffset for OwnNaturalFlow#
      set LocalFileRecord#, to TerminalRecord# for OwnNaturalFlow#

      Translate2Digits  FromCharacter 9, of SmallInputArea, setting MessageType
v2]   AcquireWorkingSpace  WorkArea#1, DUATWorkLength for MessageType characters

      TranslateScenarioParametersToWorkArea#1  from SmallInputArea
      UseCPU 70 CPUUnits
v3]   ReleaseWorkingSpace SmallInputArea

      if LoseDUATBigIOArea?  with ScenarioDirectedInputType
         if ParallelWeatherRetrieval,
p69]                            if WeatherRecordCountDown > 0, StopTheRetrieval
v3]      if OwnedLength of (BigIOArea) > 0, ReleaseWorkingSpace BigIOArea

p94]  MakeLegal&AnalysisRecording

      go to  cell DealWithDUAT for ScenarioDirectedInputType
```

```
                    [the cells referred to above are those in the next section
                    [They all come back eventually with a 'go to' to one of
                    [the next two cells, except in the
p31]                [case of DUATStopBenchmark, which does not come back


cell DUATNormalOutput

     CodeLength 40 CodeUnits

     if OwnedLength of (BigIOArea) < ScenarioDirectedReplyLength,
v2]                ExtendWorkingSpace  BigIOArea, to ScenarioDirectedReplyLength

     set ReplyLength, to ScenarioDirectedReplyLength
     FormatDirectResponse
     UseCPU  50 CPUUnits

v3]  ReleaseWorkingSpace WorkArea#1
v3]  ReleaseWorkingSpace AdminArea

v10] Output ReplyLength characters, OutOf&Releasing BigIOArea

p23] go to DUATInput
```

```
cell DUATWeatherDisplay

      CodeLength 40 CodeUnits

p69]  if ParallelWeatherRetrieval, EnsureEnoughWeatherToDisplay

      loop
         unless OwnedLength of (BigIOArea) < NextPageStart +
                                        ScenarioDirectedReplyLength - 1, escape loop
         if NextPageStart = 1, escape loop
         subtract AverageScreenful, from NextPageStart
      RecordLengthOfBigIOArea

      if OwnedLength of (BigIOArea) < ScenarioDirectedReplyLength,
v2]                ExtendWorkingSpace BigIOArea, to ScenerioDirectedReplyLength,
                                                                        DeAccessed
v5]   AccessPartOfWorkingSpace BigIOArea,
                        ScenerioDirectedReplyLength characters, at NextPage Start
      FormatDirectResponse at NextPageStart
      set ReplyLength, to ScenarioDirectedReplyLength
      UseCPU  50 CPUUnits

v3]   ReleaseWorkingSpace  WorkArea#1
v3]   ReleaseWorkingSpace  AdminArea

v10]  Output ReplyLength characters, OutOf&DeAccessing BigIOArea,
                                                              at NextPageStart
p23]  go to DUATInput
```

```
[          ***************** D U A T   INPUT-MESSAGES *********************


cell DUATSimpleInput

      CodeLength 6 CodeUnits
      UseCPU 11 CPUUnits
p24]  go to DUATNormalOutput


cell DUATPageBackward

      CodeLength 6 CodeUnits
      UseCPU 5 CPUUnits
      unless NextPageStart = 1,
                                subtract 2 * AverageScreenful, from NextPageStart
p26]  go to DUATPageForward


cell DUATPageForward

      CodeLength 6 CodeUnits
      UseCPU 11 CPUUnits
      add AverageScreenful, to NextPageStart
p25]  go to DUATWeatherDisplay
```

```
cell NonRouteDataForDUATFlightPlan

      CodeLength  70 CodeUnits

p104] ReadRecord  LocalFileRecord#,
                        of file DUATCurrentFlightPlanFile + LocalFileIDOffset,
                                                IntoNewWorkingSpace RecordArea
      UseCPU  30 CPUUnits     [editing local flight plan record

p98] WriteRecord  to LocalFileRecord#,
                        of file DUATCurrentFlightPlanFile + LocalFileIDOffset,
                                                  OutOf&Releasing RecordArea
      UseCPU  12 CPUUnits     [ updating screen format

p24]  Go to DUATNormalOutput


cell RequestDUATWeather

      CodeLength 10 CodeUnits

      set NextPageStart, to 1
p104] ReadRecord  LocalFileRecord#,
                              of file DUATRouteRecordFile + LocalFileIDOffset,
                                              IntoNewWorkingSpace  RecordArea
      UseCPU  20 CPUUnits
v3]   ReleaseWorkingSpace RecordArea

p69]  StartWeatherRetrieval  of DUATWeatherMenu
p25]  Go  to DUATWeatherDisplay
```

```
cell StartDUATConversationWithNewFlightPlan

      CodeLength 50 CodeUnits
v2]   AcquireWorkingSpace  BigIOArea, ScenarioDirectedReplyLength characters
      UseCPU  15 CPUUnits
p32]  Go  to OrganizeDUATFlightPlanFormat



cell RouteDataForDUATFlightPlan

      CodeLength 10 CodeUnits
p56]  ConvertRoute  [into RecordArea
      UseCPU  20 CPUUnits     [ to update the screen format
p98] WriteRecord  to LocalFileRecord#,
                              of file  DUATRouteRecordFile + LocalFileIDOffset,
                                                    OutOf&Releasing RecordArea
p32]  UpdateDUATTerminalRecord
p27]  Go  to NonRouteDataForDUATFlightPlan



cell DUATFlightLog

      CodeLength 250 CodeUnits
p104] ReadRecord  LocalFileRecord#,
                              of file DUATRouteRecordFile + LocalFileIDOffset,
                                      IntoNewWorkingSpace RecordArea, ReadOnly
v2]   AcquireWorkingSpace  WorkArea#2, FlightLogWorkArea characters
v2]   AcquireWorkingSpace  BigIOArea, ScenarioDirectedReplyLength characters
      UseCPU  ScenarioDirected#LegsToLog * 1600  CPUUnits
v3]   ReleaseWorkingSpace  RecordArea
v3]   ReleaseWorkingSpace  WorkArea#2

p24]  go to DUATNormalOutput
```

```
cell StartDUATConversationWithFiledFlightPlan

      CodeLength 15 CodeUnits

p32]  UpdateDUATTerminalRecord
      add  100 * UpperHalfOfParameter, to ScenarioDirectedFlightPlan#

      if FlightPlanToBeUpdated
p55]     GetFlightPlanForUpdating  ScenarioDirectedFlightPlan#  [into RecordArea
         UseCPU  25 CPUUnits
p32]     Go  to OrganizeDUATFlightPlanFormat  for ScenarioDirectedFlightPlan#
      else
p55]     GetFlightPlanReadOnly  ScenarioDirectedFlightPlan# [into RecordArea
         UseCPU  25 CPUUnits
p32]     go to OrganizeDUATFlightPlanFormat


cell ResumeDUATConversationWithCurrentFlightPlan

      CodeLength 10 CodeUnits

p104] ReadRecord  LocalFileRecord#,
                        of file DUATCurrentFlightPlanFile + LocalFileIDOffset,
                                                  IntoNewWorkingSpace  RecordArea
v2]   AcquireWorkingSpace  BigIOArea, ScenarioDirectedReplyLength characters
      UseCPU  20 CPUUnits
v3]   ReleaseW  ingSpace  RecordArea

p24]  go to DUATNormalOutput
```

```
cell FileFlightPlanFromDUATConversation

      CodeLength 20 CodeUnits

      add 100 * UpperHalfOfParameter, to ScenarioDirectedFlightPlan#

p32]  UpdateDUATTerminalRecord  [setting PreviousFlightPlan#
      if PreviousFlightPlan# = 0,
v29]            if ScenarioDirectedFlightPlan# = 0, CryFault NoFlightPlan#Given
      if ScenarioDirectedDataBaseStatusOfFlightPlan = NewFlightPlan
v29]     if PreviousFlightPlan# > 0, CryFault  NotANewFlightPlan
p104]    ReadRecord  LocalFileRecord#,
                         of file DUATCurrentFlightPlanFile + LocalFileIDOffset,
                                                  IntoNewWorkingSpace RecordArea
      else
         if PreviousFlightPlan# = 0,
p55]                      GetFlightPlanForUpdating  ScenarioDirectedFlightPlan#
         else
            set ScenarioDirectedFlightPlan#, to PreviousFlightPlan#
p104]       ReadRecord LocalFileRecord#,
                         of file DUATCurrentFlightPlanFile + LocalFileIDOffset,
                                                 IntoNewWorkingSpace  RecordArea

      UseCPU  25 CPUUnits
      set OriginalPort#, to AssociatedLogicalPort# for OwnNaturalFlow#

p56]  FileThisFlightPlan

p32]  UpdateDUATTerminalRecord  recording NoCurrentFlightPlan
p24]  Go to DUATNormalOutput
```

```
cell RequestDUATSendFormat

      CodeLength 6 CodeUnits
      UseCPU  5 CPUUnits                                  [to prepare the format
p24]  Go  to DUATNormalOutput


cell ExecuteDUATLinkSend

      CodeLength 11 CodeUnits

v2]   AcquireWorkingSpace  BigIOArea,
                             AverageOutputLength for MessageFromDUAT  characters
      UseCPU 15 CPUUnits
      FormatSecondOrderResponseFromOwnPort  for MessageFromDUAT

p83]  OrganizeOutput to ScenarioDirectedRecipient

p24]  Go to DUATNormalOutput


cell DUATStopBenchmark

      if ParallelWeatherRetrieval,
p69]                            if WeatherRecordCountDown > 0, StopTheRetrieval
v14]  prod AncestralNaturalFlow
v13]  AwaitProd              [which never comes
```

```
[ ************************  D U A T    SUPPORTING CELLS ***************


cell OrganizeDUATFlightPlanFormat  for integer FlightPlan# (default none)

      CodeLength 20 CodeUnits

v2]   if OwnedLength of (RecordArea) = 0, AcquireWorkingSpace  RecordArea,
                                                 FlightPlanRecordLength characters
      UseCPU  20 CPUUnits     [to organize a format for the screen
p98] WriteRecord  to LocalFileRecord#,
                          of file DUATCurrentFlightPlanFile + LocalFileIDOffset,
                                                   OutOf&Releasing  RecordArea
p32]  UpdateDUATTerminalRecord  recording FlightPlan#

p24]  Go to DUATNormalOutput


cell UpdateDUATTerminalRecord  recording integer FlightPlan# (default same)

      CodeLength 15 CodeUnits

p104] ReadRecord  LocalFileRecord#,
                             of file DUATTerminalRecordFile + LocalFileIDOffset,
                                                 IntoNewWorkingSpace RecordArea
      set PreviousFlightPlan#, to RecordedFlightPlan#
      UseCPU  15 CPUUnits     [updating the contents

      unless FlightPlan# = same, set RecordedFlightPlan#, to FlightPlan#
p98] WriteRecord  to LocalFileRecord#,
                             of file DUATTerminalRecordFile + LocalFileIDOffset,
                                                   OutOf&Releasing  RecordArea
```

```
[ ************************************************************************
[ ************************************************************************

[                    S E C T I O N   4

[                 R A D A R  -  M A N A G E M E N T

[                     (BOTH  CELLS  ARE  INNER-SET)

[ ************************************************************************
[ ************************************************************************
```

```
cell InitializeRadarManagement
                                                    [Radar Manager is born here

v14]  prod AncestralNaturalFlow
v13]  AwaitProd

p34]  go to RunTimeRadarManagement
```

```
cell RunTimeRadarManagement  LocalInteger SliceStart

      CodeLength 50 CodeUnitsIfProcessingByHardware, otherwise 100 CodeUnits

      loop
v9]      Acquire&Input  into BigIOArea, DeAccessed

v2]      AcquireWorkingSpace  WorkArea#1, RadarWorkLength characters
         TranslateRadarParametersToWorkArea#1  from BigIOArea

         if ScenarioDirectedInputType = RadarStopBenchmark
v14]        prod AncestralNaturalFlow
v13]        AwaitProd                    [which never comes

         set SliceStart, to 1
         loop #slices times
v5]         AccessPartOfWorkingSpaceForUpdating  BigIOArea,
                                          SliceLength characters, at SliceStart
            IfProcessingByHardwareUseCPU 2000/#slices  CPUUnits, otherwise ...
                  ... 4350/#slices  LargeCPUUnits [1 LargeCPUUnit = 10 CPUUnits
            if SliceStart = 1,
                     FormatSecondOrderResponseFromOwnPort for RadarDistribution

            UpdateTheRadarParameter SliceStart
v5]         DeAccessWorkingSpace BigIOArea

p101]    WriteRecordKeepingWorkingSpace to ScenarioDirectedRadarRecord#,
                                                 of RadarFile, from BigIOArea

v10]     for each AFSS, broadcast  RecordLength of RadarFile characters,
                       via AFSSPhysicalPortID for AFSS, OutOf&Keeping BigIOArea
v3]      ReleaseWorkingSpace  BigIOArea

v2]      AcquireWorkingSpace  BigIOArea, ScenarioDirectedReplyLength characters
         FormatDirectResponse
         Output  ScenarioDirectedReplyLength characters,
                                                 OutOf&Releasing  BigIOArea
v3]      ReleaseWorkingSpace WorkArea#1
```

[**********
[**********

[S E C T I O N 5

[C O M M O N - S E R V I C E I N P U T

[(ALL CELLS ARE INNER-SET)

[**********
[**********

[********** CELLS FOR INPUT-DRIVERS **********

```
cell InitializeInputDriving
                                                    [Input Drivers are born here

      if OwnNaturalFlow# = NaturalFlow#OfInputDriver for AWPLink
         clear InitializationOver
         clear record#
v14]     prod AncestralNaturalFlow [says 'ready to read initializetion messages'
v13]     AwaitProd

v2]      AcquireWorkingSpace WorkArea#1, MinimumLengthForWorkArea#1 words

         loop
v9]         Acquire&Input  into BigIOArea
            TranslateInitializationParametersToWorkArea#1   from BigIOArea

            call cell ReceivingCell for ScenarioDirectedInputType

p49]            [receiving cells are InitializeRescueAlerts
p49]            [                    InitializeFlightPlanDeletions
p45]            [                    ScheduleEndOfScheduling
p98]           [                    StandardInitializationOfFiles
p37]            [                    EndOfInitialization

            if InitializationOver, escape loop

v3]      ReleaseWorkingSpace WorkArea#1

      clear #InputsSoFar

v14]  prod AncestralNaturalFlow     [initialization over for this entry port
v13]  AwaitProd

p37]  go to RunTimeInputDriving
```

```
cell EndOfInitialization
                                [one of the 'Receiving Cells' for NADIN input

      set InitializationOver
      FormatDirectResponse

      Output   20 characters, OutOf&Releasing BigIOArea, via NADINOut



cell RunTimeInputDriving

      LocalIntegers MessageType, InputCoordinationCode
      CodeLength 20 CodeUnits

      clear InputCoordinationCode

      loop
v9]      Acquire&Input into BigIOArea
         step #InputsSoFar
         Translate2Digits  FromCharacter 9, of BigIOArea, setting MessageType
         if MessageType = EndOfBenchmark,
                                        set InputCoordinationCode, to #InputsSoFar
         UseCPU  60 CPUUnits
v14]     Prod CorrespondingInputDistributor for OwnNaturalFlow#, with BigIOArea,
                                                WithDataWord  InputCoordinationCode
         if MessageType = EndOfBenchmark, escape loop

v13]  AwaitProd        [which never comes                                  ;
                                                                           ;
```

