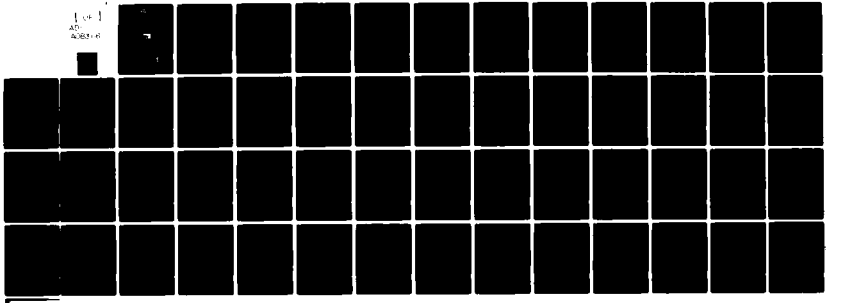
AD-A083 116 BOEING AEROSPACE CO SEATTLE WA BOEING MILITARY AIRPL--ETC F/G 9/2
IDAMST SOFTWARE MANAGEMENT PLAN. ADDENDUM 5.(U)
NOV 76 F33615-76-C-1099

UNCLASSIFIED SPEC-2-8080-IDAMST-21 AFAL-TR-76-208-ADD-5 NL



END
DATE FILMED
5-80
DTIC

AFAL-TR-76-208, Addendum #5

(14) SPEC-2-8080-IDAMST-21

AD-A047163
AD-A053940

LEVEL III

(1) (ms)

(6) IDAMST SOFTWARE MANAGEMENT PLAN. Addendum 5.

Prepared by

THE BOEING AEROSPACE COMPANY
BOEING MILITARY AIRPLANE DEVELOPMENT
SEATTLE, WASHINGTON

(18) AFAL

(11) NOV [redacted] 76

(19) TR-76-208-ADD-5

(12) 54

(15) F33615-76-C-1099

PREPARED FOR

AIR FORCE AVIONICS LABORATORY
AIR FORCE SYSTEM COMMAND
UNITED STATES AIR FORCE
WRIGHT-PATTERSON AFB, OHIO 45433

FOREWORD

This document contains a Software Management Plan for IDAMST and has been essentially derived from the software management plan provided by Appendix H to the statement of work. The presented plan incorporates the changes suggested to Appendix H as documented in the interim technical report.

This document was prepared by the Boeing Military Airplane Development organization of the Boeing Aerospace Company, Seattle, Washington under Contract F33615-76-C-1099, "Specifications for IDAMST Software", and is submitted in fulfillment of CDRL item 0001, sequence number 5. The work was sponsored and administered by the Air Force Avionics Laboratory, Wright-Patterson Air Force Base, Ohio 45433 by Mr. Lawrence Gutman, project engineer.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

IDAMST SOFTWARE MANAGEMENT PLAN

1.0 INTRODUCTION

This plan sets forth basic requirements for the conduct and application of management disciplines to be applied to the development of IDAMST Mission Software. Mission software includes all deliverable operational and support computer programs. The minimum requirements for identifying, controlling, accounting, and reviewing Mission Software elements during development and documentation are specified herein.

1.1 PURPOSE

The purpose of this plan is to provide direction and guidance to participants in the IDAMST Mission Software development effort, in order to become uniformly compliant with the disciplines specified. The extent to which the details of this plan are applicable to each element of the Mission Software shall be identified in the initial planning of affected organizations. Project planners shall identify the methods and resources needed to implement the management requirements of this plan; exceptions and deviations shall also be specifically delineated.

1.2 SCOPE

This plan establishes the practices required to maintain SPO management visibility of the IDAMST Mission Software development effort. It is primarily intended to provide coverage for the following areas:

a) Software identification, including milestone management and documentation practices.

b) Configuration control, including change classification and change processing.

c) Configuration status accounting.

d) Configuration reviews and audits.

Additional subject material is included to provide guidance in areas which impact the application of this plan.

1.3 GENERAL DEFINITIONS

IDAMST Software Management consists of directing the effort and administering the procedures and controls to further development and test of deliverable Computer Program Configuration Items (CPCI's). The specific design approach to be implemented is outlined in Section 9, following. The management guidelines contained therein conform to the concept of hierarchial, structured, top-down design principles. This methodology ensures that the requirements which are formulated evolve top-down into more detailed design specifications and, finally, verified code. Performance specification, interface definition, logical formulation, code implementation, verification, and documentation are all integral parts of the structured software develop-

ment process.

IDAMST Configuration Management applies the technical and administrative disciplines to establish the software design baseline and control change to its approved configuration.

Software Identification is the process of labeling and describing each computer program component to assure that each software element will be identified by its technical documentation, i.e., requirements, specification, and other documents associated with the software developed. The recognition and approval of this documentation establishes the configuration of the software elements. Once this documentation is approved for release, it, plus all subsequent approved changes thereto, constitutes the configuration identification.

Configuration Control provides the systematic evaluation, coordination, approval or disapproval, and implementation of all approved additions or changes to software elements.

Configuration Accounting provides for the recording and reporting of proposed and approved changes to these elements.

2.0 ORGANIZATION

2.1 CONTRACTOR CONFIGURATION MANAGEMENT RESPONSIBILITIES

The Mission Software contractor shall establish a configuration management function within its organication, in accordance with the requirements established by the contractual documentation. The implementation of the configuration management effort shall be guided by the requirements of this plan. All significant exceptions or deviations to this plan shall be identified and rationale shall be submitted to the Air Force AMST SPO for approval prior to implementation.

2.1.1 Configuration Management Plan

The contractor shall prepare a plan for Air Force AMST SPO approval which details the contractor's specific methods and procedures for implementing the general requirements of this plan. The contractor's plan shall fully delineate the organizational responsibilities, functions to be accomplished and methods of implementation, and overall phasing of the configuration management effort. This plan shall be prepared using MIL-STD-483, Appendix I as a guide; for those areas specifically related to software configuration management and control.

2.1.2 Documentation Control

The contractor shall be responsible for preparing and maintaining all required Mission Software documentation current with respect to released software versions. Control of this documentation shall be a contractor configuration management responsibility.

2.1.3 Configuration Identification

Configuration identification shall be accomplished in accordance with Section 3.0 of this plan.

2.1.4 Configuration Accounting

Configuration accounting shall be accomplished in accordance with Section 6.0 of this plan.

2.1.5 Policies/Directives

The contractor shall prepare command media establishing the necessary policies, procedures, and/or directives to implement the contractual configuration management requirements and to meet the intent of this plan. All applicable directives shall be delineated in the contractor's Configuration Management Plan, Software or Computer Program Development Plan, or through other appropriate media.

2.1.6 Software Assurance

IDAMST software assurance includes the development and maintenance of software development standards and procedures, the enforcement of formal soft-

ware development controls, and the monitoring and approval of software qualification testing.

2.2 IDAMST SOFTWARE DOCUMENTATION

IDAMST software documentation assurance responsibilities shall include monitoring and controlling software development specifications and manuals, as required, to keep them current with the software configuration. It shall also cover maintaining and reporting software configuration status to the Air Force as specified in this plan. All documentation produced under this plan shall become the sole property of the Air Force. The contractor shall ensure that the documentation evolves in a complete and correct manner from the IDAMST Systems Requirements through the Software Requirements. Documentation shall be controlled and maintained in accordance with requirements established by the CDRL; the provisions of the Statement of Work, and any governing standards, specifications, or regulations cited therein.

3.0 CONFIGURATION IDENTIFICATION

Configuration identification for IDAMST shall consist of the current authorized technical document package for each mission software item released, as set forth in requirements, specifications, and other documents. The approved technical documents describing the configuration allocation and configuration identification shall constitute a configuration baseline. Two milestones for this baseline shall be established for IDAMST:

a) the requirements document (B5 specification)

b) the specification document (C5 specification).

Overall documentation of the software development process shall conform to the requirements of the CDRL and other applicable sections of the IDAMST Statement of Work.

The Documentation Tree in Figure 3-1 illustrates how project requirements evolve in 6 steps (or milestones) to the lowest level of software specifications. The Air Force shall provide the contractor with Mission Software Requirements for the Operation Systems Software, Applications Software (Steps 1-5), and Error Handling and Recovery System (EHARS) Software. The contractor shall review and provide recommendations and critique for improving these requirements. The contractor shall also develop requirements for Mission Support and Test and Development software. After Air Force approval, the contractor shall provide the remaining documentation in the chronological order specified in the CDRL.

The Operating Systems Software shall contain the following types of Computer Program Components (CPC's):

a) Master Executive (including Bus Control);

b) Local Executive;

c) Bus Interface.

The Application Software shall contain the following types of CPC's:

a) Controller or Sequencer modules;

b) Computational modules.

The EHARS software shall contain the following types of CPCs:

a) In-flight core element tests;

b) Subsystem sensor tests.

The Mission Support Software package shall possess at least the following capabilities:

a) Mission tape build and verify functions;

b) Mission-dependent data file build and edit;

c) Post-mission DITS recording tape data reduction.

