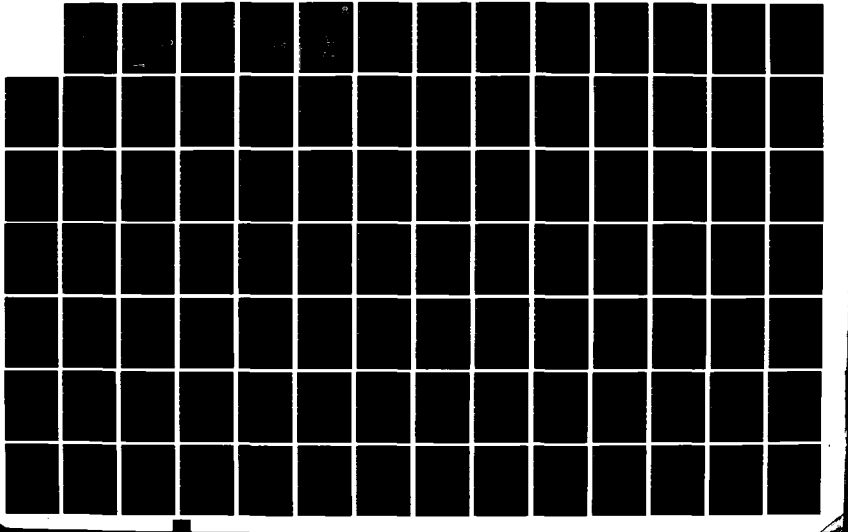
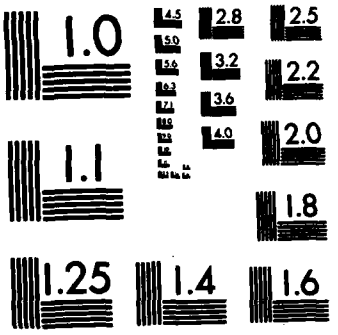
AD-A123 308 LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE MISSILE MINDER AN/TSQ-73 USING T..(U) CONTROL DATA CORP SHREWSBURY NJ GOVERNMENT SYSTEMS 09 NOV 82 1/6

UNCLASSIFIED DAAK80-81-C-0107 F/G 9/2 NL





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

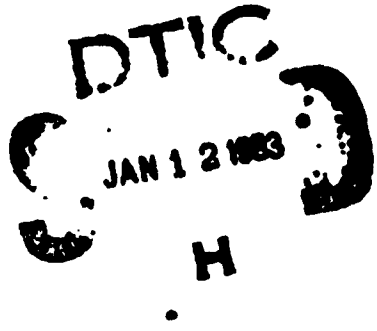
(11

AD A123308

CASE STUDY II

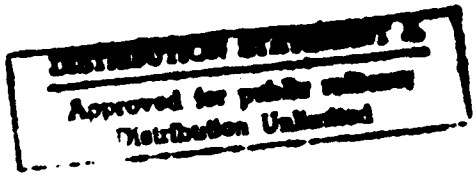
FINAL REPORT DEVELOPED FOR LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE MISSILE MINDER AN/TSQ-73 USING THE ADA PROGRAMMING LANGUAGE

U. S. ARMY CECOM
CONTRACT NO. DAAK80-81-C-0107



CONTROL DATA CORPORATION
GOVERNMENT SYSTEMS
40 AVENUE AT THE COMMON
SHREWSBURY, NJ 07701
201-542-9222

DTIC FILE COPY



83 01 12 029

"The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless designated by other documentation."

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO.	2. AD-A123308	3. Recipient's Accession No. AD-A123 308

4. Title and Subtitle
Large Scale Software System Design of the Missile Minder AN/TSQ-73 using the Ada Programming Language

5. Report Date
9 November 1982

6.

7. Author(s)
Control Data Corporation

8. Performing Organization Rept. No.

9. Performing Organization Name and Address
Control Data Corporation
Government Systems
40 Avenue at the Common
Shrewsbury, NJ 07701

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.
(C) DAAK80-81-C-0107
(G)

12. Sponsoring Organization Name and Address
USA CECOM
Center for Tactical Computer Systems (CENTACS)
ATTN: DRSEL-TCS-ADA-1
Fort Monmouth, NJ 07703

13. Type of Report & Period Covered
Final

14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

This report documents the research and development effort for the design and documentation of the Missile Minder System AN/TSQ-73 using Ada as a program design language. Issues analyzed include adequacy of design methods, system design issues, and the career types required for the system design.

DTIC
JAN 12 1983
H

17. Document Analysis a. Descriptors
Ada Programming Language
Program Design using Ada
Missile Minder

b. Identifiers/Open-Ended Terms
Military Software
System Design Language

c. COSATI Field/Group

18. Availability Statement:
~~Distribution limited to the United States.~~ Available from National Technical Information Service, Springfield, VA 22161.

19. Security Class (This Report)	21. No. of Pages
UNCLASSIFIED	572
20. Security Class (This Page)	**22. Price**
UNCLASSIFIED	

CONTROL DATA

FINAL REPORT

DEVELOPED FOR

LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE MISSILE MINDER AN/TSQ-73 USING THE ADA PROGRAMMING LANGUAGE

Contract Number DAAK80-81-C-0107

GOVERNMENT SYSTEMS
40 AVENUE AT THE COMMON
SHREWSBURY, NEW JERSEY 07701
(201) 542-9222

TABLE OF CONTENTS

DTIC COPY INSPECTED 2

Accession For
NTIS GRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A

*Ada is a Trademark of the Department of Defense (Ada Joint Program Office)

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

PART I
EXECUTIVE SUMMARY

1.0 INTRODUCTION

Control Data Corporation was selected to provide research and development effort for design and documentation of a large scale software system using the Ada programming language. As a result of this effort, Control Data developed a design methodology, and validated the methodology by the redesign and subsequent programming of selected functions of the AN/TSQ-73. The design included seven tasks:

a. Preparation of a work plan.

b. Project personnel training.

c. Development of a design methodology.

d. Validation of the design methodology.

e. Redesign of the Missile Minder System and coding.

f. Documentation of the methodology.

g. Provision of management reports on a routine basis.

The purpose, background, and scope of this report are provided in the following subparagraphs.

1.1 PURPOSE

This report documents the research and development effort for design and documentation of the AN/TSQ-73 using the Ada programming language (MIL-STD-1815).

1.2 BACKGROUND

The Army is currently developing an Ada compiler and associated support tools as part of its efforts to implement the Ada language. As a part of this Ada development effort, the Army is sponsoring separate Ada programs to gain Ada system design experience for use in developing a curriculum and materials to use Ada as a system/program design language and as an implementation language. To acquire software

1.2 BACKGROUND (continued)

design experience using Ada, two contractors were selected to develop software design methodologies based upon the Ada language. The results of the two contractual efforts, which included this contract, will be used as inputs to a training program for potential Ada users.

1.3 SCOPE

This report describes the contractual efforts for the period of twelve months, July 1, 1981 to June 30, 1982. A synopsis of project tasks and technical findings is contained within the report with detailed amplification for the following areas:

a. Training of personnel.

b. Selection of a design methodology.

c. Validation of the design methodology.

d. Application of the design methodology to the redesign of AN/TSQ-73 system.

e. Use of automated tools to redesign the AN/TSQ-73 system.

f. Use of Ada as a system design language (SDL) and a program design language (PDL).

g. Schedule.

h. Documentation.

Products which resulted from this effort are provided in Appendices to this report.

2.0 PROJECT TASKS AND SCHEDULE

The project tasks and schedules defined the contractual effort over a period of twelve months between July 1, 1981 and June 30, 1982. The major tasks included a work plan, training, development of a software design methodology, redesign of the AN/TSQ-73, coding, and the associated documentation.

2.1 WORK PLAN

The work plan documented Control Data's approach and rationale in the performance of the research and development effort. The work plan covered four phases - definition phase, design phase, programming phase and summation phase - and included the following information:

a. Description of the approach and rationale employed.

b. Phase organization.

c. Tasks and subtasks.

d. Milestone charts.

e. Project staffing which included:

   - job descriptions
   - educational and professional synopsis on project personnel
   - rationale for specific career types

f. Work apportionment plan.

The plan was followed and found to be adequate for the twelve month effort.

2.2 TRAINING

The three types of training which were given to all project personnel were orientation, formalized instruction, and informal training. The prime objectives for all types of training were standardization of project familiarization and enhancement of specific disciplines. A brief description of the different types of training given in support of the project follows:

a. Orientation Training - the orientation included an overview of the project and identification of the unique aspects of the project.

b. Formalized Instruction - this training was classroom structured and addressed three specific areas: Design Methodology, Ada programming language, and Missile Minder System introduction.

2.2 TRAINING (continued)

c. Informal Training - basically, this type of training is reinforcement instruction. It enabled project members to expand their newly acquired skills through practical application by designing, developing, and coding of small segments of the Missile Minder System.

2.3 DESIGN METHODOLOGY

The approach that was taken to construct a system design methodology was based on lessons learned, analytical studies and successful practices in the rapidly emerging field of software engineering. The following key principles which are somewhat interdependent, form the foundation for the evolved software system design methodology:

a. Life Cycle View - systems evolve through various stages from requirements to maintenance. Precision and accountability should be emphasized in the early stages.

b. Levels of Abstraction and Isolation of Detail - systems should be clearly representable at different conceptual levels. Users desiring a high level view of the system should not be distracted by lower level details.

c. Finite Human Capability - all techniques (steps) must aim to keep the design within the constraints of human understanding.

d. Graphical Communications - a great deal of information can be easily and accurately communicated through the use of graphics.

e. Automated Techniques - should enhance the limited human design capability and increase design productivity.

f. Human Communications - the role of documentation is to foster human-to-human communications and prevent computer errors.

g. User Friendliness - the user should feel comfortable in learning and using the techniques and automated tools.

2.3 DESIGN METHODOLOGY (continued)

h. Simplicity - technique. should be easy to learn and their proper utilization effectively communicated.

The above principles shaped the design methodology which was shown as three phases and eight techniques (Table 1-1).

2.4 DESIGN METHODOLOGY VALIDATION

The objectives of the validation effort were to validate and recommend improvements to the proposed design methodology. The validation task was based upon the principles described in Paragraph 2.3 above, and upon the following guidelines:

a. Support a design process which includes concurrent hardware design and software design.

b. Utilize the concept of levels of abstraction.

c. Support clear and concise descriptions and communications of the design in graphic tools wherever possible.

d. Support automated tools to free human designers from tedious and painstaking representation of their design constructs.

The methodology approach for the validation task was to take a vertical slice of the Remote Communications function of the AN/TSQ-73 and apply the design methodology, following the eight steps identified in Table 1-1. The AN/TSQ-73 redesign phase also supports the conclusions of the validation team which are summarized below.

a. The proposed design methodology supports the self-imposed goals and objectives.

b. The proposed design methodology supports, but does not force, a top-down approach.

c. The coding phase should be expanded for further evaluation.

d. The use of automated tools was invaluable and should be enhanced.

TECHNIQUES	PHASES
System Entity Diagram (SED) System Design Language (SDL)	System Design Phase
Data Flow Diagram (DFD) Data Dictionary (DD) Structure Charts (SC)	Software Design Phase
Program Design Language (PDL) Complexity Measure (CM) Structured Programming (SP)	Programming Phase

a. The system entity diagram allows for functional decomposition of the system at the top-level.

b. The system design language generates Ada-based textual representation of the system functions.

c. The data flow diagrams are graphic representations of the software system.

d. The data dictionary serves as a central repository for complete, unambiguous data definitions and process descriptions.

e. The structure charts are used to graphically establish program units, identify hierarchy, and exchange of data.

f. The program design language captures the detailed structure by detailed design based upon the Ada programming language.

g. The complexity measure is derived from a program's control flow and applied to the program design to determine the complexity of the design.

h. Structured programming is employed when writing Ada code for comprehension and maintenance of the code.

Table 1-1 - System Design Methodology

2.5 MISSILE MINDER REDESIGN TASK

The AN/TSQ-73 was redesigned and coded in accordance with the techniques of the methodology described above. The techniques were applied to the AN/TSQ-73 based on the following guidelines:

a. The AN/TSQ-73 system specifications MIS-19501C and MIS-10297J were used as prime requirement documents for the system. The system specifications were supplemented by product specifications for further delineation of requirements when necessary.

b. Design deviated from the existing specification design where improvements could be made or restructured for the Ada architecture.

c. Design is not constrained by the existing AN/TSQ-73 hardware configuration.

d. The design phase included only a sufficient level of design to validate the methodology.

e. The subfunction coded was selected by the Government for the detailed program design and coding using the Ada language.

f. No attempt was made to determine the adequacy of Ada in satisfying AN/TSQ-73 processing requirements. This limitation resulted from the use of interpreter/translator for program coding and validation.

g. The subfunction selected was related to Target Control for design to the lowest level, programmed in executable code using Ada, and verified with a limited scenario for that function.

h. The design was limited to the battalion level configuration.

The design began by reorganizing the functionality present in the currently deployed system. The reorganization revealed four major independent functional areas which are numbered in accordance with methodology requirements.

2.5 MISSILE MINDER REDESIGN TASK (continued)

- Command and Control (1.0)
- Target Control (2.0)
- Remote Communication (3.0)
- Radar Communication (4.0)

The major functional areas were subsequently decomposed to allow subsequent levels of detail to be organized. This functional decomposition was captured in graphic format by the system entity diagrams and in textual format by the system design language.

The system design language was based upon these guidelines:

a. Utilized Ada packages to represent collections of logically related system activities. The items identified in any given package should be logically independent from those represented in other packages.

b. Utilized Ada procedures and functions to represent system activities which appear to be sufficiently distinct to act as stand-alone units, but at the same time, dependent upon other stand-alone units to accomplish a more global activity.

c. Utilized Ada tasks to represent activities which can provide services to other program units in a concurrent fashion.

d. Utilized names and numbering structure provided in the SED to provide accountability and traceability.

The software design phase followed the system design phase and consisted of the development of data flow diagrams, data dictionary, and structure charts. The data flow diagrams (bubble diagrams) defined information flow at a level which included input/output, processes, and storage. The data dictionary defined the information in the data flow diagrams in clear, unambiguous terms. Using the data flow diagram and data dictionary, the structure chart superimposes the

2.5 MISSILE MINDER REDESIGN TASK (continued)

information from these previous steps into hierarchical program units that can be designed in the programming phase. The last phase, the programming phase, an Ada-based program design language, complexity measurement, and source code was generated. It was clear that more research is needed to derive the optimum format for the Ada PDL to differentiate it from the Ada code. There was difficulty experienced by project members in not writing the entire code at the PDL level. It was also clear that more research is needed for an enhanced complexity measure to take advantage of Ada's unique features (strong typing, tasking).

The application and use of the methodology was supported in the validation effort and the subfunction effort. Perhaps the greatest advantage is in the area of system design based upon Ada as a system design language. The design was relatively easy to follow since it provided a logical sequence from broad requirements to detailed design. The design process also demonstrated the need for automated tools, development of a hardware design methodology, early hardware/software trade-off decisions, and the need for simulation/modelling at the system level due to today's vast technology base.

2.6 DOCUMENTATION

The documentation produced within the contract follows:

a. Work plan which documents Control Data's approach and rationale for the performance of the contract.

b. Ada System Designer's Guide (Appendix B) which defined the procedures to be used in the development of the software design methodology.

c. Design plan which described the research and development methods to provide a top level design and documentation of a large scale software system.

2.6 DOCUMENTATION (continued)

d. Validation report which was an informal report documenting the validation of the methodology.

e. Monthly reports which provided the status of the program.

f. Final report which documented the results of the research and development for the design and documentation of the AN/TSQ-73 system.

g. Program listings.

3.0 TECHNICAL FINDINGS

The significant technical findings are provided for four major areas as follows:

a. Design methodology utility.

b. Ada language utility.

c. Project team factors.

d. Educational considerations.

3.1 DESIGN METHODOLOGY UTILITY

The system design methodology developed during this period is applicable to the system and software design phases for embedded computers. Further, a high degree of confidence has been developed that the constructs of the Ada language can be used for the entire design process, including system design. Ada can serve as both a system design language and a program design language. The use of the design methodology provided significant conclusions which included:

a. The system entity diagrams and the system design language can be used in parallel.

b. Automation of all techniques allows faster production, more efficient change, cleaner documentation and a framework for reliability checking and product visibility.

c. The methodology provided for traceability from requirements to code.

3.1 DESIGN METHODOLOGY UTILITY (conclusions)

d. The structured design techniques were compatible with the detailed design of the software.

e. The Ada-based program design language provided the design with significant power which Ada provided in the areas of abstraction.

3.2 ADA LANGUAGE UTILITY

Ada language constructs were used as a system design language and program design language. The system design language is a part of the overall system design process and acts as the initial techniques before functions are allocated to hardware or software. The program design language is used during the software design phase. The utility of each is discussed in the following subparagraphs.

3.2.1 UTILITY OF ADA AS A SYSTEM DESIGN LANGUAGE

The SDL, resulting from a number of evolutionary changes, provided the capability of defining system functional requirements. In addition, the SDL:

a. Provided for human understanding (users, developers, managers) of system requirements.

b. Provided system information for hardware/software trade-off evaluations and succeeding design phases.

c. Provided system information for management decisions.

d. Provided for system accountability.

e. Provided for system consistency.

f. Provided for ripple-effect tracing.

g. Provided for increased productivity through automation.

h. Provided a primary working system document.

3.2.2 UTILITY OF ADA AS A PROGRAM DESIGN LANGUAGE

The design of program units for the AN/TSQ-73 Missile Minder System used an Ada-like program design language to facilitate the final Ada code. As such, the program design language capitalized on the design work encapsulated in the SDL description, data flow diagrams, and the structure charts. The programming design completes the first step of the programming phase and constitutes the requirements for each program unit. The final step of the programming phase was the coding.

During the AN/TSQ-73 redesign effort, especially in the functional areas of target control, the Ada concepts proved beneficial. Special emphasis was placed on those constructs which promote readability and facilitate program reliability and maintenance. The code produced appears to be easily readable and understandable as well as maintainable.

3.3 PROJECT TEAM FACTORS

The following factors were considered during the process of assembling the project team:

a. Level of education.

b. Experience.

c. Past performance.

The individuals chosen were required to have the capability to participate in a team environment where the collective expertise possessed by the team could be used to overcome the shortcomings of any one individual.

In the selection of career types, the following prerequisites were used:

a. Knowledge of two or more high order programming languages.

b. Coding experience.

c. Familiarity with the basic concepts of large scale software systems.

3.3 PROJECT TEAM FACTORS (continued)

d. Some system design experience.

e. Demonstrated communications ability.

The selection of this depth and level of experience provided the necessary levels of career types to fulfill project requirements.

3.4 INSTRUCTIONAL ISSUES

Three elements of the project were identified to require formalized instruction. They were design methodology (Structured Analysis/Structured Design), Ada Programming Language, and Missile Minder System (AN/TSQ-73) Indoctrination. From this base, a level of understanding developed from which all project members increased their degree of expertise.

All formalized instruction followed typical lesson plans and course outlines. The length of instruction for each course was forty (40) hours with the exception of the Missile Minder Indoctrination which was twenty-four (24) hours. The forty hour courses were restricted to twenty-four hours of stand-up instruction with the remaining sixteen hours allocated to workshop sessions and small group discussions.

Reinforcement documentation was available for all three subjects to facilitate problem solution and research efforts. Formalized instruction provided a standardized base of expertise, although it was by no means all inclusive. Extensive research and supplementary reading were required by each individual to obtain more information and knowledge on specific subjects. This approach allowed each team member to supplement his/her knowledge for particular needs.

4.0 CONCLUSIONS

The following conclusions resulted from the redesign of the AN/TSQ-73:

a. The Ada language, serving in the multiple roles of a system design language, implementation language, program design language, can support a cohesive system design methodology.

b. The Ada language will require substantially more training than previous programming languages.

c. The system design methodology techniques can be automated, therefore, a need exists for further development of automated design tools.

d. Timing constraints of the Ada language require further evaluation.

5.0 RECOMMENDATIONS

The following recommendations are submitted:

a. An automated design system for the total design process be developed based upon Ada as a design language.

b. The entire tracking sequence of the AN/TSQ-73 be programmed in Ada and results evaluated against a set of representative target reports to determine the adequacy of Ada in a time dependent system.

c. A user's guide be developed and published as a part or as an adjunct to MIL-STD-1815.

d. The system design process, based upon Ada, be expanded to include the hardware design including simulation/ modelling and CAD.

e. The configuration management process be re-evaluated to permit development of the specifications as a single baseline based upon the Ada language.

f. A set of software management techniques be developed to act as a production control mechanism.

PART II
TECHNICAL REPORT

1.0 INTRODUCTION

1.1 PURPOSE

This report documents the research and development effort for the design and documentation of a large scale system using the Ada programming language (MIL-STD-1815). The system is the Missile Minder System AN/TSQ-73 as authorized by Contract DAAK80-81-C-0107. The report contains activities and findings in the performance of the project. These activities and findings include adequacy of design methods used in the design of the AN/TSQ-73 system, system design issues, Ada language issues, and the career types required for the system design.

1.2 BACKGROUND

The Ada language has been chosen as the DOD common programming language. The Army is currently developing an Ada compiler and associated support tools as part of its effort to implement the Ada language. The compiler effort is being directed by CENTACS, USA CECOM, and will be hosted on the VAX 11/780 system with code generators for the VAX, PDP 11/70 and selected target environments. Follow-on efforts will rehost the compiler as well as provide additional generators. In addition, CENTACS is sponsoring the implementation of an Ada translator that is intended solely as a training aid and not for production use. The translator is hosted on and is targetted to the VAX.

The goals of the Ada program design project are to gain Ada system design experience for use in developing a curriculum and materials for training in using Ada as a system/program design language and as an implementation language. To acquire design issues utilizing Ada, two contractors were selected to develop design methodology based upon the Ada language. The results of this effort will be used as inputs to a training program for potential Ada users. The Government also initiated a contractual design monitoring

1.2 BACKGROUND (continued)

effort separate from this contractual effort, but parallel to it. The intent of this adjunct effort is to extract and obtain information regarding design methods and approaches used, types of documentation found to be most useful, career types required and backgrounds of the personnel involved in the various aspects of the design, adaptability of certain Ada features to particular functions of the system being designed, the training period and process required to allow the design personnel to effectively use Ada, and the identification of the Ada viewpoint needed to execute the various job tasks. This information will assist in identifying and outlining the educational materials and courses that will be required to teach future designers and programmers how to use the Ada language effectively for large embedded computer systems.

The initiative for the Ada education and training plans comes from the system design area and the post-deployment software support area. Greater control over the progression from system requirements to the allocation of specific functions to a software module is required as computer systems become larger and more complex. Use of Ada during the development phase should improve management capability, but expertise at the system design level, where system architecture is developed in response to system requirements, must be generated. Therefore, the Ada program design project approach is to redesign the software architecture capturing functional requirements and system interface requirements using Ada as a system design language and as an implementation language, finally coding one subfunction of the redesigned system.

1.3 SCOPE

The final report summarizes progression in the accomplishment of the twelve-month contractual effort for the redesign and documentation of a large scale software system (AN/TSQ-73) using the Ada programming language.

1.3 SCOPE (continued)

The report describes the contractual efforts for the period of twelve months, July 1, 1981 to June 30, 1982. A synopsis of project tasks and technical findings is provided within the report with detailed amplification for the following areas:

a. Training of personnel.

b. Selection of a design methodology.

c. Validation of the design methodology.

d. Application of the design methodology to the redesign of AN/TSQ-73 system.

e. Use of automated tools to redesign the AN/TSQ-73 system.

f. Use of Ada as a system design language (SDL) and a program design language (PDL).

g. Schedule.

h. Documentation.

Products which resulted from this effort are provided in Appendices to this report.

2.0 PROJECT TASKS AND SCHEDULE

2.1 WORK PLAN

The work plan documented Control Data's approach and rationale for the performance of the research and development effort utilizing the Ada high order language for the design and documentation of a large scale software system. The work plan was divided into six sections which described the following information:

a. Description of the approach and rationale employed.

b. Phase organization.

c. Tasks and subtasks.

d. Milestone charts.

e. Project staffing which included:

- Job descriptions.
- Educational and professional synopsis on project personnel.
- Rationale for specific career types.

f. Work apportionment plan.

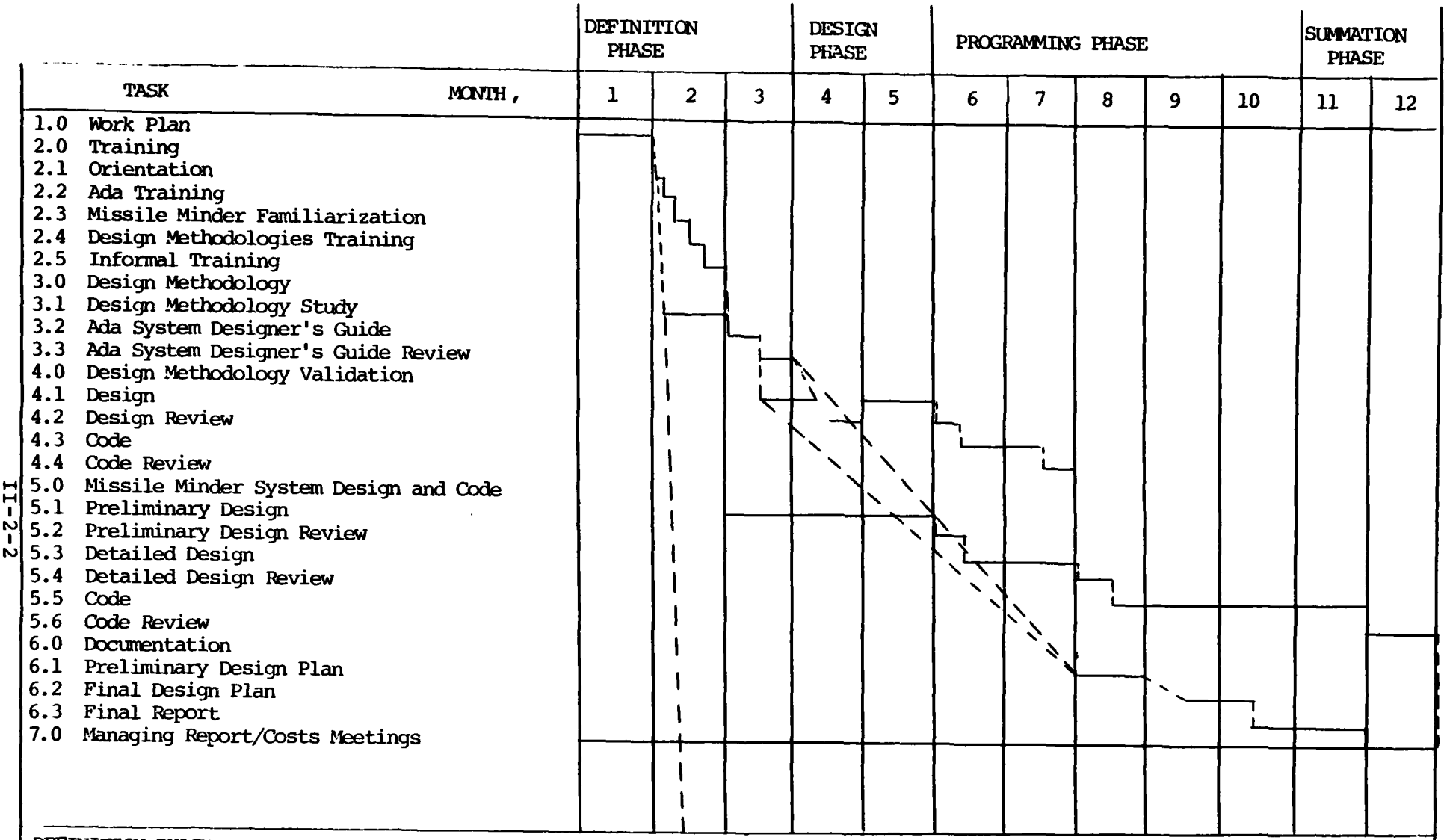
Three phases and a summation phase as shown in Figure 2-1 were followed in the accomplishment of contractual requirements. Within these phases are six major tasks. These tasks are briefly discussed in the following subparagraphs.

a. Work Plan (Task 1.0)

Based upon the input as contained in Control Data's proposal, the work plan identified the individual tasks, schedule of accomplishment, and deliverable contractual products.

b. Training (Task 2.0)

This task familiarized all project personnel with project requirements and intended objectives. Formalized instruction was provided in system design methodology, Ada programming language, and the Missile Minder System.



DEFINITION PHASE

1) Work Plan
2) Training
3) Design Methodology

DESIGN PHASE

1) System Design
2) Ada System Designer's Guide

PROGRAMMING PHASE

1) Design Plan
2) Modular Design and Analysis
3) Coding
4) Program Documentation

SUMMATION PHASE

1) Preliminary Final Report
2) Final Report

Figure 2-1 - Project Phases

2.1 WORK PLAN (continued)

c. Design Methodology (Task 3.0)

This task established a system design methodology that was used in conjunction with the Ada high order language.

d. Design Methodology Validation (Task 4.0)

The task of validation of the methodology was accomplished by designing and coding a small portion of the Missile Minder System. Design and code reviews were conducted to detect and subsequently correct discrepancies in the design methodology.

e. Missile Minder System (Task 5.0)

This task included the high level definition of criteria for logical selection and allocation of system functions and detailed design and coding. The task included a preliminary design, detailed design, and selected portions for coding.

f. Documentation (Task 6.0)

The documentation task involved the writing, correlation, and production of deliverable products. The documentation products were the work plan, draft and final design plan, final report and source code.