```
[ ****************** CELLS FOR INPUT DISTRIBUTORS ******************


cell InitializeInputDistribution
                                                [input distributors are born here

v2]   AcquireWorkingSpace  WorkArea#1, NADINInputWorkLength words

      set CorrespondingOutputRoute,
                        to WhichCommonServiceIORoute CorrespondsTo OwnNaturalFlow#
      clear #FinishedInputs
      set OriginatingPort#, to AssociatedLogicalPort# for OwnNaturalFlow#
v14]  prod AncestralNaturalFlow
                        [initialization over for distribution from this entry port

p39]  go to RunTimeInputDistribution
```

```
cell RunTimeInputDistribution

      CodeLength 12 CodeUnits

      clear LastInputRead

      loop
v13]     AwaitProdWithData receiving BigIOArea,
                                        ReceivingDataWord InputCoordinationCode
         UseCPU  35 CPUUnits

         if InputCoordinationCode > 0
            set LastInputRead
            set Total#OfInputsRead, to InputCoordinationCode - 1
                                       [to exclude EndOfBenchmark from the count

         if InputCoordinationCode = OffspringDying, step #FinishedInputs
         if InputCoordinationCode = 0       [normal case

            if ParallelDistributionOfNADINInput,
v11]p40]              SpawnTemporarily StartingAt CallRecipient, with BigIOArea
p37]        else
p40]           CallRecipient
               step #FinishedInputs

         if LastInputRead, if #FinishedInputs = Total#OfInputsRead, escape loop

v14]  prod AncestralNaturalFlow
v13]  AwaitProd                 [which never comes
```

```
cell CallRecipient

      LocalIntegers parent, ParentsProcessor
      CodeLength 30 CodeUnits

      if ParallelDistributionOfNADINInput
         CollectInheritance  BigIOArea

v2]      AcquireWorkingSpace  WorkArea#1, NADINInputWorkLength characters

         set OriginatingPort#, to AssociatedLogicalPort# for parent
         set CorrespondingOutputRoute,
                                 to WhichCommonServiceIORoute CorrespondsTo parent

      TranslateScenarioParametersToWorkArea#1  from BigIOArea
      UseCPU  70 CPUUnits

p94]  MakeLegal&AnalysisRecording

      PossibleMigrationAtNextStatement  taking BigIOArea

      call  cell  ReceivingCell for ScenarioDirectedInputType

p56]                [receiving cells are FlightPlanFromAnotherComputer
p73]                [                    IndividualUpdateFromAWP
p73]                [                    HourlyUpdateFromAWP
p52]                [                    AFOSGraphicsFromAWP
p77]                [                    MailMessage
p86]                [                    AcknowledgementFromAnotherComputer

      if ScenarioDirectedReplyLength > 0
v2]      AcquireWorkingSpace BigIOArea, ScenarioDirectedReplyLength characters
         FormatDirectResponse
p83]     OrganizeOutput  to CorrespondingOutputRoute

v14]  if ParallelDistributionOfNADINInput, Prod&Die prodding parent,
                              in ParentsProcessor, WithDataWork OffspringDying
```

```
[ ***************************************************************************
[ ***************************************************************************

[                          S E C T I O N    6

[                       T I M E   M A N A G E M E N T

[ ***************************************************************************
[ ***************************************************************************

        [NOTE the following arrangements

        [SharedWorkingSpace of Scheduler has 3 single items:
        [       TimeOfNextEvent  (in minutes)
        [       RecordWithEarliestEvent  (the record# in the scheduling files)
        [       SlotWithEarliestEvent  (the slot # in that record)

        [   and 2 tables indexed by record#:
        [       EarliestTime  (in minutes)
        [       SlotWithEarliestTime

        [each scheduling record is used in RecordArea and holds 3 tables:
        [       SlotTime  (in minutes)
        [       SlotAction
        [       SlotParameter


[       ************  P A R T  I :  P R I M E - S O U R C E  ************
```

```
[ ******************  CELLS FOR TIME MANAGER ******************************

                    [Time Manager organizes the execution of scheduled actions

cell InitializeTimeManagement
                                                  [Time Managers are born here

v14]  prod AncestralNaturalFlow                   [initialization over
v13]  AwaitProd

p43]  go to RunTimeTimeManagement


cell OneSecondRoutine

      CodeLength 15 CodeUnits
      UseCPU 5 CPUUnits


cell StopDoingScheduledActions
                                 [a scheduled cell
      PossibleMigrationPoint     [technical only

v14]  prod AncestralNaturalFlow
v13]  AwaitProd                  [which never comes
```

```
cell RunTimeTimeManagement

      LocalIntegers TimeNow, NextTime, record#, slot#

      CodeLength 12 CodeUnits

      loop

v15]     SuspendMe 1 second, setting TimeNow
         UseCPU  25 CPUUnits
p42]     OneSecondRoutine

         loop
v6]         AccessSharedWorkingSpace  of scheduler, ReadOnlyAllowingReadOnly
            set NextTime, to TimeOfNextEvent
            set record#, to RecordWithEarliestEvent
            set slot#, to SlotWithEarliestEvent
v7]         DeAccessSharedWorkingSpace

            if NextTime > TimeNow/60, escape loop
p48]        else, InitiateScheduledEvent for NextTime, in record#, at slot#
```

```
cell InitiateScheduledEvent  for integer ThisTime,
                             in integer record#,
                             at integer slot#

      LocalIntegers action, parameter

      IncludeTables ScheduledCell
      CodeLength 45 CodeUnits

v2]   AcquireWorkingSpace  WorkArea#1, TimeManagementWorkLength characters
p104] ReadRecord  record#, of SchedulingFile, IntoNewWorkingSpace RecordArea

      if SlotTime for (slot#) = ThisTime

         set action, to SlotAction for slot#
         set parameter, to SlotParameter for slot#

         UseCPU  50 CPUUnits

v3]      ReleaseWorkingSpace  RecordArea
         PossibleMigrationAtNextStatement

         call cell ScheduledCell for action,  DummyParameter,  parameter

p57]                  [Scheduled cells are TransmitFlightPlan
p58]                  [                    DeleteFlightPlan
p58]                  [                    StartRescueOperations
p42]                  [                    StopDoingScheduledActions

v14]     prod scheduler, WithDataWords (-action, and -ThisTime, also parameter)
v13]     AwaitReturnProd

v3]   ReleaseWorkingSpace  WorkArea#1
```

```
[ ******************  P A R T   I I :  S E R V I C E - H A L F  **********


[ ****************** OUTER-SET  RECEPTION-DESK  CELLS  **********************


cell Schedule  ActionToSchedule action,
               at integer time,
               WithData integer parameter (default 0)

      CodeLength 8 CodeUnits
      UseCPU  15 CPUUnits

v14]  prod  scheduler,  WithDataWords (action, and time, also parameter)
v13]  AwaitReturnProd


cell Unschedule  ActionToSchedule action,
                 at integer time,
                 WithData integer parameter (default 0)

      CodeLength 7 CodeUnits
      UseCPU  10 CPUUnits

v14]  prod scheduler,  WithDataWords (-action, and time, also parameter)


cell ScheduleEndOfScheduling

p42]  Schedule  StopDoingScheduledActions,
                         at ScenarioDirectedTimeToStop + 100*UpperHalfOfParameter
v3]   ReleaseWorkingSpace  BigIOArea
```

```
[ ********************** CELLS FOR SCHEDULER ********************************

                                [Scheduler accepts updates to the scheduling files


cell InitializeScheduling
                                [Schedulers are born here

      LocalIntegers record#, slot#

v2]   AcquireWorkingSpace  WorkArea#1, SchedulingWorkLength characters

      set SchedulingSpan, to SchedulingIncrement * #SchedulingRecords

v6]   AccessOwnSharedWorkingSpace  ReadWriteSoleAccess
      set TimeOfNextEvent, to tomorrow
      with (record#, from 1, to #SchedulingRecords),
                                        set EarliestTime for record#, to tomorrow
v7]   DeAccessSharedWorkingSpace

v2]   AcquireWorkingSpace  RecordArea, SchedulingRecordLength  characters
      with (slot#, from 1, to #SlotsPerSchedulingRecord),
                                            set SlotTime for slot#, to tomorrow
v14]  prod AncestralNaturalFlow
v13]  AwaitProd                 [waiting for permission to write to the data base

      with (record#, from 1, to #SchedulingRecords),
p99]                              WriteNewRecordKeepingWorkingSpace  to record#,
                                             of SchedulingFile, from RecordArea
v3]   ReleaseWorkingSpace  RecordArea
v14]  prod AncestralNaturalFlow
                                          [scheduling files are now initialized
p47]  go to RunTimeScheduling
```

```
cell RunTimeScheduling

      LocalIntegers ProddingNaturalFlow, proddingProcessor, action, time,
                                                        parameter, fault, record#
      LocalConditions SlotFound, ReplyExpected
      CodeLength 50 CodeUnits

      loop
v13]     AwaitProdFromClient  setting (action, and time, also parameter)
                              [action negated means 'unschedule'
                              [time negated means the prod came from Time Manager

         clear ReplyExpected

         if action > 0, set ReplyExpected
         if time < 0, set ReplyExpected
         if time < 0, negate time

         set record#, to 1 +
              (time - SchedulingSpan*(time/SchedulingSpan))/ SchedulingIncrement
```

```
        loop #SchedulingRecords times

           clear SlotFound
p104]      ReadRecord  record#, of SchedulingFile,
                                                  IntoNewWorkingSpace RecordArea
           UseCPU  55 CPUUnits

           with (slot#, from 1, to #SlotsPerSchedulingRecord)

              if action > 0 [scheduling case], if SlotTime for slot# = tomorrow
                 set SlotTime for slot#, to time
                 set SlotAction for slot#, to action
                 set SlotParameter for slot#, to parameter
                 set SlotFound

              if action < 0  unscheduling casee, if SlotTime for slot# = time,
                                 if SlotParameter for slot# = parameter,
                                              if SlotAction for slot# + action = 0
                 set SlotFound
                 set SlotTime for slot#, to tomorrow

              if SlotFound
p49]             UpdateSharedWorkingSpace  HavingUpdated slot#, in record#,
                                                              for action, at time
p98]            WriteRecord  to record#, of SchedulingFile,
                                                     OutOf&Releasing RecordArea
                 if ReplyExpected, ProdWhoeverProddedMe
                 escape loop

           step record#
v3]        ReleaseWorkingSpace RecordArea
           if record# > #SchedulingRecords, set record#, to 1

        unless SlotFound
           if action > 0, set fault, to SchedulingFilesFull
           else, set fault, to FailedToUnschedule
v29]       CryFault fault
```

```
cell UpdateSharedWorkingSpace  HavingUpdated integer slot#,
                               in integer record#,
                               for integer action,
                               at integer time

      CodeLength 25 CodeUnits

v6 ]  AccessOwnSharedWorkingSpace  ReadWriteSoleAccess

      UseCPU  25 CPUUnits

      if action > 0                    [scheduling case
         if time < EarliestTime for record#
            set EarliestTime for record#, to time
            set SlotWithEarliestTime for record#, to slot#

         if time < TimeOfNextEvent
            set TimeOfNextEvent, to time
            set RecordWithEarliestEvent, to record#
            set SlotWithEarliestEvent, to slot#

      else                   [unscheduling case
         if time = EarliestTime for record#
            set EarliestTime for record#, to tomorrow
            with (slot, from 1, to #SlotsPerSchedulingRecord),
                       if SlotTime for slot < EarliestTime for record#
               set EarliestTime for record#, to SlotTime for slot
               set SlotWithEarliestTime for record#, to slot

         if time = TimeOfNextEvent
            set TimeOfNextEvent, to tomorrow
            with (record, from 1, to #SchedulingRecords),
                                 if EarliestTime for record < TimeOfNextEvent
               set TimeOfNextEvent, to EarliestTime for record
               set RecordWithEarliestEvent, to record
               set SlotWithEarliestEvent, to SlotWithEarliestTime for record

v7 ]  DeAccessSharedWorkingSpace                [both cases
```

[This page left blank intentionally

[**********
[**********

[S E C T I O N 7

[G R A P H I C S

[**********
[**********

[********** INNER-SET RECEPTION-DESK CELL **********

```
cell AFOSGraphicsFromAWP

      LocalInteger record#
      PossibleMigrationPoint  receiving BigIOArea

      CodeLength 120 CodeUnits

      Randomise   ScenarioDirectedAFOSRecord#,
                                          of AFOSGraphicsFile, setting record#
      if OwnedLength of (BigIOArea) < RecordLength of AFOSGraphicsFile,
v2]           ExtendWorkingSpace  BigIOArea, to RecordLength of AFOSGraphicsFile

      UseCPU  115 CPUUnits

p101] WriteRecordKeepingWorkingSpace
                                 to record#, of AFOSGraphicsFile, from BigIOArea

      FormatSecondOrderResponse  for GraphicsDistribution, from AWPLogicalPort#

v10]  for each AFSS, broadcast RecordLength of AFOSGraphicsFile characters,
                        via AFSSPhysicalPortID for AFSS, OutOf&Keeping BigIOArea
      for each DUATCluster,
v10]             broadcast RecordLength of AFOSGraphicsFile characters,
                                 via DUATClusterPhysicalPortID for DUATCluster,
                                                        OutOf&Keeping BigIOArea
v3]   ReleaseWorkingSpace BigIOArea
```

```
[ ***************************************************************************
[ ***************************************************************************

[                    S E C T I O N    8

[              F L I G H T - P L A N   M A N A G E M E N T

[ ***************************************************************************
[ ***************************************************************************
```



```
[ ************** OUTER-SET RECEPTION-DESK CELLS:  INITIALIZATION **********


cell InitializeRescueAlerts

v3]   ReleaseWorkingSpace BigIOArea
p55]  InitializeRescueAlertsPtII  updating record#



cell InitializeFlightPlanDeletions

v3]   ReleaseWorkingSpace BigIOArea
p56]  InitializeFlightPlanDeletionsPtII  updating record#
```

[this page is left blank intentionally