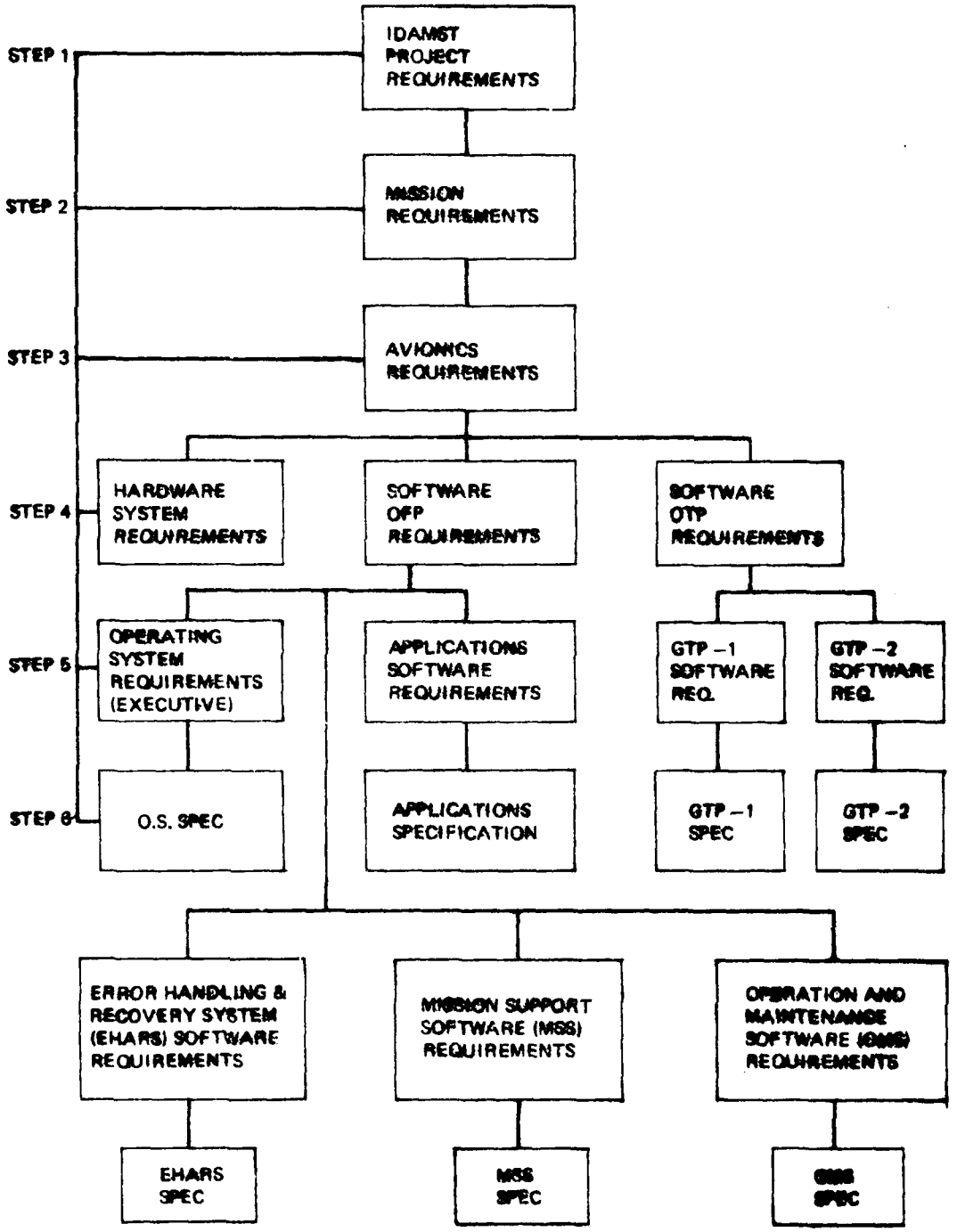


Figure 2-1. IDAMST Chronological Documentation Tree

The Operation and Maintenance (O&M) Software will be provided, in part, as GFE elements will include:

a) Ground support Computer Operating System;

b) Jovial Compiler

c) Linking Loader;

d) Data Base Management Utilities.

The software contractor is further required to identify any other development and test support software item which will be generated during the normal course of IDAMST computer program implementation and qualification. The contractor will similarly establish the need for, or desireability of, providing similar functional capability in addition to the basic O&M support software elements listed above.

The software development contractor shall assess the probable usage levels and areas of employment of all deliverable and potentially deliverable software and specify hardware and facilities requirements for their use. The analysis and support data shall be prepared as an input to the overall AMST Integrated Logistics and Configuration Management planning efforts. The schedule for delivery of this data shall be coordinated with the schedules defined in this SOW and corresponding CDRL delivery and update requirements for the applicable documents.

3.1 MILESTONE MANAGEMENT

The IDAMST software development effort shall utilize milestone management techniques as a means of facilitating configuration identification and control. Under this concept, technical control points, or milestones, shall be established for computer program specification development which are coordinated with overall AMST milestones and schedules. Evaluation, coordination, and disposition of all proposed changes to these software specification milestones shall be considered with respect to the controlling documentation for that milestone at formally convened reviews. All software milestone reviews and documentation schedules and data delivery requirements shall be responsive to applicable paragraphs in the AMST Statement of Work, the Contract Data Requirements List and the following paragraphs of this Software Management Plan document.

3.1.1 Software Requirements Milestone

The Software Requirements Milestone is accomplished by Steps 4 and 5 of Figure 3-1. The process of defining the Computer Program Configuration Item requirements for this milestone is described in the Structured Requirements Design Guidelines, Section 9. Requirements are documented in the form of B5 specifications (as defined in Paragraph 3.2.1, below) and are reviewed by the Air Force at the Preliminary Design Review (PDR).

3.1.2 Software Specification Milestone

This milestone is accomplished by Step 6 of Figure 3-1. The process of defining the detailed software design for this milestone is described in

the Program Architecture Design Guidelines, Section 10. The design is documented primarily in the form of partial C5 specifications (as defined in Paragraph 3.2.3, below). The milestone documentation is reviewed by the Air Force at the Critical Design Review (CDR).

For this specification design milestone, the documentation of the requirements design is taken to a greater level of detail, maintaining the same format and program structure, but with more information supplied. This milestone is supported by performance analysis of algorithms (e.g., timing, recovery, etc.) and architecture interfaces (e.g., data, control, etc.). The analysis ensures that the software will function per allocated requirements and specifications over its designed-for range of data and control inputs. The analysis shall further show that the effect of incorrect or anomalous control states and input data shall not unacceptably degrade software performance or shall result in predefined error responses suitable for the condition(s) encountered. Based on this analysis, the specification is either approved, and the design is implemented, or it is disapproved, and the milestone is repeated with a redirection from the Air Force reviewing agencies. The C5 specification is completed by inclusion of the as-built code listings and flows.

3.2 SPECIFICATION TYPE IDENTIFICATION

3.2.1 B5 Type Specifications

All Mission Software modules developed by the contractor shall be documented in accordance with MIL-STD-490 Type B5 specifications (as per MIL-STD-483). Part I (Paragraph 60.4) entitled, "Detailed Instructions for the preparation of computer program development specification," of Appendix VI, entitled, "Computer Program Configuration Item Specification," of MIL-STD-483, shall be strictly adhered to for the computer program development specification with the exception of specific paragraphs which have been interpreted as indicated below:

> Paragraph 60.4.4 Section 4 Quality Assurance Provisions. Although, in general, this paragraph is written to imply a "Bottom-Up" approach to the qualification testing requirements, the IDAMST software contractor shall develop the IDAMST qualification testing requirement specifications using a "top-down" approach for development testing and software module integration prior to formal CPCI and software subsystem verification testing (qualification).
>
> Paragraph 60.4.4.1.5 Paragraph 4.1.5 Category II System Test Program. This paragraph shall be interpreted to cover the Initial Operational Test and Evaluation (IOT&E) phase of system test and shall, in general, be the responsibility of the AMST prime (or integrating) contractor.

3.2.2 Software Module Groups

Upon completion of the computer program development specifications (TYPE B5) described above and the Preliminary Design Review (PDR) as described in Section 8.2.1 of this Appendix, the contractor, with Air Force approval,

shall divide the Mission Software Modules into two groups for the preparation of computer program product specifications (TYPE C5) as per MIL-STD-490.

3.2.2.1 Application Modules

The first of these Mission Software Module groups is the Application Modules group. All IDAMST Mission Software modules identified at the conclusion of the PDR, which can be described or defined by existing and available algorithms of math flow, shall be placed in the Application Modules group. Modules in this group shall include, but not be limited to, the following types:

a. Radar Cursor Acquisition
b. Navigation Acquisition
c. Flight Director Manual Heading and Automatic Navigation
d. Flight Director ILS
e. Flight Director TACAN
f. Air Data
g. Radar Beacon Offset Acquision
h. Radar Position Fix
i. Velocity Fix
j. Loran Velocity Aid
k. TACAN Fix
l. Manual Fix
m. Wind Velocity
n. Air Data Backup Navigation
o. Inertial Strapdown
p. Inertial Navigation
q. Strapdown IMU Navigation
r. Air Mass Navigation
s. IMU leveling
t. Navigation Kalman Filter
u. Along course, cross course distances
v. Radar and TACAN Fixes and Steering
w. Radar Seek/Track
x. Computed Air Release Point (CARP)

3.2.2.2 Control Modules

The second of these Mission Software Module groups is the Control Modules group. All IDAMST Mission Software modules identified at the conclusion of the PDR, which cannot be described or defined by existing and available algorithms or math flow, shall be placed in the Control Module group. Modules in this group shall include, but not be limited to, the following types:

A. Executive Software
   A.1 Master Executive
       Data Bus Control
       Systems Error Management
       Configuration Management
       Mass Memory Management

   A.2 Local Executive
       Process Control Interface
       Event Control
       Data Control
       CPU Fresh Start, Restart

B. Application Software
       Master Sequencer
       Request Processor
       Display Management Control
       Configurator
       Mission Operational Sequencers
       Specialists Functions

C. Error Handling and Recovery System (EHARS) Software
       Core Element Status
       Subsystem Status
       Reconfiguration Management

D. Operational Test Programs (OTP)
       Ground Test Program-1 (GTP-1)
       Ground Test Program-2 (GTP-2)

3.2.3 Partial C5 Type Specifications

All Mission Software Modules shall be documented in accordance with MIL-STD-490 Type C5 Specifications (as per MIL-STD-483). Part II (Paragraph 60.5) entitled, "Detailed instructions for the preparation of computer program product specification," of MIL-STD-483, shall be strictly adhered to for the computer program product specification (Application Modules) with the exception of specific paragraphs which have been either deleted or rewritten as shown below:

Paragraph 60.5.3 Section 3 Requirements (Technical Description)
This section shall specify the detailed configuration of the CPCI. It shall contain a complete technical description of the CPCI structure, functions, and the relationships between the individual Computer

Program Components (CPCs) and the CPCI with the Software Subsystem. General and descriptive material may be included in the basic Section 3 lead paragraph.

Paragraph 60.5.3.2 Paragraph 3.2 Functional Description. The individual Computer Program Components (CPCs) shall be described in separate paragraphs as required. This description shall be given at the level of detail that will define the design and configuration of the CPC sufficiently to allow for CPC modification and adaption in the operation phase. Each CPC shall be described in words and flow charts. In addition, the internal interfaces (data and control) between this CPC and other CPC's and the operational CPCI shall be completely defined (or appropriately referenced) to the degree necessary to assess the impact of changes, additions, or modifications to the component modules. The basic Paragraph 3.2 shall contain the following lead phrase, "This paragraph contains the detailed technical descriptions of the computer program components identified in Paragraph 3.1 of this specification. The following subparagraphs shall be repeated for each CPC."

Paragraph 60.5.3.2.1 Paragraph 3.2.1 Computer Program Component 1. The basic paragraph shall identify the CPC by including, as a minimum, the title, tag (symbolic code), and the CPC identification number. It shall also include a brief abstract of the tasks of the CPC, and its major functional interfaces. The CPC shall then be described in detail in subparagraphs.

Paragraph 60.5.3.2.1.1. Subparagraph 3.2.1.1 CPC No. 1 Description. This subparagraph shall describe in words, figures, equations and references to the flow charts of Subparagraph 3.2.1.2, the operation and design of the CPC. This paragraph shall contain, as appropriate, a description of the program logic and data flow; equations to be solved; algorithms used to solve these equations; timing and accuracy characteristics; and any special conditions for operation of the CPC. The description shall be sufficiently detailed to facilitate understanding of, and modification to, the CPC. Equation derivations and numerical analysis shall not be included herein, but may be included in Section 6 (Notes).

Paragraph 60.5.3.2.1.2 Subparagraph 3.2.1.2 CPC No. 1 Flow Chart. This subparagraph shall graphically portray the operations performed by the CPC. This shall be done by a (series of) flow chart(s) and/or structured logic tree diagrams which depict the processing described in Subparagraph 3.2.1.1, including the sequence of operations and decision points in the CPC. The highest-level flow chart shall depict on a single sheet the overall information flow of the CPC and shall reference the flow charts in Paragraph 3.4 that identify the CPC. In general, the lowest-level flow chart identifies all decision points in the CPC and references higher level charts as appropriate. All symbology used in the flow chart shall be in accordance with the American Standard Flow Chart Symbols for Information Processing, X3.12-1970, or latest edition, unless deviations are approved by the procuring activity.