2.2 TRAINING

The three types of training which were given to all project personnel were orientation, formalized instruction, and informal training. The prime objectives for all types of training were standardization of project familiarization and enhancement of specific disciplines. A brief description of the different types of training given in support of the project follows:

a. Orientation Training: The orientation included an overview of the project and identification of the unique aspects of the project.

2.2 TRAINING (continued)

b. Formalized Instruction: This training was classroom oriented and addressed three specific areas: design methodology, Ada programming language, and Missile Minder System indoctrination. Amplifying information on each of these subjects is contained in Section 3.4.

c. Informal Training: Basically, this type of training provided reinforcement and enhancement for the formal instruction. It enabled project members to expand their newly acquired skills by practical application by designing, developing, and coding of small segments of the Missile Minder System. The informal training was supported by extensive research and supplementary reading by team members.

2.3 DESIGN METHODOLOGY DEVELOPMENT

The design methodology definition task established a system design methodology to be utilized in conjunction with the Ada programming language. An additional requirement was for the system design methodology to be compatible with large scale software system construction. The following subsections discuss the rationale for selecting the methodology, the eight techniques which compose the methodology, and alternate methodologies which were examined.

2.4 RATIONALE FOR SELECTING METHODOLOGY

The approach that was taken to construct a system design methodology was based on analytical studies and successful practices in the rapidly emerging field of software engineering. The following key principles which are somewhat interdependent, form the foundation for the evolved software system design methodology:

a. Life Cycle View - Systems evolve through various stages from requirements to maintenance. Precision and accountability should be emphasized in the early stages.

b. Levels of Abstraction and Isolation of Detail - Systems should be clearly representable at different conceptual levels. Users desiring a high level view of the system should not be distracted by lower level details.

2.3.1 RATIONALE FOR SELECTING METHODOLOGY (continued)

c. Finite Human Capability - All techniques (steps) must aim to keep the design within the constraints of human understanding.

d. Graphical Communications - A great deal of information can be easily and accurately communicated through the use of graphics.

e. Automated Techniques - Should enhance the limited human design capability and increase design productivity.

f. Human Communications - The role of documentation is to foster human-to-human communications and prevent computer errors.

g. User Friendliness - The user should feel comfortable in learning and using the techniques and automated tools.

h. Simplicity - Techniques should be easy to learn and their proper utilization effectively communicated.

Thus, these key principles shaped the system design methodology and are reflected in the techniques utilized. Figure 2-2 illustrates the adherence of the proposed techniques of the software system design methodology with the stated guiding principles.

2.3.2 SYSTEM DESIGN METHODOLOGY TECHNIQUES

The key principles discussed in Paragraph 2.3.1, form the foundation upon which the methodology rests. This section presents the eight individual techniques which are the methodology's key components. Refer to Appendix B for a comprehensive explanation and application of the methodology. The techniques serve as creative aids for system development in three phases: (1) System Design Phase, (2) Software Design Phase, and (3) Programming Phase.

The techniques perform purposeful activity with their respective output representing boundable products. As such, the system design phase is composed of two boundable

BASIC PRINCIPLES \ TECHNIQUES	SYSTEM ENTITY DIAGRAM	SYSTEM DESIGN LANGUAGE	DATA FLOW DIAGRAM	DATA DICTIONARY	STRUCTURE CHARTS	PROGRAM DESIGN LANGUAGE	COMPLEXITY MEASURE	STRUCTURED PROGRAMMING	SOURCE CODE	
LIFE CYCLE VIEW	x	x	x	x	x	x	x	x	x	
LEVELS OF ABSTRACTION	x	x	x	x	x	x	x	x	x	
ISOLATION OF DETAIL	x	x	x	x	x	x	x	x	x	
FINITE HUMAN CAPABILITY	x	x	x	x	x	x	x	x		
GRAPHICAL COMMUNICATIONS	x		x		x		x			
AUTOMATED TECHNIQUES	x		x	x	x					
HUMAN COMMUNICATIONS	x	x	x	x	x	x	x	x	x	
USER FRIENDLINESS	x	x	x	x	x	x	x	x	x	
SIMPLICITY	x	x	x	x	x	x	x	x	x	
SOFTWARE'S MATHEMATICAL NATURE						x	x	x	x	

Figure 2-2 - Basic Principles/Design Technique Matrix

2.3.2 SYSTEM DESIGN METHODOLOGY TECHNIQUES (continued)

products: (1) System Entity Diagrams, and (2) System Design Language production. The software design phase is composed of three boundable products: (1) Data Flow Diagrams, (2) Data Dictionary, and (3) Structure Charts; and the Programming Phase also has three boundable products: (1) Program Design Language, (2) Complexity Measure, and (2) Structured Programming. Figure 2-3 illustrates the three phases of system development, major activities of these phases, and resulting boundable products.

The initial technique utilized is the System Entity Diagram. Graphical in nature, the system entity diagram allows for the functional decomposition of the system at the highest levels. The approach results in the system's major functions (entities) being partitioned into respective levels of detail. One of the immediate advantages of this technique is that it affords the user with an early view of the system composition, and establishes an audit scheme for accountability, visibility, and project scheduling and control.

Supporting the system entity diagram is the system design language, the second technique. The purpose of the system design language is to generate an Ada-based textual representation of the system functions. The overall structure of the system will be reflected at an early stage through utilization of Ada constructs. The joint information captured by the system entity diagram and the system design language concludes the system design phase and is input for hardware/software trade-offs.

Since the scope of the contractual effort emphasized software system redesign, the methodology continued only on the software side. The software design phase utilizes the data flow diagram, the data dictionary, and structure charts. The first technique employed is the data flow diagrams, which are graphic representations of the software system.

PHASE	ACTIVITY	PRODUCT
1. SYSTEM DESIGN	DESIGN OF SYSTEM ARCHITECTURE	A) SYSTEM ENTITY DIAGRAM B) SYSTEM DESIGN LANGUAGE
2. SOFTWARE DESIGN	DESIGN OF SOFTWARE PROGRAM	C) DATA FLOW DIAGRAM D) DATA DICTIONARY E) STRUCTURE CHARTS
3. PROGRAMMING	PROGRAM UNIT DETAIL DESIGN, CODE AND TEST.	F) PROGRAM DESIGN LANGUAGE G) COMPLEXITY MEASURE H) STRUCTURED PROGRAMMING

Figure 2-3 - System Development Phases, Activities and Products

2.3.2 SYSTEM DESIGN METHODOLOGY TECHNIQUES (continued)

They model the system in terms of input data, output data, stored data, and transformations of data. In support of the data flow diagram is the data dictionary and structure chart. The data dictionary serves as a central repository for complete, unambiguous data definitions and process descriptions. The structure charts are used to graphically establish program units, identify hierarchy, and data exchanged.

The programming phase begins with the Ada-based program design language. The purpose of the program design language is to capture detailed structure by detail design yet allow for a flexible range of coding expressiveness. A complexity measure derived from a program's control flow can be applied to the program design language to estimate logical weight. Those program units above a set quantitative value can be re-examined for simpler alternatives. Finally, structured programming is employed when writing the actual Ada code for comprehension and maintenance purposes.

2.3.3 ALTERNATIVE METHODOLOGIES

The system design methodology evolved for the project was briefly described in the previous section. It has been significantly influenced by the established camp of structured analysis and structured design formalized by Myers, Constantine and Yourdon. The common theme includes the concepts of levels of abstraction, graphical representations, facility for automating the chosen techniques and other principles described previously. In addition, and in recognition of the unique requirements of embedded tactical computer systems and the advanced features of the Ada programming language, other compatible techniques have been added.

2.3.3 ALTERNATIVE METHODOLOGIES (continued)

During the design methodology study segment of this contract, a number of other design methods were reviewed. A detailed description of these methodologies can be found in the design plan, a deliverable under this contract. The reviewed methodologies include:

a. Jackson Methodology

b. Warnier/Orr

c. SADT

d. PSL/PSA

e. HOS

f. SREM

2.4 VALIDATION

The validation of the design methodology was begun in mid September, 1981, and ended at the end of February, 1982. The effort took place over a period of 5.5 months for a total of 17.5 man-months. The following subparagraphs detail the results of the validation effort.

2.4.1 OBJECTIVES

The objectives of the validation effort were to validate and recommend improvements to the proposed design methodology that was created by Control Data's Shrewsbury Facility in the redesign of the Missile Minder AN/TSQ-73 System. The key principles of the evolving methodology were utilized by the validation team and are reiterated below:

a. Life Cycle View - systems evolve through various stages from requirements to maintenance.

b. Levels of Abstraction and Isolation of Detail - the ability to clearly describe systems at different conceptual levels. Users desiring a high level view of the system should not be distracted by lower level details.

2.4.1 OBJECTIVES (continued)

c. Finite Human Capability - all techniques (steps) must aim to keep the design within the constraints of human understanding.

d. Graphical Communications - a great deal of information can be easily and accurately communicated through the use of graphics.

e. Automated Techniques - should enhance the limited human design capability and increase design productivity.

f. Human Communications - the role of documentation is to foster human-to-human communications.

g. User Friendliness - the user should feel comfortable in learning and using the techniques and automated tools.

h. Simplicity - so that the techniques can be properly utilized and effectively communicated.

Based on the above key principles, the design methodology must:

a. Support a design process which includes concurrent hardware design and software design.

b. Utilize the concept of levels of abstraction.

c. Support clear and concise descriptions and communications of the design in graphic tools wherever possible.

d. Support automated tools to free human designers from tedious and painstaking representation of their design constructs.

In addition, the following questions were asked for answers based upon the team's validation experience:

a. Do the techniques represent the proper level of abstraction? Is there too much or too little of a conceptual leap between the sequential application of the techniques?

2.4.1 OBJECTIVES (continued)

b. Are the techniques complete enough to fully represent their level of abstraction?

c. Are the automated tools beneficial and/or necessary for the methodology? Are there any recommendations for the automated tools?

d. Is the System Designer's Guide descriptive enough to properly utilize the techniques? Are there any recommendations to improve the System Designer's Guide?

e. What is Ada's strength as a System Design Language (SDL)?

f. What is Ada's weakness as a System Design Language (SDL)? Are there any additional thoughts on Ada as an SDL?

g. Are there any difficulties in proceeding from the System Entity Diagram (SED) to the System Design Language (SDL) to the Data Flow Diagram (DFD)?

h. How to you know when to apply Ada's concept of TASK, PACKAGE, PROCEDURE, FUNCTION and EXCEPTION? Other Ada concepts utilized?

2.4.2 PROPOSED ADA SYSTEM DESIGN METHODOLOGY FOR VALIDATION

The proposed Ada design methodology included eight design techniques as defined in the System Designer's Guide. These techniques, outlined below, are defined to be the beginning of the design effort.

a. System Entity Diagram (SED) - is used to graphically represent the first three abstract levels of the system being designed. The three levels of abstraction are facility, subfacility, and function.

b. System Design Language (SDL) - is used to textually represent the system being designed. The graphic

third level SEDs are translated into free-form English statements formed from a subset of the Ada language. The SDLs may be decomposed beyond the third level where no SEDs exist.

NOTE: The SED and SDL techniques are performed prior to hardware/software implementation and as such, represent a functional description of the system. The remaining methodology steps decompose only the software portion of the system.

c. Data Flow Diagram (DFD) - is a graphic representation of the system in terms of:

- Input data
- Output data
- Stored data
- Significant intermediate data forms
- Data transformations

d. Data Dictionary (DD) - is a textual representation of any one of the following:

- Data Definitions
- Module descriptions with action definitions (pseudo code)
- Process specifications with actic lefinitions (pseudo code)
- Undefined names

2.4.2 PROPOSED ADA SYSTEM DESIGN METHODOLOGY (continued)

e. Structure Chart (SC) - is a graphic representation that:

- Partitions the system described by the Data Flow Diagrams and Data Dictionary into a hierarchy of modules, each of which is viewed as performing a well-defined and programmable function in the system.
- Establishes the means and the direction of communication between modules.

f. Program Design Language (PDL) - is a textual representation of the structure charts into program units utilizing Ada constructs as program pseudo code.

g. Complexity Measure (CM) - is a graphic representation of the complexity of the PDL.

h. Structured Programming (SP) - is the Ada coding of the PDL into compilable code.

The design of the system follows a top-down approach as shown in Figure 2-4. The system design, which is the top-level of the structure, consists of system entity diagrams (SED) and the system design language. The lower level, the software design, follows the remaining hierarchical levels to the lowest level which is the executable code.

Although not stated in this version of the System Designer's Guide, the methodology provided for accountability, consistency, and ripple-effect tracing with a numbering scheme. At the initial methodology step, the SEDs form a tree structure with its first level of abstraction numbered 1.0, 2.0, 3.0, 4.0, etc. The second level of abstraction begins with numbers 1.1 through 1.n until the level is completed to 4.1 through 4.n. The third level begins 1.1.1, etc., until 4.n.n. Thus, utilizing this numbering scheme, the line of parental heritage is defined as well as the level of abstraction. Example, abstract 1.1.1 (level three) is the offspring of abstract 1.1 (level two) which is the offspring of abstract 1.0 (level one).

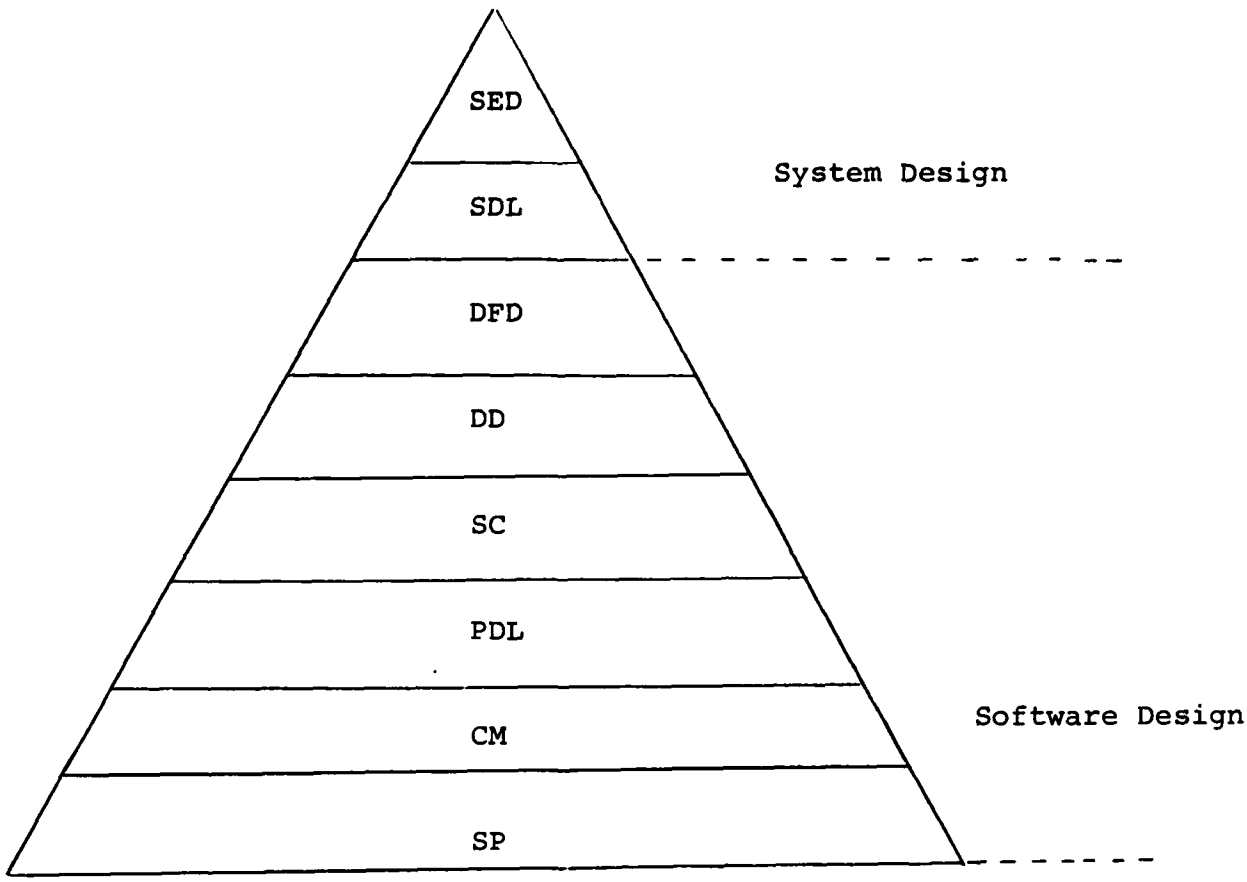


Figure 2-4 - Proposed Design Methodology Structure for Validation

2.4.3 DESIGN METHODOLOGY PROCESS

A vertical slice of the Missile Minder AN/TSQ-73 system Remote Communications was selected for validation. Since the vertical slice had not been redesigned, the validation team served as both designers and validators. As the design progressed, the eight methodology steps shown in Figure 2-4 were identified as three distinct phases. Those three distinct phases, Figure 2-5 , were followed in the redesign of the AN/TSQ-73, paragraph 2.5 of this report. The coding phase, hence, has been renamed the programming phase. The AN/TSQ-73 redesign phase also supports the conclusions reached by the validation team in most areas.

2.4.3.1 SYSTEM ENTITY DIAGRAM

The word "system" is defined in Webster's Collegiate Dictionary as, "a regularly interfacing or interdependent group of items forming a unified whole". The term was applied to the AN/TSQ-73 system to mean, "The AN/TSQ-73 system is a combination of hardware, software, and operator interaction required to meet the objectives and mission of the AN/TSQ-73".

The validation team applied these definitions to the SEDs and the SDLs of the AN/TSQ-73 Remote Communications subsystem. Structured walkthroughs were used throughout the validation effort. The System Designer's Guide stated that: (1) only the three highest system abstract levels are required for the SED, and (2) the SEDs are totally completed before beginning the next methodology step, i.e., SDL. Control Data's Structured Analysis/Structured Design (SASD) graphics packages were used to produce Remote Communications SEDs. The validatation team findings were:

a. A design methodology requiring exactly three abstract SED levels is too restrictive and provided insufficient AN/TSQ-73 system information.

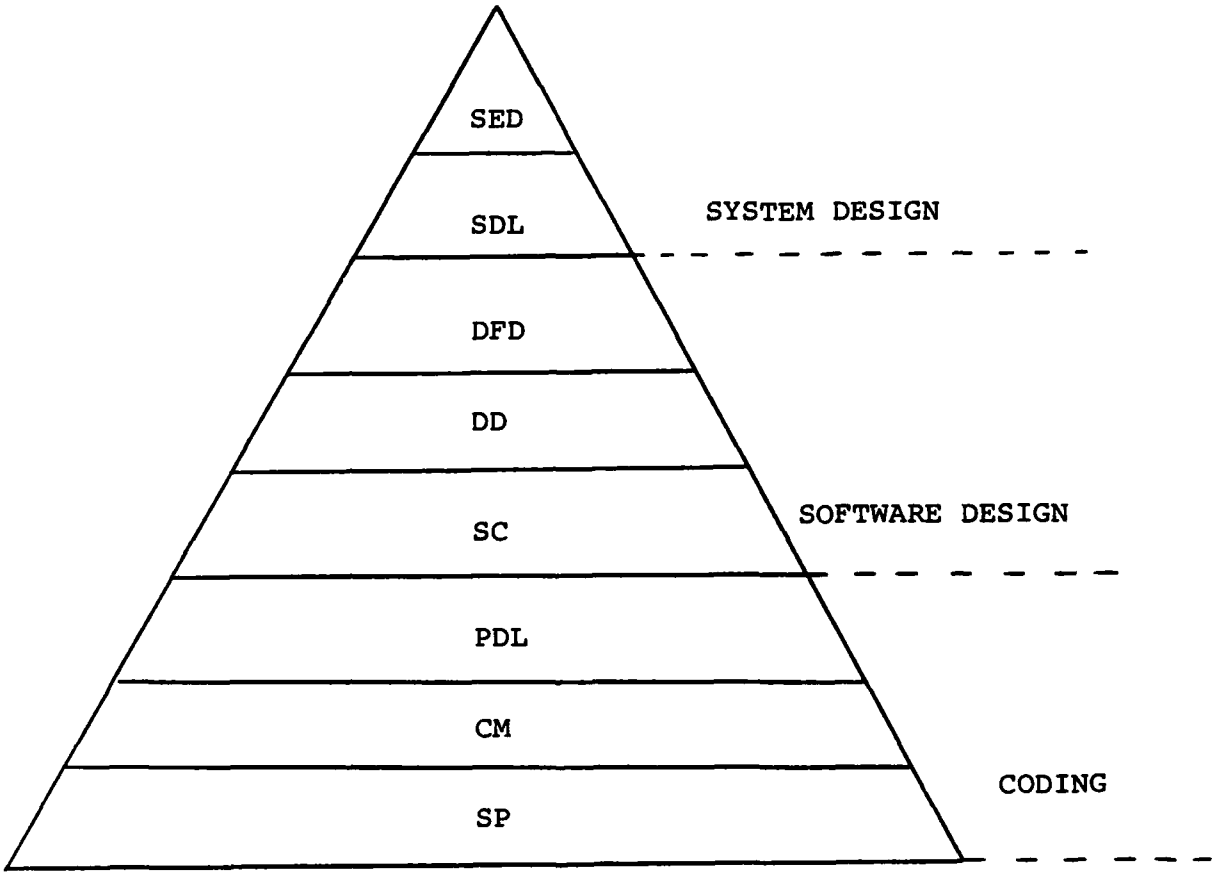


Figure 2-5 - Top-Down Design Methodology Structure
(Revised)

2.4.3.1 SYSTEM ENTITY DIAGRAM (continued)

b. In the case of Remote Communications, a fourth, fifth and sometimes sixth SED level were produced to provide a clear AN/TSQ-73 system picture.

c. The number of abstract SED levels required to express a given system is dependent upon the magnitude and complexity of the system being designed. Therefore, n number of SED levels should be permitted. To accomplish n number of abstract levels, a different graphic scheme would be required. A possible solution is shown in Figure 2-6.

As seen from that example, the numeric level of abstraction would form the inner line of the graphic. Thus, a two-lined graph could be used to represent ten levels of abstraction and a three-lined graph 99 99 system 99 99 function could be used to represent one hundred levels of abstraction.

d. It was far more productive to complete only one or two SED levels before constructing the corresponding SDL and back to the SED decomposition again than to complete the entire SED partitioning before beginning the SDL (explained more fully later in the report).

e. Control Data's SASD graphic tools were not designed for the SED, but rather the structure chart tools was adapted to meet SED requirements. Such as:

- The ability to graphically illustrate n number of SED levels did not exist. Although for this project, a maximum of six abstraction levels was determined to be sufficient.
- The number of characters per graphic description is limited which resulted in the use of many symbol identifiers being abbreviated or partially omitted.
- The amount of SED information per screen or printed page is limited. This limitation may be hardware oriented since a high resolution graphic terminal was not utilized in producing these graphics.

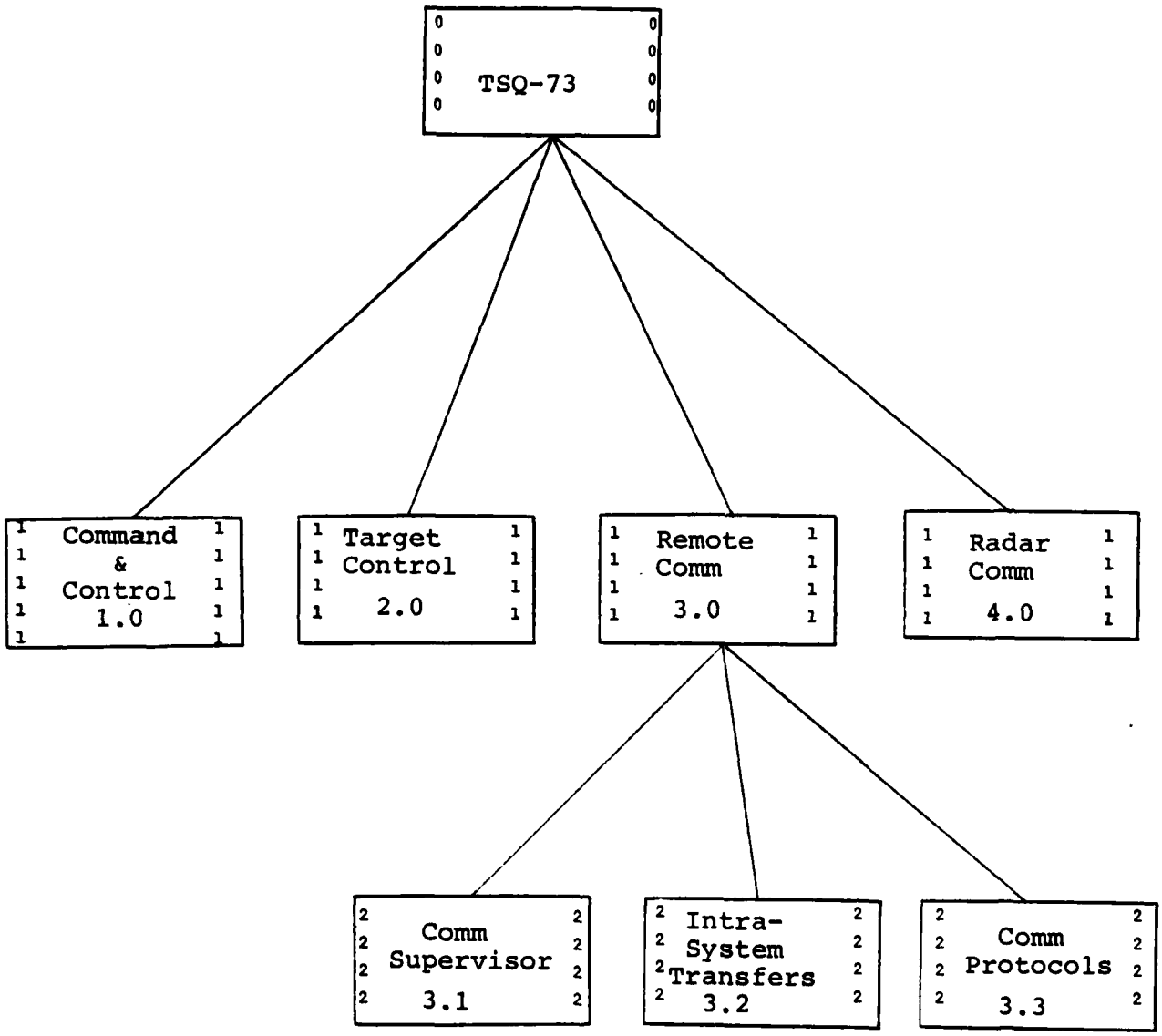


Figure 2-6 - SED With "N" Number of Abstract Levels (Example)

2.4.3.1 SYSTEM ENTITY DIAGRAM (continued)

f. The System Designer's Guide could be improved by the use of a clearly defined SED illustration.

g. The system's SEDs do not stand by themselves in providing a system description, but require clarification through an individual or the next technique (SDL) in the design methodology.

h. The SEDs do not provide a functional value in graphically illustrating a system's functions at various abstract levels. Reference Appendix C for Remote Communications SEDs.

2.4.3.2 SYSTEM DESIGN LANGUAGE

Unlike the SED technique, the initial System Designer's Guide limited SDL definition resulted in several false starts by the validation team. This eventually led to the validation team redefining the SDL technique.

The system definition was applied by the validation team to the initial SDL technique. The conclusions were that the SDL contained several weak points: accountability and consistency will be weakened and may eventually be lost if the SDL does not start at the highest SED level; the utilization of only four Ada constructs nullifies much of the power of Ada; system design concerns such as throughput, concurrency, capacity, growth, environment, etc., were not addressed by the initial SDL definition.

Based on these findings, it was determined that each SED graphic symbol would be described in the following categories based on an object-oriented design.

a. Entity
b. Object
c. Operation
d. Environment
e. Activity

2.4.3.2 SYSTEM DESIGN LANGUAGE (continued)

The validation team attempted to complete the Remote Communication SDLs with the new definition, but found:

a. It was impractical or impossible to express high level system abstractions with this technique.

b. Lower level system abstraction (fifth or sixth level) were easily expressed in this form, but did not provide sufficient system information.

The validation team was then given the additional task of defining the new SDL approach. This was accomplished over an elapsed time period of two plus months. The current SDL has evolved through several iterations of the AN/TSQ-73 Remote Communications. With each iteration, adjustments and additions to the SDL were made. The SDLs were recorded on two systems (the VAXs at Fort Monmouth and Control Data's CYBERNET system) using each system's text editor.

The validation team redefined the SDL in terms of what objectives of an SDL are, and what comprises an SDL. The objectives of the SDL are:

a. To be suitable for human (developers, users and managers) understanding.

b. To provide system input for hardware/software trade-off evaluations and succeeding design techniques.

c. To provide system input for configuration management decisions.

d. To provide for system accountability.

e. To provide for system consistency.

f. To provide for system ripple-effect tracing.

2.4.3.2 SYSTEM DESIGN LANGUAGE (continued)

g. To provide for increased productivity through automation.

h. To be utilized as the primary working system document.