```
[ ************  INNER-SET CELLS  FOR INITIALIZATION  **********


cell InitializeRescueAlertsPtII  updating integer record#

      PossibleMigrationPoint

v2]   AcquireWorkingSpace  RecordArea, FlightPlanRecordLength characters
      CollectInitializationParameters
      with (i, from 1, to 10)
         step record#

         clear RecordedDestination
         set RecordedScheduledAction, to StartRescueOperations
         set RecordedScheduleTime, to ScenarioDirectedInitialTime # i

p99]    WriteNewRecordKeepingWorkingSpace to record#,
                                          of FlightPlanFile,  from RecordArea
p58]     Schedule  StartRescueOperations,
                                     WithData record#,at RecordedScheduleTime

v3]   ReleaseWorkingSpace  RecordArea
```

```
cell InitializeFlightPlanDeletionsPtII updating integer record#

      PossibleMigrationPoint

v2]   AcquireWorkingSpace  RecordArea, FlightPlanRecordLength characters
      CollectInitializationParameters

      with (i, from 1, to 10)
         step record#
         clear RecordedDestination
p58]     set RecordedScheduledAction, to DeleteFlightPlan
         set RecordedScheduleTime, to ScenarioDirectedInitialTime # i

p99]    WriteNewRecordKeepingWorkingSpace  to record#,
                                            of FlightPlanFile, from RecordArea
p45]p58] Schedule   DeleteFlightPlan, WithData record#, at RecordedScheduleTime

v3]   ReleaseWorkingSpace  RecordArea
```

[********** INNER-SET RECEPTION-DESK CELLS (1): SCHEDULED **********

```
cell TransmitFlightPlan  record WhichPlane

      PossibleMigrationPoint
      CodeLength 110 CodeUnits

v2]   AcquireWorkingSpace WorkArea#2, WorkAreaForFlightPlanTimedCheck characters
p105] ReadRecordForUpdating   record WhichPlane,
                              of FlightPlanFile, IntoNewWorkingSpace RecordArea

v2]   AcquireWorkingSpace  BigIOArea, FlightPlanTransmissionLength characters
      FormatIndirectOutput  for ScheduledFlightPlanTransmission,
                                                          OutputID WhichPlane
      UseCPU  115 CPUUnits
p83]  OrganizeOutput to RecordedDestination

      add FlightPlanTimeOut, to RecordedScheduleTime
p58]  set RecordedScheduledAction, to DeleteFlightPlan
p99] WriteRecord&UnlockIt  to record WhichPlane,
                              of FlightPlanFile,  OutOf&Releasing RecordArea
v3]   ReleaseWorkingSpace  WorkArea#2

p45]
p58]  Schedule  DeleteFlightPlan, at RecordedScheduleTime, WithData WhichPlane
```

```
cell StartRescueOperations  record WhichPlane

      LocalInteger time
      PossibleMigrationPoint
      CodeLength 15 CodeUnits

v2]   AcquireWorkingSpace  BigIOArea,
                                   AverageOutputLength for RescueAlert characters
      FormatIndirectOutput  for RescueAlert, OutputID WhichPlane
      UseCPU  45 CPUUnits

      output AverageOutputLength for RescueAlert characters,
               OutOf&Keeping BigIOArea, via PhysicalPortForSupervisoryPosition#1
      output AverageOutputLength for RescueAlert characters,
               OutOf&Keeping BigIOArea, via PhysicalPortForSupervisoryPosition#2
      output AverageOutputLength for RescueAlert characters,
               OutOf&Keeping BigIOArea, via PhysicalPortForSupervisoryPosition#3
p83]  OrganizeOutput  to OtherHubs

p105] ReadRecordForUpdating record WhichPlane,
                                 of FlightPlanFile, IntoNewWorkingSpace RecordArea
      clear RecordedScheduledAction
p99] WriteRecord&UnlockIt to record WhichPlane,
                                   of FlightPlanFile,  OutOf&Releasing RecordArea



cell DeleteFlightPlan record WhichPlane

      PossibleMigrationPoint
      CodeLength 10 CodeUnits

      UseCPU 15 CPUUnits
p94]  MakeLegal&AnalysisRecording
p106] DeleteRecord  record WhichPlane,  of FlightPlanFile
```

[********* INNER-SET RECEPTION-DESK CELLS (2): SCENARIO-DRIVEN (RUN-TIME) ***

```
cell GetFlightPlanReadOnly  record WhichFlightPlan

      PossibleMigrationPoint  ReturningWith RecordArea
      CodeLength 7 CodeUnits
      UseCPU  11 CPUUnits

p104] ReadRecord  record WhichFlightPlan,
                              of FlightPlanFile,  IntoNewWorkingSpace RecordArea


cell GetFlightPlanForUpdating record WhichFlightPlan

      PossibleMigrationPoint ReturningWith RecordArea
      CodeLength 7 CodeUnits
      UseCPU 11 CPUUnits

p105] ReadRecordForUpdating record WhichFlightPlan,
                               of FlightPlanFile, IntoNewWorkingSpace RecordArea
```

```
cell FileThisFlightPlan

      PossibleMigrationPoint receiving RecordArea

      CodeLength 20 CodeUnits
      UseCPU  100 CPUUnits

p61]  DealWithFlightPlan


cell FlightPlanFromAnotherComputer

      PossibleMigrationPoint receiving BigIOArea

      CodeLength 10 CodeUnits

      add 100 * UpperHalfOfParameter, to ScenarioDirectedFlightPlan#

      if ScenarioDirectedDataBaseStatusOfFlightPlan = NewFlightPlan,
v2]           AcquireWorkingSpace  RecordArea, FlightPlanRecordLength characters

p105] else, ReadRecordForUpdating  ScenarioDirectedFlightPlan#,
                                   of FlightPlanFile, IntoNewWorkingSpace RecordArea
      UseCPU  25 CPUUnits

      set OriginalPort#, to OriginatingPort#    [from WorkArea#1 to RecordArea
v3]   ReleaseWorkingSpace  BigIOArea

p61]  DealWithFlightPlan
```

```
[ ********************** INNER-SET  SUPPORTING  CELL  **********************


cell DealWithFlightPlan

      LocalIntegers ActionToUnschedule, UnscheduleTime, TimeNow, OriginatingPort

      CodeLength 120 CodeUnits

v2]   AcquireWorkingSpace  WorkArea#2, FlightPlanWorkArea characters

v16]  AskTime  setting TimeNow
      divide TimeNow, by 60

      unless ScenarioDirectedDataBaseStatusOfFlightPlan = NewFlightPlan
         set ActionToUnschedule, to RecordedScheduledAction
         set UnscheduleTime, to RecordedScheduleTime

      set RecordedDestination, to ScenarioDirectedFlightPlanDestination
      set RecordedScheduledAction, to ScenarioDirectedFPScheduledAction
      set RecordedScheduleTime,
                              to TimeNow + ScenarioDirectedFlightPlanScheduleTime

      UseCPU  95 CPUUnits

      set OriginatingPort, to OriginalPort#    [latter is in RecordArea
```

```
      if ScenarioDirectedDataBaseStatusOfFlightPlan = NewFlightPlan,
p96]                WriteNewRecord  to ScenarioDirectedFlightPlan#,
                                  of FlightPlanFile,  OutOf&Releasing RecordArea

      if ScenarioDirectedDataBaseStatusOfFlightPlan = PreviouslyFiledFlightPlan,
p99]              WriteRecord&UnlockIt  to ScenarioDirectedFlightPlan#,
                                   of FlightPlanFile,  OutOf&Releasing RecordArea

      if ScenarioDirectedDataBaseStatusOfFlightPlan = FlightPlanToBeDeleted
p106]    DeleteRecord ScenarioDirectedFlightPlan#,
                                            of FlightPlanFile,  currently locked
v3]      ReleaseWorkingSpace  RecordArea
p94]     MakeLegal&AnalysisRecording

      if ScenarioDirectedTransmitNow?
v2]      AcquireWorkingSpace  BigIOArea, FlightPlanTransmissionLength characters
         FormatSecondOrderResponse  for ImmediateFlightPlanTransmission,
                                                          from  OriginatingPort
p83]     OrganizeOutput  to ScenarioDirectedFlightPlanDestination

      unless ScenarioDirectedDataBaseStatusOfFlightPlan = FlightPlanToBeDeleted,
p45]              Schedule  ScenarioDirectedFPScheduledAction,
                         at  TimeNow + ScenarioDirectedFlightPlanScheduleTime,
                                        WithData  ScenarioDirectedFlightPlan#

      unless ScenarioDirectedDataBaseStatusOfFlightPlan = NewFlightPlan,
                     unless ActionToUnschedule = 0,
                             Unschedule  ActionToUnschedule, at UnscheduleTime,
                                            WithData ScenarioDirectedFlightPlan#

v3]   ReleaseWorkingSpace  WorkArea#2
```

```
[ ************************************************************************
[ ************************************************************************

[                          S E C T I O N   9

[                    R O U T E     C O N V E R S I O N

[                          (ALL CELLS ARE INNER-SET)

[ ************************************************************************
[ ************************************************************************
```

```
[ ****************************   RECEPTION DESK   ***************************


cell ConvertRoute

      PossibleMigrationPoint  ReturningWith RecordArea
      CodeLength 245 CodeUnits

v2]   unless ConvertingRoutesInParallel, AcquireWorkingSpace ...
                    ... WorkArea#2, WorkAreaForRouteConversion characters

      with (ThisElement, from 1, to ScenarioDirected#RouteElements)
         if ConvertingRouteElementsInParallel,
v11]p68]         SpawnTemporarily  StartingAt  RouteElementsInParallel,
                                  WithCopyOf WorkArea#1, PlusDataWord ThisElement
p26]     else
p66]        ConvertElement  ThisElement
            UseCPU  5000 CPUUnits

      if ConvertingRouteElementsInParallel,
                              loop  ScenarioDirected#RouteElements  times
v13]     AwaitProd  receiving RecordArea
         UseCPU  5000 CPUUnits
v3]      ReleaseWorkingSpace RecordArea

v2]   AcquireWorkingSpace RecordArea, RecordLength of RouteRecordFile characters
v3]   unless ConvertingRouteElementsInParallel, ReleaseWorkingSpace WorkArea#2
```

```
[ ************************    SUPPORTING  CELLS   *************


cell RouteElementsInParallel

      LocalIntegers MyParent,ParentsProcessor, ThisElement
      CodeLength 30 CodeUnits

      CollectInheritance  1 DataItem, and WorkArea#1

v2]   AcquireWorkingSpace  WorkArea#2, WorkAreaForElementConversion characters

p66]  ConvertElement ThisElement
      UseCPU 80 CPUUnits

v14]  Prod&Die  prodding MyParent, in ParentsProcessor,  with WorkArea#2
```

```
cell ConvertElement  integer ThisElement

      LocalIntegers FirstLeg, Airways&FixesRecord

      CodeLength 30 CodeUnits

      Randomise  ScenarioDirectedAirways&FixesRecord, of Airways&FixesFile,
                   by ScenarioDirectedAirways&FixesOffset, instance ThisElement,
                                                        setting Airways&FixesRecord
p104] ReadRecord  Airways&FixesRecord, of Airways&FixesFile,
                                                     IntoNewWorkingSpace RecordArea
v3]   ReleaseWorkingSpace RecordArea

      set FirstLeg, to ScenarioDirected#LegsPerElement * (ThisElement - 1)  +  1

      with (leg#, from FirstLeg,
                            to  FirstLeg + ScenarioDirected#LegsPerElement - 1)
         if ConvertingLegsInParallel,
v11]p67]      SpawnTemporarily StartingAt LegsInParallel, WithCopyOf WorkArea#1,
                                          PlusDataWords (leg#, and ThisElement)
         else
p68]        ConvertLeg leg#, of ThisElement
            UseCPU  315 CPUUnits

      if ConvertingLegsInParallel, loop ScenerioDirected#LegsPerElement times
v13]     AwaitProd  receiving RecordArea
         UseCPU  315 CPUUnits
v3]      ReleaseWorkingSpace  RecordArea
```

```
cell LegsInParallel

      LocalIntegers MyParent, ParentsProcessor, ThisLeg, ThisElement
      CodeLength 30 CodeUnits

      CollectInheritance  2 DataItems, and WorkArea#1
v2]   AcquireWorkingSpace  WorkArea#2, WorkAreaForLegConversion characters

p68]  ConvertLeg ThisLeg, of ThisElement
      UseCPU 80 CPUUnits

v14]  Prod&Die  prodding MyParent, in ParentsProcessor, with WorkArea#2
```

```
cell ConvertLeg  integer ThisLeg,
                 of integer ThisElement

      LocalIntegers LocationsRecord, ThisInstance

      CodeLength 320 CodeUnits

      with (RouteRecord, from 1, to  ScenarioDirected#RecordsPerLeg)
         set ThisInstance, to ScenarioDirected#RecordsPerLeg * (ThisLeg +
                                 ScenarioDirected#LegsPerElement * ThisElement)
         add RouteRecord, to ThisInstance

         randomise ScenarioDirectedFirstLocationsRecord,
                  of ReportingLocationsFile, by ScenarioDirectedLocationsOffset,
                                 instance  ThisInstance, setting LocationsRecord

         if RequestingRouteRecordsSimultaneously,
v11]p69]           SpawnTemporarily StartingAt RequestRouteRecordSimultaneously,
                                                   WithDataWord LocationsRecord
         else
p69]        RequestRouteRecordSerially getting LocationsRecord
            UseCPU   1000 * ScenarioDirected#LocationsPerRecord  CPUUnits
v3]         ReleaseWorkingSpace RecordArea

      if RequestingRouteRecordsSimultaneously,
                              loop ScenarioDirected#RecordsPerLeg times
v13]     AwaitProd receiving RecordArea
         UseCPU   1000 * ScenarioDirected#LocationsPerRecord  CPUUnits
v3]      ReleaseWorkingSpace RecordArea
```

```
cell RequestRouteRecordSimultaneously

      LocalIntegers MyParent, ParentsProcessor, LocationsRecord

      CodeLength 30 CodeUnits

      CollectInheritance 1 DataItem
v2]   AcquireWorkingSpace WorkArea#1, MinimumLengthForWorkArea#1 words
p104] ReadRecord LocationsRecord,
                     of ReportingLocationsFile,  IntoNewWorkingSpace RecordArea
      UseCPU  25  CPUUnits

v14]  Prod&Die prodding MyParent, in ParentsProcessor, with RecordArea


cell RequestRouteRecordSerially  getting integer LocationsRecord

      CodeLength 30 CodeUnits

p104] ReadRecord  LocationsRecord,
                      of ReportingLocationsFile, IntoNewWorkingSpace RecordArea
      UseCPU 25 CPUUnits
```

[**********
[**********

[S E C T I O N 1 0

[W E A T H E R M A N A G E M E N T

[**********
[**********

[********** OUTER-SET RECEPTION-DESK CELLS **********