NOTE: Numbers of flow charts and level of detail required should be determined, based on the above, and defined further in specific terms on the CDRL backup sheet in the contract for design and development of each CPCI. In general, requirements for numbers and detail of flow charts should be held to the minimum which are adequate to facilitate understanding of the information flow, taking into account the intended uses and making provisions for follow-on maintenance of the CPCI.

Paragraph 60.5.3.4 Paragraph 3.4 Computer Program Functional Flow Diagram. To be covered in Paragraph 3.2.3.3 of this Appendix.

Paragraph 60.5.4 Section 4 Quality Assurance. Delete. However, the following subparagraphs of MIL-STD-483, Part I (Paragraph 60.4) of Appendix VI entitled, "Computer Program Configuration Item Specification" shall be adhered to during the development of the partial C5 type specifications for the Control Modules.

Paragraph 60.4.4.1.4 Paragraph 4.1.4 Formal Qualification Tests.

Paragraph 60.4.4.1.5 Paragraph 4.1.5 Category II Systems Test Program.

Paragraph 60.4.4.2 Paragraph 4.2 Test Requirements.

3.2.3.3 CPCI System Specifications

The Type C5 (MIL-STD-490) CPCI system specifications shall be documented as per MIL-STD-483. Paragraph 3.4 and its subparagraphs under Section 60.5.3.4 (entitled, "Computer Program Functional Flow Diagram,") of Appendix VI of MIL-STD-483, shall be strictly adhered to. The principal use of these diagrams shall be to relate each of the IDAMST CPCI's and constituent CPC's functional elements to requirements established in higher-level system specifications.

4.0 CONFIGURATION CONTROL

4.1 OBJECTIVES

The development process for IDAMST software shall be carried out in accordance with approved configuration management procedures to insure that the final product fulfills its intended purpose and meets all specified design and performance requirements. Implemented Computer Program Components (CPCs) should be regulated with regard to their baseline configuration and the documentation which establishes the baseline design. At the same time, the development process must not overly constrain the developers' creativeness. Two prime goals are a good product and cost-effectiveness. The guidelines of Section 9 describe some management control techniques for use in the development of "structured" computer programs. The relationships most affected during software development, and also the most difficult to control, are the software-software and hardware-software interfaces. Initial integration of the Computer Program Components and later changes to these components or to related hardware components must be closely monitored and precisely coordinated.

4.2 END-ITEM MANAGEMENT

4.2.1 Software Standards

Software end-items shall be developed and maintained in accordance with the goals, control mechanisms, and procedures described in the Software Development Guidelines, Section 9, and the Software Architecture specifications of Section 10.

4.3 INTERFACE CONTROL

An Interface Control function shall be implemented to coordinate physical, functional, and environmental interfaces which impact the IDAMST software design. IDAMST interface requirements shall be defined, established, and controlled through Interface Control Drawings (ICDs).

4.3.1 Interface Control Drawings

Interface Control Drawings (ICDs) shall be prepared and submitted for each IDAMST software or software-hardware interface.

The ICD shall delineate configuration, interface data, and characteristics which cannot be changed without affecting software design criteria in any system. The ICDs shall be approved by the Air Force and, subsequent to approval, the ICDs shall be formally released and become subject to change control procedures applicable to the governing (referencing) specification documents.

4.4 INTERNAL CONTROL BOARD - Deleted.

5.0 CONTRACT COMPLIANCE

The ILAMST contractor shall establish internal controls and procedures to assure technical compliance with contract requirements. These controls are to assure that all engineering tasks identified in the contract are accomplished on schedule, and not duplicated and that any efforts over and beyond the contract requirements are precluded.

5.1 CONTRACT CHANGE PROPOSALS

All changes in contract obligations and/or contract documentation, such as revisions to Statement of Work, but not including changes to specifications, shall be processed using the Contract Change Proposal (CCP). Changes which affect design or performance requirements set forth in approved specifications shall be processed using the Engineering Change Proposal (ECP). CCP's and ECP's shall be processed in the same manner, per MIL-STD-480.

5.2 DEVIATIONS AND WAIVERS

Requests for deviations and waivers shall be prepared based on the requirements and formats presented in MIL-STD-480. Major or critical deviations/waivers shall be subject to the same internal controls as those used for ECP's (viz, defined by an ECM, presented to the internal change board for determination of effectivity and schedule impacts, costed by the finance organization, and subjected to management review) prior to submittal to the Air Force.

6.0 CONFIGURATION STATUS ACCOUNTING

Configuration status accounting shall be established for IDAMST software development in order to provide sufficient data to assure that the as-designed configuration conforms to its technical description, i.e., specification and other associated documentation. Status accounting shall be established by developing and implementing the status records/documents described in this section of this plan.

6.1 CHANGE STATUS LISTING AND REPORT

The change status listing shall be a compilation of all proposed changes to the software specifications. It shall be used to make available current status of the software and changes throughout software development. The change status report shall be prepared from this listing and shall provide a summary of change status.

7.0 PROGRAM PHASING

IDAMST software configuration management shall be phased to be compatible with the IDAMST master scheduling and the development schedules for each of the associated system elements. The IDAMST Mission Software contractor shall develop the Mission Software phasing schedules to be compliant and compatible with the AMST program schedules. Figure 7.1 identifies the basic program milestones and their relationship to the contractor's IDAMST Mission, Mission Support, and Operation and Maintenance Software development effort.

7.1 MANAGEMENT PLAN

The Software Management Plan utilizes the techniques and facilities described in Section 9 and Section 10. The requirements of the Architecture Specification (Section 10) shall be enfored and periodically audited during the development of the software as indicated by the matrix shown in Table 7-1. Three categories shall be evaluated during this process:

a) Applications Software as a package of functions proposed to perform various applications, including Error Handling and Recovery System (EHARS).

b) Executive Software as a package of functions proposed to perform various executive and input/output operations.

c) OTP software as a package of test functions proposed to functionally test the IDAMST core elements and subsystem sensors.

Mission Support and O&M Software items shall be assessed with respect to their functional relationship to Operational Mission Software interfaces and roles. All categories begin with the requirements which are documented for Milestone 1. These categories continue development as the Type B5 specifications are added to the documentation for Milestone 2. The Type C5 specifications complete with all interfaces and supporting engineering data, comprise the final documentation for Milestone 3.

Milestone 3 contains all of the requirements for a specific application, mission, etc. Thus, the Milestone 3 requirements are a collection of a specific, top-level structured design, plus all of the functions from Milestones 1 and 2 "plugged" into the top-level structure. The task of designing Milestone 3 is, therefore, that of designing a top-level structure for each application, mission, etc., as well as collecting and integrating the required documentation baseline into one complete product specification.

FIGURE 7-1

IDAMST SOFTWARE DEVELOPMENT SCHEDULE

TO BE SUPPLIED

TABLE 7-1

SOFTWARE DEVELOPMENT MILESTONES

	MILESTONE 1 MISSIONS & OSD'S	MILESTONE 2 TYPE B5	MILESTONE 3 TYPE C5
(A) APPLICATIONS SOFTWARE PACKAGE (INCLUDES EHARS)	Software Requirements Document for Applications Function	Software B5 Specifications Additions for Applications Functions	Software C5 Specifications Additions for Applications Functions
(B) EXECUTIVE SOFTWARE PACKAGE	Software Requirements Document for Executive Functions	Software B5 Specifications Additiona for Executive Functions	Software C5 Specifications Additions for Executive Functions
(C) OTP SOFTWARE PACKAGE	Software Requirements Document for Integrated Tests Functions	Software B5 Specifications Additions for DITS Functions	Software C5 Specifications Additions for OPT Functions

8.0 PROGRAM REVIEW

A system of contractor-internal and formal reviews and audits shall be established and implemented in order to: provide visibility over the development of IDAMST Mission, Mission Support, and Operation and Maintenance Software; to obtain Air Force approval of the design approach; to assure configuration conformance of software to design specifications; and to ensure that the software meets the performance requirements of the avionics system and of the mission. These reviews will support avionics system and AMST airborne segment program reviews of similar type. Software reviews may be held in conjunction with, or independently of, other program reviews, subject to prior approval by the Air Force and subject to the requirement to support overall program schedules and delivery commitments.

8.1 IDAMST SYSTEM REVIEWS

IDAMST software design shall be an essential consideration in the IDAMST avionics system designs. Therefore, an internal team headed by the contractor's Chief Programmer shall assist in the performance of avionics system engineering tasks to assure: (a) that system functions are properly allocated to software, (b) that the optimal software and computing system partitions are selected, and (c) that all software considerations for system design are assessed and evaluated prior to final system definition. To assure accomplishment of these functions, the Chief Programmer will participate in the following reviews:

a) System Requirements Review (SRR)

b) System Design Review (SDR)

8.1.1 System Requirements Review (SRR)

The objective of the SRR is to determine the adequacy of the definition of system requirements. It shall be conducted after the system functional requirements have been established. The particular concerns for the software group in this review shall be as follows:

a) Adequacy of hardware and software trade-offs and studies.

b) Adequacy of definition of software requirements.

c) Cost effectiveness of the software allocations.

d) Software-to-system hardware interfaces.

e) Allocation of functions to software.

f) Reasonableness of performance limits for software allocated functions.

g) Initial timing and sizing estimates.

h) Correlation of software flight operational sequence diagrams to the system level sequence diagrams.

8.1.1.1 General

The SRR's are in-process reviews conducted during the initial system definition effort. Such reviews shall be conducted after the accomplishment of functional analysis and preliminary allocation of functional requirements to end items, to determine initial direction and progress of the AMST Systems Management effort in arriving at a complete and consistent set of top-level performance, design, and verification requirements.

8.1.1.2 Requirements

Representative items associated with the software design to be reviewed include the following, as appropriate:

a) Application and Executive Requirement Analysis
b) Functional Structure Analysis and Design Validation
c) Functional Partitioning
d) BITE Trade Studies
e) Specialty Discipline Studies
f) System Interface Studies
g) Generation of Specifications
h) Configuration Management

8.1.1.3 Other Considerations

Information which is useful to design analysis and available from the Procuring agency shall be requested at this review.

8.1.2 System Design Review (SDR)

This review shall be conducted when the system definition effort has proceeded to the point where system requirements and the design approach are precisely defined. The SDR is conducted in sufficient detail to insure a technical understanding for: (1) the system functions identified in the system and mission specifications, and (2) the deliverable Mission, Mission Support and O&M Software functions identified in the mission avionics system performance.

8.1.2.1 General

The SDR shall be conducted to evaluate the optimization traceability, correlation, completeness, and the risk of the candidate system implementation including the corresponding test requirements for fulfilling the System and Mission requirements. The review encompasses the total system requirements, including software. Also included shall be a summary review of the Systems Management activities, (e.g., Mission Software requirements analysis, functional structure analysis, functional partitioning, program risk analysis, Digital Integrated Test System (DITS), trade studies, intra- and intersystem interface studies, integrated test planning, and Software Management) which produced the above system definition products.