The SDLs are Ada-based textual representations of system functions which are composed of:

a. System objective(s).

b. System requirements and capabilities.

c. System capacities.

d. System constraints.

e. System exception handling.

f. System entities, subentities, their relationships, and execution type (serial, concurrent).

g. System control.

h. System data flow.

i. System environment.

j. Operator interactions.

Many questions have been answered by the SDL, but more questions still need to be answered:

a. Ada purists versus Ada generalists questions. How exact or how close must Ada as a SDL follow Ada's programming language requirements? Don't forget users or managers! Are semi-colons and underscore characters required or desirable? Can functions, procedures and tasks be represented in a single SDL statement to indicate both a declaration and usage?

2.4.3.2 SYSTEM DESIGN LANGUAGE (continued)

b. The use of Ada within the SDL enters the "gray area" arena on the ambiguous use of procedures. This ambiguous use centers on the double use of Ada's procedures to indicate both a system "PROCEDURE TSQ-73" and a low level system function "PROCEDURE ENCODE MESSAGE HEADER".

c. How do you conceptually decide which Ada construct is more appropriate (task, package, procedure)?

d. How do you express Ada constructs for different or multiple processors?

e. How do you express concurrency at a high abstraction level?

f. To what detail, if any, should an SDL describe interfaces between modules?

2.4.3.3 DATA FLOW DIAGRAMS

The Data Flow Diagrams (DFD) are the third technique in the design methodology. The DFDs provide a graphic representation of input data, output data, stored data, significant intermediate data forms and data transformations. The validation team first designed a small segment of Remote Communications using Control Data's SASD automated tools. The DFDs were easily transformed from their respective SDLs. Accountability and consistency were maintained by retaining the system design's numbers. Only the lower DFD levels (below the existing SED levels) required original work. The DFD technique was then validated with the following findings:

a. The DFD technique is a viable tool to clearly illustrate the various items mentioned above.

b. The only suggestion to improve the system designer's guide would be to provide a complete DFD example illustrating the various abstract levels.

2.4.3.4 DATA DICTIONARY

The Data Dictionary (DD) is the fourth methodology technique or step. The DDs provide textual representation of data definitions, module descriptions with action (pseudo code) definitions, process specifications with (pseudo code) definitions, and undefined names. Control Data's automated data dictionary tool was used to record and produce the DDs for a chosen subset of Remote Communications. The data dictionary was to be produced from the DFDs. The transition from DFDs to DDs was much more difficult than the transition from SDLs to DFDs.

The findings of the validation team are:

a. The data dictionary is a technique to be utilized from the SDL through structured programming. In order to complete all the required DD definitions, the SDL, DFD, SC, PDL and SP were referenced.

b. The DD is a vital technique used in defining the system's software functions and data.

c. There is some redundancy in the data dictionary's pseudo code and the code produced for the methodology's SDL and PDL.

d. The system designer's guide could be improved in the following areas:

   - Definitions of data flow, transformations, etc.
   - The use of an example in explaining the DD process.
   - The methodology should view the data dictionary as a technique that begins with the SDL and ends with structured programming.

e. Control Data's automated data dictionary tool was invaluable in providing:

2.4.3.4 DATA DICTIONARY (continued)

- Input error list generation.
- Cross-reference report.
- Cross check report.
- Flagging of undefined data definitions.
- Flagging of recursively defined definitions.

2.4.3.5 STRUCTURE CHARTS

The Structure Charts (SC) are the fifth methodology technique or step. The SCs provide a graphic representation that partitions the system described by the DFDs and DD into a hierarchy of modules, each of which is viewed as performing a well-defined function and establishes the means and the direction of communication between modules. The structure charts were to be produced from the DFDs and DDs. The transition from DFDs and a partial DD to SCs was smooth. Control Data's automated structure chart tool was used to record and produce the SCs for the Remote Communications segment. The findings of the validation team are:

a. The structure charts were finished prior to the completion of the data dictionary. The minimum amount of information that was added to the data dictionary as a result of the SCs would also be found in the PDL.

b. There is informational redundancy between the structure charts and the other design methodology techniques:

   - The system partitioning functions performed by the SCs are also performed by the SDLs, DFDs, and PDL.
   - The module interface requirements performed by the SCs are also performed by the DFDs.

c. A question of whether the structure chart is a viable technique or not within the design methodology exists. Based upon the validation effort, the structure charts are a duplication of design functions performed by the

2.4.3.5 STRUCTURE CHARTS (continued)

other design techniques. Therefore, the question of should the SC be eliminated must be raised or is there something possibly missing from the methodology's definition of structure charts. This technique needs to be re-evaluated.

2.4.3.6 PROGRAM DESIGN LANGUAGE

The Program Design Language (PDL) is the sixth methodology technique or step. The PDL provides a textual representation of the structure charts into program units utilizing Ada constructs as program pseudo code. The Army's VAX system was used to record the PDL for a segment of Remote Communications. The findings of the validation team are:

a. Although the structure charts were initially used to produce the PDL, the same information was located in the DFDs which were utilized for the final PDL.

b. Information within the PDL was used as additional input (data definitions) to the data dictionary.

c. In most areas of the PDL, the structured programming was completed before it was realized that this had occurred. The remaining portions of the PDL require almost no effort to become code. Why or how this occurred can only be surmized that the segment selected for validation coding was possibly at the PDL step when the data flow diagrams were completed. It may be that the selected validation coding was too small of a segment. The possibility exists that the validation group's decomposition of DFDs included too much detail. Would this be a common occurrence or was just a fluke occurrence? Could this occur to a large number of modules within the AN/TSQ-73?

d. The system designer's guide could provide more definition in explaining the PDL process.

2.4.3.7 COMPLEXITY MEASURE

The complexity measure (CM) is the seventh methodology technique or step. The CM provides a graphic representation of the complexity of the PDL. Unfortunately, this step would not be validated due to the ease of the PDL to structured programming that occurred during the validation effort. Several discussions were held concerning this technique and the concensus was that the benefits of the complexity measure would also be beneficial as an earlier technique to be used within the methodology. The CM could also be utilized at the SDL and/or DFD step where the effects of a modification would be less dramatic and expensive.

2.4.3.8 STRUCTURED PROGRAMMING

The structured programming (SP) is the Ada coding of the PDL into compilable code. As stated earlier in the report, this was completed.

2.4.4 VALIDATION FINDINGS

The findings and suggestions of the validation team in their critique of the proposed system design methodology for large scale systems development are:

a. The proposed system design methodology appears to have achieved most of its self-imposed goals and objectives.

b. The proposed system design methodology supports top-down design, yet no single technique or step forces the individual to use top-down design. Whether the methodology requires a technique modification or addition to enforce top-down design or the existing methodology coerces top-down design through the various techniques and peer walkthrough evaluations is not known.

c. Redundancy between and among the methodology's techniques should be minimized. Potential redundant candidates are:

- The Pseudo code in the data dictionary.
- The structure chart techniques (after its re-evaluation).

2.4.2 VALIDATION FINDINGS (continued)

d. The coding development phase (PDL, CM and SP) should be revalidated with a large segment of the AN/TSQ-73.

e. The use of automated tools for the design and validation of the AN/TSQ-73 proved to be invaluable and productive within the areas of SED, DFD, DD, and SC. The role of automated tools should be enhanced within the design methodology.

f. As viewed by the validation team, the proposed methodology would be utilized as follows:

(1) The system development process begins with the SDLs statement of the system's (specifications) objectives, requirements, capabilities, etc. The developers functionally partition or decompose the system into various levels of abstraction. After every one or two abstraction levels, the developers would request the computer aided design (CAD) system to automatically produce the SEDs for the system being developed. The data dictionary is begun and automatically updated throughout the entire development process. The developers may request the automated application of the complexity measure to the system design phase. These tools, SEDs, DD reports and the CM, would assist the developers through this iterative process.

After a number of iterative decompositions and structured walkthroughs, the system paritioning process is completed. The number of abstract levels required will be dependent upon the system being developed. It should also be expected that within a specific system, the number of abstract levels will vary.

Accountability and consistency will be maintained thrcugh the SDL numbering scheme. Ripple-effect resulting from system modifications can be traced.

2.4.4 VALIDATION FINDINGS (continued)

(2) An historical data base is accessed by the developers to locate similar system functions that have been previously developed for other systems. If functional matches are located, a copy of each function (SDL, PDL, SP) is moved into the developing system's SDL where modifications may have to be made to enable incorporation. The historical data base would be searched throughout the entire development project.

(3) The developing system's SDL is enhanced through the use of Ada constructs. Ambiguous system functions will be clarified through Ada's exactness. Additional system decomposition may also occur.

(4) Other techniques will ultimately decide (from SDL and other inputs) which functions of the system will be implemented in hardware and software. The SDL will be updated to indicate these decisions. The SDL may also require modifications as a result of these decisions.

(5) The software design phase begins by expanding and amplifying the SDL towards a PDL. The data dictionary is automatically updated. As the Ada enhancement process continues, the designers may request the CAD system to automatically produce the related DD reports, the DFDs, and SCs, for the desired design area. The designers may also request the automated application of the complexity measure. Such tools would assist designers in this iterative process.

(6) The final development phase would be detailed Ada programming. This would be developed from the Ada PDL. The data dictionary is again automatically updated.

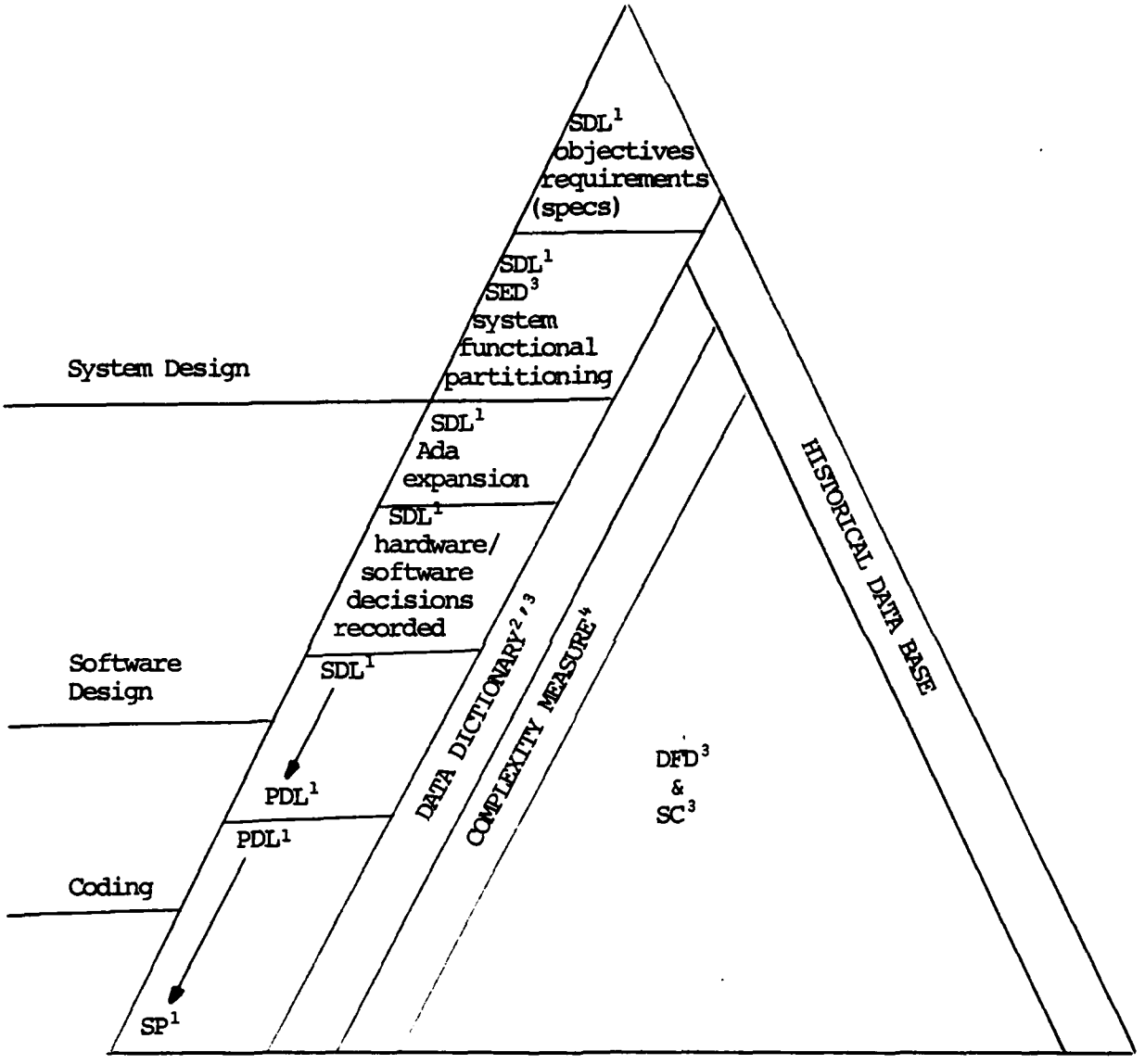
2.4.4 VALIDATION FINDINGS (continued)

The above suggested design methodology can be illustrated as shown in Figure 2-7. The modification of the design methodology as suggested by the validation team would require a Computer Aided Design (CAD) system and DD, DFDs, and SCs with enhanced meaning. The results would be:

a. A single development language, Ada, to be utilized by developers, designers, and programmers.

b. A single baseline system which grows with the system development process.

As can be seen by the later changes, the design methodology evolved to a level later shown in Figure 2-9.



[1] Ada
[2] CAD-built
[3] CAD-generated developers/designers request
[4] CAD-applied upon developers/designers request

Figure 2-7 - Revised Design Methodology Illustration

2.5 REDESIGN OF THE AN/TSQ-73 MISSILE MINDER SYSTEM USING THE ADA DESIGN METHODOLOGY

The AN/TSQ-73 Missile Minder System was designated by contract as the system to be redesigned based upon the Ada language. This section of the report describes the application of the system design methodology to the redesign of the AN/TSQ-73 Missile Minder System. The results of this task include:

a. The application of the design methodology to include the following phases:

- System design phase
- Software design phase
- Programming phase

b. Key issues.

c. Advantages and disadvantages of the design methodology as applied to the AN/TSQ-73 system.

d. Rationale for change in design methodology.

e. Verification/test methods and results.

2.5.1 INTRODUCTION

The Missile Minder AN/TSQ-73 System is the air defense command and control system (Figure 2-8) designed to coordinate actions of Hawk, Improved Hawk, and Nike/Hercules surface-to-air missiles against hostile air targets. The system is fielded in two configurations, the group/BDE level system and the BN-level system. The mission of each system is as follows:

a. The mission of the group/BDE level system concentrates on the air defense activities of subordinate BN-level systems, allocation and assignment of air defense resources, planning air defense operations, supervising the effectiveness of real-time air defense resources, serving as the tactical operations center for group operations, and interfacing with other group-level missile minder systems, and tactical command and control systems of other services.

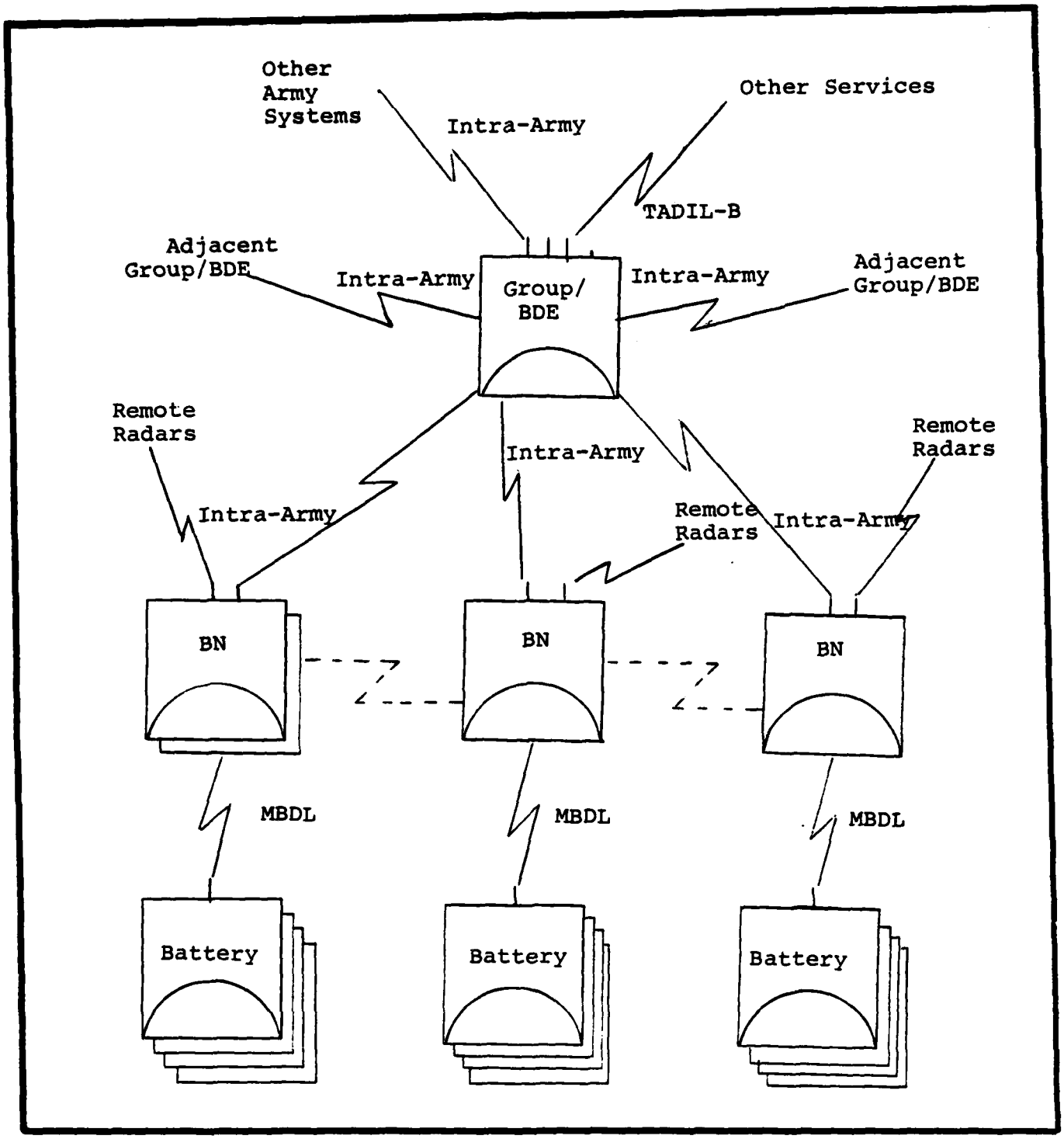


Figure 2-8 - Group/BDE and BN Operational Configuration

2.5.1 INTRODUCTION (continued)

b. The mission of the BN-level configuration is to control the actual direction of the air battle with its execution being directed at the fire unit level. The BN-level system acquires, tracks, identifies aircraft, provides real-time threat evaluation and weapons assignment for hostile aircraft engagement, and provides friendly aircraft protection. A BN-level system can be designated as a master BN-level system when a group/BDE-level system is inoperable or non-existent due to local conditions. The designated master BN-level system will provide the interface to other Army tactical data systems and tactical command and control systems of other military services.

The battalion level system was selected for redesign of the AN/TSQ-73 of which one subfunction will be programmed to the level of executable code. The overall design of the AN/TSQ-73 system has been decomposed and repartitioned into four major functional areas:

- Command and control.
- Target control.
- Remote communications.
- Radar communications.

Target control has been selected as the area for detail design, of which two program units - "smoothing and prediction" will be coded.

The redesign of the AN/TSQ-73 and coding was affected by the following:

a. The AN/TSQ-73 system specifications MIS-19501C and MIS-10297J were used as prime requirements documents for the system. The system specifications were supplemented by product specifications for further delineation of requirements when necessary.

2.5.1 INTRODUCTION (continued)

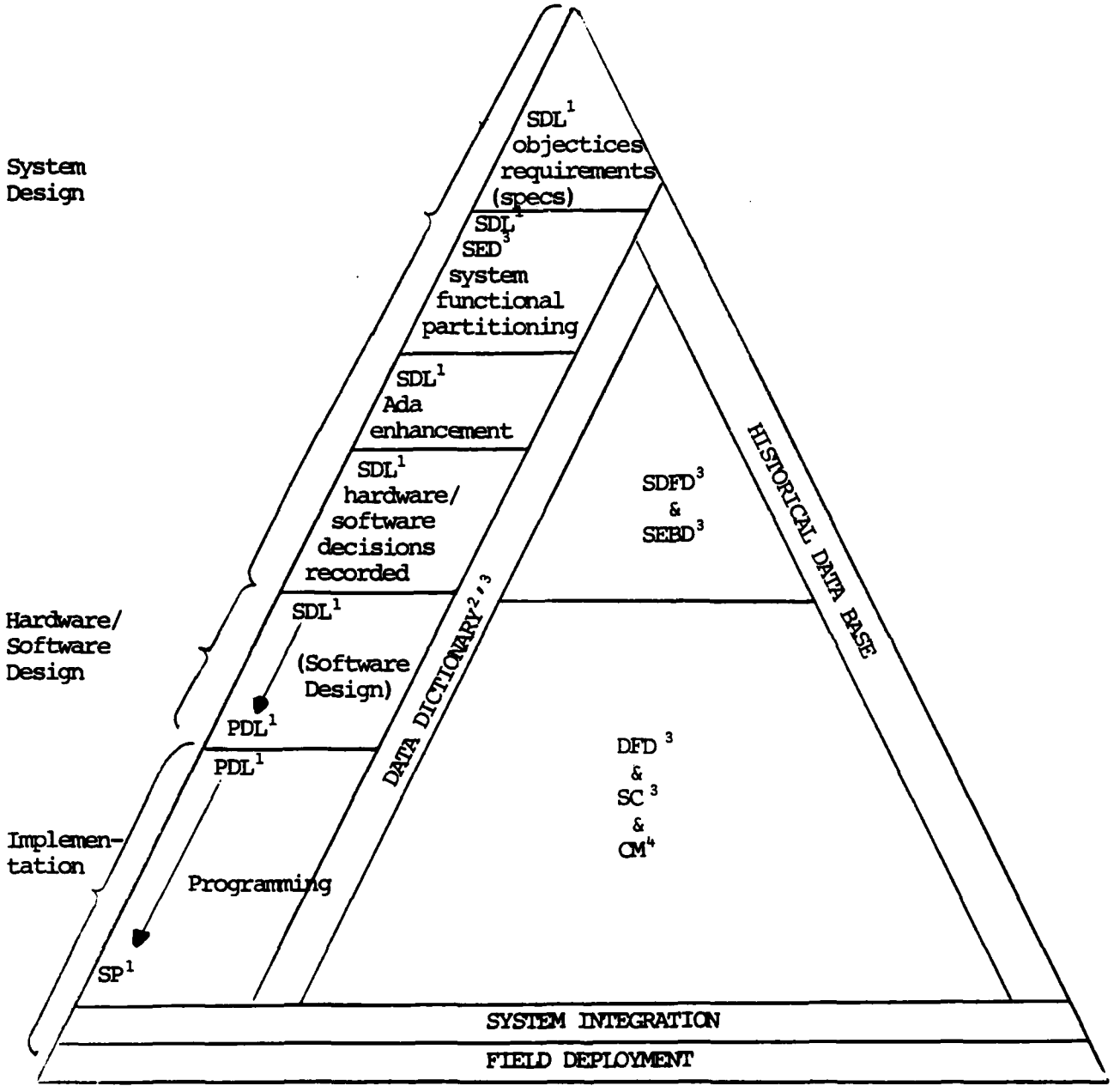
b. Design deviated from the existing specification where improvements could be made or restructuring was necessary for the Ada architecture.

c. Design was not constrainted by the existing AN/TSQ-73 hardware configuration.

d. The design phase included only a sufficient level of detail to validate the methodology.

e. No attempt was made to determine the adequacy of Ada in satisfying AN/TSQ-73 processing requirements. This limitation results from the use of interpreter/translator for program coding and validation.

Although the contract called for a large scale system software design, the top-level system design and some hardware design was necessary to show how the design methodology is applicable to the AN/TSQ-73. Figure 2-9 shows the system design methodology which is based upon the Ada language. The top level design is shown at the top of the chart and represents the system design function described in Paragraphs 2.5.2.1 and 2.5.2.2. The second level of the chart is the software and hardware design phases which are based upon the hardware and software trade-offs normally conducted as a part of the hardware and software trade-off analysis. The hardware design was limited to development of the engineering block diagram of the system (Paragraph 2.5.2.3). Therefore, the hardware design methodology has been omitted. The hardware and software design is eventually completed and integrated as a complete system.



[1]Ada
[2]CAD-built
[3]CAD-generated developers/designers request
[4]CAD-applied upon developers/designers request

Figure 2-9 - System Design Methodology

2.5.1 INTRODUCTION (continued)

The complexity measure, which is shown across all phases, is an area requiring further investigation. The complexity measure herein was the McCabe Complexity Measure which is applicable to the software design.

The design is based upon the use of computer aided design tools, but may be accomplished manually. However, the manual method defeats the use of automation to provide accountability and traceability, and the elimination of the drudgery involved in the detail design work.

2.5.2 SYSTEM DESIGN PHASE

The system design phase included three subphases:

a. Development of the system entity diagrams (SED)

b. Development of the system design specifications using Ada as a system design language (SDL)

c. Hardware/software trade-off analysis.

These three subphases were adequate to develop the system design to the specification level, commonly known as B2 hardware and B5 software specifications using Ada-like pseudo code and Ada constructs. Although the SED development started as a separate, independent phase, it actually was completed in conjunction with the other two phases. It was more productive to complete only one or two SED levels and then construct the corresponding SDL. Upon completion of the corresponding SDL, the SED was further decomposed and the corresponding SDL was prepared. Although the hardware/software trade-off analysis was limited, Control Data is satisfied that all three subphases are dependent and should be developed through an iterative process to the desired level.

The evolved design methodology lends itself to automation which was a major consideration in selecting each design methodology technique. The various levels of automation were not determined since a full complement of automatic or computer assisted design tools were not available. However, an automated tool currently under development by Control Data was used for generation and update of the SEDs. The design methodology which was based upon the use of the tool limited the number of functional levels to three. This was far too restrictive for a large system such as the AN/TSQ-73. The design plan expanded the decomposition of requirements to "n" level. Any fully developed automated tool should support these levels.

2.5.2.1 SYSTEM ENTITY DIAGRAM (SED)

The system entity diagram was produced using AN/TSQ-73 specifications MIS-19501C and MIS-10297J to define the system functional requirements. Since the SED involved a learning process for the system analyst and programmers in the understanding of both the design methodology and the AN/TSQ-73 system, its value as a top-level structure tool was well demonstrated. The SED provided a logical grouping of the functions in a graphic form, easily readable by both the users, analysts and programmers. Although the preparation of the SED was relatively easy, several iterations were required to capture the functions. Table 2-1 provides a list of the hierarchical SED entities and Appendix C provides the completed SEDs.

The SED was based to a degree upon the packaging concept of the Ada language. In fact, a detailed SED may be eventually defined in software areas to reflect packages, tasks, procedures and functions. In the system design SDL technique, note that the titles reflect the system diagram and the system entity numbers to provide traceability and accountability throughout the design.

The development of the SEDs followed the processes listed below:

a. Identify the functions of the AN/TSQ-73 at the top hierarchical level. This serves as a baseline for further decomposition of functions.

b. Develop subentities at the second, third through "n" levels.

c. Assign numbers to each entity level beginning at the top structure level and continuing to the lowest entities. The numbers provide traceability throughout the design.