```
cell StartWeatherRetrieval  of WeatherMenu WhichMenu

      LocalIntegers fileID, CountDownStart

      CodeLength 45 CodeUnits

      clear WeatherRecordCountDown

      with (PositionInMenu, from 1, to 10),
                              add ScenarioDirected#Records for PositionInMenu,
                                                        to WeatherRecordCountDown
      set CountDownStart, to WeatherRecordCountDown

      with (PositionInMenu, from 1, to 10)
                         unless ScenarioDirected#Records for PositionInMenu = 0
         UseCPU  8 CPUUnits

         if WhichMenu = SpecialistWeatherMenu, set fileID,
                         to SpecialistWeatherFile CorrespondingTo PositionInMenu
         else, set fileID, to DUATWeatherFile CorrespondingTo PositionInMenu

         if ParallelWeatherRetrieval
v11]p74]    SpawnTemporarily  StartingAt GetIndividualWeatherInParallel,
                 WithCopyOf WorkArea#1,
                       PlusDataWords (CountDownStart, and FileID,
                              also ScenarioDirected#Records for PositionInMenu)
            subtract ScenarioDirected#Records for PositionInMenu,
                                                        from CountDownStart
   p74]  else, GetIndividualWeatherInSeries from FileID,
                                 ScenarioDirected#Records for PositionInMenu
```

```
cell EnsureEnoughWeatherToDisplay

      CodeLength 20 CodeUnits
      UseCPU  10 CPUUnits

      loop
         if WeatherRecordCountdown = 0, return
         unless OwnedLength of (BigIOArea) < NextPageStart ...
                                    ... + ScenarioDirectedReplyLength - 1, return
v13]     AwaitProdWithKey  WithMessageKey WeatherRecordCountDown,
                                       Extending BigIOArea, replying IfLastProd
         subtract 1, from WeatherRecordCountDown


cell StopTheRetrieval

      LocalInteger LastCountIfEndOfBatch
      CodeLength 25 CodeUnits
      UseCPU  40 CPUUnits

      loop
v13]     AwaitProdWithKey&Data  WithMessageKey WeatherRecordCountDown,
                 Extending BigIOArea, ReceivingDataWord LastCountIfEndOfBatch,
                                                              replying DropDead
         loop
            subtract 1, from WeatherRecordCountDown
            if LastCountIfEndOfBatch > 0, escape loop
v13]        else, AwaitProdWithKey&Data  WithMessageKey WeatherRecordCountDown,
                  Extending BigIOArea, ReceivingDataWord LastCountIfEndOfBatch

         set WeatherRecordCountDown, to LastCountIfEndOfBatch - 1
         if WeatherRecordCountDown = 0, escape loop
```

```
cell HourlyUpdateFromAWP

      LocalIntegers file#, FirstRecord#, LastRecord#

      PossibleMigrationPoint receiving BigIOArea     [technical only

      CodeLength 25 CodeUnits
      UseCPU 25 CPUUnits

      if OwnedLength of (BigIOArea) < RecordLength of ScenarioDirectedFileID,
v2]                        ExtendWorkingSpace  BigIOArea,
                                     to RecordLength of ScenarioDirectedFileID

      set file#, to ScenarioDirectedFileID
      set FirstRecord#, to 100 * ThirdUpperHalf + ScenarioDirectedStartRecord#
      set LastRecord#, to FirstRecord# + (100 * SecondUpperHalf) +
                                        ScenarioDirected#BulkUpdateRecords - 1
p107] BulkUpdate  of file#, from FirstRecord#, to LastRecord#


cell IndividualUpdateFromAWP

      LocalInteger record#
      PossibleMigrationPoint  receiving BigIOArea    [technical only

      CodeLength 120 CodeUnits

      if OwnedLength of (BigIOArea) < RecordLength of ScenarioDirectedFileID,
v2]                        ExtendWorkingSpace  BigIOArea,
                                     to RecordLength of ScenarioDirectedFileID

      randomise ScenarioDirectedWeatherRecord#,
                              of file ScenarioDirectedFileID, setting record#
      UseCPU 115 CPUUnits

p98] WriteRecord  to record record#, of ScenarioDirectedFileID,
                                                   OutOf&Releasing BigIOArea
```

```
[ ************************  INNER-SET  SUPPORTING CELLS ************************


cell GetIndividualWeatherInParallel
      LocalIntegers MyParent, ParentsProcessor, CountDownStart, file#, #records

      CodeLength 25 CodeUnits

      CollectInheritance  3 DataItems, and WorkArea#1

      UseCPU 20 CPUUnits

p75]  GetIndividualWeather #records, from file#, StartingAt CountDownStart,
                              OnBehalfOf MyParent, ParentBeingIn ParentsProcessor



cell GetIndividualWeatherInSeries integer #records,
                                  from file file#

      PossibleMigrationPoint  ReturningWith BigIOArea
      CodeLength 25 CodeUnits
      UseCPU  20 CPUUnits

p75]  GetIndividualWeather  #records, from file#
```

```
cell GetIndividualWeather  OnBehalfOf integer MyParent (default 0),
                           integer #RecordsToRead,
                           from file file#,
                           StartingAt integer CountDown (default 0),
                           ParentBeingIn integer processor# (default 0)

      LocalIntegers record#, LastCount, CountDownProdKey
      LocalConditions ToldToDie?

      CodeLength 270 CodeUnits

      clear ToldToDie?

p109] if ItsBulkUpdate InTheCaseOf file#, StartBatchRead updating WhichFile
      set LastCount, to 1 + CountDown - #RecordsToRead
```

```
        with (RecordCount, from 1, to #RecordsToRead)
           Randomise ScenarioDirectedWeatherRecordStarter, of file file#,
                                        by ScenarioDirectedWeatherRecordIncrement,
                                              instance RecordCount, setting record#
           if RequestingWeatherRecordsSimultaneously
              set CountDownProdKey, to CountDown - RecordCount + 1
v11]p77]      SpawnTemporarily StartingAt RequestWeatherRecordSimulataneously,
                    WithDataWords (record#, and file#, also CountDownProdKey)
           else
p78]          RequestWeatherRecordSerially getting record#, OutOf file#
              UseCPU 3500 CPUUnits
p77]          DigestWeatherRecord OutOf file#
              if ItsBulkUpdate InTheCaseOf file#
p110]            if ToldToDie?, EndOfBatchRead for file#
p110]            else, if RecordCount = #RecordsToRead, EndOfBatchRead for file#
              if ParallelWeatherRetrieval
v14]             if ToldToDie?, Prod&Die  prodding MyParent, in processor#,
                       with RecordArea, PlusDataWords (CountDown, and LastCount)
                 if RecordCount = #RecordsToRead, ProdThenDieWhenToldTo ...
v14]                     ... prodding MyParent, in Processor#, with RecordArea,
                                     PlusDataWords (CountDown, and LastCount)
v14]             Prod&CollectReply prodding MyParent, in processor#,
                                  with RecordArea, PlusDataWords (CountDown,
                                              and ImStillAlive), setting ToldToDie?
                 subtract 1, from CountDown
v4]           else, MoveWorkingSpace  RecordArea ToBigIOArea

        if RequestingWeatherRecordsSimultaneously,
                                     with (RecordCount, from 1, to #RecordsToRead)
v13]       AwaitProdWithKey WithMessageKey CountDown, receiving RecordArea
           UseCPU 3500 CPUUnits
p77]       DigestWeatherRecord OutOf file#
           if RecordCount < #RecordsToRead
v14]          Prod MyParent, in processor#, with RecordArea,
                                       PlusDataWords (CountDown, and ImStillAlive)
              subtract 1, from CountDown
           else
              if ItsBulkUpdate InTheCaseOf file#, EndOfBatchRead for file#
v14]          ProdThenDieWhenToldTo prodding MyParent, in processor#,
                       with RecordArea, PlusDataWords (CountDown, and LastCount)
```

```
cell DigestWeatherRecord OutOf file WhichFile

      CodeLength 20 CodeUnits

p78]  if WhichFile = FGFile, WindRetrieval
p78]  if WhichFile = FGAlternateFile, WindRetrieval

      UseCPU 50 CPUUnits
v7]   DeAccessWorkingSpace RecordArea
```



```
cell RequestWeatherRecordSimultaneously

      LocalIntegers MyParent, ParentsProcessor, record#,
                                          WhichFile, CountDownProdKey

      CodeLength 20 CodeUnits

      CollectInheritance 3 DataItems
v2]   AcquireWorkingSpace WorkArea#1, MinimumLengthForWorkArea#1 words
p104] ReadRecord record#, of WhichFile, IntoNewWorkingSpace RecordArea
      UseCPU 50 CPUUnits

v14]  Prod&Die prodding MyParent, in ParentsProcessor,
                                 with RecordArea, PlusDataWord CountDownProdKey
```

```
cell RequestWeatherRecordSerially getting integer record#,
                                  OutOf file WhichFile

      CodeLength 20 CodeUnits
p104] ReadRecord record#, of WhichFile, IntoNewWorkingSpace RecordArea
      UseCPU 50 CPUUnits


cell WindRetrieval

      CodeLength 205 CodeUnits
      UseCPU  160 CPUUnits
p78]  go to WindComputations


cell WindComputations

      CodeLength  275 CodeUnits
      UseCPU  6500 CPUUnits
```

```
[ ************************************************************************
[ ************************************************************************

[                    S E C T I O N   1 1

[            M E S S A G E - P O O L   M A N A G E M E N T

[ ************************************************************************
[ ************************************************************************
```



```
[ ****************  INNER-SET  RECEPTION-DESK  CELL  **************************
```

```
cell MessageToScreen  from integer SelectedManager

      LocalIntegers MessageRecord#, AdminRecord#, FamilyOffset

      PossibleMigrationPoint ReturningWith BigIOArea
      CodeLength 20 CodeUnits

      set FamilyOffset, to SelectedManager - 1
      Randomise  ScenarioDirectedMessageListRecord#,
                          of file TerminalMessageListFile, setting AdminRecord#

p104] ReadRecord  AdminRecord#, of file TerminalMessageListFile + FamilyOffset,
                                IntoNewWorkingSpace RecordArea
      UseCPU 15 CPUUnits
v3]   ReleaseWorkingSpace RecordArea

      Randomise ScenarioDirectedMessageRecord#, of file MessageFile,
                                                        setting MessageRecord#
p104] ReadRecord  MessageRecord#, of file  MessageFile + FamilyOffset,
                                                  IntoNewWorkingSpace BigIOArea

v6]   AccessSharedWorkingSpace  of  MessageManager + FamilyOffset,
                                                          ReadWriteSoleAccess
      UseCPU  15 CPUUnits
v7]   DeAccessSharedWorkingSpace

p105] ReadRecordForUpdating  AdminRecord#,
                                of file TerminalMessageListFile + FamilyOffset,
                                                IntoNewWorkingSpace RecordArea
      UseCPU  15 CPUUnits
p99] WriteRecord&UnlockIt  to AdminRecord#,
                                of file TerminalMessageListFile + FamilyOffset,
                                                     OutOf&Releasing RecordArea

p106] if  ScenarioDirectedTimeToDelete?,  DeleteRecord  MessageRecord#,
                                        of  file  MessageFile + FamilyOffset
```

```
[ ****************  OUTER-SET  RECEPTION-DESK  CELL  ************************

cell MailMessage

      PossibleMigrationPoint  receiving BigIOArea   [technical only

      CodeLength 10 CodeUnits
      UseCPU  20 CPUUnits

      for each IndividualMessageManager,
v14]            prod NaturalFlow#OfMessageManager for IndividualMessageManager,
                                                          WithCopyOf WorkArea#1
v3]   ReleaseWorkingSpace BigIOArea

v13]  for each IndividualMessageManager, AwaitReturnProd
```

[This page left blank intentionally

```
[          ************* CELLS  FOR  MESSAGE-MANAGER  **********


cell InitializeMessageManagement
                              [Message Managers are born here.  At run-time their
                              [     job is to accept messages into the pool

      set ThisManager, to OwnNaturalFlow# - MessageManager + 1
      set MessageFileID, to MessageFile + ThisManager - 1
      set TerminalMessageListFileID,
                                 to TerminalMessageListFile + ThisManager - 1

v14]  prod AncestralNaturalFlow

p83]  go to RunTimeMessageManagement



cell RunTimeMessageManagement

      LocalIntegers ProddingNaturalFlow, ProddersProcessor, RecordInstance,
                                                            record#, specialist#

      CodeLength 215 CodeUnits
```

```
        loop
v13]      AwaitProdFromClient  receiving WorkArea#1

v2]       AcquireWorkingSpace  RecordArea, RecordLength of MessageFile characters
          Randomise  ScenarioDirectedMessageRecord#, of file MessageFileID,
                                                                  setting record#
p98]     WriteRecord  to record#, of MessageFileID, OutOf&Releasing RecordArea

          with (RecordInstance, from 1, to ScenarioDirected#OfListUpdates)
             Randomise ScenarioDirectedMessageListStarter,
                      of file TerminalMessageListFileID, instance RecordInstance,
                         by ScenarioDirectedMessageListIncrement, setting record#
p105]        ReadRecordForUpdating  record#, of TerminalMessageListFileID,
                                                   IntoNewWorkingSpace RecordArea
             UseCPU  25 CPUUnits
p99]        WriteRecord&UnlockIt  to record#, of TerminalMessageListFileID,
                                                      OutOf&Releasing RecordArea

v6]       AccessOwnSharedWorkingSpace  ReadWriteSoleAccess
          UseCPU  30 CPUUnits
v7]       DeAccessSharedWorkingSpace

v14]      ProdWhoeverProddedMe

          set Specialist#, to ScenarioDirectedFirstMessageReceiver ...
                                          ... + NaturalFlow#OffsetForSpecialists
v2]       AcquireWorkingSpace  BigIOArea, MessageWaitingAlertLength characters
          FormatSecondOrderResponse  for MessageWaitingAlert,
                                                          from OriginatingPort#
          loop
             if Specialist# > LastSpecialist#, escape loop
             if ThisManager = SelectedMessageManager for specialist#,
v10]                 broadcast MessageWaitingAlertLength characters,
                                 via AssociatedPhysicalPortID for specialist#,
                                                      OutOf&Keeping BigIOArea
             add ScenarioDirectedMessageReceiverIncrement, to Specialist#
v3]          ReleaseWorkingSpace BigIOArea

v3]       ReleaseWorkingSpace WorkArea#1
```

```
[ ************************************************************************
[ ************************************************************************

[                    S E C T I O N    1 2

[              C O M M O N - S E R V I C E   O U T P U T

[                      (ALL CELLS ARE INNER-SET)

[ ************************************************************************
[ ************************************************************************
```

```
[NOTE: SharedWorkingSpace of OutputDrivers holds the following items:
[              NextEmptyRecord
[              NextOutputToGo
[              IODriverIsBusy
[              FileBeingWrittenTo
[              BenchmarkOver
```