A technical understanding shall be reached or the validity and completeness of the System, Mission, Software and other requirements with respect to the system architectural design.

8.1.2.2 Purpose

An SDR shall be conducted as the initial full-scale development review before proceeding with the preliminary functional design of the system component elements. The SDR is primarily concerned with the overall review of the operational/support requirements, updated/completed system specification requirements, allocated performance requirements, and the accomplishment of the Systems Management activities to insure that the definition effort products are "necessary and sufficient". The purpose of the SDR inlcudes assuring that the updated/completed systems requirements are adequate, that the allocated requirements represent a complete and optimal synthesis of the system requirements, and that the technical program risks are identified, ranked, avoided, and reduced. A system architecture is presented for evaluation. It also assures adequate DITS design tradeoffs.

8.1.2.3 Content

The SDR shall include a summary review of results of significant trade studies and the following Systems Management activities which relate to IDAMST software development:

a) Applications or Executive Requirements Analysis

b) Functional Structure Analysis

c) Requirements Allocation

d) System Growth Capability

e) DITS Hardware/Software Trade Off

8.2 IDAMST SOFTWARE DESIGN REVIEWS

Two series of formal software design reviews shall be conducted between the Air Force and the contractor. These software design reviews shall be scheduled by and conducted at times and locations established by program schedules approved by the Air Force. The two sets of technical reviews shall consist of a Preliminary Design Review (PDR) and a Critical Design Review (CDR). The contractor shall be responsible for establishing the agenda for the formal reviews and shall coordinate preparation of the review data with the Air Force and the integrating contractor.

8.2.1 Preliminary Design Review (PDR)

A PDR shall be conducted for each of the IDAMST software deliverables, as allocated from the System Mission Requirements. It shall be conducted prior to the detailed design process, subsequent to submittal of the Software Functional Requirements for Air Force approval. In general, at this point the software design activity has progressed to the point where software functions have been fully defined and where functional flow diagrams show the data and control between the executive, structured functions, and external inter-

faces. (Type B5 specifications).

The PDR shall constitute an evaluation of the software design approach and the initial design, prior to introducing implementation criteria (high order language, etc.). The contractor shall not proceed with detailed software design until after Air Force approval of the Software Functional Requirements, unless otherwise directed. Separate PDRs may be held for independent CPCI's or groups of functionally-related CPCI's, subject to prior approval by the integrating contractor and the Air Force.

8.2.1.1 PDR Objectives

The PDR shall accomplish the following:

a) Final approval of the Software Functional Requirements to establish the software design baseline, if not previously approved. The compatibility of the design approach with the requirements shall be established by a presentation of functional flow diagrams, memory maps, control maps, timing estimates, descriptions of significant algorithms and other appropriate data and/or engineering documentation.

b) Evaluation of the progress and technical adequacy of the design approach. This shall be supported by the presentation of algorithms, proposed programming techniques, software standards and practices, and functional and design simulation results, estimates of storage and timing requirements, control and data structures, etc.

c) Establishment of the existence and compatibility of the physical and functional interfaces between the computing system and the system operational hardware. This shall be accomplished by a presentation of the proposed data formats, timing constraints, interface specifications and drawings, the applicable system design data, and other appropriate systems engineering data and documents.

8.2.1.2 General

The PDR shall be a formal technical review of the basic design approach. It shall be held after Air Force approval of the functional requirements and the accomplishment of preliminary design efforts, but prior to start of the detail design. A collective PDR for a group of Mission, Mission Support, and Operation and Maintenance Software components, treating each individually, may be held when such an approach is established as effective in terms of cost savings, project schedules, or other desiderata.

8.2.1.3 Items to be Reviewed

In general, the PDR shall provide a review of the following:

a) Preliminary design synthesis of the approved functional requirements.

b) DITS trade-offs and design studies results.

c) Functional flows and requirements allocated data.

d) Interface data, internal and external to the software.

e) Development schedule.

8.2.1.4 Software Components

The software PDR's shall be conducted for one or a group of software components after approved functional requirements, including detailed interface definitions, are available. The design approach shall be made available by the contractor for review at the PDR. As a minimum, the following shall be performed:

a) Review all detailed functional interfaces with AMST system hardware. Review word types, message data, storage estimates available within the computer, timing, and other considerations which were established in the functional requirements. At this time, the interfaces between the software and hardware facilities shall be defined at a level low enough to preclude subsequent definition at a lower level.

b) Review all functional interfaces between segments within the system. (A more detailed review of these interfaces at a lower level is conducted at the CDR).

c) Review the structure of the Mission, Mission Support, and O&M Software as a whole, with emphasis on the following:

   1) Allocation of software components to the functions delineated in the functional flow diagrams.

   2) Estimated memory requirements and allocation.

   3) Software operating sequences and control.

   4) Design of the common data bases.

   5) Adherence to Design Standards.

   6) Validation of structural interfaces, data and control.

d) Analyze critical timing requirements of the system as they apply to the computer program to insure that the proposed design will satisfy the timing requirements.

e) Review the computer program interactions with the user (pilot, operator, etc.) requirements.

The following data shall be presented at the software PDR:

a) Functional Control Map - This structure identified all control interfaces between software functions in a hierarchical order. The type of control is indicted (e.g., real-time) as are any dependencies involved in this control (e.g., time, event).

b) Data Structure Maps - For each functional component in the control map, this description identifies data characteristics and the intersections or common data for each components' subfunctions.

c) Functional Flow Diagrams - These structured diagrams depict the sequence and control of the function performed by each component identified on the control map.

The SOW shall be consulted to establish data to be presented at the PDR.

8.2.1.5 Post Review Action

After completing a PDR, the contractor shall publish and distribute copies of the Review minutes as specified by the Contract Data Requirements List (CDRL).

8.2.2 Critical Design Reviews (CDR)

A CDR shall be conducted when the detailed design and development of software modules are essentially completed and the documentation reflecting the software design is ready for release. The CDR shall include a review of the detailed design of the IDAMST software, based on preliminary design specification documentation or other appropriate engineering design media.

8.2.2.1 Scheduling

Depending on the complexity of the Software System, one CDR may be conducted for the entire Mission, Mission Support, or O&M Software package or a series of CDR's may be scheduled covering various components of these packages.

8.2.2.2 CDR Objectives

The CDR shall accomplish the following:

a) Determination that the detailed software design satisfies the requirements established by the Software Functional Requirements and the design approach established by the PDR. This shall be accomplished by a presentation of the module design to the detailed flow diagram level.

b) Establishment of exact internal interface relationships among operational software modules, between these modules and the Executive and jointly between all elements of the software sub-system. This shall be accomplished by a presentation of the internal interface design for the modules and the Executive, including data flows between all functional elements of the IDAMST Mission Software. The presentation shall also cover interface control drawings applicable to software.

8.2.2.3 General

The software CDR shall be a formal technical review of the design. The CDR is normally accomplished for the purpose of establishing the integrity of software design at the level of detailed flow diagrams. The primary product of the CDR is formal identification of specific software documentation which will be released for implementation and verification. By mutual agreement between the contractor and the Air Force, CDR's may be scheduled concurrently for two or more software components.

8.2.2.4 Items to be Reviewed

a) Adequacy of the detail design reflected in the specification in satisfying the functional requirements for the software component being reviewed.

b) Adequacy of the detailed design.

c) Control Structure Maps.

d) Design studies and analysis

e) Data Scope Maps

f) DITS Design.

The following data shall be presented at the software CDR:

a) The complete specifications, excepting source code listings.

b) Supporting documentation describing results of studies and analyses.

c) Software partitions or allocations among processors.

8.2.2.5 Detailed Evaluation

a) Detailed design diagrams shall be compared with control structure maps to determine system compatibility

b) Establish compatibility and correlation of design with the Requirements.

c) Establish system design compatibility and review all interfaces between software functional modules by analysis of detailed flow diagrams and other descriptive documentation.

d) Establish design integrity by review of available analytical data in the form of data scope maps and detailed flow diagrams for all program components.

e) Review interfaces between the software and equipment items to insure that changes, etc. have not affected compatibility.

8.2.2.6 Post Review Action

After completion of CDR, the contractor shall publish and distribute copies of Review minutes.

9.0 STRUCTURED PROGRAMMING STANDARDS

9.1 THE TOP-DOWN CONCEPT

There are three basic principles to acquiring a top-down-structured program. First, the program should be designed and implemented by top-down methods. The second principle is to plan the software in a structured manner. This requires rules and enforcement of these rules on the part of everyone involved. The third principle is to program in a HOL which (1) enforces structured programming rules, (2) contains static and dynamic debug features, and (3) automates designs in the software development process.

The concept of top-down can be thought of as planning each level of the program and each level of the accompanying data modules from top to bottom completely.

When writing a paper or preparing a talk, one first jots down notes. Then an outline is developed. After the outline is expanded by way of a few iterations, the paper is rewritten. Many revisions are usually necessary if the paper or speech is of any significance. A software program shouldn't be much different in the way it is created. Better organized papers and speeches are, of course, much easier to follow and understand than a paper or speech that rambles back and forth. The same holds true for an individual program; even more so for a whole software system. However, an iterative process is necessary, just as when writing a paper. Each iteration in the design of the software system will bring the definition of each level closer to the best modular, top-to-bottom concept for the particular system in mind. During the design process, it will become apparent that some modules on one level will be applicable for use on another level; some data modules must be accessible to more than one program module; and, in some instances, it will be important that some data modules be inaccessible to particular program modules.

Top-down design is analagous to a tree where each level of planning is another branch. Each branch in turn can be a node with branches of its own. But, if one chooses any point at an outermost branch, it is always possible to retrace the growth of the tree and follow each mode back to the original trunk. Likewise, one never loses sight of the original problem by designing a software system top-down.

At each level of planning the data modules are planned just as carefully as the program modules. Top-down data module design will indicate the data on the outer levels that must be available to the inner, or lower, levels; and at the same time will indicate those data modules that require no interfaces. The "scope" of the data is therefore an important concept.

9.2 A CASE FOR STRUCTURED PROGRAMMING

In the past, a programmer's objective was to generate code as efficiently as possible; there was not enough concern for those people who had to understand, modify, and many times debug a program long after the original programmer had disappeared from the programming effort. At present, it has not been possible to be 100% sure that there are no errors in actively used software. The task of proving any program correct by conventional means is expensive

and is not guaranteed to be reliable. This is mainly due to the older techniques of generating and testing a program. The older method invariably shows the presence of errors in the program, but in fact, there is no way that testing can detect the absence of errors.

New concepts in programming style make it possible to attempt to prove a program is correct. The technique involves two basic steps. The first is to prove certain programming constructs to be correct. The second is to allow the programmer the luxury of using these constructs in the same manner that one uses a mathematical theorem as a building block. By concatenating these building blocks, a simple sequence for a structured program develops. Not only can the entire sequence be proved correct, but the modularity of each building block anticipates future program modification.