1.0 COMMAND AND CONTROL

1.1 General Operating System

1.1.1 System Bootstrap Initialization

1.1.1.1 Executes Initial Start-Up Procedures
1.1.1.2 Loads and Executes Operating System
1.1.1.3 Initializes System Console Device
1.1.1.4 Initializes System Log
1.1.1.5 Initializes Remaining Peripherals
1.1.1.6 Initializes System Coprocessors
1.1.1.7 Loads and Executes System Command Interpreter

1.1.2 Resource Management

1.1.2.1 Processor Management
1.1.2.2 Memory Management
1.1.2.3 Interrupt Management
1.1.2.4 Time Management
1.1.2.5 Peripherals Management
1.1.2.6 Coprocessors Management

1.1.3 Task Management

1.1.3.1 Task Characteristics Manager

1.1.3.1.1 Task Type
1.1.3.1.2 Task Status
1.1.3.1.3 Task Priority
1.1.3.1.4 Site Execution Location Determination
1.1.3.1.5 Critical Tasks

1.1.3.2 Task Scheduling Policies Manager
1.1.3.3 Task Synchronization Manager
1.1.3.4 Task Invocation
1.1.3.5 Task Suspension
1.1.3.6 Task Resummation
1.1.3.7 Task Termination
1.1.3.8 Task Creation
1.1.3.9 Task Deletion

1.1.4 Task Communications

1.1.4.1 Low Level

1.1.4.1.1 Device Communications Using I/O Bus
1.1.4.1.2 Device Communications Using Memory I/O Bus
1.1.4.1.3 Intertask Communications Using Shared Variables
1.1.4.1.4 Intertask Communications Using Message Buffers

Table 2-1 - Hierarchical Listing of AN/TSQ-73 (page 1 of 7)

1.1.4.2 High Level

1.1.4.2.1 Channels
1.1.4.2.2 Files
1.1.4.2.3 Intrasystem Transfers

1.1.5 Power Fail/Automatic Restart

1.1.5.1 Power Fail
1.1.5.2 Automatic Restart

1.1.6 Device I/O Handlers

1.1.6.1 Physical Device Interface
1.1.6.2 Physical to Logical Interface Task
1.1.6.3 Logical Device Channels

1.1.7 File Management

1.1.7.1 Data Structures
1.1.7.2 File Manager
1.1.7.3 File Manager Utilities
1.1.7.4 Files to Logical Device Channels

1.1.8 Error Recovery and System Reporting

1.1.8.1 Error Recovery
1.1.8.2 System Reporting

1.1.9 System Command Interpreter

1.1.9.1 Assembler
1.1.9.2 Ada Language
1.1.9.3 Libraries
1.1.9.4 Editors
1.1.9.5 Utilities

1.1.10 Maintenance and Diagnostics Utilities

1.1.10.1 Destructive (Off-Line) Utilities
1.1.10.2 Non-Destructive (On-Line) Utilities

1.2 Runtime Support Software

1.2.1 Runtime Operating System
1.2.2 Runtime System Library
1.2.3 Mathematical Library
1.2.4 Character String Library
1.2.5 Runtime On-Line Debugger

1.3 Runtime Initialization

1.3.1 Loads Runtime Support Software
1.3.2 Optionally Loads On-Line Debugger
1.3.3 Loads User Application Library
1.3.4 Optionally Loads System Exercisor
1.3.5 Loads I/O Message Interpreter
1.3.6 Optionally Opens Transaction Message Log
1.3.7 Loads Site Adaptation
1.3.8 Loads Target Control
1.3.9 Loads Remote Communications
1.3.10 Loads Radar Communications

Table 2-1 - Hierarchical Listing of AN/TSQ-73 (page 2 of 7)

1.4 User Application Library

1.4.1 Transverse Mercator
1.4.2 Sterographic Function
1.4.3 Slant Coordinates
1.4.4 Geographic Function

1.5 I/O Message Interpreter

1.5.1 Message Routing Manager

1.5.1.1 Input Messages
1.5.1.2 Output Messages

1.5.2 Display Update I/O Manager

1.6 Runtime Termination

1.6.1 Generate Termination Message to Local/ Remote Users
1.6.2 Terminates Radar Communications
1.6.3 Terminates Remote Communications
1.6.4 Terminates Target Control
1.6.5 Optionally Closes Transaction Message Log
1.6.6 Termination Options

1.7 Runtime System Exercisor

1.7.1 Exercisor Command Supervisor
1.7.2 User Application Library Exercisor
1.7.3 I/O Message Interpreter Exercisor
1.7.4 Site Adaptation Exercisor
1.7.5 Target Control Exercisor
1.7.6 Remote Communications Exercisor
1.7.7 Radar Communications Exercisor

2.0 TARGET CONTROL

2.1 Start-Up Initialization

2.1.1 Real-Time Clock
2.1.2 Initialize Track Algorithms
2.1.3 Initialize Threat Evaluation Algorithms
2.1.4 Initialize Weapons Assignment Algorithms
2.1.5 Initialize Track Tables
2.1.6 Sector Definition

2.2 Tracking

2.2.1 IFF/SIF

2.2.1.1 Auto Beacon Interrogation
2.2.1.2 Manual Interrogation
2.2.1.3 Verify IFF Code
2.2.1.4 Mode 4 Interrogation
2.2.1.5 Interrogate Beacon

2.2.2 Target Report Processing
2.2.3 Track While Scan

Table 2-1 - Hierarchical Listing of AN/TSQ-73 (page 3 of 7)

2.2.3.1 Correlation Actions

2.2.3.1.1 Computer Deviation Vector
2.2.3.1.2 Innergate Correlation
2.2.3.1.3 Outergate Correlation
2.2.3.1.4 Non-Auto Beacon Correlation
2.2.3.1.5 Altitude Gate Correlation
2.2.3.1.6 Correlation Score

2.2.3.2 Association

2.2.3.2.1 Sort Pairs by Track Category
2.2.3.2.2 Compare Correlation Scores
2.2.3.2.3 Merge Condition Test
2.2.3.2.4 IFF Emergency Test
2.2.3.2.5 Update Association Count

2.2.3.3 Smoothing

2.2.3.3.1 Smooth Position
2.2.3.3.2 Smooth Velocity
2.2.3.3.3 Smooth Altitude
2.2.3.3.4 Detect Track Maneuverability

2.2.3.4 Prediction

2.2.3.4.1 Computer Assoc Predicted Position
2.2.3.4.2 Computer Non-Assoc Predicted Position
2.2.3.4.3 Update Track Record

2.2.4 Track While Scan Utilities

2.2.4.1 Automatic Initiate New Track
2.2.4.2 Drop Track
2.2.4.3 Determine Track Quality

2.3 Track Manager

2.3.1 Track Update
2.3.2 Manage Tables
2.3.3 Determine Track Status
2.3.4 Data Control

2.3.4.1 Routing
2.3.4.2 Filtering

2.3.5 Operator Switch Action

2.3.5.1 Operator Tracking Actions

2.3.5.1.1 Initiate Tracks
2.3.5.1.2 Hook Display Element
2.3.5.1.3 Update
2.3.5.1.4 Enter Track Information
2.3.5.1.5 Drop Track
2.3.5.1.6 Change Mode of Tracking
2.3.5.1.7 Obtain Close Control
2.3.5.1.8 Enter Maneuver Sensitivity
2.3.5.1.9 Enter Alternate Track Number
2.3.5.1.10 Enter Jam Strobes
2.3.5.1.11 Enter Clutter Map Area
2.3.5.1.12 Enter Method of Tracking
2.3.5.1.13 Challenge IFF Modes

Table 2-1 - Hierarchical Listing of AN/TSQ-73 (page 4 of 7)

2.3.5.2 Operator Tactical Actions

2.3.5.2.1 Couple/Decouple Fire Units
2.3.5.2.2 Make Primary/Secondary Assignments
2.3.5.2.3 Reference/Dereference Track for Trans
2.3.5.2.4 Enter Fire Unit Information
2.3.5.2.5 Designate Display Elements
2.3.5.2.6 Terminate Engagements
2.3.5.2.7 Accept Reject Remote Site Commands

2.4 Counter Measure Process

2.4.1 Jam Strobe

2.4.1.1 Jam Strobe Correlation
2.4.1.2 Jam Strobe Association
2.4.1.3 Jam Strobe Initiation
2.4.1.4 Jam Strobe Update
2.4.1.5 Drop Jam Strobe

2.4.2 Chaff

2.5 Engage Target

2.5.1 Threat Evaluation
2.5.2 Evaluate Weapons Assignment Score
2.5.3 Weapons Assignment
2.5.4 Monitor Engagement

3.0 REMOTE COMMUNICATIONS

3.1 Communication Supervisor

3.1.1 Consume Internal Message
3.1.2 Initialize Remote Communication

3.1.2.1 Housekeeping
3.1.2.2 Initialize Communication Subsystem Parameters
3.1.2.3 Initiate Manage Message
3.1.2.4 Initiate Manage Data Links

3.1.3 Manage Data Links

3.1.3.1 Manage Data Link Status Table
3.1.3.2 Manage Protocols

3.1.4 Manage Messages

3.1.4.1 Outgoing Messages
3.1.4.2 Incoming Messages
3.1.4.3 Manage Message Buffers and Queues

3.1.5 Manage Fault Detection, Correction and Reporting

3.1.5.1 Identify Faults
3.1.5.2 Update Error Counters
3.1.5.3 Attempt Corrective Action
3.1.5.4 Report Faults

3.1.6 Terminate Remote Communications

3.1.6.1 Build Termination Message
3.1.6.2 Start Terminate Data Link
3.1.6.3 Update Data Link Status Table

Table 2-1 - Hierarchical Listing of AN/TSQ-73 (page 5 of 7)

3.1.6.4 Abort Data Link Manager
3.1.6.5 Abort Message Manager
3.1.6.6 Update Transaction Message Log

3.1.7 Produce Internal Message

3.2 Intrasystem Transfers

3.2.1 Get Intrasystem Transfers
3.2.2 Put Intrasystem Transfers
3.2.3 Message Posting

3.2.3.1 Consumer Timer
3.2.3.2 Producer Timer

3.3 Communication Protocols

3.3.1 TADIL-B Protocol Supervisor
3.3.2 ATDL-1 Protocol Supervisor
3.3.3 MBDL Protocol Supervisor

3.3.3.1 Consume Comm Supervisor Message
3.3.3.2 Initialize Data Link
3.3.3.3 Transmit Message
3.3.3.4 Receive Message
3.3.3.5 Protocol Utilities
3.3.3.6 Terminate Data Link
3.3.3.7 Comm Supervisor Message Product

4.0 RADAR COMMUNICATIONS

4.1 Communication Supervisor

4.1.1 Consume Internal Message
4.1.2 Initialize Radar Interface Communication
4.1.3 Manage Data Links

4.1.3.1 Manage Data Link Status Table
4.1.3.2 Manage Protocols

4.1.4 Manage Messages

4.1.4.1 Message Supervisor

4.1.4.1.1 Determine Message Action
4.1.4.1.2 Select Highest Priority Message
4.1.4.1.3 Post Transmission Message Action

4.1.4.2 Outgoing Messages

4.1.4.2.1 Select Highest Priority Message
4.1.4.2.2 Validate Outgoing Message

4.1.4.3 Incoming Messages
4.1.4.4 Internal Messages
4.1.4.5 Manage Message Buffers and Queues

4.1.5 Manage Fault Detection, Correction and Reporting

4.1.5.1 Identify Faults
4.1.5.2 Update Error Counters
4.1.5.3 Attempt Corrective Action
4.1.5.4 Report Faults

4.1.6 Terminate Radar Communication

Table 2-1 - Hierarchical Listing of AN/TSQ-73 (page 6 of 7)

4.1.6.1 Build Termination Message
4.1.6.2 Start Terminate Data Link
4.1.6.3 Update Data Link Status Table
4.1.6.4 Abort Data Link Manager
4.1.6.5 Abort Message Manager
4.1.6.6 Update Transaction Message Log

4.1.7 Product Internal Message

4.1.7.1 Select Highest Priority Message
4.1.7.2 Dequeue Message from Produce Internal Message Queue

4.2 Intrasystem Transfers

4.2.1 Get Intrasystem Transfers
4.2.2 Put Intrasystem Transfers
4.2.3 Message Posting Timer

4.3 ATDL-1 Protocol Supervisor

4.3.1 Consume Comm Supervisor Message
4.3.2 Initialize Data Link
4.3.3 Transmit Message
4.3.4 Receive Message
4.3.5 Protocol Utilities
4.3.6 Terminate Data Link
4.3.7 Comm Supervisor Message Product

Table 2-1 - Hierarchical Listing of AN/TSQ-73 (page 7 of 7)

2.5.2.1 SYSTEM ENTITY DIAGRAM (SED) (continued)

d. Enter system entity diagrams into data base for update and accountability.

e. Update the diagram as required.

f. Place the system entity diagram under configuration management control. This will not be accomplished until the design using SDL is complete; i.e., the system level and the subsystem level.

An early statement above indicated that the system entity diagram was relatively easy to prepare. This was in part as a result of the provision of AN/TSQ-73 requirements by the government. In practice, the system entity diagram is a cumulative product based on a thorough analysis of user requirements. The greatest benefit of the SED, perhaps, is that it reflects user requirements in relatively easy-to-read functions. This is especially true with military personnel who are oriented toward the use of block and line diagrams to reflect functions. The software designer found the SED to be especially useful because it presented the overall functional picture.

The SED also has significant disadvantages:

a. The SED does not provide interfaces between entities.

b. The SED is a large diagram which restricts its use as a total single picture of the system. It must be reviewed in segments which is allowed by the automated tools.

2.5.2.2 DEVELOPMENT OF SYSTEM DESIGN SPECIFICATIONS

The system design specifications were completed using Ada as a system design language (SDL). Application of Ada's structuring features was based upon these guidelines:

a. Utilize packages to represent collections of logically related system activities. The items identified in any given package should be logically independent from those represented in other packages.

b. Utilize procedures and functions to represent system activities which appear to be sufficiently distinct to act as stand-alone units, but at the same time, dependent upon other stand-alone units to accomplish a more global activity.

c. Utilize tasks to represent activities which can provide services to other program units in a concurrent fashion.

d. Utilize the names and numbering structure provided in the SED to provide accountability and traceability.

Using the above guidelines, the SDL captured all the functions necessary to define the system. In comparing the system design and performance characteristics with those guidelines of MIL-STD-490, the conclusion can be reached that Ada can be used as a system design language. At first review, the system design was not readily apparent to the casual reader although written in English and in narrative format based upon Ada-like pseudo code. As the design further developed, it was relatively easy to follow since it provided a logical sequence from broad requirements to detail design. The design specifications were easier to follow then the traditional A-level, B-level, and C-level specifications, in that the design provided a logical sequence and avoided the duplication normally found between levels of specifications. Although the logical steps in the design methodology should be followed, the SDL can be used to document the design to the level of executable

2.5.2.2 DEVELOPMENT OF SYSTEM DESIGN SPECIFICATIONS (continued)

code. Therefore, the design methodology may be used to specify design of the software to a single baseline.

The SDL above provides no graphic information. Therefore, the SDL should be supported with graphic information to depict the design, particularly interfaces for clarity. First, the hardware design section of the system should show the partitioning of the hardware in graphic form. To provide this capability, graphic tools programmed in Ada should be provided as a part of the system design phase.

2.5.2.3 HARDWARE/SOFTWARE TRADE-OFF ANALYSIS

Hardware and software trade-off anlaysis in the design process was limited; however, the design methodology provides a high degree of decoupling in its design which, in turn, provides a high degree of flexibility for the design engineer. A preliminary analysis indicates that the hardware design can be specified to the B2 level using Ada as a design language. A shortcoming exists in that graphic tools are required to generate the supporting graphics tools, however, the methodology techniques have the capability of generating the graphic information via on an automated design system.

In the past, the selection of design, both hardware and software, has to a large extent resulted from the way the user defines his requirements. Design factors have included such considerations as size, weight, constraints, processing times, response times, bench markings, permanancy of design, redundancy, flexibility, reliability, etc. With the exception of selected time functions that can be measured, the actual partitioning of the design and selection of the components is, to large degree, a judgement factor in the early days of design. The partitioning of hardware and software in commercial systems is a lesser problem in that software design is based upon existing product lines. In contrast, embedded

2.5.2.3 HARDWARE/SOFTWARE TRADE-OFF ANALYSIS (continued)

military computer systems are based upon militarization of existing products, new designs and potentially the Military Computer Family. With the liberal use of microprocessors, the partitioning becomes an even more critical issue, even within the top-level design.

Models have been used in the past, but unfortunately, the models have been almost as costly and time consuming as the actual design and fabrication. Secondly, the models have been application-dependent. Nevertheless, this area must be investigated in more detail with the goal being the inclusion of the development of automated simulation/modelling techniques.

Although a detailed hardware/software trade-off analysis was not accomplished as a part of the system design, some redesign of the AN/TSQ-73 was accomplished to provide a better understanding of the hardware and software partitioning. The design also takes maximum advantage of Ada's packaging, increases the radar processing capabilities, improves communication interfaces with several baud rates and provides continuity of operation through redundancy. Figure 2-10 is a representation of the redesigned AN/TSQ-73 engineering block diagram but does not represent a final design, although this design is considered an improvement over the original design and may be fully implemented within the AN/TSQ-73 shelter. The design provides a number of capabilities:

a. Judicious selection of components from today's data base will provide the additional flexibility for any additional overhead and processing time, if any, in the use of Ada.

b. It enhances communication and improves error rate through the use of variable, selectable bit rates and error detection correction schemes. The program for accomplishing this task is a part of remote communication Ada package, which may be implemented as firmware or software. The trend is implementation in firmware.

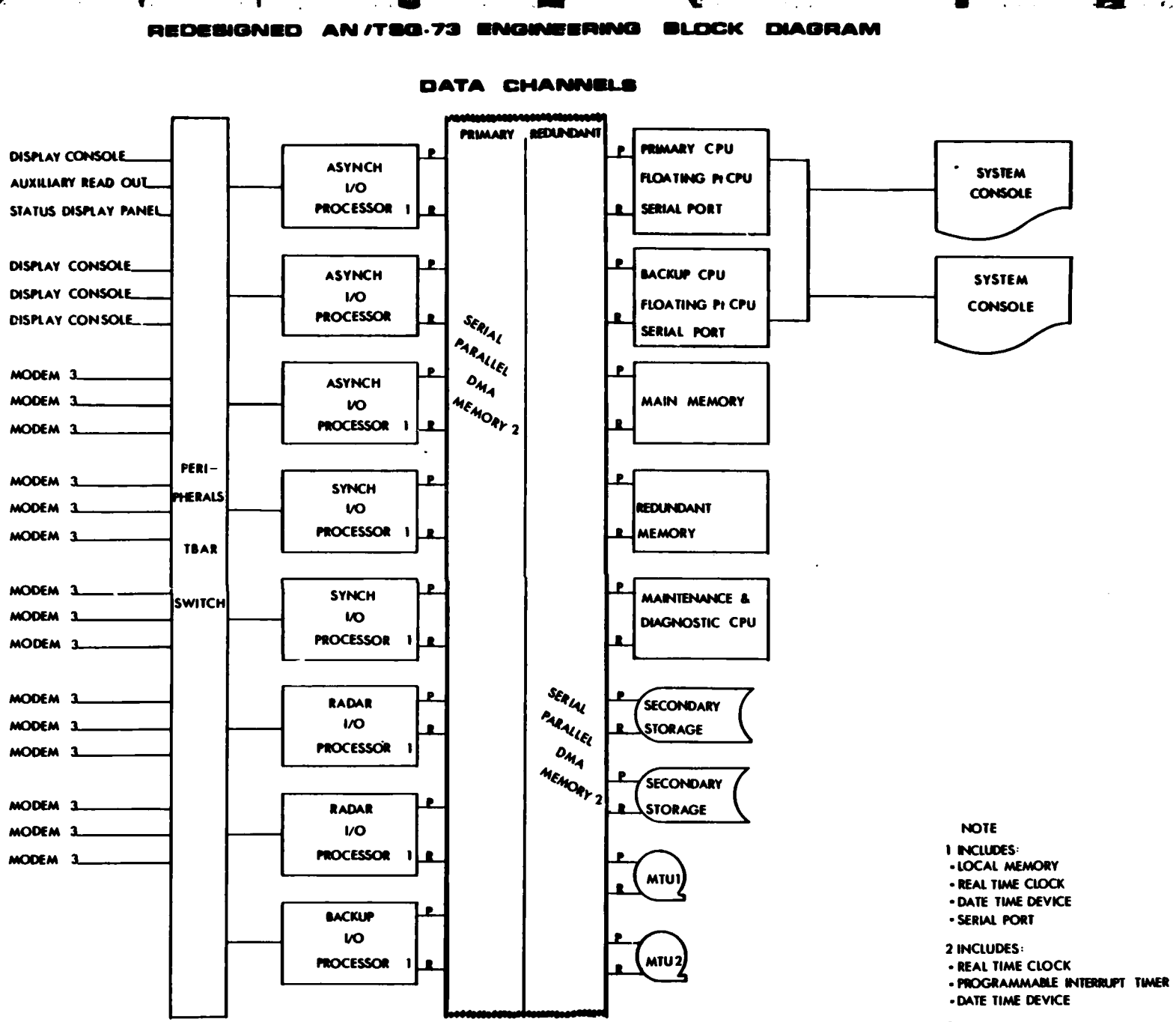


Figure 2-10 - Redesigned AN/TSQ-73 Engineering Block Diagram

2.5.2.3 HARDWARE/SOFTWARE TRADE-OFF ANALYSIS (continued)

c. Multi-radar inputs can be processed simultaneously. The target control function in the proposed design permits simultaneous processing of a multiple target.

d. It provides continuity of operation through:

(1) Switching of peripherals.

(2) Redundancy of CPUs, memories, and data channels.

(3) Multi-data channels.

(4) Distributed real-time clocks and processing functions.

e. Multi-power sources.

f. Flexibility in election of peripherals.

2.5.3 SOFTWARE DESIGN PHASE

The software design phase consisted of three steps:

a. Development of data flow diagrams.

b. Preparation of data dictionary.

c. Structure charts.

These three subphases were easier to prepare as a sequence than the system design phase. However, some iterative changes were necessary primarily because of the learning process involved in using the methodology.

2.5.3.1 DATA FLOW DIAGRAMS

The data flow diagrams were identified to include these four areas:

a. Inputs.

b. Transformations (processes).

c. Storage areas.

d. Outputs.

2.5.3.1 DATA FLOW DIAGRAMS (continued)

The data flow diagrams were developed to the lowest level and provided traceability back to system design phase by extending the numbering scheme. The names associated with the processes, unlike the system entity diagram which included titles of the functions, denotes action, i.e., "Engage Target". The data flow diagrams initially were developed only for the four functional entities of command and control, target control, remote communication and radar interfaces. Later the top-level flow was provided separately from the command and control which initially provided the relationship between the entities.

The data flow diagrams begin with a top level (context) data flow diagram (Figure 2-11) which shows the data flows for the overall system. The second level data flow (Figure 2-12) shows the top level diagram for the battalion system which is subsequently decomposed into detailed flow to the lower entities. The data flow diagrams are included as Appendix E.

The major shortcomings of the data flow diagrams are that they do not provide control information or the direction of the data flow. The direction of flow is reflected later in the structure chart where control is also illustrated. In developing the detailed flows for target control, the programmer analyst had a tendency to look for data control that is normally associated with the traditional method of flow charting. Secondly, the data flows do not show parallelism or sequence of events that are traditionally found in data flow charting. Once the traditional views were overcome, the development of data flows was relatively easy.

The data flow diagrams were based upon the AN/TSQ-73 documentation provided by the USA CECOM. In the development of a new system, the data flow diagram would be produced in close coordination with the user and would be based upon

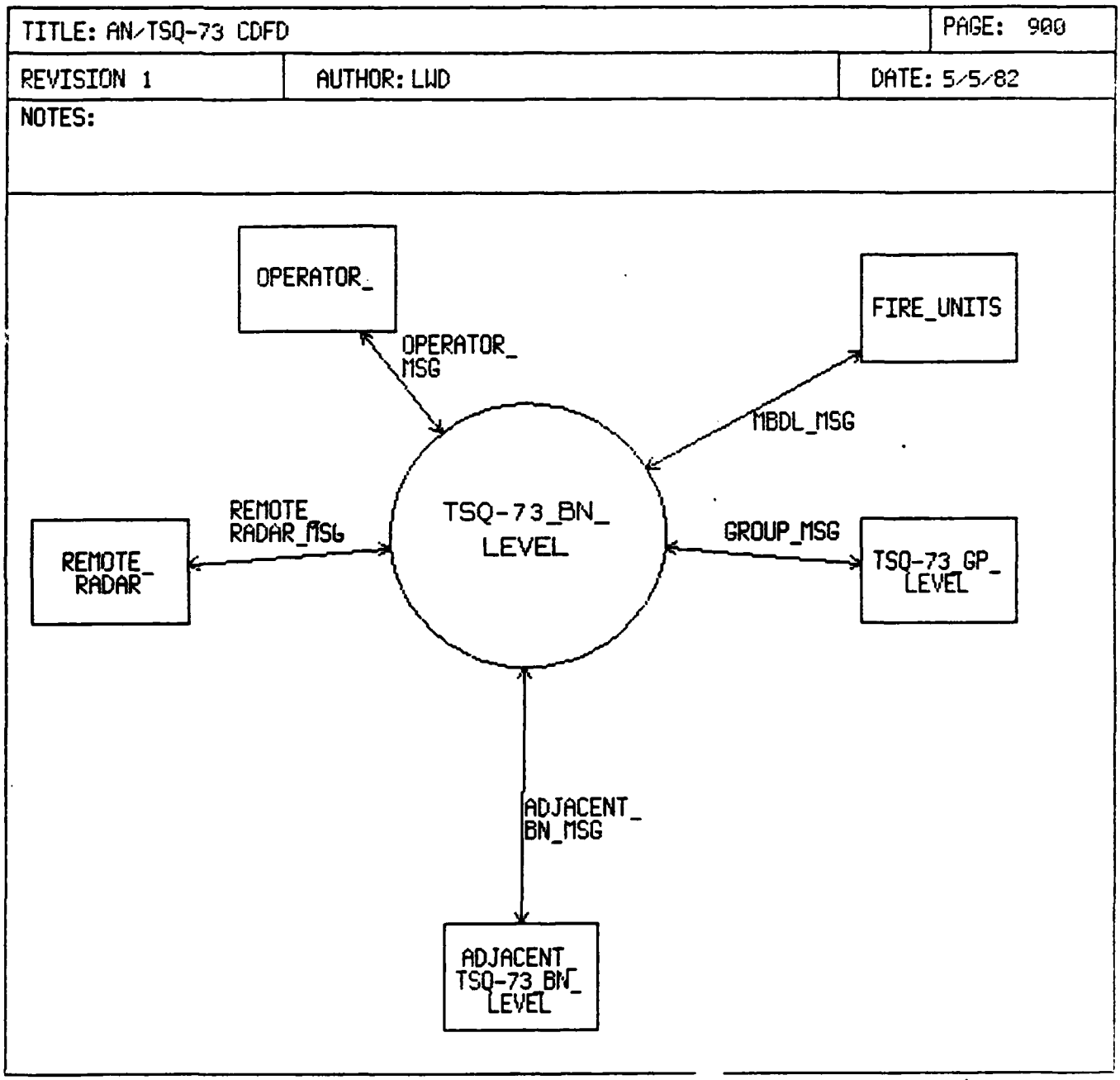


Figure 2-11- Top-Level Flow Diagram

TITLE: AN/TSQ-73 BN-LEVEL SYSTEM DFD		PAGE: 901
REVISION 1	AUTHOR: LWD	DATE: 5/5/82
NOTES:		ROOT:

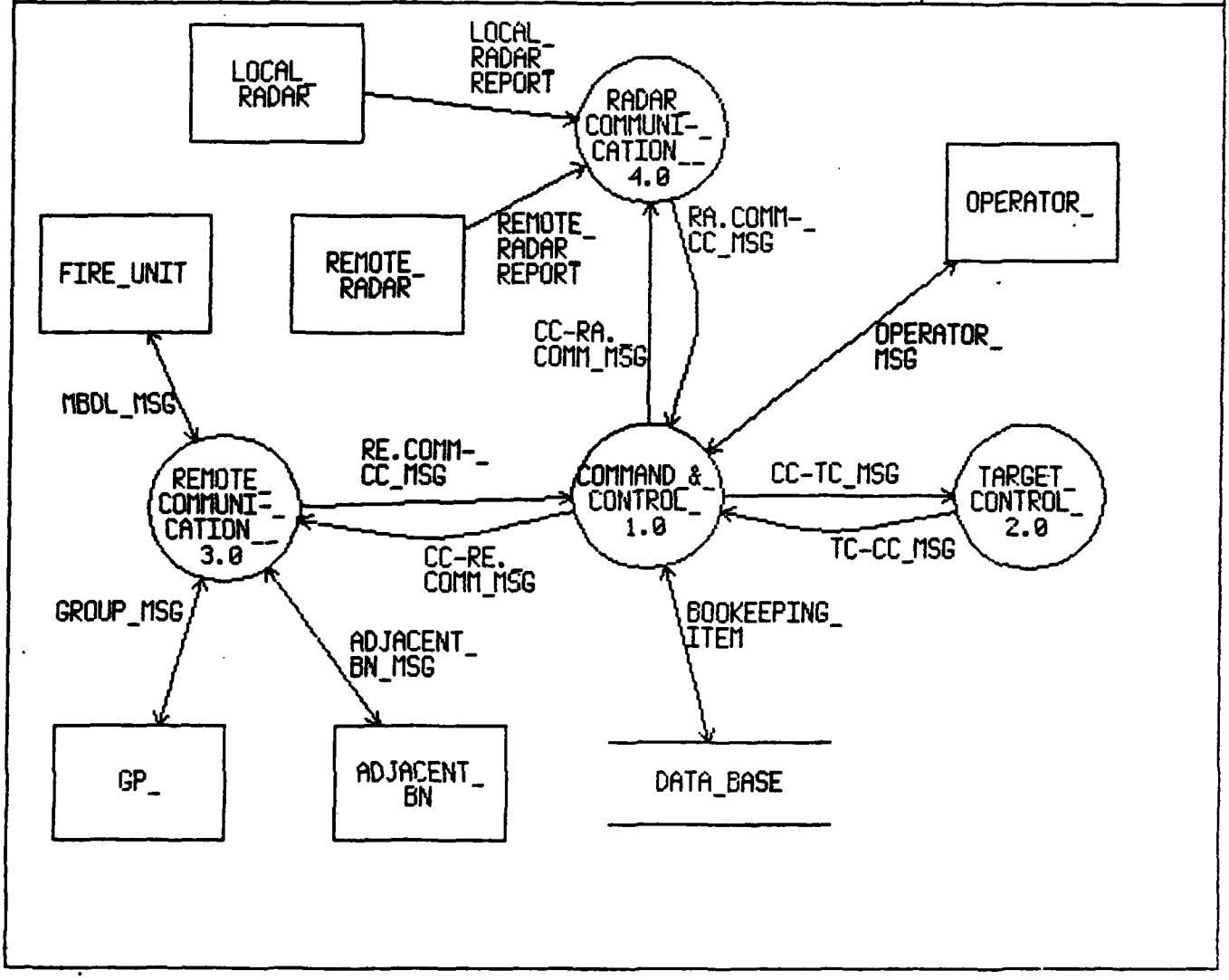


Figure 2-12- Second Level Flow Diagram

2.5.3.1 DATA FLOW DIAGRAMS (continued)

a thorough analysis of the requirements. The provided specification defined the requirements to a level of detail that permitted development of the data flow diagrams. The input, output and storage were more readily identifiable from the AN/TSQ-73 specifications. The identification of the processes was more difficult in that the descriptions were not apparent or were not provided in the specifications. An exception is the algorithms for the tracking functions. As a result, the program analyst relied heavily upon data flows and his own analysis of the requirements. The analyst also and a learning process in some area such as radar operations and interfaces. To avoid loss of productive time, the analyst used his own initiative to increase his knowledge in areas of need. In addition, Control Data provided a consultant in air defense operation.