```
[ ************************** RECEPTION-DESK CELLS ********************


cell AcknowledgementFromAnotherComputer

      LocalIntegers FileID, record#, OutputDriver

      PossibleMigrationPoint receiving BigIOArea     [an unlikely one

      CodeLength 15 CodeUnits
      UseCPU  25 CPUUnits

      set FileID, to OutputFileID for CorrespondingOutputRoute
      set OutputDriver,
                     to NaturalFlow#OfOutputDriver for CorrespondingOutputRoute

v3]   ReleaseWorkingSpace  BigIOArea

v6]   AccessSharedWorkingSpace of OutputDriver, ReadOnlyAllowingReadOnly
      set record#, to NextEmptyRecord            [to get random record#
v7]   DeAccessSharedWorkingSpace

p106] DeleteRecord  record#, of FileID
```

```
cell OrganizeOutput  to CommonServiceIORoute OutputRoute

      LocalInteger OutputRecord#
      LocalConditions MustTryAgain, DriverToBeProdded

      PossibleMigrationPoint  receiving BigIOArea

      CodeLength 35 CodeUnits
      UseCPU 60 CPUUnits

p94]  MakeLegal&AnalysisRecording

      if OwnedLength of (BigIOArea) < RecordLength of OutputFiles,
v2]                 ExtendWorkingSpace  BigIOArea, to RecordLength of OutputFiles

      loop
v6]      AccessSharedWorkingSpace of NaturalFlow#OfOutputDriver for OutputRoute,
                                                                  ReadWriteSoleAccess
         if FileBeingWrittenTo, set MustTryAgain
         else
            set FileBeingWrittenTo
            clear MustTryAgain
            set OutputRecord#, to NextEmptyRecord
v7]      DeAccessSharedWorkingSpace

         if MustTryAgain,
v15]                  SuspendMeMomentarily  WithSuspensionKey 512 + OutputRoute
         else, escape loop

p101] WriteRecordKeepingWorkingSpace to record OutputRecord#,
                                  of OutputFileID for OutputRoute, from BigIOArea
```

```
v6 ]   AccessSharedWorkingSpace of NaturalFlow#OfOutputDriver for OutputRoute,
                                                                ReadWriteSoleAccess
       step NextEmptyRecord
       if NextEmptyRecord > #Records in OutputFiles, set NextEmptyRecord, to 1
v29]   if NextEmptyRecord = NextOutputToGo, CryFault OutputSpoolOverflow
       clear FileBeingWrittenTo

       if IODriverIsBusy
v3 ]      ReleaseWorkingSpace  BigIOArea
          clear DriverToBeProdded
       else
          set IODriverIsBusy
          set DriverToBeProdded

v7 ]   DeAccessSharedWorkingSpace

v14]   if DriverToBeProdded, prod  NaturalFlow#OfOutputDriver for OutputRoute,
                                                                  with BigIOArea
```

```
cell DrainSpool  for CommonServiceIORoute ThisOutputRoute

      LocalInteger driver

      PossibleMigrationPoint

      set driver, to NaturalFlow#OfOutputDriver for ThisOutputRoute

v6]   AccessSharedWorkingSpace of driver, ReadWriteSoleAccess
      set BenchmarkOver
v7]   DeAccessSharedWorkingSpace

v2]   AcquireWorkingSpace  BigIOArea, 20 characters

      FormatIndirectOutput  for EndOfSpooledOutput
p83]  OrganizeOutput to ThisOutputRoute
```

```
[ ************  CELLS  FOR  OUTPUT-DRIVERS  ********************************


cell InitializeOutputDriving
                                                    [output drivers are born here

      LocalInteger RouteID

v2]   AcquireWorkingSpace  AdminArea, AdminAreaLength words

      set RouteID, to WhichCommonServiceIORoute CorrespondsTo OwnNaturalFlow#
      set File#ForThisSpool, to OutputFileID for RouteID

v6]   AccessOwnSharedWorkingSpace ReadWriteSoleAccess
      set NextEmptyRecord
      set NextOutputToGo
      clear BenchmarkOver
      clear IODriverIsBusy
      clear FileBeingWrittenTo
v7]   DeAccessSharedWorkingSpace

v2]   AcquireWorkingSpace  WorkArea#1, OutputDriverWorkLength characters

v14]  prod AncestralNaturalFlow
                                                    [initialization is over
p91]  go to RunTimeOutputDriving
```

```
cell RunTimeOutputDriving

      LocalIntegers OutputLength, type, NextOutput
      LocalConditions MoreToOutput, AllOver

      CodeLength 45 CodeUnits

      clear AllOver
      set NextOutput
```

```
        loop
v13]       AwaitProd  receiving BigIOArea
           UseCPU  75 CPUUnits
           set MoreToOutput

           loop
              EstablishOutputType  setting type
              if type = DirectResponse, EstablishOutputLength setting OutputLength
              else, set OutputLength, to AverageOutputLength for type

              if OutputLength  >  OwnedLength of BigIOArea,
v2]                                ExtendWorkingSpace  BigIOArea, to OutputLength
v10]          Output  OutputLength characters, OutOf&Keeping BigIOArea

p98]         WriteRecord  to record NextOutput,
                                  of File#ForThisSpool, OutOf&Releasing BigIOArea

v6]           AccessOwnSharedWorkingSpace  ReadWriteSoleAccess
              step NextOutputToGo
              if NextOutputToGo > #Records in OutputFiles,
                                                      set NextOutputToGo, to 1
              set NextOutput, to NextOutputtoGo
              if NextOutputToGo = NextEmptyRecord
                 clear IODriverIsBusy
                 clear MoreToOutput
                 if BenchmarkOver, set AllOver

v7]           DeAccessSharedWorkingSpace

p104]         if MoreToOutput, ReadRecord  record NextOutput,
                               of File#ForThisSpool, IntoNewWorkingSpace BigIOArea
              else, escape loop

           if AllOver
v14]          prod AncestralNaturalFlow
v13]          AwaitProd                                [which never comes
```

```
[ ***************************************************************************
[ ***************************************************************************

[                          S E C T I O N    1 3

[                L E G A L   &   A N A L Y S I S    R E C O R D I N G

[                        (ALL CELLS ARE INNER-SET)

[ ***************************************************************************
[ ***************************************************************************
```

```
        [NOTE: SharedWorkingSpace of Legal&AnalysisDrivers holds the following:
        [                #MessagesInBuffer
        [                BufferIsFull
        [                the buffer itself
```

```
[ *************************  RECEPTION-DESK  CELL **********************
```

```
cell MakeLegal&AnalysisRecording

      LocalIntegers WhichProcessor, driver
      LocalConditions DriverToBeProdded, DriverBusy
      PossibleMigrationPoint
      CodeLength 30 CodeUnits

v12]  AskWhichProcessorIamOn  setting WhichProcessor
      set driver, to RecommendedLegal&AnalysisDriver for WhichProcessor

      loop
v6]      AccessSharedWorkingSpace  of driver, ReadWriteSoleAccess

         if BufferIsFull, set DriverBusy
         else
            UseCPU  55 CPUUnits
            clear DriverBusy
            step #MessagesInBuffer
            if #MessagesInBuffer = Max#MessagesInLegal&AnalysisBuffer
               set BufferIsFull
               set DriverToBeProdded
            else, clear DriverToBeProdded

v7]      DeAccessSharedWorkingSpace

v15]     if DriverBusy, SuspendMeMomentarily WithSuspensionKey 256 + driver
         else
v14]        if DriverToBeProdded, prod driver, WithDataWord KeepGoing
            escape loop

      .
      .
```

```
cell MakeLegal&AnalysisRecording

      LocalIntegers WhichProcessor, driver
      LocalConditions DriverToBeProdded, DriverBusy
      PossibleMigrationPoint
      CodeLength 30 CodeUnits

v12]  AskWhichProcessorIamOn  setting WhichProcessor
      set driver, to RecommendedLegal&AnalysisDriver for WhichProcessor

      loop
v6]      AccessSharedWorkingSpace  of driver, ReadWriteSoleAccess

         if BufferIsFull, set DriverBusy
         else
            UseCPU  55 CPUUnits
            clear DriverBusy
            step #MessagesInBuffer
            if #MessagesInBuffer = Max#MessagesInLegal&AnalysisBuffer
               set BufferIsFull
               set DriverToBeProdded
            else, clear DriverToBeProdded

v7]      DeAccessSharedWorkingSpace

v15]     if DriverBusy, SuspendMeMomentarily WithSuspensionKey 256 + driver
         else
v14]        if DriverToBeProdded, prod driver, WithDataWord KeepGoing
            escape loop

      .
      .
```

```
[ ***************   CELLS  FOR  LEGAL-&-ANALYSIS  DRIVERS   *****************


cell InitializeLegal&AnalysisDriving
                                                    [Legal&Analysis drivers are born here
                                                              .

v6]   AccessOwnSharedWorkingSpace  ReadWriteSoleAccess
      clear BufferIsFull
      clear #MessagesInBuffer
v7]   DeAccessSharedWorkingSpace

v14]  prod AncestralNaturalFlow
                                                         [ initialisation is over
p96] go to Legal&AnalysisDriving
                                                              .
```

```
cell Legal&AnalysisDriving

      LocalCondition AllOver?
      CodeLength 25 CodeUnits

      clear AllOver?

      loop
v13]     AwaitProdWithData  ReceivingDataWord AllOver?

v2]      AcquireWorkingSpace  BigIOArea, Legal&AnalysisBufferSize characters

v6]      AccessOwnSharedWorkingSpace ReadWriteSoleAccess
         FormatIndirectOutput for Legal&AnalysisRecording
         UseCPU  35 CPUUnits
v10]     Output  Legal&AnalysisBufferSize characters, OutOf&Releasing BigIOArea

         clear BufferIsFull
         clear #MessagesInBuffer
v7]      DeAccessSharedWorkingSpace

         if AllOver?
v14]        prod AncestralNaturalFlow
v13]        AwaitProd                          [which never comes
```

```
[ ***********************************************************************
[ ***********************************************************************

[                        S E C T I O N   1 4

[                  D A T A - B A S E   M A N A G E M E N T

[ ***********************************************************************
[ ***********************************************************************


          [note the following arrangements:

          [SharedWorkingSpace is used for the MasterDirectory.
          [It holds 2 arrays indexed by FileID:
          [        DirectorySpaceID (negated means 'file directory locked')
          [        AlternateDirectorySpaceID
          [                (zero means 'Fixed Information Directory')
          [                (negated means 'other copy is current')

          [and 4 arrays indexed by membership number in family of
          [bulk-update files:
          [           FileUserCount
          [           AlternateFileUserCount
          [           AlternateFileIsCurrent?
          [           #BulkUpdateRecordsRead

          [DirectoryArea is used for accessing File Directories, which have
          [one array indexed by record# (>0):
          [         RecordSpaceID
          [                  (zero means 'record deleted')
          [                  (negated means 'record locked')


          [WorkArea#2 holds 2 arrays indexed by membership number in family
          [of Bulk-update files:
          [           CountDown
          [           #RepliesExpected
```

```
[ ************  OUTER-SET  RECEPTION-DESK  CELL:  INITIALIZATION  ******


cell StandardInitializationOfFiles
                                    [driven from scenario initialization messages

      LocalInteger record#

      for each file, if StandardInitialization for file
         if file = RadarFile, AcquireWorkingSpace RecordArea,
                                 RecordLength of RadarFile characters, DeAccessed
v2]      else,  AcquireWorkingSpace RecordArea,
                                      RecordLength of file characters, ReadWrite
         with record#, from 1, to #records in file,
p99]                 WriteNewRecordKeepingWorkingSpace  to record#,
                                                        of file, from RecordArea
v3]      ReleaseWorkingSpace RecordArea

v3]   ReleaseWorkingSpace BigIOArea
```

[********** INNER-SET RECEPTION-DESK CELLS (1): NORMAL RUN-TIME *******

```
cell WriteNewRecordKeepingWorkingSpace to record record#,
                                       of file file#,
                                       from WorkingSpace WhichWS

     PossibleMigrationPoint  ReceivingCopyOf DesignatedWorkingSpace

     CodeLength 15 CodeUnits
     UseCPU  20 CPUUnits

     set ThisRecord, to record#
     set ThisFile, to file#
     set SpecifiedWorkingSpace, to WhichWS
     set RequiredAccessCode, to SameCodeAsBefore
     set ExpectedRecordState, to EmptyRecord
     set RequiredRecordState, to unlocked

p114] CollectFileStatistics

     go to cell ExtensionToWriteNewRecord for ThisFileType

p107]        [DirectoryNotModelled: to WriteRecordDirect
p109]        [FixedInformationDirectory: to WriteRecord&Directory
p109]        [FixedAddressRecords: to WriteRecord&Directory
p109]        [VariableAddressRecords: to WriteVariableAddressRecord
```

```
cell WriteNewRecord  to record record#,
                     of file file#,
                     OutOf&Releasing WorkingSpace WhichWS

      PossibleMigrationPoint  receiving DesignatedWorkingSpace

      CodeLength 15 CodeUnits
      UseCPU  20 CPUUnits

      set ThisRecord, to record#
      set ThisFile, to file#
      set SpecifiedWorkingSpace, to WhichWS
      set RequiredAccessCode, to empty
      set ExpectedRecordState, to EmptyRecord
      set RequiredRecordState, to unlocked

p114] CollectFileStatistics

      go to cell ExtensionToWriteNewRecord for ThisFileType

p107]        [DirectoryNotModelled: to WriteRecordDirect
p109]        [FixedInformationDirectory: to WriteRecord&Directory
p109]        [FixedAddressRecords: to WriteRecord&Directory
p109]        [VariableAddressRecords: to WriteVariableAddressRecord
```

```
cell WriteRecordKeepingWorkingSpace  to record record#,
                                     of file file#,
                                     from WorkingSpace WhichWS

      PossibleMigrationPoint  ReceivingCopyOf DesignatedWorkingSpace

      CodeLength 15 CodeUnits
      UseCPU  20 CPUUnits

      set ThisRecord, to record#
      set ThisFile, to file#
      set SpecifiedWorkingSpace, to WhichWS
      set RequiredAccessCode, to SameCodeAsBefore
      set ExpectedRecordState, to MaybeLocked
      set RequiredRecordState, to AsItWas

p114] CollectFileStatistics

      go to cell ExtensionToWriteRecord for ThisFileType

p107]            [DirectoryNotModelled: to WriteRecordDirect
p109]            [VariableAddressRecords: to WriteVariableAddressRecord
p107]            [else: to WriteRecordSimpleCase
```

```
cell WriteRecord  to record record#,
                  of file file#,
                  OutOf&Releasing WorkingSpace WhichWS