The modular building process encompasses the two basic modular types: data modules and program modules.

9.2.1 Data Modules

Data modules are structured by 1) the individual programmer when defining non-local variables, or 3) by the language automatically.

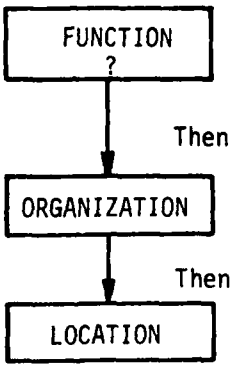


FIGURE 9.1 The Data Module

The function of a data module is determined by program constraints, timing constraints, data interface constraints, error recovery constraints or I/O constraints. The organization of a data module is derived from its function and specifies the data type (bit, integer, vector, matrix, character, scaler) and the array and structural qualities. The size, specified in the organization, and the function make it possible to define a block for the data module. The location for each block can be a location such as the COMPOOL, can be specified as outer level, or can be specified as local to a particular program. In addition, this location can be an absolute location such as a sensor known to the program via a particular I/O device.

9.2.2 Program Modules

A program module can be an open block, which is in-line (the IF construct) or a closed block (the PROCEDURE). A module is characterized by the particular function it performs. It has a single entrance, i.e., the single entry point of a procedure or the first statement of an in-line block. It also has a single exit in the sense of returning to the same place from which is was invoked, i.e., procedures return normally to the place from which they were called and all in-line blocks exit only to the statement immediately following the block.

The one entrance, one exit structure of module linkage assures the programmer that the state of the program is always defined. That is, at any point of execution a simple dump reveals the current state of the program in terms of the set of active modules and their calling relationships. In the event a dynamic error does occur, the simple sequenced structure makes it possible to identify the error using the building blocks as coordinates of the program.

The simplest construct to prove correct is a group of sequential statements or blocks of code. These are nothing more than an ordered list in which all possibilities are clearly visible by simply following the code.

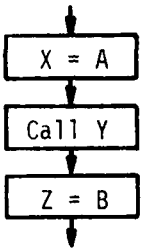


FIGURE 9.2 Structured Program Consisting of Sequential Blocks of Code

A programming style which advocates the use of branching (GOTO) for the main purpose of producing efficient code would produce a program difficult to understand, modify, debug, or prove correct. One could not simply follow the code. If a language contains the proper control statements, it is theoretically possible to construct an efficient program with GOTO's. These control statements are:

IFTHENELSE

DOUNTIL

DOWHILE

DOCASE

It is also true that these control statements can be proved correct from a logical point of view.

The IFTHENELSE provides a simple choice between two possibilities. Since this statement is always entered at the beginning and has a single EXIT, the entire construct can be thought of as a single module whose internal structure is not relevant to the context in which it is used.

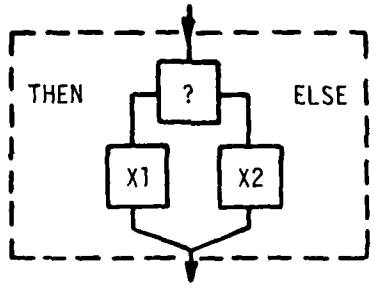


FIGURE 9.3 If? Then X1 Else X2

The DOCASE construct is just a selection process easily proven correct by enumerative reasoning and, again, has the characteristic of a single entry and exit.

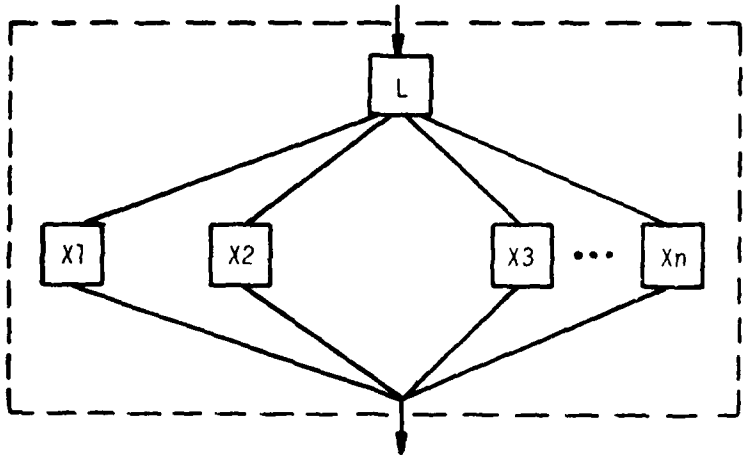
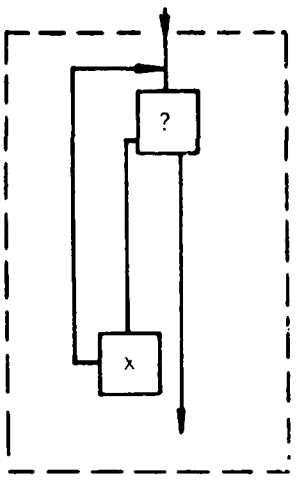
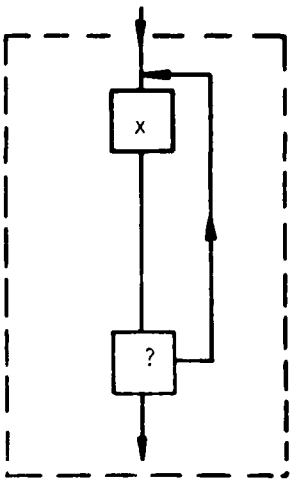


FIGURE 9.4 DO CASE L: Case L of (X1; X2;... Xn)

Repetition via DOWHILE or DOUNTIL statements (Figure 9.5a) and REPEAT (Figure 9.5b) can be proved correct by mathematical induction. Again, we have a control statement with a single entry and exit. As long as we enforce the rule that the loop variables of DOWHILE or DOUNTIL must be a local variable (changed only within the realm of the DOWHILE or the DOUNTIL the loop will not depend on an outside variable and the proof for correctness will not be difficult. Imagine how difficult it would be to prove programming logic correct where a GOTO depends on a non-local variable. In this case, branching via a GOTO could lead to an infinite repetition of a particular set of statements.



(a) DO WHILE or DO FOR:
while? do X



(b) DO X UNTIL ?

FIGURE 9.5 Repetitive HOL Statements

9.3 Structured Flow Charts

IDAMST software is to be characterized by a combination of two basic programming styles: structured programming and top-down techniques. The structured programming concept is characterized by an ordered set of program instructions. Accompanying structured data modules directly convey the flow of data. The use of top-down techniques results in program flow which can be compared to the organization of a book: the "table of contents" specifies the entire program flow on page one; each "chapter" is the expansion of a particular block. Conventional flow chart techniques cannot adequately convey these organizations. The block structure, the scope, and the data flow inherent in any structured top-down program must be represented by a structured flow chart.

Structured flow charting is based on the premise that the functional flow of a program includes 1) decision statements based on program data, 2) CALLs to sub-modules or other programs which manipulate data and 3) in-line equations, which can be thought of as language supplied CALLs or "degenerate" CALLs which manipulate data.

The functional representation of a program is the first page of the structured flow chart. The complete data module is represented on the second page. The functional program is the third page. The succeeding pages expand the modules found on page three. A complete data module should accompany each PROCEDURE, TASK, or functional block.

A complete data module, associated with a functional block, is defined as the set of data referenced and assigned within a module. This set includes data referenced or assigned within each sub-module and each CALL to an outside program. The complete data module is an inherent structure to any program module and should, therefore, accompany the functional flow of the module.

The basic unit of a structured flow chart is the "block". A "block" is a module which has a single entrance and a single exit. It will be represented as:

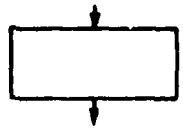


FIGURE 9.6 The Baisc Unit of a Structured Flowchart

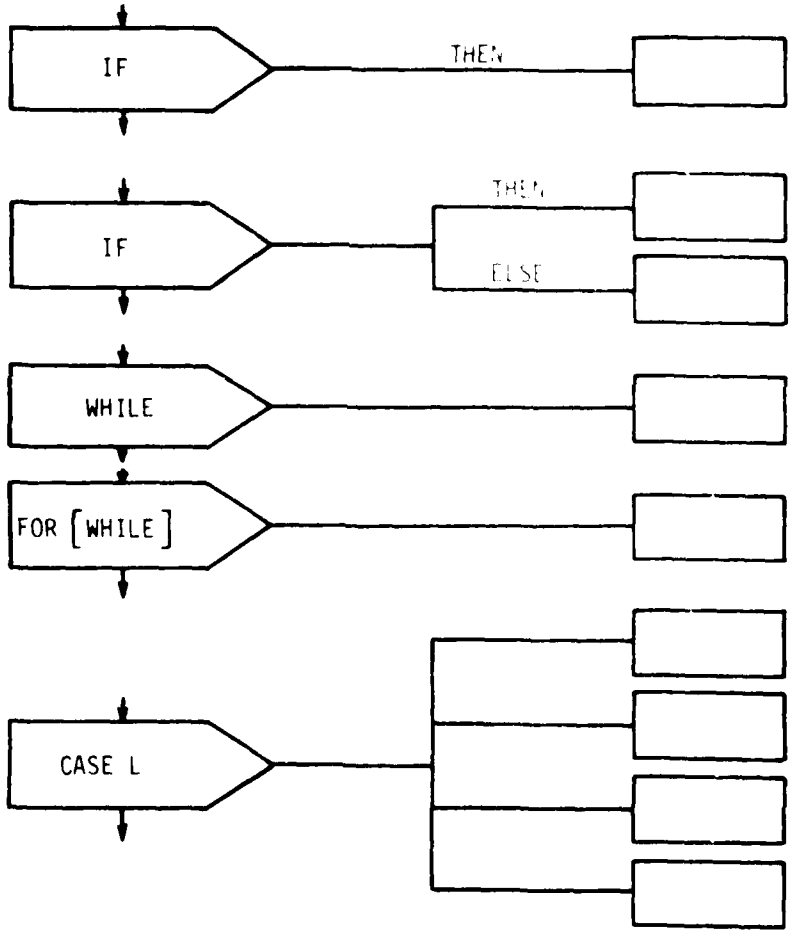
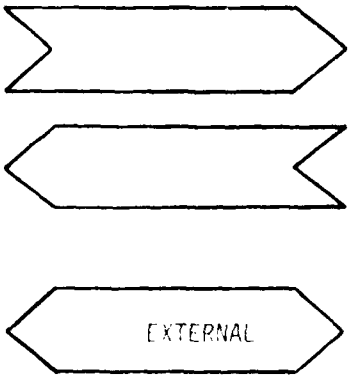


Figure 9.7 Decision Statements

The functional flow of the program depends on the decision structure where the object of that decision can be thought of as a sub-module to the decision statement. The statements used to make decisions are the basic structured statements shown in Figure 9.7. Inherent in this representation is the knowledge that any decision statement performs the required sub-module resulting from the decision and immediately returns to the next statement in the main program flow.