The four packages, command and control, target control, radar communications, and remote communications, were developed by separate analysts; therefore, there were some duplications in the flows. This resulted in some changes in the SEDs, SDL and the data flows. A more detailed walkthrough of the design earlier would have prevented the duplication. Nevertheless, the duplication was identified and eleminated early.

The remote communications package was selected as the package to validate the methodology; therefore, a more thorough analysis was accomplished in this area. The package with the most detail is target control since the subfunction to be designed and programmed is a part of this function.

2.5.3.1 DATA FLOW DIAGRAMS (continued)

The preparation of the data flow diagram was assisted by use of computer aided design tools still under development. Automated tools show a significant advantage in that they can identify all other changes resulting from a single change in flow diagrams and provided updates with relative ease.

2.5.3.2 DATA DICTIONARY

The data dictionary is one of the most important tools in the design methodology. The data dictionary defines all data, data flows, transformations, data files, storage areas, reports, etc., provided in the data flow diagrams. In normal practice, the data dictionary would be developed as a result of a thorough analysis in conjunction with the user.

The structure of the process descriptions in the data dictionary is based upon Ada-like pseudo code. To arrive at this structure, segments of the target control were developed using both Ada-like pseudo code and pure Ada constructs based upon MIL-STD-1815. In addition, unambiguous definitions of data flows (vectors) and storage areas (files) are provided in the data dictionary.

2.5.3.3 STRUCTURE CHARTS

The structure chart was based upon a concept by Steven C. Myers and Larry Constantine, entitled "Structured Design", IBM System Journal, 1974, (Reference Appendix A). The structure chart utilized the information from the data flow diagrams and the data dictionary. In addition, the structure charts separate data and control communications and shows the direction of flows for each. The structure chart may also show loops in the system. The reviewer of the design can use the charts as a focus of the design evaluation process as long as all program and modules names,

2.5.3.3 STRUCTURE CHARTS (continued)

functions and interfaces are defined rigorously in the data dictionary. The structure chart can be viewed as follows:

a. A top-level hierarchical software design for use by major decision makers who do not want to be involved in the intricate details of the design.

b. A hierarchical design of lower program units for analysts and programmers who are concerned with the minute details of the software design.

In order to create a structure chart from a data flow diagram, the strategy of transform analysis was employed. Transform analysis locates the "central transform" by tracing the data from physical input for logical input and from logical output to physical output. The bubble or bubbles where the data is most abstracted from the physical become the central transform or transforms. On the structure chart these bubbles would become boxes on the second level. A controller box is placed above. Lower level boxes represent the processing of the data from physical inputs to physical outputs. By utilizing transform analysis, the role of intuition in building a structure chart is greatly reduced. The advantages of the structure chart are that the system:

a. Is easier to develop and maintain because it reflects the software design in a highly structured and detailed format.

b. Provides systematic, teachable approach for transfer of information and training to the analysis.

2.5.3.3 STRUCTURE CHART (continued)

c. Intuition is avoided.

Major disadvantages are that it does not reveal concurrent tasks and does not reflect the entities to the lowest level of an Ada package, procedure, function or task when using Ada HOL.

2.5.4 PROGRAMMING DESIGN PHASE

The programming design phase consisted of three subphases: (1) selection of program unit (module) to code, (2) program design, and (3) coding. The two concepts upon which the design is based is structured design and the application of the complexity measure described in the Ada System Designer's Guide, Appendix B.

2.5.4.1 SELECTION OF PROGRAM UNIT FOR PROGRAM DESIGN AND CODING

The selected subfunctions for coding were "Smoothing" and "Prediction", part of the Target Control package (Appendix D, Ada-Based System Design Language). These subfunctions were selected for the following reasons:

a. The subfunctions are virtually independent of the other system functions.

b. Each subfunction is further divisable into discrete program units for coding by individuals.

c. Each subfunction is easily expandable by inclusion of the "front-end" since both require the same support software.

d. Both subfunctions require non-trival mathematical manipulation.

e. Both subfunctions can be tested independent of the remainder of the system.

2.5.4.1 SELECTION OF PROGRAM UNIT FOR PROGRAM DESIGN AND CODE (continued)

f. Both subfunctions enable the use of Ada's unique features.

g. The subfunction represents functions typical of an air defense system with real-time constraints.

The program units for these subfunctions are identified as "procedure SMOOTHING" and "procedure PREDICTION" (Appendix H - Ada-Based Program Design Language).

The tracking function (Figure 2-13) consists of two sequential actions, the "front-end" which processes data received from the radar interface equipment and outputs associated track pairing to the "back-end", smoothing and prediction. The "back-end" transforms these pairings into a collection of data about a specific target. This collection of data includes the targets predicted position at the next radar sweep and the targets current corrected (smoothed) position, velocity and altitude. In order to complete the smoothing process, maneuver detection analysis must also be performed. This information is then used by the system to update the master file of current track information (track storage file). Since these update functions are actually the responsibility of the subfunction of track manager (2.3) or more specifically manage tables (2.3.2) (Appendix C - System Entity Diagram), those procedures necessary to perform the update were also included in the coding effort.

The Ada package that contains the procedures for smoothing and prediction consists of two major areas: (1) the smoothing algorithms, files and the code to manipulate them, and (2) the support routines.

a. Algorithms - the smoothing algorithms are implemented as Ada procedures or functions which are called by the control logic of the smoothing package.

b. Files - the major file involved is the track storage file. This file would be seen by the system as a package which offered file access procedures (e.g., get, put) and functions (e.g., find). The package

TITLE: 2.2.3 TPACK WHILE SCAN SYSTEM DFD		PAGE: 906
REVISION 1	AUTHOR: LWD	DATE: 5/12/82
NOTES:		ROOT: 2.0

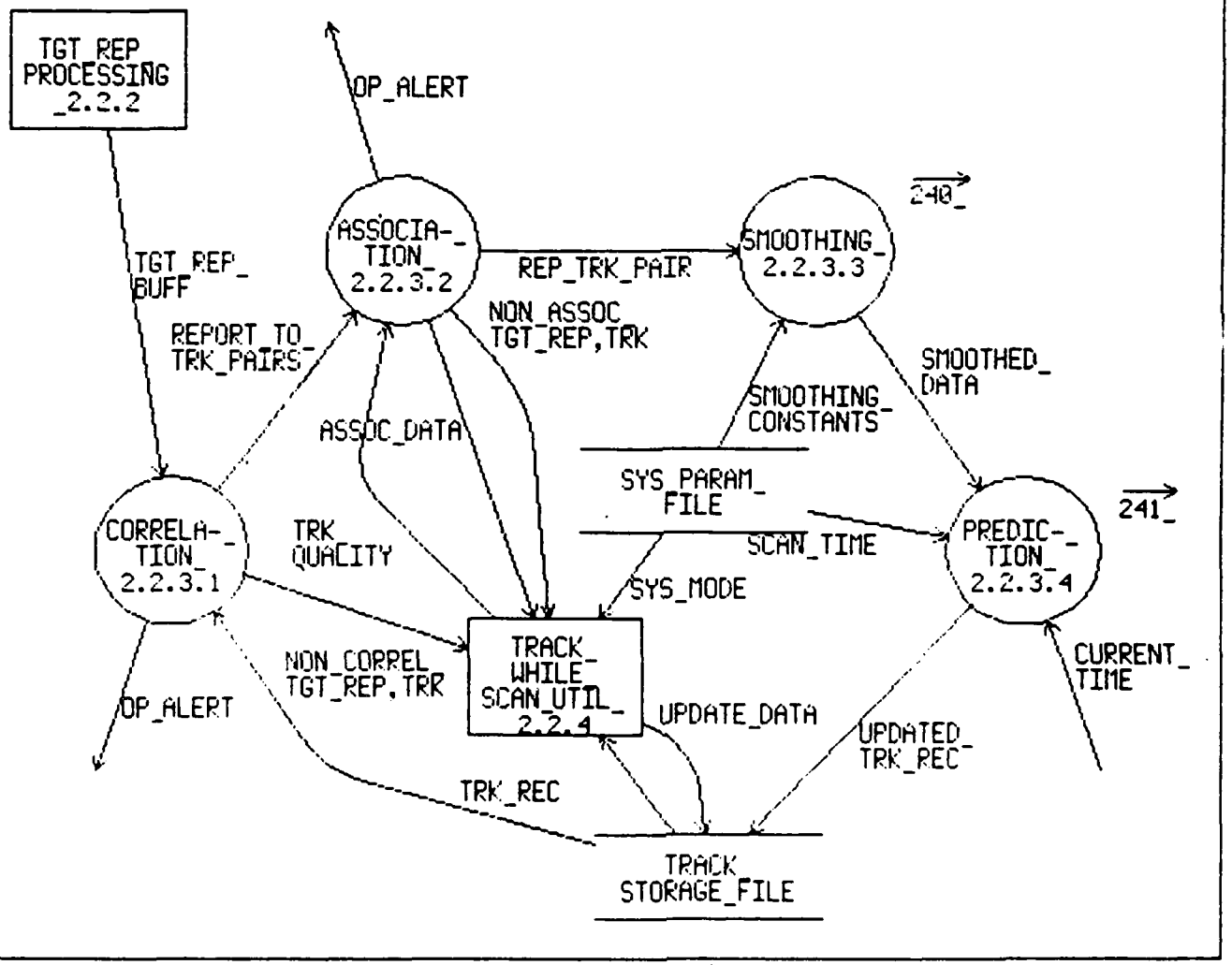


Figure 2-13 - Tracking Function

2.5.4.1 SELECTION OF PROGRAM UNIT FOR PROGRAM DESIGN AND CODE (continued)

itself contains a task that manages the actual physical representation of the file and at the same time allows queuing of access requests.

c. Code - the code that manipulates the files consists of user-defined data types, exception, and exception handlers and numerical algorithms necessary to transform the radar report into its finished form.

d. Support Routines - the support routines are comprised of: (1) a vector and arithmetic package containing overloaded definitions of standard scalar operations that will allow vector operations, and (2) trigonometric functions.

2.5.4.2 PROGRAM DESIGN

The design of the program units selected for coding is based upon the data flow diagrams, data dictionary and the structure charts. The program unit design consists of two parts - specification and the body defined in MIL-STD-1815. The following constructs were identified in Appendix B, System Designer's Guide, Design Plan, and below for use as a PDL:

- Accept
- Begin...End
- Case Is...End Case
- Do...End
- Entry
- Exception
- Exit
- Function
- For Loop...End Loop
- If Then...Else...(ELSIF)...End If
- Is
- Loop...End Loop
- Package
- Raise
- Record

2.5.4.2 PROGRAM DESIGN (continued)

- Return
- Select...End Select
- Task
- Type
- Use
- When
- While
- With

Although originally the use of an Ada subset seemed attractive for a PDL definition, this approach eventually proved to be restrictive. Therefore, when the program design was completed, it was accomplished using the full Ada language as defined in MIL-STD-1815 as the PDL. The use of PDLs was probably one of the first design tools that was created for software development. Even before the use of flow charts became widespread, programmers found that an English-like description of what was to be coded often identified potential problems prior to coding. This description process evolved to the program design language as known today, which is commonly referred to as the tool and phrase by the same name.

The PDL tool that is most widely used today is pseudo code. A pseudo code is a non-executable programming language that allows a designer/programmer to "write" sections of the program without being concerned with the details of the intended implementation language. The PDL description is intended to embody much of the control structure and algorithmetic content of the program so that these components can be verified as correct without the added complication of logically checking lower level details that do not impact upon this higher, logical level. Pseudo code enforces this separation of logic and detail by providing little or no capability for expressing detail. The use of pseudo code also enforces a structured phase design approach because, regardless of the completion of the PDL description, a final PDL-to-code phase is always required.

2.5.4.2 PROGRAM DESIGN (continued)

In approaching the problem of creating an Ada PDL, the problem was stated simply as; "In what areas of an Ada program design is the traditional pseduo code definitive?" Investigation of this question quickly revealed two things (1) traditional pseudo code could be used for "low level" Ada constructs such as simple functions and procedures, but was inadequate for describing tasks, generic packages, etc., and (2) our original estimation of the problem was grossly understated. Our second iteration was based on the premise that a pseudo code could be developed suitable for Ada design. The natural choice was Ada itself, based upon some subset of Ada that could describe these areas not described by traditional Ada code. The pseudo code would not be exact Ada "code"; therefore, separate distinct phases would be maintained. The approach had merit and was investigated further. When the Ada subset was completed and subsequently revised, the subset invariably resulted in the omission of some Ada feature that was expressly designed to describe a specific issue. Using the subset resulted in a poor and overly complicated description of features that were elegantly described by the omitted Ada features. Finally, our decision was to use the full Ada language as the PDL, but with reluctance. The use of the full Ada language offers a potentially more serious problem; there was no distinction or step between PDL and code. Every line of PDL was potentially a part of the program which the PDL was intended to describe; therefore, PDL could be viewed as coding.

Our decision implied the following:

> Ada's power of abstraction is sufficiently greater to allow the elimination of one of the abstraction tools used in the traditional design methodology, or an artificial "breakpoint" must be introduced to separate design and coding.

Both were partially true. The use of the full Ada language eliminated the final design step, but in name only. Since

2.5.4.2 PROGRAM DESIGN (continued)

the traditional coding, i.e., writing executable code now began at the PDL step, the PDL-to-code transition ceased to exist. However, the only program units completely coded, i.e., executable, were those at the highest logical levels in the structure charts. The remainder of the program design was coded only as specification sections, with studies or just comments indicating how the design should be implemented or coded. This separation of those program units "at the top" from those "lower level" program units constituted an artificial breakpoint. The levels of design which were retained through the separation are no longer obvious or easily defined. In the AN/TSQ-73 design, the specification and body identified, as a minimum, the structure chart program units in terms of execution type (packages, procedures, tasks and functions). In addition, the program design expanded upon the structure charts in areas where the structure charts could not capture all the functionality due to constraints in the Ada design language.

The program design phase used two techniques during the program design and coding phase which were McCabe's Complexity Measure and Structured Programming Techniques. Both techniques are defined in the Ada System Designer's Guide, Appendix B.

McCabe's measure was designed to constrain excessive branching within any given module. Decision statements (IF_THEN_ELSE, DO_WHILE, CASE) increase logical complexity and, therefore, must be kept within a numeric bound. This measurement was found useful for Ada procedures and functions, but will need to be enhanced for Ada tasks, packages, and strong typing capability.

2.5.4.2 PROGRAM DESIGN (continued)

The second technique was structured programming which utilized three control structure forms commonly referred to as sequence, conditional, and looping. Hierarchical indentification on the computer print-out listing also aided human readability and comprehension.

2.5.4.3 CODING

Basically, the coding phase is the completion of the body in the program design. The coding was accomplished by using the VAX/PDP 11/780 located at Fort Monmouth, New Jersey using a remote VT-100 terminal. The use of an off-site terminal and the Ada/Ed Translator was slow and time-consuming to the programmer. In addition, the translator had very poor runtime debugging facilities which added a degree of difficulty to the debugging effort. Although the use of the translator caused a large degree of frustration to the programmer, it served as an excellent learning tool.

The coding of the selected program units in Ada provided the best learning tool for gaining an in-depth knowledge of the Ada language. The programmers estimated that eighty (80) percent of the Ada language knowledge was gained during this phase of the development. The programmers concluded that Ada is not easy to use, but makes problem solving much easier.

The Ada language offers a large number of options for solving any given problem. The variety of options produce a situation in which the best method to use Ada is difficult to determine. A good example of this problem is illustrated in our choice of a method to process radar data. Control Data used a collection of tasks with a controller for the task management, but Control Data and Softech identified several other methods that were equally

viable. Experience and hardware limitations will probably determine the method for using Ada in the future.

The programmers found that Ada is a very powerful language. The language allows the implementation of complexed ideas while still thinking of the ideas in terms of abstraction. The result is that Ada tends to expand the traditional limits of programming language which was the main motivation for the PDL. The PDL worked because Ada provided both the power and abstraction capability needed at the PDL stage.

The most significant problem that was encountered during coding was the use of Ada's concept of scope. Scope is an encapsulation technique that promotes low coupling. For example, an object declared in a procedure cannot be accessed outside of the procedure. This limits the use of scope to the procedure in which it was declared. Obviously, this limitation prohibited the object from being accessed and changed from outside of the procedure. This limitation also applies to Ada program units, types and blocks. Since programmers were not used to the use of scope, they tended to violate the rules for its usage.

Ada does not force the programmer to write understandable code. Therefore, design has a profound effect on the ensuing code. The lack of a good design can result in poor code even when written in Ada. Our initial experiences indicate that the software design must consider the use of the Ada language early in the design phase.

2.5.5 VERIFICATION AND TEST

2.5.5.1 TEST SCENARIO

The Ada code was tested by providing a table of simulated "front-end" output to drive the "back-end". The output of the "back-end" will be before and after "snapshots" of the system tables, especially the track storage files, showing the changes in the files produced by a specific input.

By observing the changing track position values in the track storage file, a simulated view of the targets movements will be observed. Since the display update portion of the system uses the track storage coordinates to assemble the synthesized video portion of the display, this file contains the same information as required by the display. Testing will occur at two levels. Individual program units will be tested after separate compilation and the integrated separate parts will be tested as a unit.

2.5.5.2 PROGRAM FILES

The support package for the smoothing and prediction include files, the code to manipulate files and algorithms and the support routines. The files and support routines include:

- Track sto age files, and radar reports.
- Exception handlers and the numeric algorithms to transform radar reports in finished form.
- Any special software to execute the program.

2.5.5.3 TEST EXECUTION

The test is a single thread test as described in the test scenario above.

2.5.5.4 TEST RESULTS

The program unit tests, Appendix I, were successfully completed on 18 June 1982. The integration tests will be completed 30 June 1982.

2.6 DOCUMENTATION

This section briefly describes the documentation delivered as a part of this contract.

2.6.1 WORK PLAN

The work plan was completed and delivered to the Government on August 14, 1981. The work plan was described previously in Paragraph 2.1.

2.6.2 ADA DESIGNER'S GUIDE

The Ada Designer's Guide (Appendix B) defined the procedures to be used in the development of the software design methodology. The designer's guide included:

a. Statement of the problem.

b. Methodology rationale and overview.

c. Description of the system entity diagram and system design language.

d. Description of data flow diagrams.

e. Data dictionary.

f. Structure charts.

g. Program design language.

h. Complexity measure.

i. Structured programming.

j. Design reviews.

k. Program design methodologies.

2.6.2 ADA DESIGNER'S GUIDE (continued)

The Ada Designer's Guide was not a deliverable under the contract, but copies were provided to the Government and Adjunct Contractor. The Ada Designer's Guide is also provided as Appendix B of this report.

2.6.3 DESIGN PLAN

The design plan described the research and development effort for a twelve-month period to provide a top-level design and documentation of a large scale software system. The design plan included the following:

a. Summary.

b. Design methodology.

c. Design methodology validation.

d. Application of the design methodology to the AN/TSQ-73 Missile Minder System.

e. Technical skills acquisition.

f. Data collection and recording.

g. Schedules.

h. Description of Control Data's automated tools.

i. Software support center.

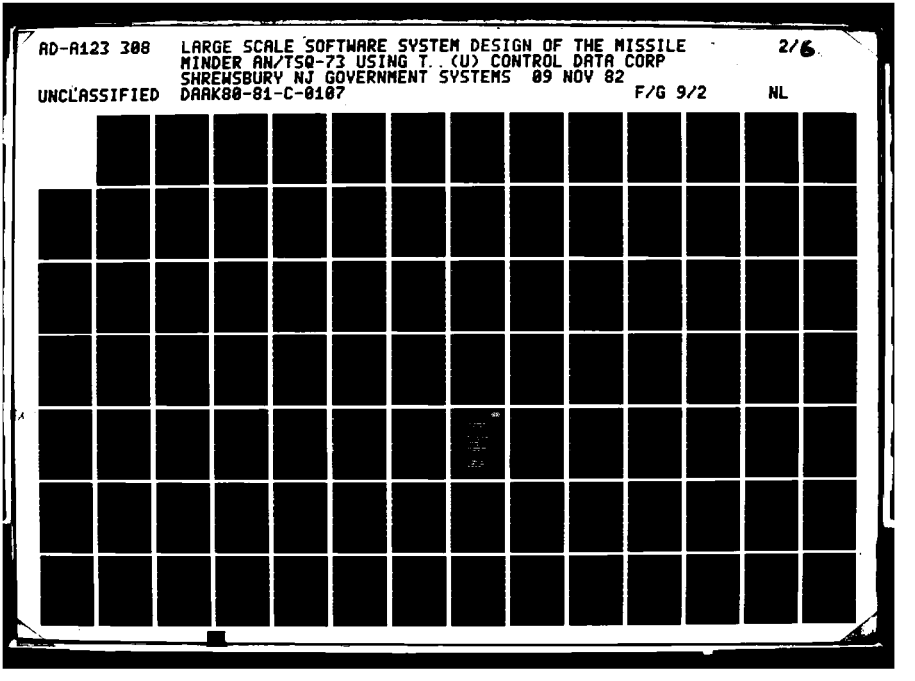
j. Ada PDL - reserved words and syntax diagrams.

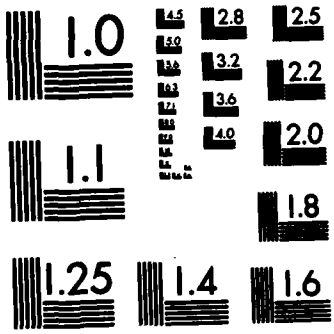
The design plan was prepared late in the contract, therefore, the design plan reflected the latest results of the "lessons learned" throughout the early days of the contract. To date, deviations from the plan have been minimized. Any deviations appear in the depth of the design required by the design plan.

The draft design plan was delivered February 25, 1982, and later revised to include additional information.

AD-A123 308 LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE MISSILE MINDER AN/TSQ-73 USING T..(U) CONTROL DATA CORP SHREWSBURY NJ GOVERNMENT SYSTEMS 09 NOV 82 2/6

UNCLASSIFIED DAAK80-81-C-0107 F/G 9/2 NL





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

2.6.4 VALIDATION REPORT

The validation report validates the proposed design methodology that was created by Control Data's Shrewsbury Facility in conjunction with the redesign of the Missile Minder AN/TSQ-73 System. The results of the validation were provided to the Government as a courtesy copy and is further detailed in Paragraph 2.4.

2.6.5 FINAL REPORT

This report documents the results of the entire contract period and includes the products developed as a part of this contract.

2.7 TECHNICAL INTERCHANGE MEETINGS

The technical interchange meetings were informal meetings between the Ada support contractor and the Adjunct Contractor (Softech) for the purpose of discussing problems, issues, the use of the Ada language, clarifying areas of MIL-STD-1815, status, and other information. The meetings, chaired by the Contracting Officer's Representative were held on the following dates and locations:

December 16, 17, 1981 - Shrewsbury, New Jersey

January 20, 21, 1982 - Shrewsbury, New Jersey

March 11, 12, 1982 - Shrewsbury, New Jersey

April 15, 16, 1982 - Shrewsbury, New Jersey

June 2, 3, 1982 - Shrewsbury, New Jersey

3.0 TECHNICAL FINDINGS

This section of the document reports on significant technical findings in four major areas. These are:

a. Design methodology utility.
b. Ada language utility.
c. Project team factors.
d. Educational considerations.

3.1 DESIGN METHODOLOGY UTILITY

The basic system design methodology is shown in Figure 3-1. This Ada-based design methodology is viewed as three distinct phases encompassing eight separate techniques:

System Entity Diagram (SED) System Design Language (SDL)	System Design Phase
Data Flow Diagram (DFD) Data Dictionary (DD) Structure Charts (SC)	Software Design Phase
Program Design Language (PDL) Complexity Measure (CM) Structured Programming (SP)	Programming Phase

As the contract required a software system design methodology, the phases after system design were oriented toward the software design of the system with the hardware design more of a mental picture than a formalized structure. The following sections report results from utilizing the system design methodology during the AN/TSQ-73 redesign.

3.1.1 SYSTEM DESIGN PHASE

The system design phase is intended to accept system performance requirements (traditionally known as system or "A" level specifications) as input and provide a logical method

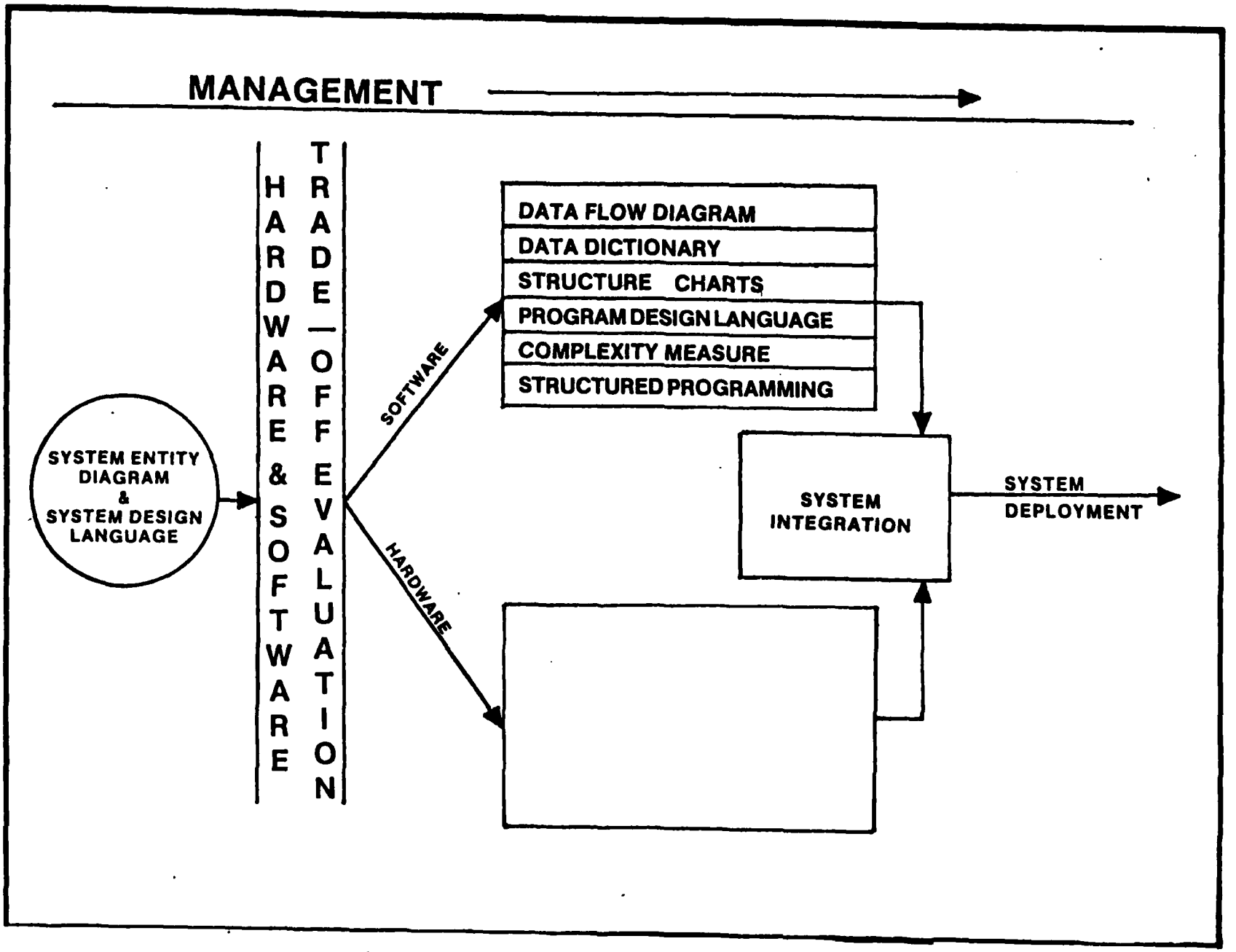


Figure 3-1 - System Design Methodology

3.1.1 SYSTEM DESIGN PHASE (continued)

by which subsystem ("B2" & "B5") level specifications may be produced from the given inputs.

The system design began with a functional requirement specification in which functionality was treated independently of ultimate hardware or software implementation. The functional requirements and subsequent functional decomposition were illustrated in graphical form by system entity diagrams which were generated for the uppermost levels of the entire redesign, (see Appendix C). Each functional "block" was hierarchically numbered in order to provide accountability, consistency, and traceability. The numbering scheme employed allowed for ease of modification and the required ripple-effect tracing.