      PossibleMigrationPoint  receiving DesignatedWorkingSpace

      CodeLength 15 CodeUnits
      UseCPU  20 CPUUnits

      set ThisRecord, to record#
      set ThisFile, to file#
      set SpecifiedWorkingSpace, to WhichWS
      set RequiredAccessCode, to empty
      set ExpectedRecordState, to MaybeLocked
      set RequiredRecordState, to AsItWas

p114] CollectFileStatistics

      go to cell ExtensionToWriteRecord for ThisFileType

p107]             [DirectoryNotModelled: to WriteRecordDirect
p109]             [VariableAddressRecords: to WriteVariableAddressRecord
p107]             [else: to WriteRecordSimpleCase
```

```
cell WriteRecord&UnlockIt  to record record#,
                           of file file#,
                           OutOf&Releasing WorkingSpace WhichWS

      PossibleMigrationPoint   receiving DesignatedWorkingSpace

      CodeLength  15 CodeUnits
      UseCPU  20 CPUUnits

      set ThisRecord, to record#
      set ThisFile, to file#
      set SpecifiedWorkingSpace, to WhichWS
      set RequiredAccessCode, to empty
      set ExpectedRecordState, to locked
      set RequiredRecordState, to unlocked

p114] CollectFileStatistics

      go to cell ExtensionToWriteRecord&UnlockIt for ThisFileType

p109]         [FixedAddressRecords: to WriteRecord&Directory
p109]         [VariableAddressRecords: to WriteVariableAddressRecord
              [else: disallowed
```

```
cell ReadRecord  record record#,
                 of file file#,
                 AccessCode HowAccessIt default ReadWrite,
                 IntoNewWorkingSpace WorkingSpace WhichWS

     PossibleMigrationPoint  ReturningWith DesignatedWorkingSpace

     CodeLength  15 CodeUnits
     UseCPU 20 CPUUnits

     set ThisRecord, to record#
     set ThisFile, to file#
     set SpecifiedWorkingSpace, to WhichWS
     set RequiredAccessCode, to HowAccessIt
     set ExpectedRecordState, to MaybeLocked
     set RequiredRecordState, to AsItWas

p114] CollectFileStatistics

     go to cell ExtensionToReadRecord for ThisFileType

p113]                    [DirectoryNotModelled: to ReadRecordDirect
p113]                    [FixedInformationDirectory: to ReadRecordSimpleCase
p114]                    [else: to ReadRecordHoldingDirectory
```

```
cell ReadRecordForUpdating  record record#,
                            of file file#,
                            AccessCode HowAccessIt default (ReadWrite),
                            IntoNewWorkingSpace WorkingSpace WhichWS

      PossibleMigrationPoint  ReturningWith DesignatedWorkingSpace
      CodeLength  15 CodeUnits

      set ThisRecord, to record#
      set ThisFile, to file#
      set SpecifiedWorkingSpace, to WhichWS
      set RequiredAccessCode, to HowAccessIt
      set ExpectedRecordState, to unlocked
      set RequiredRecordState, to locked

p114] CollectFileStatistics

p116] GetLockedDirectoryUpdateEntry&GetSpaceID
      UseCPU  100 CPUUnits
p114] WriteDirectory&UnlockIt
v20]  ReadFromPermanentMemory  RecordLengthForThisFile characters,
               from ThisRecord, of ThisFile, at SpaceIDForThisRecord,
                              to SpecifiedWorkingSpace, AccessCode HowToAccessIt
```

```
cell DeleteRecord  record record#,
                   of file file#
                   currently LockState state (default unlocked)

      PossibleMigrationPoint

      CodeLength  20 CodeUnits

      set ThisRecord, to record#
      set ThisFile, to file#
      set ExpectedRecordState, to state
      if PretendDeletion for ThisFile, set RequiredRecordState, to PretendEmpty
      else, set RequiredRecordState, to EmptyRecord

p114] CollectFileStatistics

p116] unless ThisFileType = DirectoryNotModeled,
                                    GetLockedDirectoryUpdateEntry&GetSpaceID
      UseCPU  100 CPUUnits
p114] unless ThisFileType = DirectoryNotModeled, WriteDirectory&UnlockIt

      unless PretendDeletion for ThisFile,
v19]                ReleasePermanentMemory  RecordLengthForThisFile characters,
                          for ThisRecord, of ThisFile, at SpaceIDForThisRecord

      TraceDeletion of record#, in file#   [only compiled into debugging version
```

[********** INNER-SET RECEPTION-DESK CELLS (2): BULK-UPDATE RUN-TIME ****

```
cell BulkUpdate  from integer FirstRecord#,
                 of file file#,
                 to integer LastRecord#

      LocalIntegers IndexToFamily, controller, DirectoryID
      LocalCondition CompleteTrainArrived?

      PossibleMigrationPoint  receiving BigIOArea
      CodeLength 100 CodeUnits
```

```
      set IndexToFamily, to BulkUpdateIndex for file#
      set controller, to ControllingBulkUpdateManager for IndexToFamily

v6]   AccessSharedWorkingSpace of controller, ReadWriteSoleAccess
      if AlternateFileIsCurrent? for IndexToFamily, set ThisFile, to file#
      else, set ThisFile, to AlternateFileID for IndexToFamily

      UpdateMessageTrainCount for IndexToFamily, Setting CompleteTrainArrived?,
                                   HavingReadFrom FirstRecord#, to LastRecord#
v7]   DeAccessSharedWorkingSpace

      clear ThisRecord
p114] CollectFileStatistics

      if ThisFileType = DirectoryNotModelled, clear DirectoryID
      else
         set DirectoryID, to DirectoryArea
p116]    GetLockedDirectoryUpdateEntry&GetSpaceID
                                          [just gets directory if ThisRecord= 0

      UseCPU  55 CPUUnits

v23]  BulkWrite  RecordLengthForThisFile characters, from BigIOArea,
          to FirstRecord#, UpTo LastRecord#, of ThisFile, SpaceIDsTo DirectoryID

p114] unless ThisFileType = DirectoryNotModelled, WriteDirectory&UnlockIt

v3]   ReleaseWorkingSpace BigIOArea
v14]  if CompleteTrainArrived?, prod controller,
                                  WithDataWords (EndOfBulkUpdate, and ThisFile)
```

```
cell StartBatchRead  integer.processor# (default 0),
                     updating file file#

     LocalInteger IndexToFamily

     PossibleMigrationPoint

     CodeLength 20 CodeUnits
     UseCPU  20 CPUUnits

v12] AskWhichProcessorImOn  setting processor#
     set IndexToFamily, to BulkUpdateIndex for file#
v6]  AccessSharedWorkingSpace  of LocalBulkUpdateManager for (IndexToFamily +
                     #BulkUpdateFiles * (processor# - 1)), ReadWriteSoleAccess

     if AlternateFileIsCurrent? for IndexToFamily
        set file#, to AlternateFileID for IndexToFamily
        step AlternateFileUserCount for IndexToFamily
     else, step FileUserCount for IndexToFamily

v7]  DeAccessSharedWorkingSpace
```

```
cell EndOfBatchRead  integer processor# (default 0),
                     for file file#,
                     integer IndexToFamily (default 0),

      LocalCondition FileIsObsolete

      PossibleMigrationPoint

      CodeLength 15 CodeUnits
      UseCPU  20 CPUUnits

      clear FileIsObsolete
v12]  AskWhichProcessorIamOn setting processor#
      set IndexToFamily, to BulkUpdateIndex for file#

v6]   AccessSharedWorkingSpace of LocalBulkUpdateManager for (IndexToFamily +
                        #BulkUpdateFiles * (processor# - 1)), ReadWriteSoleAccess

      if ThisIsTheAlternateFile? InTheCaseOf file#
         subtract 1, from AlternateFileUserCount for IndexToFamily
         unless AlternateFileIsCurrent? for IndexToFamily, set FileIsObsolete
      else
         subtract 1, from FileUserCount for IndexToFamily
         if AlternateFileIsCurrent? for IndexToFamily, set FileIsObsolete

v7]   DeAccessSharedWorkingSpace

v14]  if FileIsObsolete, prod ControllingBulkUpdateManager for IndexToFamily,
                                          WithDataWords (EndOfRead, and file#)
```

[********** EXTENSIONS TO INNER-SET RECEPTION-DESKS **********

```
cell WriteRecordDirect

      GoToTypeExtensionOfMigration

      CodeLength 10 CodeUnits
      UseCPU  20 CPUUnits

      WriteToPermanentMemory  RecordLengthForThisFile characters,
v21 or v22]   from SpecifiedWorkingSpace, at SpaceIDForThisRecord,
                        with RequiredAccessCode, to ThisRecord, of ThisFile


cell WriteRecordSimpleCase

      GoToTypeExtensionOfMigration

      CodeLength 10 CodeUnits

p115] GetSpaceIDForRecord
      UseCPU  30 CPUUnits
      WriteToPermanentMemory  RecordLengthForThisFile characters,
v21 or v22]   from SpecifiedWorkingSpace, with RequiredAccessCode,
                             to ThisRecord, of ThisFile, at SpaceIDForThisRecord
```

```
cell WriteRecord&Directory

      GoToTypeExtensionOfMigration
      CodeLength 20 CodeUnits

p116] GetLockedDirectoryUpdateEntry&GetSpaceID
      UseCPU  80 CPUUnits

      WriteToPermanentMemory  RecordLengthForThisFile characters,
v21 or v22]   from SpecifiedWorkingSpace, with RequiredAccessCode,
                              to ThisRecord, of ThisFile, at SpaceIDForThisRecord
p114] WriteDirectory&UnlockIt



cell WriteVariableAddressRecord

      LocalInteger NewSpaceID
      GoToTypeExtensionOfMigration
      CodeLength 12 CodeUnits

v18]  AcquirePermanentMemory  RecordLengthForThisFile characters,
                                 for ThisRecord, of ThisFile, setting NewSpaceID
      WriteToPermanentMemory  RecordLengthForThisFile characters,
v21 or v22]    from SpecifiedWorkingSpace, with RequiredAccessCode,
                                       to ThisRecord, of ThisFile, at NewSpaceID

      DirectoryNotReallyBeingUpdated
p116] GetLockedDirectoryUpdateEntry&GetSpaceID
      UseCPU  80 CPUUnits

      set RecordSpaceID for ThisRecord, to NewSpaceID
p114] WriteDirectory&UnlockIt

      unless ExpectedRecordState = EmptyRecord,
v19]                ReleasePermanentMemory  RecordLengthForThisFile characters,
                              for ThisRecord, of ThisFile, at SpaceIDForThisRecord
```

```
cell ReadRecordDirect

      GoToTypeExtensionOfMigration
      CodeLength 10 CodeUnits
      UseCPU  20 CPUUnits

v20]  ReadFromPermanentMemory  RecordLengthForThisFile characters, ReadOnly,
                        from ThisRecord, of ThisFile, to SpecifiedWorkingSpace,
                                                    AccessCode RequiredAccessCode


cell ReadRecordSimpleCase

      GoToTypeExtensionOfMigration
      CodeLength 10 CodeUnits

p115] GetSpaceIDForRecord
      UseCPU  30 CPUUnits

v20]  ReadFromPermanentMemory  RecordLengthForThisFile characters,
                  from ThisRecord, of ThisFile, at SpaceIDForThisRecord,
                        to SpecifiedWorkingSpace, AccessCode RequiredAccessCode
```

```
cell ReadRecordHoldingDirectory

      GoToTypeExtensionOfMigration
      CodeLength 15 CodeUnits
      DirectoryNotReallyBeingUpdated

p116] GetLockedDirectoryUpdateEntry&GetSpaceID
v3]   ReleaseWorkingSpace DirectoryArea

v20]  ReadFromPermanentMemory  RecordLengthForThisFile characters,
                    from ThisRecord, of ThisFile, at SpaceIDForThisRecord,
                         to SpecifiedWorkingSpace,  AccessCode RequiredAccessCode
      UseCPU  40 CPUUnits

v6]   AccessSharedWorkingSpace  of DataBaseManagerForThisFile,
                                                          ReadWriteSoleAccess
      negate DirectorySpaceID for ThisFile
v7]   DeAccessSharedWorkingSpace
```

```
[ ************************** SUPPORTING  CELLS  **********************


cell GetSpaceIDForRecord

      CodeLength 15 CodeUnits

v6]   AccessSharedWorkingSpace  of DataBaseManagerForThisFile,
                                ReadOnlyAllowingReadOnly,
                                      DirectorySizeForThisFile CharactersNeededSoon
      set SpaceIDForThisDirectory, to  AlternateDirectorySpaceID for ThisFile

      if SpaceIDForThisDirectory <= 0,
                 set SpaceIDForThisDirectory, to  DirectorySpaceID for ThisFile
v20]  ReadDirectoryFromPermanentMemoryToDirectoryArea for ThisFile, ReadOnly
               DirectorySizeForThisFile characters, from SpaceIDForThisDirectory
v7]   DeAccessSharedWorkingSpace

      UseCPU  40 CPUUnits
p117] Find&UpdateRecordEntry
v3]   ReleaseWorkingSpace  DirectoryArea
```

```
cell GetLockedDirectoryUpdateEntry&GetSpaceID

      CodeLength 15 CodeUnits

      loop
v6]      AccessSharedWorkingSpace of DataBaseManagerForThisFile,
                    ReadWriteSoleAccess, DirectorySizeForThisFile +
                                 RecordLengthForThisFile  CharactersNeededSoon
         set SpaceIDForThisDirectory, to  DirectorySpaceID for ThisFile
         set SpaceIDForAlternateDirectory,
                                    to  AlternateDirectorySpaceID for ThisFile
         if SpaceIDForThisDirectory > 0,
                           unless ThisFileType = FixedInformationDirectory,
                                        negate DirectorySpaceID for ThisFile
v7]      DeAccessSharedWorkingSpace

         if SpaceIDForThisDirectory > 0, escape loop
v15]     else, SuspendMeMomentarily  WithSuspensionKey ThisFile

      if ThisFileType = FixedInformationDirectory,
                  set SpaceIDForAlternateDirectory, to SpaceIDForThisDirectory
      if SpaceIDForAlternateDirectory < 0
         negate SpaceIDForAlternateDirectory
         exchange SpaceIDForAlternateDirectory, with SpaceIDForThisDirectory

      UseCPU  40 CPUUnits

v20]  ReadDirectoryFromPermanentMemoryToDirectoryArea for ThisFile,
          DirectorySizeForThisFIle characters, from SpaceIDForAlternateDirectory

p117] unless ThisRecord = 0, Find&UpdateRecordEntry
```

```
cell WriteDirectory&UnlockIt

      CodeLength 20 CodeUnits
      UseCPU 100 CPUUnits

v22]  WriteDirectoryToPermanentMemory&LoseDirectoryArea for ThisFile,
                 DirectorySizeForThisFile characters, to SpaceIDForThisDirectory
      unless ThisFileType = FixedInformationDirectory
v6]      AccessSharedWorkingSpace  of DataBaseManagerForThisFile,
                                                          ReadWriteSoleAccess
         negate  DirectorySpaceID for ThisFile
         negate AlternateDirectorySpaceID for ThisFile
v7]      DeAccessSharedWorkingSpace


cell CollectFileStatistics


      CodeLength 35 CodeUnits
      UseCPU  115 CPUUnits

      set ThisFileType, to FileType for ThisFile
      set DirectorySizeForThisFile, to SizeOfWord * #Records for ThisFile
      set RecordLengthForThisFile, to RecordLength for ThisFile
      set DataBaseManagerForThisFile,
                                 to NaturalFlow#OfDataBaseManager for ThisFile
```