The basic symbols required to understanding a data module are



for input variables or variables referenced

for output variables or variables assigned

to indicate data modules from sub-modules referenced within the program called by the main program

FIGURE 9.8 Basic Data Module Symbols

The notation shown in Figure 9.9 is to be used to specify location and organization for data module elements. The loaction of a data sub-module is indicated within the basic symbol described in Figure 9.8. The data elements, with organization indicated, are listed within the block accompanying the location notation.

The RETURN statement shall be represented as

FIGURE 9.10 The Structured RETURN Symbol

to distinguish it from the decision statements.

9.4 Program Structuring

The task of program structuring is of major importance; decisions made for this task determine and dictate the total structure of the software both statically and dynamically. Program structuring defines the building blocks, the control mechanisms and the interfaces of the software. It is in this effort that the interrelationship of systems software, applications software and data is entirely mapped out. In the same way that structured programs require rules and enforcement of these rules, the software system itself must be

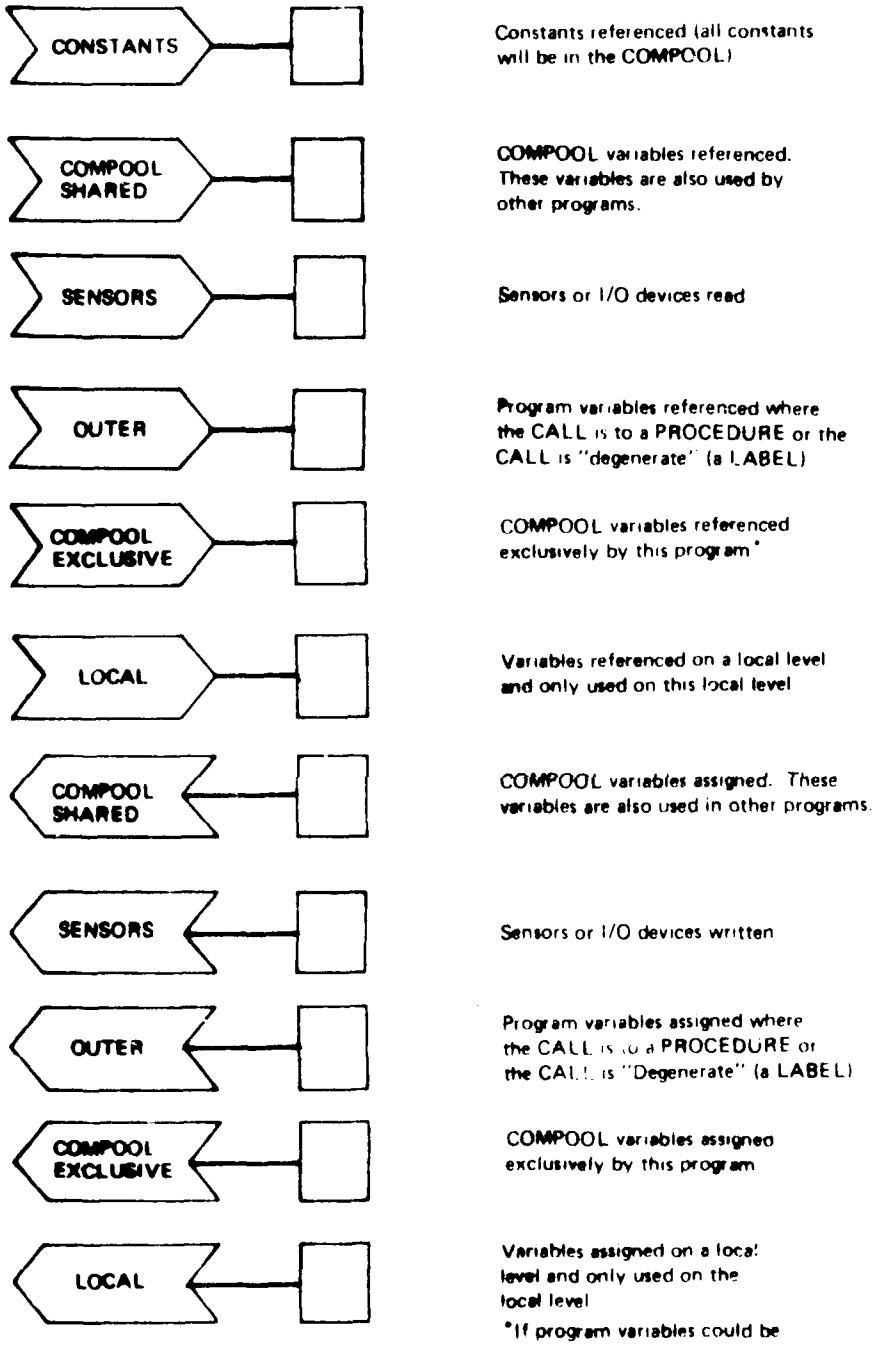


Figure 9.9. The Data Flow Performed by the Object of the CALL or by the Object of the LABEL

structured according to rules and provision must be made for techniques to enforce these rules. Software tools, including the language and the digital simulator, act as automatic aids for correct software structure. Likewise, the executive, including the systems software, can be an effective tool in the dynamic enforcement of these rules.

Program structuring encompasses the definition of 1) the software executive structure (systems software), 2) modularity (including program and data modules), 3) structure within each block, 4) the interface points between levels and within levels, 5) the structure of the total program including such considerations as timing, memory, priority, error recovery, etc., 6) the requirements and interfaces of data modules common to more than one program (COMPOOL), and 7) automatic sequencing and the constraints imposed on the system due to non-automatic sequence.

There are many difficult problems in laying out software. One of the main problems is that of defining modularity. One definition of a module is a unit which performs a specific function. It has an internal structure and local (private) variables which are unknown to the outside world. We all talk about modularity, but are we taling about the same thing? Both program and data modules might be divided according to:

Mission phases

Mission functions

Blocks of memory

Divisions of software error recovery

Critical vs. non-critical mission phases

Subroutines

Control vs. calculations routines

Data divisions

Components of the assembly, e.g., systems vs. mission modules

Synchronous vs. asynchronous logic

Instruction sets, e.g., DOCASE

Real time vs. non-real time logic

The system designers must determine the scope of the modules as well as determine which modules are assigned to programs, procedures, tasks, functions, etc. Section 10 of this Appendix is written to provide a baseline for the IDAMST Software Systems Architecture.

10.0 SOFTWARE SYSTEM ARCHITECTURE

10.1 SYSTEM ARCHITECTURE

The architecture of a system implies a separation of functional components, the control of one component over another, and a dependence of one component on another. The DAIS system architecture is depicted in Figure 10.1-1 showing the separation of hardware and software functions. The applications software is functionally separated from the hardware by the executive software just as the avionic subsystems are separated from the computers by the remote terminals and data bus.

The executive system provides service functions to allow the applications software to symbolically control the avionics system while the executive retains physical control over the entire hardware system (processing, bus control interface units, remote terminals, etc.). In this manner, changes to the hardware system operation may be isolated in the executive system; functional changes to the avionics systems affect only the application software, not the executive control system. Thus, the software system achieves virtually complete separation of functional and operational responsibilities.

10.1.1 Software Architecture

Software is separated into two basic functional components: Executive and applications. The executive masks the applications from the hardware system architecture, i.e. multiple computers and remote terminals configuration. The applications control the execution of all software functions by invoking the executive to schedule tasks, events and I/O. The executive functions are created to service application requests and are therefore dependent on what these requests are.

The local executive provides services to tasks for the computer in which it resides. The master executive manages the multicomputer configuration and the data bus traffic. The master executive coordinates the I/O and service requests from each computer through each computer's local executive. The master executive itself uses the local executive in the master computer to invoke its operations for bus control and configuration management and, as such, is a straight forward extension of the local executive.

The applications software architecture consists of a master control program (master sequencer), mission phase operational sequencers (OPS), specialist mission functions (SPEC), display processes (DISP) and avionics equipment moding programs (Equip). The master sequencer controls the major application system functions which consist of mode requests monitors, subsystem failure detection and an applications configurator.

10.2 ARCHITECTURE REQUIREMENTS

The following requirements have been devised to effect a high level of functional separation and control. These requirements affect both the functions of the executive system and the manner in which the applications software modules communicate with other application modules. The application software must adhere to the program structure and data structure imposed by these requirements. For applications software structuring purposes, all program modules appear as if they reside in one large processor. The mechanics of inter-processor communications are discussed fully in Section 10.2, the executive software system, and are invisible to the application programs.