The system entity diagram technique proved to be a valuable communication device in that it allowed several levels of functionality to be represented in a single illustration. The system entity diagram was important in that it provided a logical grouping or an overall functional grouping that was easily understood by both the user and designer. This technique was found beneficial for system changes. For example, upon a design walkthrough, the entity "OPERATOR SWITCH ACTION" component under command and control was reassigned as an entity "Track Manager" component under Target Control. The impact was minimal and the automated tool generated the new documentation. The SED does not specify interface description between functional parts of the system. The interface description is provided by the use of the complimentary SDL technique which describes these interfaces by Ada constructs.

It was originally decided to limit the degree of decomposition by the System Entity Diagram to three levels. This was

found to be inappropriate because many functional areas appeared to require further definition before hardware or software tradeoff decisions could be made. Therefore a guideline was required which was flexible enough to allow for varying degrees of decomposition. A maximum of six levels of decomposition was found to provide the flexibility needed for the redesign.

Upon completion of the system entity diagrams the system design language (SDL) technique was employed. Various schemes were tested. These included:

1. Labelling all SED blocks as procedures.
2. Proceeding as in #1 but including comments appropriate to each procedure.
3. Renaming lower level procedures to indicate execution type (serial or concurrent), packaging higher level program unit groupings which were logically related, including the numbering sequence taken from the corresponding SED, and providing Ada-based pseudo-code descriptions for each procedure or task.

As a result of the experimentation performed with the SDL, sets of objectives, characteristics and user guidelines were defined.

SDL Objectives:

The SDL must

- Be suitable for human understanding (users, developers, managers).
- Provide system input for hardware/software trade-off evaluations and succeeding design techniques.
- Provide system input for management decisions.
- Provide for system accountability.

3.1.1 SYSTEM DESIGN PHASE (continued)

- Provide for system consistency.
- Provide for ripple-effect tracing.
- Provide for increased productivity through automation.
- Provide a primary working system document.

SDL Characteristics:

The SDL must provide for specification of:

- System objectives.
- System entities, subentities, and their relationships.
- System control.
- System data flow.
- System requirements and capabilities.
- System capacities.
- System constraints.
- System exception handling.

User Guidelines:

To aid in the use of the SDL, the following guidelines are offered:

- Utilize all features of the Ada language necessary to provide the required system descriptions, while maintaining a structured approach.
- Employ the Ada program units according to the following criteria:
  - Utilize packages to represent collections logically related system activities. The items identified in any given package should be logically independent from those represented in other packages.
  - Utilize procedures and functions to represent system activities which appear to be sufficiently distinct to act as stand-alone

units to accomplish a more global activity.

- Utilize tasks to represent activities which can provide services to other program units in a concurrent fashion.
- Proceed to a level of detail which allows hardware/software trade-offs to be made, or additional clarity is warranted.
- Provide appropriate comments for each high level program unit.
- Maintain consistency with numbering conventions.
- Keep in mind that the SDL is a communicative device, the purpose of which is to describe system functionality.
- Do not attempt to write executable source code.

Development of the SDL based on the information provided by the completed SED's proved to be difficult for several reasons. The completed set of SED's caused a degree of confusion for the designer because the lower level details had already been revealed before the SDL was begun. Maintaining the thought process required to achieve consistent high levels of abstraction was hindered by the revelation of these lower level details. Therefore, it was decided to develop the SED and SDL simultaneously, with one technique proceeding no further than two levels ahead of the other at any given time.

Although the SED's were complete before the SDL was developed, the conceptual leap required to move from the SED to the SDL proved to be too large because the designer had too much information to generate for each functional "block" or entity. The execution type, the interface, the numbering sequence, a functional description, and Ada-like psuedo code had to be generated for every entity.

3.1.1 SYSTEM DESIGN PHASE (continued)

The system design phase purpose is ultimately to provide input to the hardware and software design phases. In order to provide a smooth transition between phases, the techniques comprising each phase should be compatible so that the transfer of information does not result in any deletions, additions, or modifications. The software design phase begins by utilizing the data flow diagram and data dictionary techniques. Use of the SED-SDL techniques in the system design phase did not provide a strong enough link to the software design because of the conceptual differences between both the graphical and textual techniques utilized in the different phases.

The system design phase thus was deficient in several ways. The major problems included:

- The lack of interface definitions in graphic form.
- And the poor link between the system and software design phases.

To alleviate these problems, two techniques were tried at the system design level, the system data flow diagram and the accompanying data dictionary.

The system data flow diagram illustrates system functional units (Bubbles), functional unit interfaces (Vectors) and system boundaries (Rectangles). The data dictionary entires for these items are maintained as they would be for a software data dictionary.

The system data flow diagram and data dictionary were found to illustrate linkage by providing for the graphical identification of functional interfaces, and by providing functional and interface descriptions prior to PDL generation.

```
package REMOTE COMMUNICATION (3.0) is
--  The objective of the REMOTE COMMUNICATION SYSTEM
--(3.0) is to enable the AN/TSQ-73 to communicate
--digitally with various remote sites by means of the
--previously defined military protocols, TADIL-B,
--ATDL-1, and modified MBDL.  The remote communication
--subsystem shall provide for:
--     -2 group data links (ATDL-1)
--     -2 battalion data links (ATDL-1)
--     -1 tactical operations system data link (ATDL-1)
--     -1 air traffic management system data link (ATDL-1)
--     -1 inter-service communication data link (TADIL-B)
--     -4 fire unit data links (MBDL)
end REMOTE COMMUNICATION;
```

Figure 3-2 - Ada SDL Exhibits System Objectives

3.1.1 SYSTEM DESIGN PHASE (continued)

The system design phase purpose is ultimately to provide input to the hardware and software design phases. In order to provide a smooth transition between phases, the techniques comprising each phase should be compatible so that the transfer of information does not result in any deletions, additions, or modifications. The software design phase begins by utilizing the data flow diagram and data dictionary techniques. Use of the SED-SDL techniques in the system design phase did not provide a strong enough link to the software design because of the conceptual differences between both the graphical and textual techniques utilized in the different phases.

The system design phase thus was deficient in several ways. The major problems included:

- The lack of interface definitions in graphic form.
- And the poor link between the system and software design phases.

To alleviate these problems, two techniques were tried at the system design level, the system data flow diagram (Figure 3-2) and the accompanying data dictionary.

The system data flow diagram illustrates system functional units (Bubbles), functional unit interfaces (Vectors) and system boundaries (Rectangles). The data dictionary entires for these items are maintained as they would be for a software data dictionary.

The system data flow diagram and data dictionary were found to illustrate linkage by providing for the graphical identification of functional interfaces, and by providing functional and interface descriptions prior to PDL generation.

```
package REMOTE COMMUNICATION (3.0) is
--  The objective of the REMOTE COMMUNICATION SYSTE'
--(3.0) is to enable the AN/TSQ-73 to communicate
--digitally with various remote sites by means of t
--previously defined military protocols, TADIL-B,
--ATDL-1, and modified MBDL.  The remote communica
--subsystem shall provide for:
--     -2 group data links (ATDL-1)
--     -2 battalion data links (ATDL-1)
--     -1 tactical operations system data link (ATDL-1)
--     -1 air traffic management system data link (ATDL-1)
--     -1 inter-service communication data link (TADIL-B)
--     -4 fire unit data links (MBDL)
end REMOTE COMMUNICATION;
```

Figure 3-2 - Ada SDL Exhibits System Objectives

3.1.1 SYSTEM DESIGN PHASE (continued)

The fact that the system data flow diagrams and the data dictionary can be continued in the software design phase provides a strong link between design phases.

The drawback to this configuration is that the system data flow diagrams alone do not represent a large easily comprehendable view of the system on any given illustration and they may be too complex to promote communications . This is alleviated by the continued use of the SED to provide a simple overall illustration.

3.1.2 SOFTWARE DESIGN PHASE

The software design phase followed the completion of high level system design. The distinct steps of the software design phase are:

a. Preparation of data flow diagrams.
b. Preparation of data dictionary.
c. Preparation of structure charts.

The utility of the data flow diagrams, also known as "bubble charts", in the design is that they defined graphically all the input/output data, stored data, and the processes for a software entity. This technique provided a clear and concise view of the major functions and the data exchanged between functions. The DFD is not intended to capture control elements or establish hierarchy for program units. Project members with a background of utilizing flow charts as a software design tool initially found it difficult not to impose control on data flows.

It was found also that there was not a one-to-one correspondance necessitated by the number of bubbles on the data flow and the Ada program units. The bubble for smoothing (2.2.7-Appendix E) eventually subdivided into four bubbles. The

actual code produced only one Ada program unit, namely a procedure called smoothing which incorporated the functionality of those four bubbles into it. Creating four separate units would have resulted in unnecessary overhead by causing multiple calls to access the data base. This example points out that the DFD is useful in developing a conceptual view of smoothing as having four distinct functions, while the implementation results in just one Ada program unit.

The data dictionary defined all flows, files, and transformations described by the data flow diagrams. It is utilized as a central repository for data definitions and process descriptions. It provided:

a. A glossary of terms.
b. Standard terminology.
c. Definition of data on data flow diagrams.
d. Definition of transformations on data flow diagrams.
e. Cross-referencing.
f. A bridge to program design.
g. Input error list generation.

It was found during the AN/TSQ-73 redesign effort that the data dictionary can be started at the system design phase and continued in the software design phase and programming phase. Furthermore, data described in the dictionary can easily become Ada types or objects at the code level.

The structure charts provided a hierarchical structure of modules, partitioned the software, provided data control, and showed the direction of communication between modules. Definitions of data were provided by the data dictionary. Structure charts provide the programmer with clearly established relationships between program units, directional flow of data and control of information.

3.1.2 SOFTWARE DESIGN PHASE (continued)

The structure chart was designed for sequential processing systems and the representation of tasks and other Ada features (generics, packages, etc.), in the charts is a fruitful area for further investigations.

3.1.3 PROGRAMMING PHASE

The programming phase is the last phase in the developed methodology. The steps of the programming phase are:

a. Prepare program design language.
b. Utilize complexity measure.
c. Prepare code.

The program design language was based on the Ada language constructs. The program design language provides precisely the definition of each interface; relationships of packages, procedures, tasks and functions; package and subprogram descriptions; dependencies, parameters, dependencies on other packages, target and host dependencies, if any, etc. The largest difficulty encountered was the tendency to elevate all the details of the implementation code to the PDL level. The objective of the PDL to capture the essential structure of the software was met.

The complexity measure attempted to measure the logical weight of a program unit to assist is designing manageable and maintainable software. McCabe's measure is concerned with the control flow complexity of individual program units. The measurement of data flow and concurrency (TASKING) was not meant to be addressed by this technique. The strong typing and tasking capability features of Ada are candidates for extending the metric to more fully represent the total complexity of a program unit.

3.1.3 PROGRAMMING PHASE (continued)

The final technique employed was structured programming. This technique allowed for only a limited number (three) of control concepts to be used in combination when generating software units. Structured programming eliminated GOTO statements and unnecessary branching.

3.1.4 SYSTEM DESIGN METHODOLOGY CONCLUSIONS

The system design methodology developed during this effort is applicable to the system and software design phases for embedded computers. A high degree of confidence has developed that the contructs from the Ada language can be used in the design process. Specifically, Ada can serve as both a system design language and a program design language. Sections 3.2.1 and 3.2.2 provide further discussion on Ada as a design language as well as examples from the AN/TSQ-73 redesign effort.

The use of the design methodology described in this report offers significant advantages which include:

- SED/SDL Technique for System Design Process - These design tools can be used in sequence or concurrently to produce a high level system specification.
- Automation - All techniques in the methodology can be automated. For this contract, the automated techniques were the system entity diagram, system design language, data flow diagrams, data dictionary, structure charts, and program design language. Automation allowed faster production, more efficient change, cleaner documentation and a framework for reliability checking and product visibility.
- Traceability - The methodology provided for the traceability of requirements to code. This traceability

3.1.4 SYSTEM DESIGN METHODOLOGY CONCLUSIONS (continued)

assured that (1) all requirements have been included in the design and (2) that the code (or any intermediate design documents) actually fulfills the requirements.

- Structured Design Techniques for Software Design Process The design techniques (DFD, DD, and SC) are extremely compatible for producing detailed software design.
- Ada-Based Program Design Language - Using Ada as the target language demanded that the PDL offer the module designer ways of using the considerable power which Ada offers in the areas of abstraction. An Ada derived PDL used Ada syntax to describe the Ada abstractions, but still allowed the PDL to maintain a level of abstraction above Ada code.

In addition, a number of areas were identified to enhance the system design methodology. These include the following items and will be further expanded in section 5.0.

- Further refinement of the requirements for an Ada program design language to be utilized during the programming phase of large scale software system construction.
- Development of a set of software management techniques to act as production control mechanisms.
- Development of a full complement of automated tools to achieve the maximum effective use of the system design methodology.
- Expansion of the system design methodology to include detail steps for the hardware design and fabrication process.

3.2 ADA LANGUAGE UTILITY

The Ada programming language constructs were well established in the reference manual issued by the U. S. Department of

Defense, 1980. These syntactical statements must be submitted in a rigorous and executable manner to all validated compilers. During the course of this project, a serious attempt was made to utilize the Ada constructs at two levels of abstraction above the code level. The Ada constructs were used as both a system design language and a program design language.

It should be noted that the originators of Ada envisioned it as a high powered programming language only. However, the unique features of Ada which support the general principles of information hiding and abstraction seem to invite design language experimentation.

The system design language is part of the overall system design process and acts as the initial technique before functions are allocated to hardware or software. Two stated goals for the SDL are to capture relevant information to aid the hardware/software evaluation trade-offs and to serve human communication between user and developer. The SDL was found capable for capturing system objectives, requirements, control, data flow, capacities, and constraints.

The program design language is used during the software design process. The data flow diagrams, data dictionary and structure charts serve as input to the PDL. Modules and their interfaces have been identified. The PDL establishes the detailed structure which the actual Ada code will assume.

The following sections report on the utility of the Ada language in the multiple roles of system design language, program design language, and programming language.

3.2.1 ADA AS A SYSTEM DESIGN LANGUAGE (SDL)

The AN/TSQ-73 redesign utilizing an Ada-based system design language witnessed a number of evolutionary changes in the SDL objectives, content and recommended user guidelines. The resulting SDL achieves the objectives:

a. Provide for human understanding (users, developers, managers) of system requirements.
b. Provide system information for hardware/software trade-off evaluations and succeeding design.
c. Provide system information for management decisions.
d. Provide for system accountability.
e. Provide for system consistency.
f. Provide for ripple-effect tracing.
g. Provide for increased productivity through automation.
h. Provide a primary working system document.

In order to guarantee accomplishing these objectives, the SDL provided informational content for the specification of:

a. System objectives.
b. System entities, subentities and their relationships.
c. System control, data flow and exception handling.
d. System requirements, capacities, and constraints.

The unique features of the Ada programming language allowed these desired qualities to be brought to the system level. This point is significant enough to warrant a detailed description. The following examples are excerpted from the AN/TSQ-73 redesign for illustrative purposes:

a. Ada's Capability to Exhibit System Objectives

System objectives can be made visible by successfully deploying the Ada construct of the comment along with the four basic programming units (package, task function and procedure). This point is illustrated for package Remote Communications in Figure 3-2, Page II-3-8.

3.2.1 ADA AS A SYSTEM DESIGN LANGUAGE (SDL) (continued)

b. Ada's Capability to Exhibit System Entities, Subentities and Their Relationships

Entities, subentities, and their respective relationships can be clearly expressed due to Ada's unique nesting of programming unit constructs. Packages, especially, provide a logical mechanism for grouping related system activities and maintaining a high degree of functional independence. Figure 3-3 illustrates the functional decomposition of the AN/TSQ-73 into four major areas. Remote Communications is further partitioned in the following illustrations.

c. Ada's Capability to Exhibit System Control, Data Flow, and Exception Handling

System control, system data flow, and system exception handling can be captured by the Ada control structures such as IF-THEN-ELSE, CASE and EXCEPTION. These constructs are found in the program unit's body and may be necessary for a hardware/software trade-off decision. Figure 3-4 illustrates the use of Ada constructs for exhibiting control, data flow, and exception handling.

d. Ada's Capability to Exhibit System Requirements, Capacities and Constraints

Requirements, capacities, and constraints can be captured by Ada's strong typing facility. Figure 3-5 illustrates these points.

Overall, the Ada programming language appears capable of serving as a system design language.

3.2.2 ADA AS A PROGRAM DESIGN LANGUAGE (PDL)

Historically, the use of Program Design Language has been a tool in the software design area. As described by Caine and Gordon its purpose being "for the production of structured designs in a top down form." The data currently

```
procedure TSQ-73 is
 ---
   package COMMAND AND CONTROL (1.0) is
   ---
   end COMMAND AND CONTROL;
   package TARGET CONTROL (2.0) is
   ---
   end TARGET CONTROL;
   package REMOTE COMMUNICATION (3.0) is
       ---
       package COMMUNICATION SUPERVISOR (3.1) is
       --
       --
       end;

       package INTRASYSTEM TRANSFERS (3.2) is
       --
       --
       end;
       package COMMUNICATION PROTOCOLS (3.3) is
       --
       --
       end;
   end REMOTE COMMUNICATION (3.0);
   package RADAR COMMUNICATION (4.0) is
   ---
   end RADAR COMMUNICATION;
end TSQ-73;
```

Figure 3-3 - Ada SDL Exhibits Entities, Subentities, and Their Relationships

```
procedure CONSUME INTERNAL MESSAGE (3.1.1) is
begin
  if Command and Control Message or Communication
    Protocol Message = then
    case Message Type is
      when Initialize Remote Communication =>
        Queue Message to Initialize Remote Communication (3.1.2);
      when Manage Data Links =>
        Queue Message to Manage Data Links (3.1.3);
      when Manage Messages =>
        Queue Message to Manage Messages (3.1.4);
      when others =>
        set error code for invalid message type;
        queue to produce Internal Message (3.1.7)
    end case;
  else
    set error code for invalid message originator;
    queue to produce Internal Message (3.1.7);
  end if;
end CONSUME INTERNAL MESSAGE;
```

Figure 3-4 - Ada SDL Exhibits System Control, Data Flow, and Exception Handling

```
type PROTOCOL CHOICE is (TADIL-B, ATDL-1, MBDL);
  BAUD RATE CHOICE:constant array (PROTOCOL CHOICE) of
  integer := (1200, 1200, 600);
type ORIENTATION CHOICE is (BIT ORIENT, CHAR ORIENT);
type TRANSMISSION CHOICE is (PT TO PT, MULTI DROP);
type SYNCH CHOICE is (SYNCHRONOUS, ASYNCHRONOUS);
type MODE CHOICE is (PRIMARY, REDUNDANT);

type PRI MBDL LINK SPEC is
  record
    PROTOCOL : PROTOCOL CHOICE := MBDL;
    BAUD RATE : INTEGER := BAUD RATE CHOICE (PROTOCOL);
    ORIENTATION : ORIENTATION CHOICE := CHAR ORIENT;
    TRANSMISSION : TRANSMISSION CHOICE := PT TO PT;
    SYNCH : SYNCH CHOICE := SYNCHRONOUS;
    MODE : MODE CHOICE := PRIMARY;
  end record;

type RED MBDL LINK SPEC is
  record
    PROTOCOL : PROTOCOL CHOICE := MBDL;
    BAUD RATE : INTEGER := BAUD RATE CHOICE
      (PROTOCOL);
    ORIENTATION : ORIENTATION CHOICE := CHAR ORIENT;
    TRANSMISSION : TRANSMISSION CHOICE := PT TO PT;
    SYNCH : SYNCH CHOICE := SYNCHRONOUS;
    MODE : MODE CHOICE := REDUNDANT;
  end record;

PRIMARY MBDL LINK : PRI MBDL LINK SPEC range 1...4;

REDUNDANT MBDL LINK : RED MBDL LINK SPEC range 1...4;
```

Figure 3-5 - Ada SDL Exhibits System Requirements Capacities and Constraints

available concerning increased productivity, decreased debugging efforts, and superior products comes from projects using a PDL within that scope. The PDL used during this project maintains this historical perspective.

An underlying assumption of the development of Ada as a PDL is that Ada would be the implementation language. This immediately raised the issue of identifying a distinct separation between design and code. Using the implementation language as the design language makes this a gray area.

The objective of the PDL used during this project was to create a framework for the code by:

- Identifying all procedures.
- Identifying all functions.
- Identifying all tasks.
- Identifying all packages.
- Defining the parameters of the functions and procedures.
- Defining the global data.
- Defining the logic of the control blocks.

The following guidelines were employed:

a. Program units will be designed to solve a particular problem and have logic which is easy to describe.

b. Simple solutions, designs and interfaces will be utilized.

c. Program units will be enclosed by identifiable boundaries.

d. Program units can only be referenced from other parts of the program by its name.

e. It is desireable for program units to have only a single, common entry and a single common exit. With tasks and task types this may not be possible.

3.2.2 ADA AS A PROGRAM DESIGN LANGUAGE (PDL) (continued)

f. Program units will be designed using the top-down concept.

g. Logic flow will begin at the top and flow to one bottom exit point except for "Exceptions".

h. Logically related program units will be identified for possible implementation as Ada packages.

i. The comment mechanism is utilized where information can be best transmitted for human communication purposes by English text.

The general strategy of the PDL was to define completely the higher level control blocks, in particular the task SECTOR_PROCESSING_CONTROLLER and the task type TRACK_WHILE_SCAN (Appendix H). The lower level functions and procedures called by these tasks were identified in the specification sections of packages and had only their interfaces defined. This establishes program unit identity and detailed communication interfaces to other program units. Additionally, many types were required for these interfaces, and were defined and placed in packages. By taking this approach the implementation of lower level details were left until coding time. Comments were used extensively in the PDL in order to describe how these details could be accomplished.

A major issue that arose during the development of the PDL was that of a relaxed syntax format versus Ada's exact syntax requirements. The latter was selected over the relaxed syntax for a number of reasons. Primarily, the PDL was intended to be a subset of the code and therefore exact syntactical requirement had to be met. Furthermore in keeping with the spirit of a research project, the PDL was compiled to see if the semantic errors that occured would be helpful to the design creating process. A machine processable PDL has been identified as a major issue and the only processing available, that of the NYU translator, also drove the requirement of exact syntax. Processing the PDL in this manner could be useful in identifying scope and visibility errors, locating interface inconsistencies and identifying function and procedure requiring definitions.

3.2.2 ADA AS A PROGRAM DESIGN LANGUAGE (PDL) (continued)

Using this strategy as a step prior to coding made the actual coding effort relatively simple. All interfaces were defined as well as higher level control logic leaving only the logic of lower level details. There was a tendency, however, during the writing of the PDL to slip into coding. No formal rules about wher. design would stop were defined. Without defining a formal guideline, the largest difficulty experienced was the tendency to elevate all the details of the implementation problem to the PDL level of abstraction.

Also, special care had to be taken to preserve the PDL on separate files for documentation purposes. Since the PDL was a subset of the final code, some code was created by adding to the PDL, thus gobbling up the audit trail. Also, after compilation it became necessary to locate some of the items of the package in order to solve some visibility problems and it was helpful to create some new package names to reflect these changes.

In conclusion, the program design language should first be used to complete the specification portion for each program unit. This establishes program unit identity and detailed communication interfaces to other program units. These specification sections can then be compiled before details are provided in the program unit bodies. Both the specification section and the body section carry a description of the purpose for the program unit. This information is usually available from the SDL description. The data dictionary provides type descriptions of data and process descriptions for each bubble in the data flow diagram and structure charts. This information can guide the generation of Ada code.

3.2.3 ADA AS A PROGRAMMING LANGUAGE

The Ada programming language incorporated, by technology transfer, proven constructs from a variety of research efforts. Special emphasis was placed on constructs which promote human readability, and program reliability and maintenance. These constructs include packages, generics, separate compilation sections, types, and tasks. In a larger sense, these features support encapsulation, information hiding, data abstraction, and parallel processing.

During the AN/TSQ-73 redesign effort, especially in the functional area of target control which was chosen for coding, the Ada constructs proved beneficial. The code produced appears to be easily readable and understandable as well as maintainable. The major concepts utilized and which will be explored in this section are Packages, Tasks, Task Types, Rendevous, Overloading, and Strong Typing. The code for the selected subfunction of the Missile Minder System can be found in Appendix I. The format followed below presents the concept definition and displays expamples of Ada code with a discussion of the merit of the application in the AN/TSQ-73 air defense system.

3.2.3.1 PACKAGES

Package - a program unit specifying a collection of related entities such as constants, variables, types, subprograms, and tasks.

3.2.3.1 PACKAGES (continued)

The package facility provides a major structuring capability to software system design. In addition, it provides a mechanism for information hiding implementation details from other program units. Other program units need only see the specification section and not the body where the work is accomplished. A major design issue is how many packages are available and how to group related program units within a package. Strong cohesiveness of package components will strengthen comprehension and consequently aid maintenance.

The packages utilized during the coding effort were:

a. VECTOR_OPERATIONS PACKAGE

b. TRACKING_OPERATIONS PACKAGE

c. TRACKING_DATA PACKAGE

d. TRACK_STORES PACKAGE

It was found useful to maintain strong logical cohesiveness among the components of the package. The VECTOR_OPERATIONS package, for instance, can be viewed as a mathematical library with functions related to applications area (operations on two dimensional vectors). The TRACK_STORES package contains the track storage file as well as the access to the track storage file. In addition, it provides the memory management and access management. The user, therefore, only needs to know that reading and writing to the track storage file can be

3.2.3.1 PACKAGES (continued)

accomplished by using this package. Thus, the details of memory locations or updating mechanics need not be the user's concern.

Figure 3-6 illustrates the specification or interface portion of these packages visible to outside program units desiring services for either vector, manipulation (add, subtract, multiply) or file management of track storage data.

3.2.3.2 TASKS, RENDEVOUS, TASK TYPES

Tasks - program units with the capability to operate in parallel with other program units.

Rendevous - the mechanism by which two tasks, one issuing an entry call and the other accepting the call, achieve proper synchronization.

Task Type - A specification that permits the subsequent declaration of any number of similar tasks. The task specification establishes the interface between tasks of a given type and task of a different type.

During the coding effort, a number of tasks were implemented to serve functionally distinct purposes. Tasks proved to be a valuable mechanism for more than just addressing concurrency. The tasks described in the coding phase include:

```
PACKAGE VECTOR_OPERATIONS IS
  TYPE VECT IS
   RECORD
    X:FLOAT;
    Y:FLOAT;
   END RECORD;
  FUNCTION "+" (A,B:VECT) RETURN VECT;
  FUNCTION "*" (A:FLOAT;B:VECT) RETURN VECT;
  FUNCTION "-" (A,B:VECT) RETURN VECT;
END VECTOR_OPERATIONS;

package TRACK_STORES is
  task FILE_MGR is
    entry READ_SECTOR (S : in SECTOR_NO; D : out SECTOR_DATA);
    entry WRITE_SECTOR (S : in SECTOR_NO; D : in SECTOR_DATA);
    entry READ (S : in SECTOR_NO; T : in TRACK_NO; R : out TRACK_RECORD);
    entry WRITE (S : in SECTOR_NO; T : in TRACK_NO; R : in TRACK_RECORD);
  end FILE_MGR;

  type TRACK_RECORD is
  type SECTOR_DATA is array (TRACK_NO'FIRST..TRACK_NO'LAST) of TRACK_RECORD;
  type TRACK_STORAGE_FILE is array (SECTOR'FIRST..SECTOR'LAST;
       TRACK_NO'FIRST..TRACK_NO'LAST) of TRACK_RECORD;
end TRACK_STORES;
```

Figure 3-6 - Visible Specifications of Cohesive Packages

3.2.3.2 TASKS, RENDEVOUS, TASK TYPES (continued)

a. RADAR_INTERFACE EQUIPMENT (RIE)

b. SECTOR_PROCESSOR CONTROLLER

c. TRACK_WHILE_SCAN

d. FILE_MANAGER

The TRACK_WHILE_SCAN is a task type. Figure 3-7 illustrates the relationships between the tasks in the coding effort.

The RADAR_INTERFACE EQUIPMENT task simulates the Radar Communication function (4.0) in the AN/TSQ-73. It acts as a master clock rendevousing with the SECTOR_PROCESSOR_CONTROLLER task on a periodic basis. During each rendevous, the RIE provides one sector worth of data which a portion of one sweep of the actual radar hardware would normally produce. The SECTOR_PROCESSOR_CONTROLLER task, in turn, has a rendevous with TRACK_WHILE_SCAN tasks in order to pass sector data, and stop and start the proper tasks. TRACK_WHILE_SCAN is a task type in order to generate one task for each of the twenty 18^{o} sectors described by the radar sweep. Each interaction between the SECTOR_PROCESSOR_CONTROLLER task and the TRACK_WHILE_SCAN task is started and passed its corresponding sector data. The task three sectors "ahead" is stopped in anticipation of imminent updating.

Finally, a task was used as a FILE_MANAGER mechanism. Since Ada procedures and functions are re-entrant, the possibility exists for data losing its integrity by being accessed by more than two routines. In light of the twenty TRACK_WHILE_SCAN tasks, the need for an orderly updating process of common records was a requirement.