```
[ ******************  CELLS FOR  DATA-BASE MANAGERS  ********************

cell InitializeDataBaseManagement

      LocalIntegers record#, SizeOfDirectory

                                                [data base managers are born here

v2]   AcquireWorkingSpace  AdminArea, AdminAreaLength words
v6]   AccessOwnSharedWorkingSpace ReadWriteSoleAccess
```

```
      for each file, if NaturalFlow#OfDataBaseManager for file = OwnNaturalFlow#
         set SizeOfDirectory, to  SizeOfWord * #records in file

v2]      AcquireWorkingSpace  DirectoryArea, SizeOfDirectory characters
         with (record#, from 1, to #records in file),
                                                clear RecordSpaceID for record#

v18]     AcquirePermanentMemory  SizeOfDirectory characters, for directory,
                                       of file, setting DirectorySpaceID of file
v21]     WriteToPermanentMemory&KeepWorkingSpace from DirectoryArea,
                        SizeOfDirectory characters, to directory, of file,
                                                   at DirectorySpaceID of file

         if FileType of file = FixedInformationDirectory,
                                        clear AlternateDirectorySpaceID of file
         else
v18]        AcquirePermanentMemory SizeOfDirectory characters, for directory,
                              of file, setting AlternateDirectorySpaceID of file
v21]        WriteToPermanentMemory&KeepWorkingSpace from DirectoryArea,
                        SizeOfDirectory characters, to directory, of file,
                                          at AlternateDirectorySpaceID for file
            negate AlternateDirectorySpaceID of file

v3]         ReleaseWorkingSpace  DirectoryArea

v2]   AcquireWorkingSpace  WorkArea#2, 2 * #BulkUpdateFiles words
      with (i, from 1, to #BulkUpdateFiles)
         clear FileUserCount slot i
         clear AlternateFileUserCount slot i
         clear AlternateFileIsCurrent? for i
         clear CountDown for i
         clear #BulkUpdateRecordsRead for i
         clear #RepliesExpected for i

v7]   DeAccessSharedWorkingSpace
v2]   AcquireWorkingSpace  WorkArea#1, MinimumLengthForWorkArea#1 words

v14]  prod AncestralNaturalFlow     [announcing data base ready to use
p121] go to DataBaseRunTime
```

```
cell DataBaseRunTime

      LocalIntegers IndexToFamily, ReasonForProd, file#,
                                    ProdData, NaturalFlow#,CountDownIncrement

      LocalCondition ItsABulkUpdateManager?
                        [at run time the manager looks after bulk-index files

      CodeLength 25 CodeUnits
```

```
      loop
v13]     AwaitProdWith3DataWords  setting (ReasonForProd, and file#,
                                                                 also ProdData)
         UseCPU  15 CPUUnits

         set IndexToFamily, to BulkUpdateIndex for file#

         if ReasonForProd = EndOfBulkUpdate
            clear CountDown for IndexToFamily
            clear #RepliesExpected for IndexToFamily
            for each IndividualDataBaseManager
               set NaturalFlow#,
                           to DataBaseNaturalFlow# of IndividualDataBaseManager
               EstablishBulkUpdateStatus of NaturalFlow#,
                                                 setting ItsABulkUpdateManager

               if ItsABulkUpdateManager
v14]              prod NaturalFlow#,
                            WithDataWords (AskingForCountDown, and file#)
                  step #RepliesExpected for IndexToFamily

         if ReasonForProd = AskingForCountDown
p123]       ReverseBulkUpdateSwitch for IndexToFamily,
                                                 setting CountDownIncrement
v14]        prod ControllingBulkUpdateManager for IndexToFamily,
                     WithDataWords (ReturningACountDown, and file#,
                                                    also CountDownIncrement)

         if ReasonForProd = ReturningACountDown
            add ProdData, to CountDown for IndexToFamily
            subtract 1, from #RepliesExpected for IndexToFamily
p123]       TestIfReadyToDelete file#, WithFamilyMembership IndexToFamily

         if ReasonForProd = EndOfRead
            subtract 1, from CountDown for IndexToFamily
p123]       TestIfReadyToDelete file#, WithFamilyMembership IndexToFamily
```

```
cell ReverseBulkUpdateSwitch for integer IndexToFamily,
                             returning integer CountDown

      CodeLength 8 CodeUnits
      UseCPU  15 CPUUnits

v6 ]  AccessOwnSharedWorkingSpace  ReadWriteSoleAccess
      if AlternateFileIsCurrent? for IndexToFamily
         set CountDown, to AlternateFileUserCount for IndexToFamily
         clear AlternateFileIsCurrent? for IndexToFamily
      else
         set CountDown, to FileUserCount for IndexToFamily
         set AlternateFileIsCurrent? for IndexToFamily
v7 ]  DeAccessSharedWorkingSpace


cell TestIfReadyToDelete file file#,
                         WithFamilyMembership integer IndexToFamily

      LocalInteger DirectoryID


      CodeLength 20 CodeUnits
      UseCPU  35 CPUUnits
```

```
      if CountDown for IndexToFamily = 0,
                                        if #RepliesExpected for IndexToFamily = 0

v6]       AccessOwnSharedWorkingSpace  ReadOnlyAllowingReadOnly
          if AlternateFileIsCurrent? for IndexToFamily,
                                                  set ThisFile, to IndexToFamily
          else, set ThisFile, to AlternateFileID for IndexToFamily
v7]       DeAccessSharedWorkingSpace

          clear ThisRecord
p114]     CollectFileStatistics

          if ThisFileType = DirectoryNotModelled, clear DirectoryID
          else
            set DirectoryID, to DirectoryArea
p116]       GetLockedDirectoryUpdateEntry&GetSpaceID
v24]      DeleteFile  ThisFile, #records in ThisFile InFile,
                                RecordLengthForThisFile CharactersPerRecord,
                                                     SpaceIDsIn DirectoryID
          unless ThisFileType = DirectoryNotModelled
             with (record#, from 1, to #records in ThisFile),
                                             clear RecordSpaceID for record#
p114]        WriteDirectory&UnlockIt