DAIS M. S. STRUCTURE

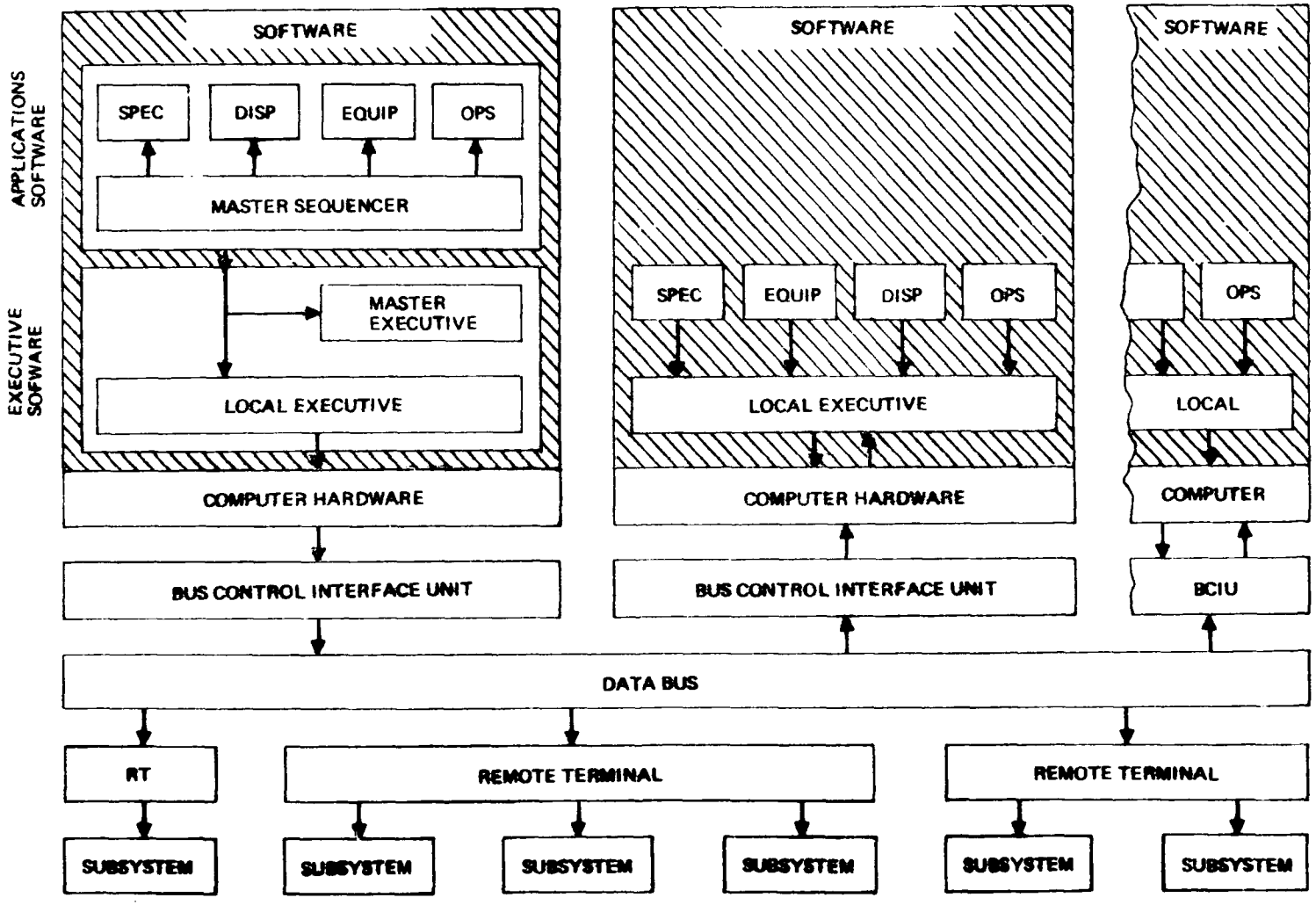


Figure 10.1-1 DAIS Architecture

ROBERTS 107164-4

10.2.1 Program Structure

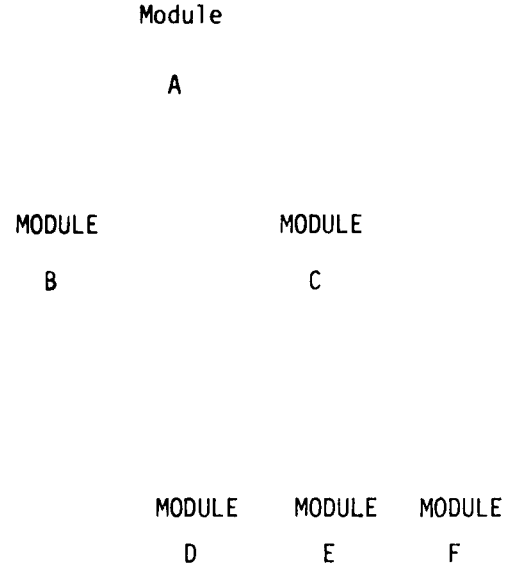
All applications software programs consist of modules that are either procedures or in-line functional blocks. A procedure is a compliable unit which is either a task or a comsub. A task is a program module unit that accomplishes a single function, whether to control sub-modules, or to perform a calculation under control of a parent-task. A comsub is a re-entrant program module unit used only for specific calculation purposes. An in-line block is a HOL assignment statement or a group of such statements within a begin-end construct which performs a single function within a procedure. Control over procedures is established by their positions within the program's hierarchical control structure. Communication is established by declared common data between the communicating modules. Procedures all have a single entry and a single exit point. Procedures cannot be nested for compilation purposes since it is then not possible to fully specify the interface structure of the nested procedures.

A task can reside in any one processor; a comsub is duplicated in another processor if a task in that processor requires use of the comsub.

10.2.2 Data Structure

10.2.2.1 Internal Data

All task data interfaces are declared in unique common data pools that exist for each nodal family. For example, a data pool is declared in module A for the nodal family A-B-C; one in module C for the C-D-E-F nodal family. Data blocks exist for each communicating set of modules within the nodal family. These data blocks are completely distinct portions of the common data pool for the nodal family. A data variable is uniquely known in a data block of a common data pool. For example, data blocks exist for A-B, B-C, A-C and A-B-C, all part of the A-B-C nodal compool.



10.2.2.2 External Data

All data external to the applications software structure (sensor, controls, displays, etc.) is declared in common program data pools. All procedures requiring access to an external data compool are not restricted to a nodal family. The organization of the external data compools is on a data bus message basis, i.e. one bus message, one compool message bundling, i.e. the grouping of several disparate data variables into one bus message, is mirrored in the block organization of the compool such that unbundling of a message is performed automatically by block access.

10.2.3 Communication Between Modules

Communication is established between two modules in the manner one module controls the other, and with the passing of data from one to the other. Control is established by the relative position of the modules on the hierarchical control structure. Data communication is provided through the structure of data pools.

10.2.3.1 Control Communication

A task is activated by the local Executive when a set of events associated with the task reaches a desired, predetermined configuration, and the task's controller has, at some time in the past, invoked the task. The events can be set by other tasks (e.g. when an altitude is reached) or by the Executive (e.g. when bus minor cycle No. 2 is recognized). See Section 10.2.3.2.1.).

A task can be either invoked by only one other task. The invoking task is the parent and is one level higher in the hierarchical control structure. A comsub can only be called in a conventional fashion. It cannot be scheduled. A comsub appears in the hierarchical control structure for each task or other comsub calling it. A comsub can in no manner invoke any task, but it may invoke another comsub. A comsub has no access to any Executive services.

Invocation of a task is accomplished when an invoke statement is encounted in the controlling task. Every task in the system is either a schedulable task or a callable task. The designation is specified at linkage editing time and does not change during system execution. Invocation of each type results in different actions:

1) Call Task - the action of invoking a call task causes suspension of execution of the parent task until the call task terminates. The call task is immediately set to the active state and may run when its priority is highest in the computer. Under normal circumstances, since the priority of the call task is one less than the priority of the parent, the call task will run immediately. This need not be the case for the invocation of a call task in a different computer.

2) Scheduled Task - invocation of a scheduled task does not suspend execution of the parent. The scheduled task will then become active whenever a set of events, specified for the task reaches a predetermined configuration. Scheduled tasks have two sub-categories:

A) Sequential - A scheduled task is sequential if it is not allowed to execute during the times its parent is in the active state.

B) Concurrent - A scheduled task is concurrent if it is allowed to execute while its parent is active.

The definition of sequential and concurrent are not dependent on procedure partitioning.

Invocation of a comsub is a convential call and, as such, is invisible to the executive. Tasks can be directly terminated (de-activated) or cancelled (de-activated and de-invoked) by only the module invoking it. If a task is terminated, it is de-activated. All scheduled tasks invoked by it are terminated, and all called tasks invoked by it are cancelled. If a task is cancelled, the task and all its invoked tasks are cancelled.

Priorities are globally and uniquely assigned to each task. Once a program module is activated, it competes for execution with all other activated program modules in the same processor on a priority basis.

10.2.3.2 Data Communication

Data communicated between modules falls into three areas: real-time events, shared variables in common data pools and explicit parameter passing.

10.2.3.2.1 Events

Events are elementary data variables with numeric value of 1 or 0. Events are associated with:

a) arrival of asynchronous messages over the data bus;

b) interrupts from the data bus interface unit;

c) certain local executive services, and

d) encountering a signal-event statement in the procedure code during execution. Multiple events may be signalled with one whole statement.

The desired event condition configuration is pre-determined and fixed at the time the system is linkage edited (by the PALEFAC support software). A task is activated when a set of events, associated with the task, reaches a desired, configuration. The set of events act to enable the task's execution when they read the desired state. Events are either latched or unlatched with respect to a individual task. Latched events remain fixed at a value until explicitly changed by action of the applications software. Unlatched events are reset for the task when that task becomes active. Any number of the conditioning events can be declared as unlatched. The latched events associated with the task may be set whether the task is scheduled or not.

Events are classified as global, nodal or control events. The classification reflects the scope of the event.

1) Global events can be known to any task. This category includes only those events which originate external to the computers; plus the minor cycle event.

2) A nodal event is known only by the tasks of one nodal family. Such an event is used for purposes of real-time sequencing among the members of the nodal family.

3) A controlled event can be:

    a) signaled only by the offspring of one nodal family.

    b) known by one task acting in a controlled capacity, which is reachable by traveling only upward from the signaling task of the control hierarchy.

Control events are used in real-time to explicitly obtain the decision making capability of a controller. An event classification naming convention will be adopted for the mission software. The classifications will be incorporated by PALEFAC to enforce event scope.

10.2.3.2.2 Compool Data

Date variables are classed as either:

1) External - External data is available to all tasks. This is I/O data to the computer system from the environment, or to the environment from the computer system. Examples are:

    a) pilot input

    b) displays

    c) sensor data

    d) IMU data

2) Inter-Task Data - This is data which is supplied by one task to another task. It is part of the interface specification for a task. Inter-task data is know, by the name to only one nodal family. The same data can be known in a nodal family and a nodal family on the next lower level only by explicit transfer from one compool to another by the controller of the lower nodal family.

3) Local - Local data is known only to a specific task. Local data must always be either initialized by the task at the start of execution or assigned as a specific function of compool data acquired through a read statement. This is a guarantee restartability of the task.

Data is further classified as synchronous or asynchronous. This classification is specified to the linkage editor (PALEFAC). PALEFAC uses this information in the construction of a data descriptor block to be associated with the data. The executive takes different action depending on the classification.

The classification, the effect on PALEFAC, and the effect on the executive are not explicitly visible to the mission software.

10.2.3.2.3 Compool Purpose

Compools exist to identify blocks of data for explicitly defined functions. They are classed as:

1) Synchronous external data - I/O into or out of synchronous external data compools proceeds without Executive service.

2) Asynchronous external data - I/O data requires Executive service to route data and, for some situations, to activate tasks that have been invoked on the reception of the asynchronous data block.

3) Inter-Task Data (Asynchronous) data - There is one inter-task data compool for each nodal family.

No distinction as to classification is made within a compool itself. All such information is used by PALEFAC in the construction of data descriptor blocks and executives tables.

10.2.3.2.4 Compool Organization and Operation

Compools are organized into blocks of variables. Blocks are entirely read or entirely written by a program module.

A HOL syntax could be:

```
READ (X,Y);
Write (X,Y).
```

Here X is assumed to be a compool block name and Y is a local data area of equivalent size to X. Read and write is performed on a block level.

a) Organization of variables into compool blocks is prescribed so that no information, extraneous to a module, is communicated in a compool read or write.

b) Any compool block may have copies in several processors as required by partitioning. A write statement, referencing a block in one processor, will automatically cause an update to be performed on copies in other processors.

c) Except for copies required by partitioning, a compool variable appears in only one compool block.

d) Synchronous I/O is directly into and out of compools.

e) Each program module maintains its own local copy (using local names) of compool blocks referenced by the module.

f) A program module may not both read and write the same compool block.

g) All inter-task compool blocks utilized by a nodal family are gathered into a single compool. This is the only inter-task compool associated with the nodal family.

h) Each external compool comprises a single bus message. The external compool is divided into blocks which reflect the collection of distinct data variables into a single transmission message (bundling).

Particular organization of the blocks is constrained by a) above.

The values of a compool variable can be assigned in more than one program module if and only if:

a) there is no more than one such module on any one control branch, and;

b) the program modules are mutually exclusive, i.e., absolutely non-concurrent in real time.

10.2.3.2.3 Parameter Passing

Passing of parameters occurs only between a program module calling a comsub. The calling task or comsub explicitly declares the parameters to be passed to the called comsub. Being re-entrant, comsubs require parameter and do not have access to compools. There is no parameter passing when a task is called or scheduled; all data on the interface between the invoking task and the invoked task is declared in their compool.