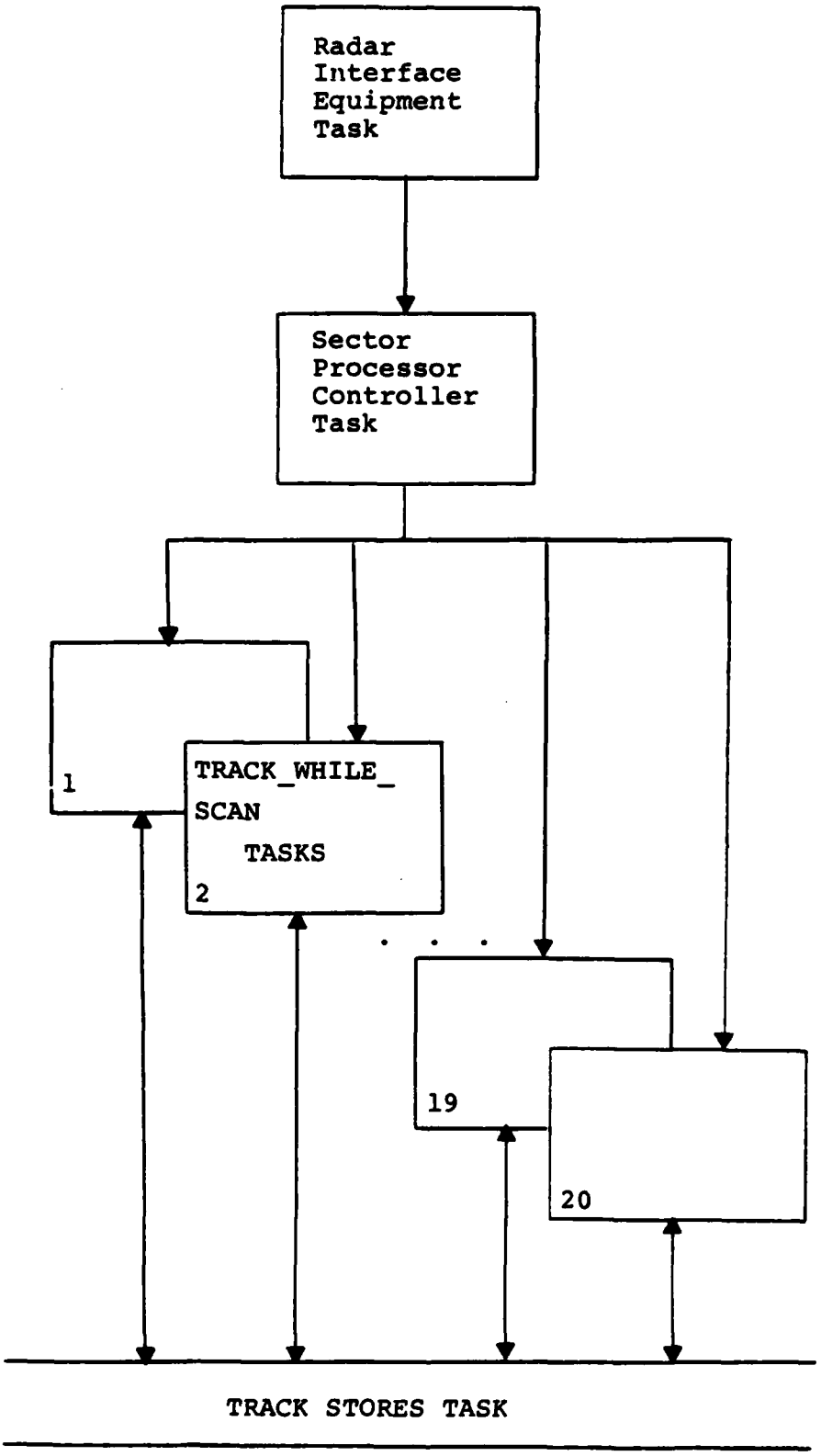


Figure 3-7 Relationship Between Tasks In Coding Effort

3.2.3.2 TASKS, RENDEVOUS, TASK TYPES (continued)

The select statement of Ada tasks provides a guard against this unfortunate occurrence by its instrinsic queueing scheme. All requests are automatically queued. The programmer is relieved of the responsibility for designing and implementing a scheduler.

3.2.3.3 OVERLOADING

Overloading - the property of literals, identifiers, and operators that can have several alternative meanings within the same scope.

Overloading is a convenient mechanism for hiding implementation details from the user. During the coding effort, the familiar operators for addition (+), multiplication (*), and subtraction (-) were overloaded to apply to vector operations. The user will find within the package VECTOR_OPERATIONS specification section three available functions to manipulate vectors:

```
FUNCTION "+" (A, B:VECT) RETURN VECT;
FUNCTION "*" (A:FLOAT;B:VECT) RETURN VECT;
FUNCTION "-" (A,B:VECT) RETURN VECT;
```

These functions free the programmer from the implementation details or mechanics which underlie sum, product, or difference. The programmer need only utilize these functions and provide data in the correct format.

3.2.3.4 STRONG TYPING

Strong Typing - all variables and expressions at compile time will be one member of a possible set of well determined types.

3.2.3.4 STRONG TYPING (continued)

The strong typing mechanism employed in Ada mandates declaration by type for all objects. This maximizes the compiler's ability to detect typing mismatches in expression evaluation and parameter passing. The advantage of the strong typing capability is to define the nature of an identifier's attributes as well as a range of values which an identifier might possess.

```
type SECTOR is NEW INTEGER range 1..20;
```

In this example, the identifier SECTOR must be an integer from one to twenty. An attempt to give SECTOR a value of twenty-one will result in an error notice at compile time.

The subtype feature of Ada was utilized to increase readability and comprehension. The subtype does not introduce a new and distinct type, but allows the operations and values of an established type to be transferred. Subtypes DEVIATION_VECT, PREDICTED_POS, SMOOTH_POS, and SMOOTH_VEL were used in PACKAGE_TRACKING_OPERATIONS. These terms are ordered pairs like VECT.

```
TYPE VECT IS
  RECORD
    X;FLOAT;
    Y;FLOAT;
  END RECORD;

SUBTYPE DEVIATION_VECT IS VECT;
SUBTYPE PREDICTED_POS IS VECT;
SUBTYPE SMOOTH_POS IS VECT:
SUBTYPE SMOOTH_VEL IS VECT:
```

3.2.4 ADA LANGUAGE REFERENCE MATERIALS

During the course of this project, various reference materials were obtained for assistance in learning the Ada language constructs and proper usage. These materials include the following books, tutorials, reference manuals, and audio-visual films. Articles may be found in the bibliography.

BOOKS:

a. Barnes, J., Programming in Ada, Prentice-Hall, Inc. 1981
b. Hibbard, P., et. al., Studies in Ada Style, Springer-Verlag, 1982
c. Ledgard, H., Ada an Introduction, Springer-Verlag, Inc., 1981.
d. Pyle, I., The Ada Programming Language, Prentice-Hall, Inc., 1982.
e. Wegner, P., Programming With Ada: An Introduction by Means of Graduated Examples, Prentice-Hall, Inc., 1979.

TUTORIALS:

a. Massachusetts Computer Associates, Low-Level Language Features, Cambridge, Mass., 1981.
b. Massachusetts Computer Associates, The Use of Ada Packages, Cambridge, Mass., 1980.
c. Massachusetts Computer Associates, Tutorial on Ada Exceptions, Cambridge, Mass., 1981.
d. Massachusetts Computer Associates, Tutorial on Ada Tasking, Volume 1: Basic Interprocess Communication, Cambridge, Mass., 1981.
e. Massachusetts Computer Associates, Type Structures, Cambridge, Mass., 1981.

3.2.4 ADA LANGUAGE REFERENCE MATERIALS (continued)

REFERENCE MANUALS:

a. U. S. Department of Defense, Reference Manual for the Ada Programming Language, Washington, D. C., July, 1980.
b. U. S. Department of Defense, Military Standard-1815. Ada Programming Language, Dec., 1980.

AUDIO-VISUAL FILMS:

a. Honeywell/Alsys, Ada Programming Course, Presented by Jean Ichbiah, John Barnes, and Robert Firth Minneapolis, Minnesota, 1980.

3.2.5 ADA LANGUAGE FINDINGS

In theory, the technology transfer which has produced the unique features of Ada promises a radical change in the traditional software development process. Features and attributes such as strong typing, packages, tasks, abstraction, and information hiding have implications beyond the programming phase. They have the capability of forging and influencing both the software design and system design phases. These powerful features force the consideration of program structure at an earlier time.

There seems certain a significant amount more to be learned about Ada's viability which will be expedited by actual experience with a validated compiler. Thus, the eventual acceptance or rejection of Ada will depend on the users. A number of points and issues were surfaced during the AN/TSQ-73 redesign effort which other Ada users will probably experience also. These include:

a. The Need for User Guidelines
   The Ada constructs are understood well enough, but knowing when to properly apply the constructs is lacking.

The reference manual does not offer guidance on when to apply; for example, a task or a procedure. There have been examples cited at the technical exchanges from the AN/TSQ-73 that could be either a task or a procedure. The conclusion is that there is a great deal to be further understood about the proper usage of Ada constructs. Premature usage before firm guidelines are established to confidently apply generics, tasks, procedures, and packages may have the opposite effect than that which is desired in the development and maintenance phases. This approach portends awkward and inefficient software structures.

b. The Need for Overhead and Timing Consideration Data

The application of programming constructs for real-time processing requires knowledge of which timing considerations are associated with the various Ada constructs. In addition, the overhead associated with tasking is necessary to confidently apply the feature.

c. The Need for Enhancing Ada as a System Programming Language

The Ada language attempts to satisfy two traditional programming environments: system and applications. The Ada language does not completely satisfy the needs of system programming, yet it offers expanded capability to the applications programmer by having system responsibilities, such as tasking. The major problem anticipated by the system programmer is the inability of Ada's high level constructs to address the real-time environment. Specifically, the writing of an operating system or kernal scheduler requires more capabilities than the Ada language provides. These capabilites include the ability to:

- terminate a task
- kill a task

- ready a task
- suspend a task
- dequeue a task

Interrupt handling requires predefined timing requirements which must be met or the event is lost. Ada does not have this capability.

d. Ada Will Require Substantially More Educational Training Than Previous Programming Languages

The project members experienced educational training in a one week formal course and access to Ada language reference materials (3.2.4). The general impressions were that learning Ada is a non-trival task. The sophistication of the Ada programming language features and uniqueness will require substantially more educational training than in previous programming languages. Both the motivation behind the concept and the proper application needs to be well understood.

It appears that the best approach is to proceed from the familiar to the unfamiliar. This could be achieved by two Ada courses. The first (40 hours) concentrating on the syntactical composition and rudimentary constructs found in familiar languages (eg. Fortran, Cobol). The second course (80 hours) would address advanced features and novel constructs of Ada. This course should include examples from military examples. The courses would be separated by some period of time where the student would have access to the Ada films, books, computer assisted courses, and a computer translator to proceed at the individual's own pace.

As with other learning endeavors, proficiency can be expected to be a function of language usage (experience).

3.3 PROJECT TEAM FACTORS

To realize the goals from the redesign, coding and documentation of a large scale software system using the Ada programming language, the selection of the project team members was thorough and detailed. The factors of level of education, experience, and past performance were scutinized to provide the required level of expertise and capability. The selected career types had to qualify individually and have a high degree of team participation. An emphasis was placed on team effort as collective expertise formulated by the project team could alleviate possible problems.

3.3.1 CAREER TYPES SELECTION

In the selection of career types, the following prerequisites were used to select project members:

- Knowledge of two or more high order programming languages.
- Coding experience.
- Familiarity with the basic concepts of large scale software systems.
- Some system design experience.
- Demonstrated communications ability.

The selection of this depth and level of experience provided the necessary levels of career types to fulfill project requirements. The following subparagraphs identify, by title, and briefly describe the career types which were utilized for this project.

3.3.1.1 CAREER TYPES (TITLES)

The title of the career types which participated on this project are:

a. Project manager.

b. Principal consultant.

3.3.1.1 CAREER TYPES (TITLES) (continued)

c. Principal programmer analyst.

d. Principal systems analyst.

e. Senior programmer analyst.

f. Programmer analyst.

g. Applications programmer.

h. Software systems programmer.

i. Associate applications analyst.

3.3.1.2 PROJECT PERSONNEL

The following subparagraphs briefly describe the individual career types which participated on this project:

Project Manager - Grade Level 38

Requires experience in the management of system design, programming projects, in-depth knowledge of system design of hardware/software, program design, coding and testing and in the preparation of reports and forms associated with major government projects.

Principal Consultant - Grade Level 38

Requires expert programming knowledge in addition to a high level of technical expertise in automated system, both hardware and software. Thorough knowledge of government standards and procedures is mandatory.

Principal Programmer Analyst - Grade Level 15

Requries seasoned experience and special study of at least one major application system. Expert capability with at least one high order programming language and a programming background sufficient to render judgements about programming problems. Also requires the ability to evaluate and write programming specifications and code.

3.3.1.2 PROJECT PERSONNEL (continued)

Principal Systems Analyst - Grade Level 15

Requires expert knowledge on one or more major application fields, including theoretical and practical aspects. Thorough familiarity with software systems and programming is required and the ability to write clear, concise system specifications which can be readily translated into programmable code is required.

Senior Programmer Analyst - Grade Level 13

Requires thorough knowledge of overall functioning of software systems and expert knowledge of at least one application system, to the subsystem level. Must have expert knowledge of at least one high order programming languages and advanced familiarity with computer operations.

Programmer Analyst - Grade Level 11

Requires advanced capability with at least one high order language and a thorough to advanced knowledge of computer operations. Also requires general to thorough understanding of operating systems including utility routines, subroutines, and assembly language usage.

Applications Programmer - Grade Level 11

Requires advanced programming skills in at least one language plus thorough capability in at least one additional languages. Also requires thorough knowledge of various software systems features and routines and must be knowledgeable in peripheral hardware.

Software Systems Programmer - Grade Level 11

Requires advanced programming ability in two languages and working knowledge of various software systems features and routine. Advanced experience with operating systems and subroutines is also required for this position.

3.3.1.2 PROJECT PERSONNEL (continued)

Associate Applications Analyst - Grade Level 9

Requires general knowledge of one major area of application and general programming and computer operations skills including thorough systems analysis skills. Thorough knowledge of at least one high order programming language.

3.3.1.3 PROJECT FUNCTIONS

The career types listed below performed the described functions in the project:

a. Project Manager - The responsibility for overall project integrity was vested with the project manager. Specific tasks included submission of required project status and accounting reports, coordination and assignment of resources, final review authority, and principal interface to the government organization.

b. Prinicipal Consultant - Provided preliminary data on design techniques and submitted research data in support of deliverable products; design plan, final report. Provided analysis services for design methodology validation and source code review.

c. Principal Systems Analyst - As the technical focal point for the project coordinated individual tasks and assignments, reviewed inputs for completeness and technical accuracy, was primarily responsible for the design plan and final report, and was the principle designer for the development of the proposed design methodology with direct participation in data assessment, design and analysis techniques and technical validation of project findings.

d. Principal Programmer Analyst - Was the lead participant in the design methodology validation task. Organized and directed the program coding effort to ensure delivery to the government of the source code for the designated subfunction of the Missile Minder System. Provided inputs for both the design plan and final report.

3.3.1.3 PROJECT FUNCTIONS (continued)

e. Senior Programmer Analyst - Involved in the development of the design methodology, with primary inputs for the design techniques of system entity diagrams, system design language and structure charts. Was a participant in the program coding effort.

f. Programmer Analyst - Primarily dedicated to the coding effort. Was the recognized project authority on Ada programming and was responsible in the development of the source code for the selected subfunction of the Missile Minder system. Participated in the validation effort of the Ada system design methodology.

g. Applications Programmer - Was a major contributor in the development of the system design methodology. Provided significant inputs to the design plan and final report. Worked with the programmer analyst in the development of the source code for the selected subfunction of the Missile Minder system.

h. Software Systems Programmer - Performed in-depth research and analysis into the design techniques as expressed in the design plan. Participated in the validation effort of the Ada system design methodology. Provided inputs for both the design plan and final report.

i. Associate Applications Analyst - Responsible for all data collection and recording. Provided computer aided design documentation for the design plan and final report. Was a participant in the validation effort for the Ada system design methodology.

3.3.1.4 ADEQUACY OF PERSONNEL

The selection of project personnel was derived from two categories - professional discipline and project peculiar requirements. The professional disciplines were identified to provide the best cross section of vertical and horizontal expertise in satisfaction of typical programming projects. The category of

3.3.1.4 ADEQUACY OF PERSONNEL (continued)

project peculiar materialized subsequent to careful and in-depth analysis of the identified individual's actual experience and on-the-job performance. The major factors that were considered included:

- Programming experience, based upon actual coding and results achieved.
- Degree of familiarity with software systems.
- Proven ability to perform original design work.
- Demonstrated technical writing ability.
- Performance as "team" player.
- Exposure to the Ada programming language.

There was a recognized overlap of functions, but this was by design rather than exception. The individuals originally selected had to possess the quality and flexibility to permit such action. Here surfaces the exception rather than the rule. Typically defined career patterns would not permit such a luxuary, however, the uniqueness of this project directed the establishment of such a requirement.

Project manning of similar efforts would not require this degree of flexibility, however, when into the area of "new" territory it is not unrealistic to formulate such actions.

It was concluded that the adequacy of the personnel did meet project requirements and no major voids of expertise were identified, however, the following findings were considered pertinent:

a. Top-level design work should be assigned higher corporate level personnel, particularly when dealing with Ada as a programming language.

b. The Ada language requires a better than average programmer.

c. In-depth knowledge of Government systems and familiarity with the use of Government documentation is strongly recommended.

3.4 INSTRUCTIONAL ISSUES

Three elements of the project were identified to require formalized instruction. They were design methodology (Structured Analysis/Structured Design), Ada Programming Language, and Missile Minder System (AN/TSQ-73) Indoctrination. From this base, a level of understanding developed from which all project members increased their degree of expertise. Informal training, which was primarily derived from books, technical journals and training aids such as video tapes, supported the formalized instruction. All instructional issues had the underlining objective of providing a uniform knowledge base, enhancement of design principles and increased Ada programming skills.

3.4.1 INSTRUCTIONAL CURRICULUM

All formalized instruction followed typical lesson plans and course outlines. The length of instruction for each was forty hours with the exception of the Missile Minder Indoctrination which was twenty-four hours. The forty hour courses were restricted to twenty-four hours of stand-up instruction with the remaining sixteen hours allocated to workshop sessions and small group discussions. Reinforcement documentation was available for all three subjects to facilitate problem solutions and research efforts. Formalized instruction provided a standardized base of expertise, although it was by no means all inclusive. Extensive research and supplementary reading were required by each individual to obtain more information and knowledge on specific subjects. This approach allowed each team member to supplement his/her knowledge for particular needs.

3.4.1.1 DESIGN METHODOLOGY

A design approach and documentation was studies from a Structured Analysis/Structured Design course presented by a Control Data Consultant from the Corporate's Consulting and Educational Services Division. Textbook

3.4.1.1 DESIGN METHODOLOGY (continued)

utilized was Structured Analysis by Victor Weinberg supplemented by a Control Data Corporate-developed workbook titled Structured Analysis and Structured Design Syllabus. The course provided project members with the disciplines required to evolve the current design methodology.

3.4.1.2 ADA PROGRAMMING LANGUAGE

The formalized course of instruction for Ada programming was locally developed and presented by a senior level team member who had extensive exposure to the Ada programming language including practical experience in early research and development efforts. The objective of the course was to introduce the Ada programming language to project members from which they could build a working knowledge of the language. Supportive documentation for the course included:

a. The Ada Programming Language by I.C. Pyle.

b. Programming With Ada: An Introduction by Means of Graduated Examples by P. Wegner.

3.4.1.3 MISSILE MINDER INDOCTRINATION

This course of instruction was structured from information contained in the development specification MI-PD19501, FM 44-70 (Field Manual for all Air Defense Artillery Command and Control Systems, AN/TSQ-73), and the ACN 18006 (Interop III, A-Technical Feasibility).

The course was presented by a Control Data Consultant familiar with the Missile Minder System. Subsequent to this instruction, project members had a thorough working knowledge of system capabilities, hardware/software configurations, and functional composition.

3.4.1.4 INSTRUCTIONAL ADEQUACIES

As previously stated, the formalized instruction provided an introductory base for the three major elements of the project - design methodology, Ada programming language, and the Missile Minder System (AN/TSQ-73). The very concept of training, however, is never ending and for any project and/or program must be regarded as a very large function of the effort. The paramount objective must be to raise the level of competence and understanding of every individual in the organization and to this end the training must be dynamic and continuous. From the viewpoint of this particular project the following opinions are offered:

- Design methodology - formalized training was considered adequate, however, to adequately train individuals to the level of implementation of the various techniques, the amount of instructional workshop sessions should be increased. Restructure of the courses schedule of content would be expanded to sixty (60) hours versus the current forty (40). Also as the project progressed four to eight (4 to 8) hour refresher courses to intensify individual levels of competence would be beneficial.
- Ada programming language - thorough knowledge of either PASCAL, PL/1, ALGOL or any other block structured language is a definite asset. In retrospect, the formalized course of instruction of only forty hours was considered insufficient and only through extensive reinforcement knowledge and research were project members able to fully comprehend the full concepts of this programming language. Once ag in, in retrospect a formalized course of eighty hours with emphasis placed on hands-on programming would have served the project goals better. Continuous programming exposure must be afforded all concerned individuals throughout their association with Ada as it is only through actual hands-on training will increase levels of competence be realized. Additional adequate reference

3.4.1.4 INSTRUCTIONAL ADEQUACIES (continued)

materials must be available such as MIL-STD-1815, however, it was found that Programming in Ada by J. G. P. Barnes was the most informative and beneficial.

- Missile Minder Indoctrination - the instructional period should be expanded to eighty (80) hours. Subsequent to the formalized instruction project members were introduced to the system, however, the level of competence required to fully appreciate the functions and concepts of the system requires more time. Extensive reinforcement reading is mandatory for this subject and adequate and detailed reference materials must be made available for all concerned individuals. The reference materials recommended are stated in Paragraph 3.4.1.3.

4.0 CONCLUSIONS

The Ada language features proved beneficial for coding a typical military application, namely a section of the target control function from the An/TSQ-73 (Missile Minder). The more sophisticated features such as packages, tasks, task types, overloading, and strong typing found efficient application and provided significant structuring capability. The task types, for instance, were a novel mechanism for allowing one task for each of the twenty sectors processed by a typical radar sweep. It is strongly recommended, however, that to guard against possible misuse and assure proper usage of the powerful constructs, the user should have thorough educational training and a set of clear guidelines for applying the Ada constructs. This will require more application experiments to further understand the implications inherent in the usage of this syntactically rich language.

The Ada language is a large, syntactically rich language and consequently learning Ada is a non-trival task. This becomes a more serious concern when one considers that the eventual acceptance of the language depends on the users. It seems obvious that Ada will require substantially more educational training than devoted to previous programming languages. Both the motivation behind the concepts as well as the proper application needs to be well understood.

Teaching Ada will require substantially more Ada training than was devoted to traditional languages. This will require a number of formal courses as well as access to an Ada translator/compiler, self paced instruction modes, and audio-visual media. It is envisioned that a series of Ada courses will be necessary for individuals to adroitly manipulate the powerful features.

The Ada language has demonstrated the capability for serving multiple roles (system design language, program design

4.0 CONCLUSIONS (continued)

language, implementation language), and can support a cohesive software system design methodology. During the AN/TSQ-73 redesign effort, such a methodology was developed. This methodology encompasses a series of techniques for the system design phase, software design phase, and programming phase. The strength of this methodology was that the techniques employed were both graphical and textual, utilized a number of compatible structured design techniques, were mostly automated, and utilized Ada in each phase.

The methodology's techniques at the system design phase experienced the most evolution. The original methodology was composed of a system entity diagram and a system design language. The SED provided a logical grouping of functions in a graphic form, earily readable by both the users, analysts and programmers. The SDL captured information including system objectives, requirements, capacities, and constraints in Ada format. Two additional techniques were later added including a system data flow diagram and system data dictionary. This was not entirely unexpected since addressing system design independent of hardware/software implementation is such a new approach. These additional techniques enhance the linkage to the software design phase techniques, especially the data flow diagram and data dictionary.

Although the hardware/software trade-off analysis was limited during this project, Control Data envisions an iterative process between the system level techniques and trade-off analysis until requirements are adequately developed.

The system design language captured all the functions necessary to define the system. Based on the guidelines described in MIL-STD-490, an Ada-based SDL is a viable vehicle for capturing broad requirements and detail design. The design

specifications appeared easier to comprehend than traditional A-level, B-level, and C-level specifications. The system design language possesses a logical structure, sequence, and non-redundancy not found in these documents.

As the contract required a software design methodology, the subsequent phases after system design were oriented towards the software design of the system with the hardware design more of a mental picture than a formalized structure. As such, before a unified system design methodology could be firmly established, for both hardware design and software design, techniques for the hardware design and fabrication phases need to be established. This effort would most likely further define the techniques necessary at the system design stage. It is clear from the AN/TSQ-73 redesign effort, that such a unified system design methodology must make maximum use of automated techniques lest the sheer documentation becomes overwhelming.

The techniques composing the software design phase (data flow diagrams, data dictionary, and structure charts) were proven quite adequate and cohesive for their mission. The structure charts proved an excellent vehicle for transitioning to the programming phase's program design language. Although a number of issues have been addressed under this contract concerning Ada's role as a PDL, this remains a promising area for further research. There exists a need to finalize the requirements which will define an Ada program design language (PDL), and how these requirements can be realized. Finally, from a methodology viewpoint, there remains a good deal more to be understood about the impact of the powerful Ada language on methods of design. Further application efforts on military system's requirements appears to be the viable vehicle.

4.0 CONCLUSIONS (continued)

The Ada language clearly has the capability to impact more than the implementation phase of the software system life-cycle. This effect is because the language embodies features which support the principles of encapsulation, information hiding, data abstraction, and parallel processing. These qualities are actually design issues and naturally invite experimentation for utilizing Ada as a design language. Ada was applied as a system design language and found capable of providing informational content for the specification of:

a. System objectives.

b. System entities, subentities and their relationships.

c. System control, data flow and exception handling.

d. System requirements, capacities, and constraints.

This was accomplished at a global logical level before hardware and software assignments were made.

Ada was applied as a program design language and found capable of creating a framework for the code and supporting the following objectives:

a. Identifying all Ada program units (PACKAGE, TASK, PROCEDURE, FUNCTION).

b. Definition of logic for the control structures.

c. Parameter definition for procedures and functions.

d. Global data definitions.

5.0 RECOMMENDATIONS

The automated tools used to support the software design methodology were based upon the use of software design tools. However, the existing automated tools that were used in the application of the methodology were adopted and adapted because a full complement of automated tools did not exist. The limited use of automated tools during the design of the AN/TSQ-73 convincingly demonstrated the need for further development of software design tools for the entire design.

The program design and coding phase consisted of two subfunctions - Smoothing and Prediction. These two subfunctions are a part of the larger sequence (Tracking) which must be completed during the radar scan. Since the operation of the AN/TSQ-73 is time critical, the timing aspects should be examined in greater detail to determine the capability of programs written in Ada to meet critical timing requirements.

The system design methodology lends itself to a single baseline that can be generated and maintained through the use of automated design tools. The design methodology, based upon Ada as a system design language, offered significant advantages:

a. Captures all the requirements in a single baseline.

b. Avoids duplication of requirements.

c. Can be automated to include maintenance and distribution of changes.

d. Permits early hardware/software trade-off decisions.

The system and software design methodologies presented herein provide a viable means for system software development and should be utilized as one of the Government's design methodologies; therefore, it is recommended that the design methodology be accepted. To further enhance the design process, the following actions are recommended:

5.0 RECOMMENDATIONS (continued)

a. An automated software design system for the total design process be developed based upon Ada as a design language.

b. The entire tracking sequence be programmed in Ada and results evaluated against a set of representative target reports to determine adequacy of Ada in a time dependent system.

c. A users guide be developed as a part of or as an adjunct to MIL-STD-1815.

d. The system design process, based upon Ada, be expanded to include the hardware design including simulation/ modelling, and computer aided design.

e. The configuration management process be re-evaluated to permit development of the specification as a single baseline based upon the Ada language.

f. A set of software management techniques be developed to act as production control mechanisms.

Since learning Ada will require substantially more Ada training than was previously devoted to programming languages, it is recommended that a graduated Ada curriculum be developed. A minimum of two forty (40) hour courses covering basic and advanced topics is envisioned. This educational area deserves more attention from researchers.

APPENDIX A

REFERENCES

APPENDIX A

REFERENCES

FINAL REPORT

LARGE SCALE SOFTWARE SYSTEM DESIGN
FOR THE
MISSILE MINDER AN/TSQ-73
USING
THE ADA PROGRAMMING LANGUAGE

PREPARED FOR

U.S. ARMY COMMUNICATIONS ELECTRONIC COMMAND
FORT MONMOUTH, NEW JERSEY 07703

REFERENCES

1. Department of Defense, Reference Manual for the Ada Programming Language, July, 1980.

2. Fisher, D.A., "DOD's Common Programming Language Effort", IEEE pp. 24-33, July 1978.

3. Boehm, B.W., "Keynote Address - The High Cost of Software", Proceedings of a Symposium on the High Cost of Software, Menlo Park, California, Stanford Research Institute, pp. 27-40, 1973.