EndOfModel
```

APPENDIX D

MODEL CALIBRATION

D.1 GENERAL

The benchmark demonstration run has three phases; initialization, measured and termination. Those cells which are executed during the measured phase will be calibrated to use the proper amount of storage and CPU time on each vendor's proposed FSDPS configuration. References to cell(s) below are directed only toward those which are executed during the measured period of the benchmark demonstration run.

The calibration will be accomplished as described below under two separate but related topics using two independent sets of measurements on the vendor's proposed FSDPS configuration. The first topic is description of the approach used in determination of cell target sizes for each proposed FSDPS. These are called target sizes since these sizes are to be approximated on the proposed FSDPS. The second topic is a description of the techniques used and measurements made in measuring the model machinery and padding out the cells to the target size.

D.2 CELL TARGET SIZE

Each model cell contains a CodeLength and one or more UseCPU statements. The target sizes of these statements are abstractly identified as CodeUnits and CPUUnits, respectively. Measurements of the Sample Program (provided with the Processor Benchmark) executable size and execution time are used to translate these abstract units to real, measurable units on each vendor's proposed FSDPS. Explained in this section is the methodology used by the Government, and inputs required of the vendors, in determining cell target size (i.e., stor[illegible] and processing time) on each vendors propose[illegible] FSDPS configuration.

The translation of code units to bytes is largely dependent upon the code expansion efficiency of the proposed compiler. However, it is implementation dependent to a limited extent. The implementation dependent portion of the translation depends on whether the library and its data space are to be permanently main memory resident with the operating system or whether it is to be included as part of the

applications code in the vendor's proposed design. The translation algorithm specified below allows the vendor the opportunity to select the appropriate value for P. The vendor shall chose P as that percentage of the library instruction and data space which is to be permanently main memory resident. If P is chosen as zero, the code size of the model will assume that the library is included with the applications code. If P is chosen as greater than zero, then that portion of the library instruction and data space shall be permanently resident in main memory during the benchmark demonstration. The selected value of P and the resulting value of R1 shall be provided to the Government within 30 days after date of design verification contract award.

The translation of code size first requires the computation of a number (R1) as follows;

$$R_{1R} = \frac{S_R}{242.96} \quad \text{and}$$

$$R_{1N} = \frac{S_N}{404.48} \quad ,$$

then if $R_{1R} < R_{1N}$

$$R_1 = \frac{P \cdot R_{1R}}{100} + \frac{(100 - P) \cdot R_{1N}}{100}$$

else $R_1 = R_{1R}$;

where:

S_R is the Sample Program executable program size in bytes including instructions, local data and variables, and common for MAIN, ATARET, ATAWX, TERP, COPY, NREC, JDATE, INIT, FMSIO, ATAN*, and SQRT* in bytes.

---

*Library subroutine probable names.

S

N is the Sample Program executable program size in bytes defined as the minimal storage allocated at run time to enable correct execution of the Sample Program on the offeror's proposed machine under the proposed compiler and operating system and shall include all instruction and data space resulting from compilation, from linking, all library routines, and buffer(s) reserved for I/O data transfer and file record handling.

P is the percentage of library instruction and library data space intended by the offeror to be permanently main memory resident during execution of FSAS applications software in the Model 2 FSDPS. (This portion of the library instruction and data space shall be permanently resident in main memory during the benchmark demonstration.)

The number of CodeUnits, in the CodeLength statement of each cell is multiplied by the factor R1 to obtain the target code size in bytes for that cell.

The translation of CPUUnits first requires the computation of a number, R2, equal to the Sample Program execution time in seconds divided by 1.445. Then, CPUUnits when multiplied by R2, yield execution times in microseconds for the UseCPU statements.

An adjustment for CPU's having large high-speed cache memories is further required. The Sample Program was developed as representative of the FSAS application; however, looping was used to increase the measured number of executed instructions. As a result the Sample Program has a higher cache hit rate in general than the FSAS application; hence, the Sample Program execution time and R2 as defined above are biased in favor of cache memories. While the substantial benefit of large (1K + bytes) high-speed cache memories to the FSAS application must not be discounted; the bias will be eliminated. The functions represented by the cells have cache hit rates which vary from 0% to greater than 90% of the executed instructions. Due to the variation in cache hit rates across the functions represented by the cells, a different adjustment factor will be used by the Government as a multiplier of the target CPU time required for each cell. Those cells which represent functions having cache hit rates in excess of 90%

require no adjustment, hence a factor of 1.0, and those representing functions with cache bit rates of zero percent require the greatest adjustment. Because of the large number of factors and variables characterizing the CPU design, the Government will calculate a different set of factors for each different CPU proposed with cache memory.

As an example, the Government has derived the adjustment factors which would be used for a PDP 11/70 if it were proposed for the FSDPS. These are shown in Table D-1. The cache memory assumed for this example is 2K bytes and the effective memory cycle time for a read from main memory is approximately 4.4 times that of a read from cache memory. In this example, as in the general description above, the Government would determine the processing time in microseconds for any UseCPU statement as the product of the CPUUnits times R2 times the factor associated with the cell containing the UseCPU statement.

D.3 CELL CALIBRATION

Having found cell target sizes, it is the intent of the calibration process to approximate the storage and processing time resource consumption requirements represented by the target sizes. Storage and processing time is consumed in each cell by modeling machinery and by padding, if padding is found to be necessary. Padding will be required in most cells.

The calibration process involves at least two sets of measurements (not counting the Sample Program measurements), plus analysis and tabulation of the measurements and automated production of the calibrated cells. The calibration measurements shall be made by the vendor using the proposed CPU and compiler (A "SECNDS" routine as specified for the Sample program is also required). The calibration programs to be run will be supplied by the Government along with instructions. The first set of calibration measurements will be used by the Government in assessing machinery sizes on the proposed FSDPS. The second set of calibration measurements shall also be made by the vendor using Government supplied programs and instructions. These will be measurements of fully and partially calibrated cells. The Government will use these measurements in assessing and adjusting the precision of the automated production of calibrated cells. Additional sets of measurements of calibrated cells may be necessary to obtain the calibration precision

TABLE D-1

CACHE HIT RATE ADJUSTMENT FACTOR EXAMPLE

CELL	NAME	FACTOR	CELL	NAME	FACTOR
2	GetFlightPlanReadOnly	1.5	34	MailMessage	2.0
3	GetFlightPlanForUpdating	1.5	39	SimpleInput	1.5
4	FileThisFlightPlan	1.2	40	PageForward	1.5
7	ConvertRoute	1.0	41	PageBackward	1.5
8	GetIndividualWeatherInSeries	2.6	42	DirectionFinding	1.5
9	MessageToScreen	2.6	43	RequestSendFormat	2.6
10	OrganizeOutput	1.3	44	FlightLog	1.0
12	MakeLegal&AnalysisRecording	1.3	45	ExecuteLinkSend	1.3
13	WriteNewRecord	1.4	46	DisplayMessage	1.3
14	WriteNewRecordKeepingWorkingSpace	1.4	47	PrintScreen	1.3
15	WriteRecord	1.4	48	StartConversationWithNewFlightPlan	1.6
16	WriteRecordKeepingWorkingSpace	1.4	49	StartConversationWithFiledFlightPlan	1.3
17	WriteRecord&UnlockIt	1.4	50	ResumeConversationWithCurrentFlightPlan	1.5
18	ReadRecord	1.4	51	RouteDataForFlightPlan	2.0
19	ReadRecordForUpdating	1.4	52	NonRouteDataForFlightPlan	2.3
20	DeleteRecord	1.4	53	FileFlightPlanFromConversation	2.0
21	BulkUpdate	1.6	54	RequestWeather	2.6
22	StartBatchRead	2.0	56	DUATSimpleInput	1.5
23	EndOfBatchRead	2.0	57	DUATPageForward	1.5
24	StartRescueOperations	2.6	58	DUATPageBackward	1.5
25	DeleteFlightPlan	2.0	59	RequestDUATSendFormart	2.6
26	TransmitFlightPlan	1.5	60	DUATFlightLog	1.0
29	FlightPlanFromAnotherComputer	2.0	61	ExecuteDUATLinkSend	1.3
30	IndividualUpdateFromAWP	1.5	62	StartDUATConversationWithNewFlightPlan	1.6
31	HourlyUpdateFromAWP	1.5	63	StartDUATConversationWithFiledFlightPlan	1.3
32	AFOSGraphicsFromAWP	1.5	64	ResumeDUATConversationWithCurrentFlightPlan	1.5
33	AcknowledgementFromAnotherComputer	2.0	65	RouteDataForDUATFlightPlan	2.0

TABLE D-1

(CONT'D)

CELL	NAME	FACTOR
66	NonRouteDataForDUATFlightPlan	2.3
67	FileFlightPlanFromDUATConversation	2.0
68	RequestDUATWeather	2.6
73	SpecialistInput	1.7
74	SpecialistFormattedOutput	1.3
75	SpecialistFreeTextOutput	1.3
76	SpecialistWeatherDisplay	1.3
77	OrganizeFlightPlanFormat	2.6
78	UpdateTerminalRecord	2.0
80	DUATInput	1.7
81	DUATNormalOutput	1.3
82	DUATWeatherDisplay	1.3
83	OrganizeDUATFlightPlanFormat	2.6
84	UpdateDUATTerminalRecord	2.0
86	RunTimeRadarManagement	1.0
87	DealWithFlightPlan	1.5
88	RouteElementsInParallel	1.0
89	ConvertElement	1.0
90	LegsInParallel	1.0
91	ConvertLeg	1.0
92	StartWeatherRetrieval	2.6
93	EnsureEnoughWeatherToDisplay	2.6
94	StopTheRetrieval	1.5
95	GetIndividualWeatherInParallel	2.6
96	GetIndividualWeather	1.0
98	RunTimeMessageManagement	1.6
100	RunTimeInputDriving	1.6
102	RunTimeInputDistribution	2.6
103	CallRecipient	1.5
105	RunTimeOutputDriving	1.3
107	RunTimeTimeManagement	2.6

CELL	NAME	FACTOR
108	InitiateScheduledEvent	1.6
110	RunTimeScheduling	1.6
111	UpdateSharedWorkingSpace	1.6
112	Schedule	2.6
113	Unschedule	2.0
114	OneSecondRoutine	1.8
116	Legal&AnalysisDriving	1.3
118	WriteRecordDirect	2.6
119	WriteRecordSimpleCase	1.6
120	WriteRecord&Directory	1.6
121	WriteVariableAddressRecord	1.6
122	ReadRecordDirect	2.6
123	ReadRecordSimpleCase	2.0
124	ReadRecordHoldingDirectory	2.0
125	DataBaseRunTime	2.0
126	ReverseBulkUpdateSwitch	2.0
127	GetSpaceIDForRecord	2.6
128	GetLockedDirectoryUpdateEntry&GetSpaceID	2.6
129	WriteDirectory&UnlockIt	1.3
130	Find&UpdateRecordEntry	2.6
131	CollectFileStatistics	1.3
132	TestingIfReadyToBeDelete	2.0
133	WindRetrieval	1.8
134	WindComputations	1.0
135	RequestRouteRecordSimultaneously	1.0
136	RequestRouteRecordSerially	1.0
137	DigestWeatherRecord	1.0
138	RequestWeatherRecordSimultaneously	1.0
139	RequestWeatherRecordSerially	1.0

required by the Government. The Government reserves the right to have its representatives make any or all of these sets of measurements. If any run is made by the vendor, the Government reserves the right to verify the results by having Government representatives make the same run on the vendor's equipment or on other identical equipment.

In a small number of cases, the machinery in a cell may consume more storage and/or processing time than allowed by the target size. This can happen for example, when a cell must perform the functions it is modeling in addition to model control functions. In these cases, the Government will reduce resource consumption elsewhere in the model to balance out to the proper resource consumption level.

APPENDIX E

MODEL VERSIONS

E.1 GENERAL

As has been stated at points in the text of Parts II and IV, there exist several versions of the FSAS benchmark model. Some of these versions have proven to be of value to the Government in development and testing and may be of equivalent value to the vendor. In particular, the Test Model, Small Scale Model and Full Scale Debugging Model are useful in integration and test of the VIP interfaces. The basic characteristics of the various model versions are indicated in Table E-1.

The Test Model will be distributed, discussed and demonstrated at the workshop. The full scale models all have the characteristics outlined in Part IV except for their unique properties outlined in Table E-1.

E.2 SMALL SCALE MODEL

The Small Scale Model (SSM) contains a subset of the functional areas, the cells and the natural flows of the full sized models described in Part IV. It is intended for use in a first dynamic test of the VIP. It should readily fit into a single CPU system, thus allowing the vendor to avoid the added complexity of a multiprocessor configuration in early testing of the VIP. The SSM contains 68 cells of which 12 are used during initialization, 52 during the measured phase and 4 during the termination phase of a benchmark demonstration run. The details of this information as well as the association of cells with functional area and the identification of cell type are provided in Table E-2.

The input transactions used with the SSM are listed in Table E-3 as well as the logical ports over which they are to be input. Fourteen Natural Flows are used, as listed in Table E-4. The SSM requires the use of logical ports 1, 2, 132, 133, 134 and 138. The Natural Flows which are associated with each of these logical ports are shown in Table E-5. The association of logical port with physical port remains as required for the full scale model, except that printer output will be on logical port 134 for the SSM which is directed to physical port 14.

TABLE E-1

MODEL VERSIONS

MODEL VERSION	CALIBRATED	HOL-EDITABLE OPTIONS	DYNAMIC OPTIONS	OTHER OPTIONS	MODEL TRACE	MISC. DEBUG[3]	REMARKS
Test Model	NA[1]	NA	NA	NA	Yes	NA	Contains five cells, used for static VIP testing.
Small Scale Model	No	V[2]	Yes	V	V[6]	Yes	Subset of the cells, two specialist and two NADIN Natural Flows.
Medium Scale Model[4]	No	V	Yes	V	No	Yes	Contains all of the cells and all Natural Flows except reduced to five specialist and two DUAT.
Full Scale Debug Model	No	V	Yes	V	No	Yes	Uncalibrated full scale model with debug switches set.
Trial Models	Yes	V	V	V	No	No	Two trial versions are allowed for vendor's optimization efforts.
Demonstration Model	Yes	No[5]	No[5]	V	No	No	Official model to be used in the demonstration.

NOTES:
1. NA = Not applicable.
2. V = Vendor selectable.
3. Miscellaneous Debug = Stack overflow test and Simulate CPU.
4. The medium scale model will be made available if the Government has sufficient time and resources to do so. To be announced at the workshop.
5. These options will be set in fixed alphanumeric form in the HOL.
6. Will be discussed at the workshop.

TABLE E-2

SMALL SCALE MODEL - CELLS BY FUNCTIONAL AREA

FUNCTIONAL AREA	CELL NAME	ACTIVE PHASE**	CELL TYPE*
Top Level	InitializeBenchmark	A	3.
	BenchmarkRunTime	B	3.
	WrapUpBenchmark	C	3.
Specialist	InitializeSpecialist	A	3.
	SpecialistInput	B	3.
	SpecialistFormattedOutput	B	3.
	SpecialistWeatherDisplay	B	3.
	SimpleInput	B	1.,3.
	PageBackward	B	1.,3.
	PageForward	B	1.,3.
	RequestWeather	B	1.,3.
	StopBenchmark	C	1.,3.
Common-Service Input	InitializeInputDriving	A	3.
	EndOfInitialization	A	3.
	RunTimeInputDriving	B	3.
	InitializeInputDistribution	A	3.
	RunTimeInputDistribution	B	3.
	CallRecipient	B	3.,5.

NOTES:
* 1. Input Message Processor
2. Outer-Set
3. Inner-Set
4. Reception Desk
5. Contains BigGoTo

** Phase A = Model Initialization Phase
Phase B = Measured Period
Phase C = Model Termination Phase

TABLE E-2

(CONT'D)

FUNCTIONAL AREA	CELL NAME	ACTIVE PHASE**	CELL TYPE*
Time Management	InitializeTimeManagement	A	3.
	OneSecondRoutine	B	3.
	StopDoingScheduledActions	C	3.,5.
	RunTimeTimeManagement	B	3.
	InitiateScheduledEvent	B	3.,5.
	Schedule	B	2.,4.
	ScheduleEndOfScheduling	A	2.,4.
	InitializeScheduling	A	3.
	RunTimeScheduling	B	3.
	UpdateSharedWorkingSpace	B	3.
Weather Management	StartWeatherRetrieval	B	2.,4.
	EnsureEnoughWeatherToDisplay	B	2.,4.
	StopTheRetrieval	B	2.,4.
	HourlyUpdateFromAWP	B	2.,4.,5.
	GetIndividualWeatherInParallel	B	2.,4.
	GetIndividualWeatherInSeries	B	2.,4.,5.
	GetIndividualWeather	B	2.,4.
	WindRetrieval	B	2.,4.
	WindComputations	B	2.,4.

NOTES:
- * 1. Input Message Processor
  - 2. Outer-Set
  - 3. Inner-Set
  - 4. Reception Desk
  - 5. Contains BigGoTo
- ** Phase A = Model Initialization Phase
  - Phase B = Measured Period
  - Phase C = Model Termination Phase

TABLE E-2

(CONT'D)

FUNCTIONAL AREA	CELL NAME	ACTIVE PHASE**	CELL TYPE*
Common-Service Output	OrganizeOutput	B	3.,4.,5.
	DrainSpool	C	3.,4.,5.
	InitializeOutputDriving	A	3.
	RunTimeOutputDriving	B	3.
Legal&Analysis Recording	MakeLegal&AnalysisRecording	B	3.,4.,5.
	InitializeLegal&AnalysisDriving	A	3.
	Legal&AnalysisDriving	B	3.
Data-Base Management	StandardInitializationOfFiles	A	2.,4.
	WriteNewRecordKeepingWorkingSpace	B	3.,4.,5.
	WriteNewRecord	B	3.,4.,5.
	WriteRecordKeepingWorkingSpace	B	3.,4.,5.
	WriteRecord	B	3.,4.,5.
	WriteRecord&UnlockIt	B	3.,4.,5.
	ReadRecord	B	3.,4.,5.
	BulkUpdate	B	3.,4.,5.
	StartBatchRead	B	3.,4.,5.
	EndOfBatchRead	B	3.,4.,5.
	WriteRecordSimpleCase	B	3.,5.
	WriteRecord&Directory	B	3.,5.
	WriteVariableAddressRecord	B	3.,5.
	ReadRecordSimpleCase	B	3.,5.
	ReadRecordHoldingDirectory	B	3.,5.
	GetSpaceIDForRecord	B	3.

NOTES:

* 1. Input Message Processor
2. Outer-Set
3. Inner-Set
4. Reception Desk
5. Contains BigGoTo

** Phase A = Model Initialization Phase
Phase B = Measured Period
Phase C = Model Termination Phase

TABLE E-2

(CONC)

FUNCTIONAL AREA	CELL NAME	ACTIVE PHASE**	CELL TYPE*
Data-Base Management	GetLockedDirectoryUpdate Entry&GetSpaceID	B	3.
	Find&UpdateRecordEntry	B	3.
	WriteDirectory&UnlockIt	B	3.
	CollectFileStatistics	B	3.
	InitializeDataBaseManagement	A	3.
	DataBaseRunTime	B	3.
	ReverseBulkUpdateSwitch	B	3.
	TestIfReadyToDelete	B	3.

TABLE E-3

SMALL SCALE MODEL - INPUT TRANSACTIONS

USER CLASS		TRANSACTION TYPE		LOGICAL
ID	NAME	ID	NAME	PORTS
01	Specialist	01	Simple Input	1, 2
		02	Page Forward	1, 2
		03	Page Backward	1, 2
		16	Request Weather	1, 2
		17	Specialist Stop Benchmark	1, 2
03	NADIN	04	Hourly Update	132,133
		09	Schedule End Of Scheduling	132
		10	Standard Initialization Of Files	132
		11	End Of Inialization	132
		12	End Of Benchmark	132,133

TABLE E-4

SMALL SCALE MODEL NATURAL FLOWS

NATURAL FLOW	ID
Ancestral	1
Legal&AnalysisDriver	2
DataBaseManager	3
Scheduler	4
OutputDriver (AWP Link)	5
OutputDriver (AWP Hourly)	6
OutputDriver (Printer)	7
InputDriver (AWP Link)	8
InputDriver (AWP Hourly)	9
InputDistributor (AWP Link)	10
InputDistributor (AWP Hourly)	11
TimeManager	12
Specialists	13,14

TABLE E-5

SSM MAPPING OF NATURAL FLOWS TO LOGICAL PORTS

NATURAL FLOW ID	NATURAL FLOW NAME	LOGICAL PORT NO.
02	Legal&AnalysisDriver	138
05	OutputDriver (AWP Link)	132
06	OutputDriver (AWP Hourly)	133
07	OutputDriver (Printer)	134
08	InputDriver (AWP Link)	132
09	InputDriver (AWP Hourly)	133
10	InputDistributor (AWP Link)	132
11	InputDistributor (AWP Hourly)	133
13	Specialist	1
14	Specialist	2

Table E-6 lists the shared working space sizes for the SSM. The other working spaces used in the full scale model are also used in the SSM. However, the six largest areas are reduced in size.

See Table E-7 for file sizes in the SSM.

TABLE E-6

SMALL SCALE MODEL SHARED WORKING SPACE (SWS) SIZES

NATURAL FLOW	NUMBER OF SWSs	SIZE - GENERAL[1]	SIZE FOR TWO BYTE WORD[1]
OutputDrivers	3	8 * SizeOfWord	16
Scheduler	1	SizeOfWord*(3+2* #SchedulingRecords)	806
Legal&AnalysisDriver	1	2 * SizeOfWord + Legal&Analysis BufferSize	44
DataBaseManager	1	SizeOfWord*(2*#Files+ 3*#BulkUpdateFilePairs)	76

Note: 1. In characters size of word is number of bytes per word.
* Equals multiply

TABLE E-7

SMALL SCALE MODEL FILES

FILE ID	FILE NAME	#RECORDS[2]	RECORD LENGTH[1]
1	HourlyWeatherObservationsFile	5	20
2	FGFile	5	20
3	FAFileUnexpanded	5	20
4	PIREPFileUnexpanded	5	20
5	HourlyWeatherObservationsAlternateFile	5	20
6	FGAlternateFile	5	20
7	Airways&FixesFile	5	20
8	ReportingLocationsFile	5	20
9	FlightPlanFile	10	30
10	SchedulingFile[3,4]	200	24
11	OutputFile	20	30
12	OutputFile	20	30
13	OutputFile	20	30
14	TerminalRecordFile	2	20
15	CurrentFlightPlanFile	2	30
16	RouteRecordFile	2	20

Note:
1. In characters
2. Plus directory
3. Directory size = SizeOfWord * #OfRecords
4. See also paragraph 3.5.2 of Part IV for discussion of directories

MITRE Department
and Project Approval: _____

S. Koslow

END
DATE
FILMED
3-13-82
NTIS