10.2.3.3 External Communication

External communication is that communication involving system components outside of a processor. There are two types of external communication:

1) Intertask communcation across partition boundaries. Because of features built into both the local executive and the PALEFAC, intertask communication, control and data, between processors is completely invisible to the applications software.

2) Tasks that depend on subsystems at remote terminals must communicate with the subsystems. The executive system commands transmission between processor and remote terminal, either on a periodic basis or asynchronously. The data sent or received is therefore known to be externally communicated data and is declared as such to the executive. The applications software has no specific knowledge that this data is external as opposed to intertask. The applications software reads and writes compool data in the usual manner. The executive performs the chores necessary to communicate this data to or from the outside world. Discrete signals which are communicated to and from a remote terminal are bundled and unbundled by the compool organization.

10.3 EXECUTIVE SOFTWARE SYSTEM ARCHICTECTURE

10.3.1 MASTER EXECUTIVE

The master executive manages the avionics operating configuration and includes:

a) Data Bus Control - allocates time segments on data bus for synchronous processor-remote terminal communication and for asynchronous processor-processor messages.

b) System Error Management - Monitoring and analyzing errors related to avionics configuration.

c) Configuration Management - initializes multiple computer system at startup and after severe system errors.

d) Mass Memory Management - provides the retrieving of information from mass memory.

10.3.2 Local Executive

The local executive, responsible for only those activities within its CPU, is dependent only on characteristics of that CPU. The local executive exists to provide specific functions for application software and for the master executive. Therefore, its architecture isolates these executive functions from each other so that an application process uses only those functions necessary for its current operation.

Each avionics processor is to have it own local executive. It is the same for all processors, and it manages all activities occurring within a processor. These activities include:

a) Process Control Interface - Uses a task table to activate and deactivate periodic or non-periodic tasks when appropriate conditions have been met. These conditions are based on a logical setting of real-time events.

b) Event Control - uses a table of real-time events to communicate conditions signalled between processors.

c) Data Control - guarantees interlocks between shared data, provides mechanism for data bus transmission/reception, prepares message formats for transmission and decodes messages received.

d) CPU Fresh Start/Restart - used to initialize CPU, to recover from transient CPU failures and to perform self-test.

10.4 APPLICATIONS SOFTWARE ARCHITECTURE

The applications software architecture is based on hierarchical control structure principles listed in Paragraph 10.4.1 and the requirements to facilitate maintenance and growth. The highest level of this structure is determined by the controlling factors involved in operating an avionics system. The are:

Inputs	-	External Request Processor
	-	Failure Detection
Outputs	-	Pilot Display Control
	-	Device Control
Processing	-	Mission Mode Control Sequencing
	-	Mission Functions

10.4.1 Functional Definitions

Mission functions are divided into three categories:

1) Mission operation sequencers (OPS) - Phases of a mission scenario (e.g., preflight, waypoint steer, weapon delivery, landing, etc.). Only one OPS is active at one time.

   OPS correspond to mission phases that perform the operations for mission scenarios. Some OPS are used for all scenarios while others are used only for particular mission situations as required. Waypoint steering and weapon delivery are sequences in these categories.

2) Specialist Mission Functions (SPECS) - Supporting functions that an OPS requires or the pilot desires (e.g., navigation, system tests, weapon stores setup, failure recovery sequence, etc.). More than one spec can be active at one time, depending on certain allowable criteria.

   Spec functions agument or support the operations being performed by an OPS. Spec sequences are suitable for tasks which are not always required, or which may be required ad hoc, not being synchronized with identifiable points in any OPS sequence. Stores setup and navigation control are tasks in the SPEC category. Although they are required by one or more OPS sequences, they can be invoked at any time within the OPS sequences to change operating modes.

3) Equipment Moders (EQUIP) - Functions that mode the avionics equipment (inertial, air data, communication, etc.). These device moding functions are associated with each piece of the avionics equipment set. These functions are selected by the master sequencer when requests are made by the pilot, from other mission software tasks (i.e., OPS or SPEC functions), or from the failure detection functions (subsystem status).

4) Display Processes (DISP) - Calculators for pilot text message construction and mission parameters and symbol positions which are to be placed on the cockpit display devices.

   The master sequencer allows only one OPS to be active at one time, but provides facility to automatically sequence from one OPS to another (e.g., enter waypoint steer after weapon delivery phase completed).

   Requests to select an OPS, SPEC or EQUIP come from external and internal sources. External requests are detected by the request processor and the subsystem status program. The configurator function determines for the master sequencer what SPEC and EQUIP functions are allowed to be invoked concurrent with OPS and other SPEC and EQUIP functions.

   When a detected failure has a high severity, the subsystem status program causes the configurator to take the appropriate corrective

Based on present status and input from an external request or from a detected failure, the mission mode control and sequencer invokes mission functions that generate information for display for controlling avionics devices.

To invoke the proper application functions, the master sequencer must be able to interpret requests or detect failures. It must be able to configure the avionics software functions based on present conditions as well as new requests. The configurator performs this processing function. The basic structure shown in Figure 10.4-1, then, is the highest level of the applications software architecture.

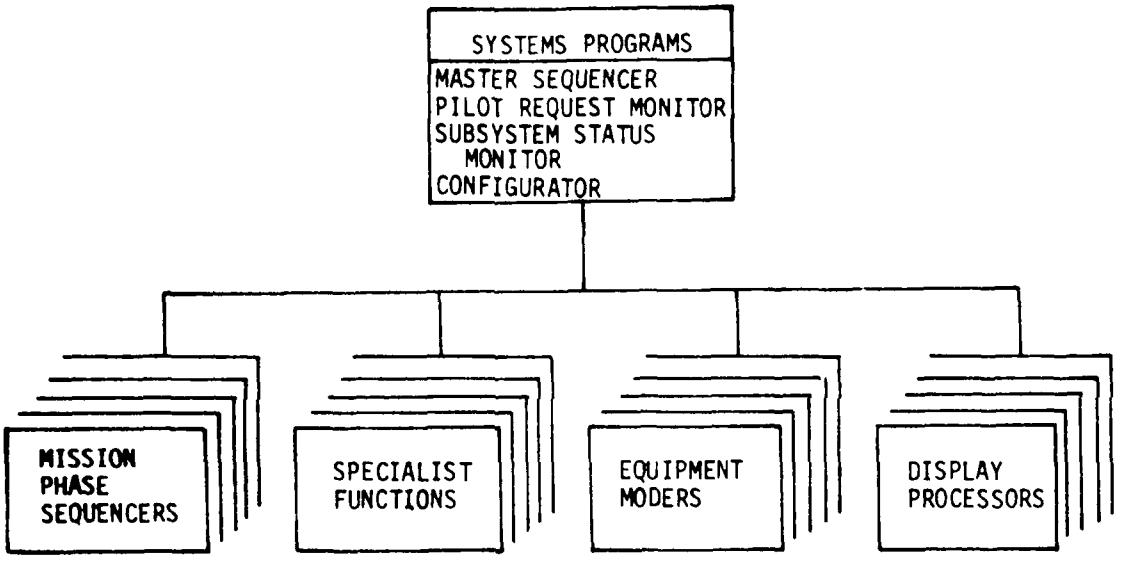


Figure 10-2 Application Software Architecture

action. The subsystem status program maintains the status of avionics equipment modes as well as monitor for equipment failures.

Processing for display devices is handled for each device separately. Display information is computed from OPS and SPEC data and is made available to the CRT display programs. Both OPS and SPEC can request new CRT formats, but SPEC requests take precedence over concurrent OPS requests if the formats requested are not compatible. The OPS and SPEC display requests are compatible when their data can be displayed on the same format template.

10.4.2 Hierarchical Control Structure Principles

All possibilities of control over software functions can be represented as a hierarchical tree structure. Each node (and all of its dependents represent a unique tree structure.

A function is expressed as output elements resulting from input elements, where the elements can be in the form of variables or real-time flags (signals). In order to execute a function, there must be a controller. The controller exists at the node just immediately higher on the tree relative to the function it controls.

Therefore, each member of a requirements hierarchical control structure is either a controller function or a calculator function. Based on its inputs the controller sets up the appropriate functions to be executed, whether another controller or a calculator. This implies a control hierarchy as well as a functional hierarchy. A lower level function cannot be invoked without having been set up by its controller.

The controller has the responsibility to perform a function. For that purpose, the controller has direct control over other functions only on the immediate lower level. This is done by invocations (call or schedule), by the determination of ordering (sequence, priority, etc.) the functions on that level. Since every function receives input from and produces output for its controller, controllers and calculators are relative definitions; the upper level is a controller with respect to its immediate lower level functions. The lower levels of any tree contains no controllers; the highest level of the tree is only a controller.

Application software is dependent on avionics functions (i.e. sensors, aircraft class, display device, etc.). The architecture is designed so that controller functions are dependent on its subfunctions - a subfunction cannot be removed from the tree structure without affecting its controller.

The following list of principles has been used to set up the architecture requirements and structure the functions defined in Sections 10.2 and 10.3.

1) Every decision necessary within a module is directly related to the module's function.

2) Each module is singular in purpose. If a module can be removed from the structure without also changing its controller's input or output, the function is not necessary for the controller's

functional purpose.

3) A module can only invoke modules on its next lower level, not on the same level, and not itself.

4) Two formulations of the same function can be inter-changed only if the input-output relationships remain the same.

5) Modules have no information about their controller.

6) A module can receive input from either its controller, another module with the same controller, or a read request.

7) A module's output can either be an input to another module with the same controller, an output from its controller or a write request.

8) A module cannot receive data from another module unless the data it receives is "known" by its controller.

9) A controller assigns the access rights to variables of only its immediate lower level modules; the controller cannot assign access rights to its own I/O.

10) The priority of a process is higher than the priority of any process on its most immediate lower level.

11) If two processes have the same controller such that the first has a lower priority than the second, then all processes in the control tree of the first are of lower priority than the processes in the control of the second.

12) The ordering between members of any given process tree cannot be alterted by an interrupt.

13) A module controls the priority relationships of those processes on the immediate - and only the immediate - lower level.

14) An explicit priority relationship exists among each pair of processes at the same level which branch from the same most immediate higher level node.

15) If a process can interrupt another process, the control tree of the first can interrupt the second and any member of the second's control tree.

16) Since a module controls the priority relationships of a set of processes, then the module itself cannot interrupt any member of the set of processes.

17) A process cannot interrupt itself nor can a process put itself into a wait state.

18) A process cannot interrupt its controller.

19) A process that can interrupt another process also affects its absolute time loss.

20) There can be no decisions within a module that are made with respect to real-time. Controlling functions with respect to real-time is handled by invokation the function through the executive on a time maintained by the executive.

21) If a function is invoked by a call or a schedule, its controller can invoke another function by a call.

22) The scheduling of two processes can occur in any order; the calling of two functions cannot.

23) If a function is invoked by a call, it cannot schedule a process.

24) A schedule must always cause the processes invoked to be dependent so that the higher level maintains control at all times.

25) The maximum time for a cycle of a process to be completed or delayed can be determined so that static time analysis is possible.

26) A module cannot modify its own input.