4. Stephan, D.G., et. al., "DOD Digital Data Processing Study - A Ten-Year Forecast", EIA Fall Symposium, October, 1980.

5. Department of Defense Directive, Number 5000.29, April, 1976.

6. Department of Defense Directive, Number 5000.31, November, 1976.

7. Myers, G.J., Software Reliability, John Wiley & Sons, Inc., 1976.

8. Glass, R.L., Software Reliability Guidebook, Prentice Hall, 1979.

9. Carlson, W.E., Druffel, L.E., Fisher, D.A., Whitaker, W.A., "Introducing Ada", Proc. ACM Annual Conference, pp. 263-271, October, 1980.

10. Department of Defense, "Steelman, Requirements for High Order Computer Programming Languages", June, 1978.

11. Department of Defense, "Stoneman, Requirements for Ada Programming Support Environments", February, 1980.

12. Wegner, P., Programming with Ada: An Introduction by Means of Graduated Examples, Prentice Hall, 1980.

13. Wegner, P., "The Ada Language and Environment", Technical Report CA-56, Department of Computer Science, Brown University, May, 1980.

14. Pyle, I.C., The Ada Programming Language, Prentice Hall, 1981.

15. Hibbard, P., et. al., Studies in Ada Style, Springer-Verlag, 1981.

16. Ledgard, H.F., Ada, An Introduction, Springer-Verlag, 1981.

17. Control Data Corporation, "Impact of the Ada Programming Language on the Software Components of Interoperability", Technical Report Submitted to U.S. Army, February, 1981.

18. Walsh, T.J., Texel, P.P., "Ada "+" Style:= Reliability", Control Data PSI Excerpts, 1981.

19. Boehm, B.W., "Software and Its Impact: A Quanitative Assessment", Datamation, Vol. 19, Number 5, 1973.

20. Boehm, B.W., "Software Engineering", IEEE Transactions on Computers, Vol. C-25, No. 12, 1976.

21. Mills, H.D., "Software Engineering", Science, Vol. 195, 1977.

22. Dijkstra, E.W., "Complexity Controlled by Hierarchical Ordering of Function and Variability", in P. Naur and B. Randell, Eds., Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Brussels, Belgium, NATO Scientific Affairs Division, pp. 181-185, 1968.

23. Dijkstra, E.W., "The Structure of the THE Multiprogramming System, Communictions of the ACM", pp. 341-346, Vol. 11, 1968.

24. Wirth, N., "Program Development by Stepwise Refinement", CACM, Vol. 14, No. 4, 1971.

25. Liskov, B.H., "A Design Methodology for Reliable Software Systems", Proceedings of the 1972 Fall Joint Computer Conference, AFIPS Press, pp. 191-199, 1972.

26. Parnas, D.L., "On the Criteria to be Used in Decomposing Systems into Modules", Communications of the ACM, Vol. 15, pp. 1053-1058, 1972.

27. Shneiderman, B., Software Psychology: Human Factors in Computer and Information Systems, Winthrop Publishers, 1980.

28. Miller, G.A., "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", Psychological Review, Vol. 63, pp. 81-97, 1956.

29. Newman, W.M., Sprould, R.F., Principles of Interactive Computer Graphics, McGraw-Hill, 1979.

30. Weinberg, G.M., The Psychology of Computer Programming, Van Nostrand Reinhold, 1971.

31. Mills, H.D., Linger, R.C., Witt, B.I., Structured Programming, Addison-Wesley, 1979.

32. Dijkstra, E.W., "On a Methodology of Design", MC-25, Information Symposium, Mathematical Centre Traits, 1971.

33. Whitehead, A.N., An Introduction to Mathematics, Oxford University Press, 1948.

34. Russell, B., An Introduction to Mathematical Philosophy, Simon and Schuster, 1981.

35. Rein Turn, "Hardware-Software Trade-Offs in Reliable Software Development", 11th Asilomar Conference on Circuits, Systems, and Computers", 1977.

36. Jensen, R.W., Tonies, C. C., Software Engineering, Prentice Hall, 1979.

37. McCabe, T.J., "A Complexity Measure", Software Engineering, Vol. SE-2, Number 4, pp. 308-320, 1976.

38. Bohm, C., Jacopini, G., "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules", CACM, Vol. 9, pp. 366-371, 1966.

39. Yourdon, E., Constantine, L.L., Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, 2nd ed., Yourdon Press, 1978.

40. Weinberg, V., Structured Analysis, Prentice Hall, Inc., 1980.

41. Control Data Corporation, Structured Analysis and Structured Design Syllabus and Workbook, 1981.

42. Yourdon, E., How to Manage Structured Programming, Yourdon Press, Inc., 1976.

43. Caine, S.H., Gordon, E.K., "PDL - A Tool for Software Design", AFIPS Conference Proceedings, Vol. 44, National Computer Conference, pp. 271-276, 1975.

44. Sammet, J.E., Waugh, D.W., Reiter, R.W., "PDL/Ada -- A Design Language Based on Ada", ACM '81, November, 1981.

45. Mills, H.D., "Mathematical Foundations for Structured Programming", Federal Systems Division, IBM Corporation, Gaithersburg, Maryland, FSC 72-6012, 1972.

46. Perlis, A.J., Sayward, F.G., Shaw, M., Software Metrics: An Analysis and Evaluation, The MIT Press, 1981.

47. Hanson, W.J., "Measurement of Program Complexity by the Pair Cyclomatic Number, Operator Count", SIGPLAN Notices, Vol. 13, No. 3, pp. 29-33, March, 1978.

48. Myers, G.J., "An Extension to the Cyclomatic Measure of Program Complexity", SIGPLAN Notices, Vol. 12, No. 10, pp. 62-64, October, 1977.

49. Woodward, M.R., Hennell, M.A., Hedley, D., "A Measure of Control Flow Complexity in Program Test", Software Engineering, Vol. SE-5, No. 1, pp. 45-50, January, 1979.

50. Schneidewind, N.F., Hoffman, H.M., "Experiments in Software Error Data Collection and Analysis", Proceedings of the Sixth Texas Conference on Computing, University of Texas at Austin, pp. 4A1-4A12, November, 1977.

51. Green, T.F., Schneidewind, Howard, G.T., Pariseau, R., "Program Structures, Complexity and Error Characteristics", Computer Software Engineering, Polytechnic Institute of New York, pp. 139-154, 1976.

52. Schneidewind, N.F., "Software Metrics for Aiding Program Development and Debugging", Proceedings of the 1979 NCC, AFIPS Press, 1979.

53. Sunohara, T., et. al., "Program Complexity Measure for Software Development Management", Fifth International Conference on Software Engineering, 1981.

54. Walsh, T.J., "A Software Reliability Study Using a Complexity Measure", Proceedings of the 1979 NCC, AFIPS Press, 1979.

55. Jackson, M.A., Principles of Program Design, Academic Press, 1975.

56. Jackson, M.A., "Data Structure as a Basis for Program Design", Proc. INFO 74 Conf., Infotech, 1974.

57. Orr, K.T., Structured Systems Development, Yourdon Press, 1977.

58. Orr, K.T., Structured Requirements Definition, Ken Orr and Associates, 1981.

59. Warnier, J.D., Logical Construction of Systems, Von Nostrand Rheinhold, Co., 1981.

60. Ross, D.T., "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Transactions on Software Engineering, Vol SE-3, No. 1, 1977.

61. Teichroew, D., Hershey, E.A., "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions on Software Engineering, 1977.

62. Hamilton, M., Zeldin, S., "Higher Order Software - A Methodology for Defining Software", IEEE Transactions on Software Engineering, Vol. SE-2, No. 1, pp. 9-32, March, 1976.

63. Hamilton, M., Zeldin, S., "The Relationship Between Design and Verification", The Journal of Systems and Software, Vol. 1, pp. 29-56, 1979.

64. Davis, C.G., Vick, C.R., "The Software Development Systems", IEEE Transactions on Software Engineering, January, 1977.

APPENDIX B

ADA SYSTEM DESIGNER'S GUIDE

APPENDIX B

ADA SYSTEM DESIGNER'S GUIDE

FINAL REPORT

LARGE SCALE SOFTWARE SYSTEM DESIGN FOR THE MISSILE MINDER AN/TSQ-73 USING THE ADA PROGRAMMING LANGUAGE

PREPARED FOR

U.S. ARMY COMMUNICATIONS ELECTRONICS COMMAND

FORT MONMOUTH, NEW JERSEY 07703

ADA SYSTEM DESIGNER'S GUIDE

DEVELOPED FOR

LARGE SCALE SOFTWARE SYSTEM DESIGN

OF THE

MISSILE MINDER AN/TSQ-73

USING

THE ADA PROGRAMMING LANGUAGE

Contract Number DAAK80-81-C-0107

GOVERNMENT SYSTEMS

40 AVENUE AT THE COMMON

SHREWSBURY, NEW JERSEY 07701

(201) 542-9222

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS

INTRODUCTION

The Ada programming language is a significant recent effort initiated by the Department of Defense (DOD) to establish a common High Order Language (HOL) to use on embedded tactical military systems. It has been designed with the intent of facilitating a marked improvement in the performance of embedded computer systems and addressing software deficiencies which have been identified in numerous studies as common impediments to such systems.

In addition to the development of a high powered common programming language (Ada); the need for compatible software system design methods has also been recognized. Currently, no system has been designed or coded using the Ada language. Yet, to effectively evaluate and analyze the full impact of the Ada programming language relative to software system design, an actual system design must be attempted. Therefore, the U.S. Army at Fort Monmouth (specifically the Center for Tactical Computer Systems (CENTACS)), has initiated contractual research and development design efforts to acquire information concerning large scale software system design related to utilizing the Ada language efficiently.

Associated with various Ada language activities, Software Technology Division, CENTACS, is seeking to establish software design methods which are compatible with the Ada language. In addition to the design issues expected to surface during this effort, information is being sought to develop a total educational training program for various skill level personnel to effectively employ the Ada language.

The Ada R&D redesign efforts will provide Ada system design experience for use in developing a curriculum and materials for training in using Ada as a system/program design language and as an implementation language. This information should result from the combination of a design methodology analysis and then applying the results to a large scale software system design.

The objective of this designer's guide is to present a description of the design methodology to be utilized by Control Data Corporation project personnel for the software system design of the AN/TSQ-73 Missile Minder using the Ada programming language under contract number DAAK80-81-C-0107. This document was developed to explain the rationale for selection and delineation of the methods and standardized usage by the project team members. Additional information, especially concerning design issues encountered and lessons learned on this project will be delivered in the final report (contract CDRL C002). The methodology goal is to establish a framework of Ada compatible techniques which are linked to and facilitate the evolution of software exhibiting high reliability and subsequently increased maintainability. As a result of the application of these techniques during the redesign of the TSQ-73 system using Ada, it is anticipated that recommendations will emerge to enhance the methodology contained within this document.

The guide is organized into the following sections:

- Introduction
- Statement of the Military's Problem
- Rationale for Software System Design Methodology
- Software System Design Methodology Overview
- The Software System Design Methodology Components
- System Entity Diagram
- System Design Language
- Data Flow Diagrams
- Data Dictionary
- Structure Charts
- Program Design Language
- Program Complexity Measure
- Structured Programming
- Design Reviews and Walkthroughs
- Summary
- Appendix A (Program Design Methodology Reviews)
- Appendix B (References)

STATEMENT OF THE MILITARY'S SOFTWARE PROBLEM

The two central problems with the tactical software component of military computer embedded systems are the same as with software in general - cost and reliability. However, since the Department of Defense is by far the largest single consumer of software and related products, the problems are especially large and complex. The following facts are from recent DOD and Army studies.

1. Studies conducted in 1973 and 1974 give conservative estimates for lower bounds on DOD software costs. Figure 1 indicates the breakdown of the estimated $3 billion annual DOD software costs. More than one-half of these costs are associated with embedded computer systems written in five hundred (500) general purpose languages. Embedded computer software often exhibits characteristics that are strikingly different from those of other computer applications. The programs are frequently large (50,000 to 100,000 lines of code) and long-lived (10 to 15 years). Personnel turnover is rapid, typically two years. Change is continuous because of evolving system requirements, and annual revisions are often of the same magnitude as the original development.

2. The majority of military costs spent on computer systems is in the software arena and this amount is growing in proportion to other computer costs. Figure 2 clearly indicates that the rise in software development costs is as significant as the reduction in hardware costs. Notice also the rise in the proportion of software maintenance costs, now the single most expensive phase in the software life-cycle. It is not unreasonable and, in fact, quite common to say that 70% of the life-cycle costs occur in the maintenance phase.

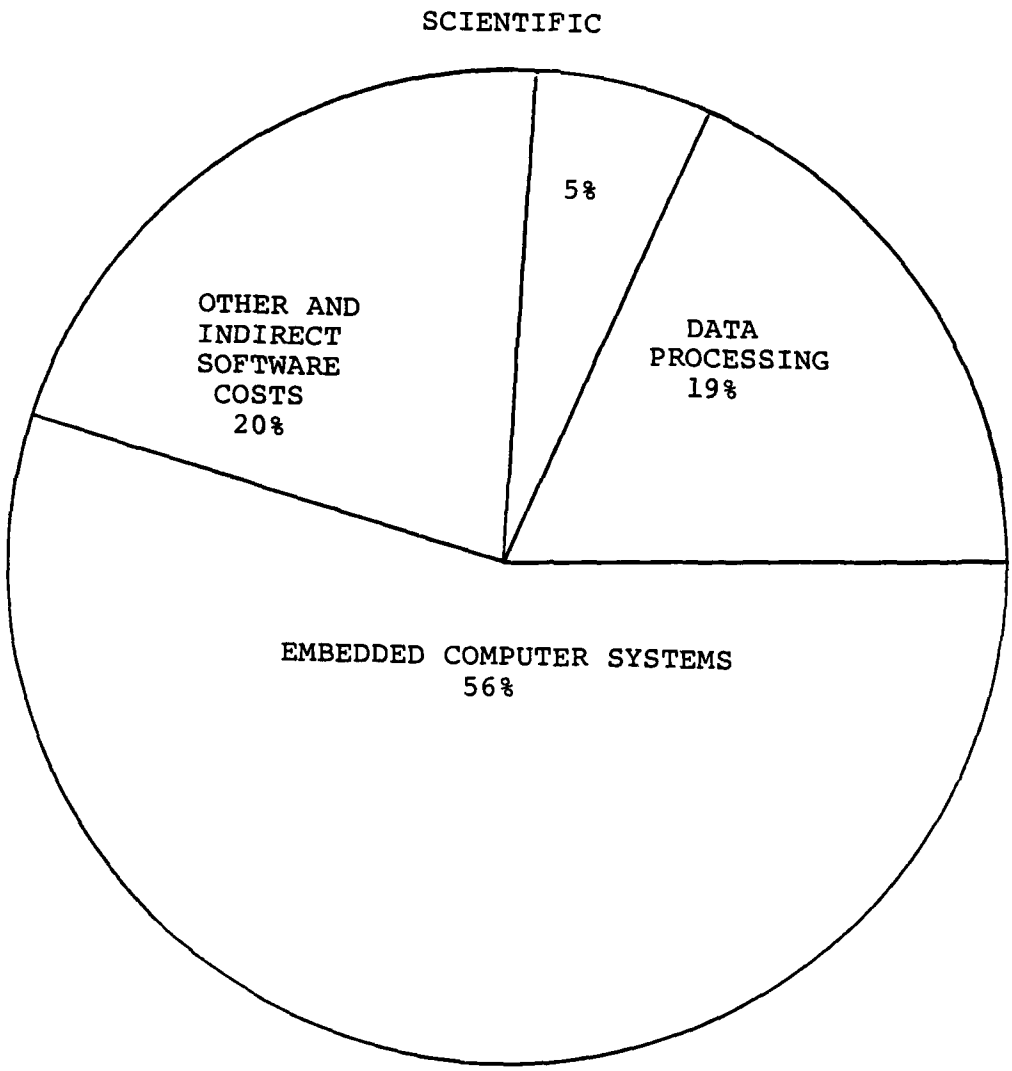
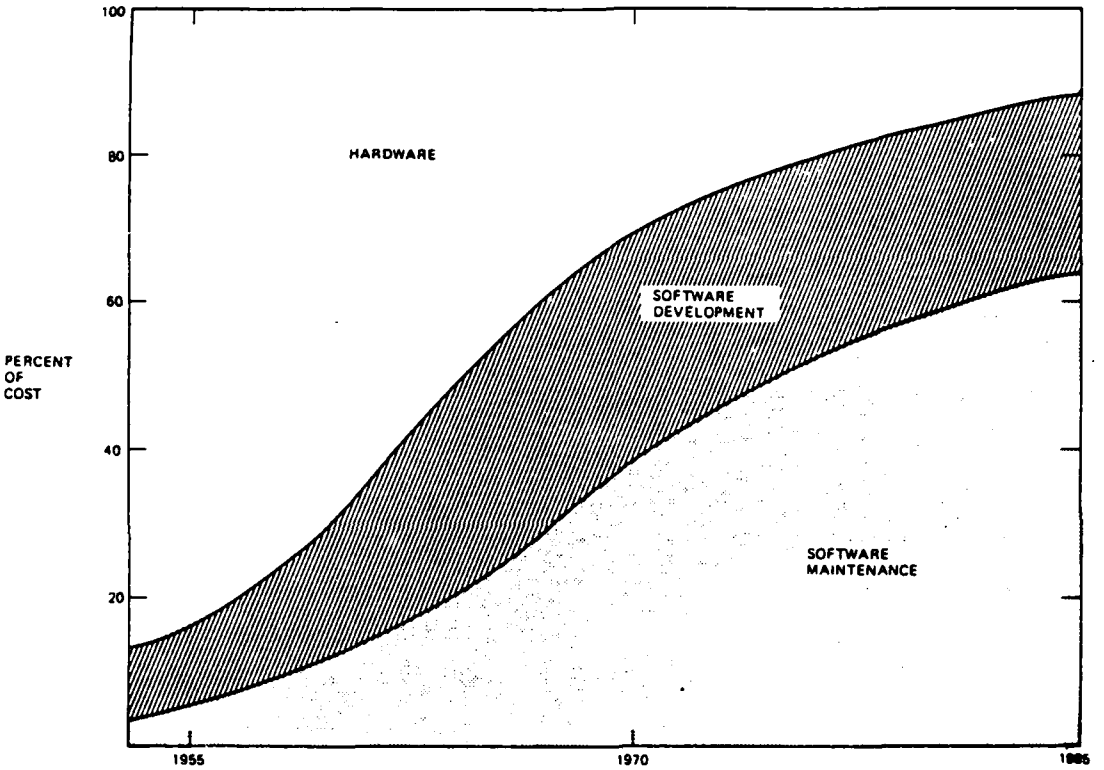


Figure 1 - Breakdown of DOD Software Costs

Source: David A. Fisher (2)



Source: B. Boehm (3)

Figure 2 - Hardware Software Cost Trends

3. A recent Army Post Deployment Software Support (PDSS) study revealed that the Army employed a minimum of 59 different computer types from 29 diverse manufacturers. Software was found to be written in 10 higher order languages and approximately 32 machine-oriented languages. These proliferation problems have a multiplying effect on costs for management and technical activities ranging from personnel training through technical documentation.

4. Unfortunately, the future projections of DOD expenditures, Figure 3, reveal even more alarming trends than previous studies (4). This graph illustrates that as the DOD budget increases 2.8 times, the software price increases 8.1 times. Clearly, management and technical issues relating to software methodology and productivity must be surfaced and addressed.

In addition to cost concerns, studies have also revealed software reliability as a second major area for software difficulties. Reliability should be viewed as far more than quantitative Mean Time Between Interruption (MTBI) statistics or automated recovery from a hardware failure. Software reliability is a function of the difficulties a particular software system causes for the users. A software system would have high reliability if it satisfied the initial requirements, performed to the users satisfaction, and was designed for change and accountability. Unfortunately, this is rarely the case. Reliability spans the entire software life-cycle from design through maintenance and is the primary culprit responsible for excessive costs (7,8). The high costs of software for defense systems are a symptom of poor reliability. This poor reliability is partially a result of language proliferation. But, in addition, recent studies indicate a strong correlation between software's structural composition and the amount of errors encountered.

The foremost key to software reliability is the degree of precision and accuracy achieved during the initial system life cycle phases of requirements/specifications and design (Figure 4). It is these

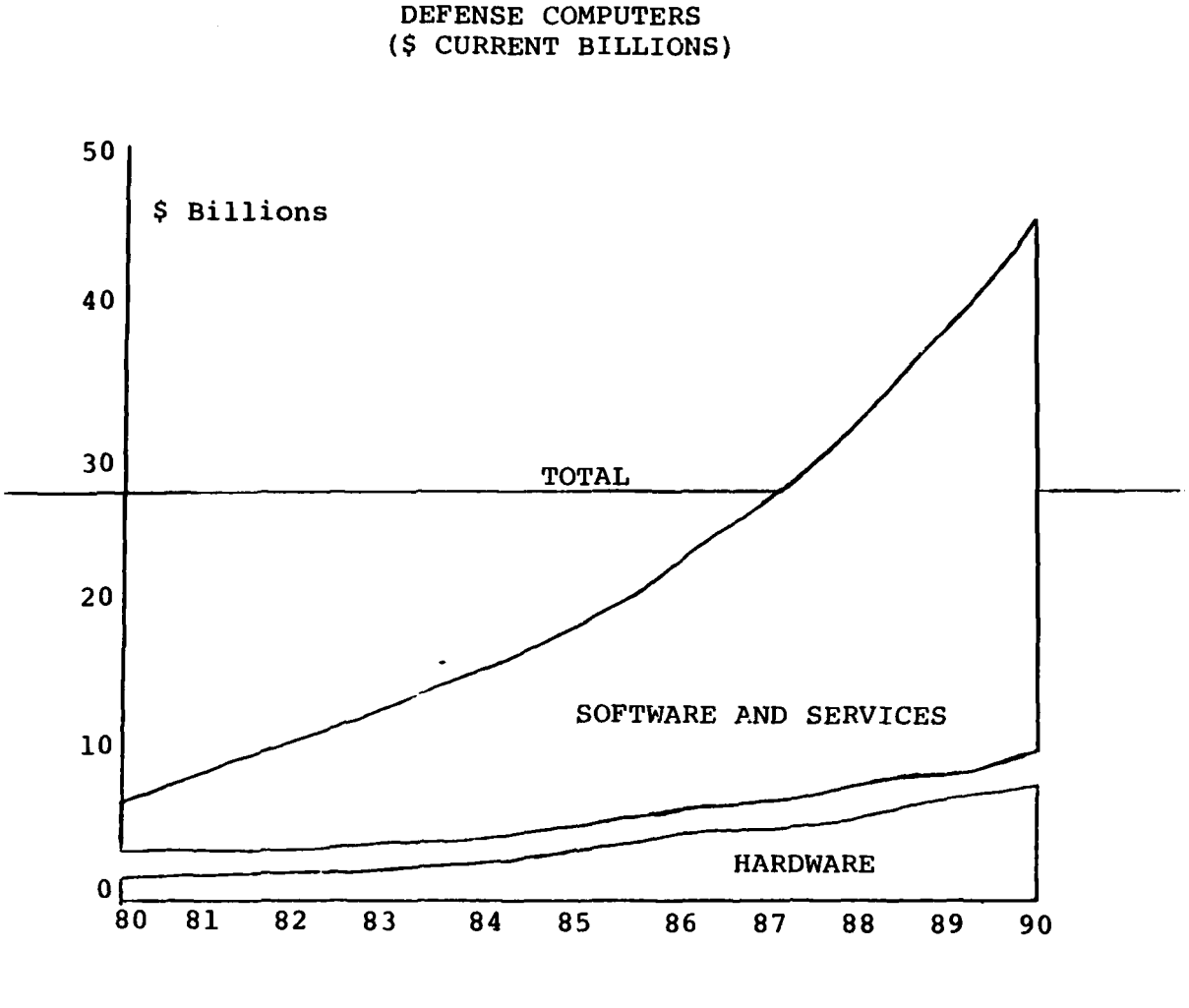


Figure 3 - Defense Computer Expenditures Till 1990

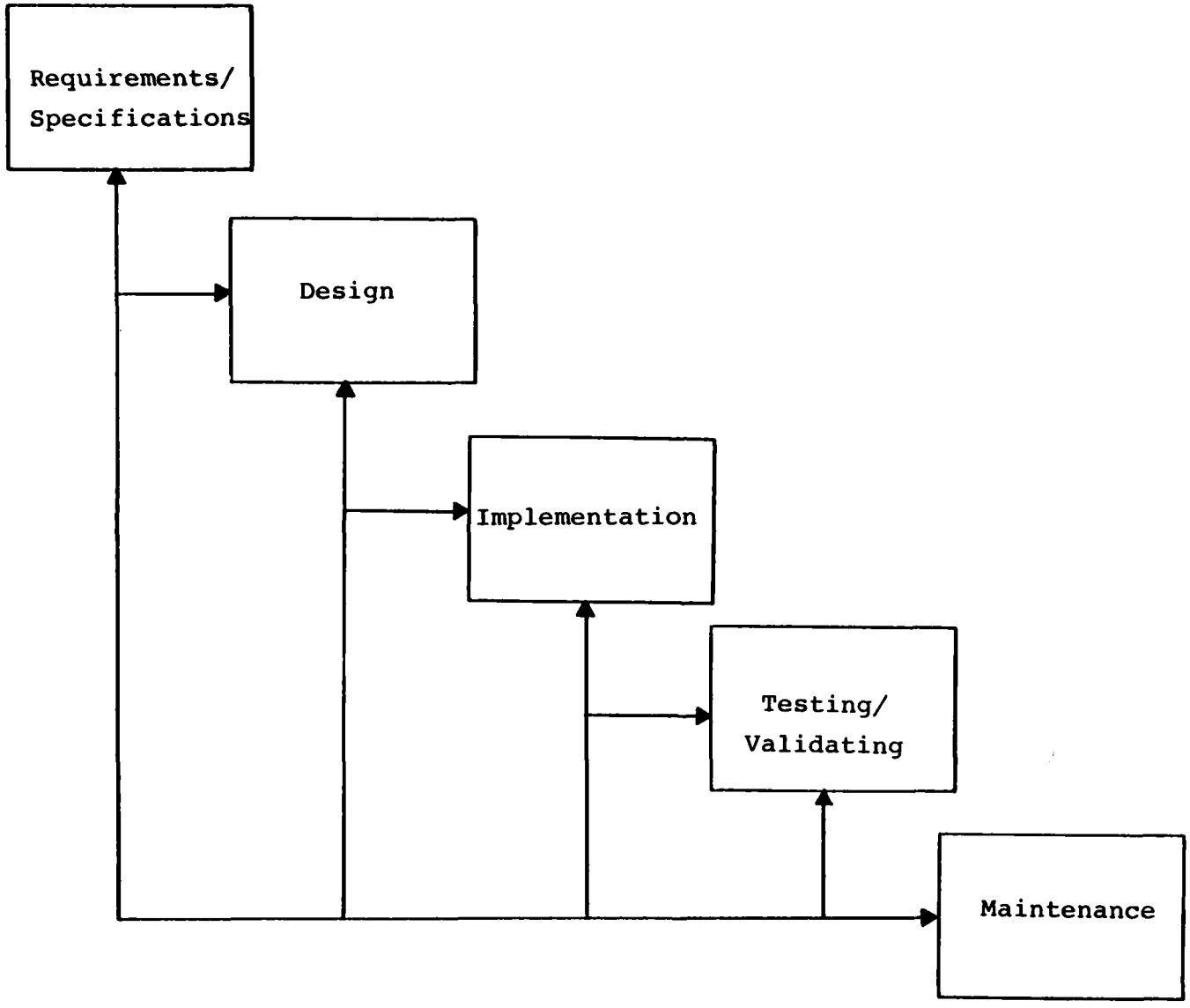


Figure 4 - System Life-Cycle Phases

phases which have the greatest impact on the overall efficiency of the system and consequently, the overall cost. This point cannot be over-emphasized. The requirements/specifications phase must reflect a complete and unambiguous statement of the problem to be solved. The design phase must fit a solution to the previously defined problem. In addition, the system software should be designed to facilitate testing efforts and not to constrain the impact of the inevitable change. What is needed is a methodology for software system development which makes the concern for reliability an integral part of the development process, especially in the design phase. Careful attention should be paid to those methods which address the early phase of development and which are strongly linked to testing/validation and maintenance efforts.

One promising avenue of attacking these central problems is by reducing the proliferation of languages, with the ultimate goal of standardization on one high order advanced language possessing sophisticated facilities which address tactical requirements.

The Department of Defense directives (5,6) are clearly aimed at the reduction of proliferation to a small set of approved languages. The result has been the development of one standard programming language for embedded computer systems. This language is known as Ada. Ada's data abstraction capability, modularity, block structure, single entry-single exit control constructs, embedded program documentation, and finally specification/implementation section separation should enhance the production of more reliable tactical system software and programmer maintenance efforts. Additionally, the unique, dynamic exception handling capability, along with real-time access, parallel processing capability, and low level I/O, provide real-time control constructs not previously found in high order languages. These language features, utilized within the Ada Programming Support Environment (APSE) should help reduce support costs and more closely fit the requirements of military systems than do previously available languages and weak support environments when available.

The advantages promised by standardizing on one language include the following:

1. Since the programming language pervades all phases of the software life-cycle, it will play a substantial role in how well the computer system fulfills original ambitions.
2. The limiting of programming languages will allow greater focusing and achievments by research efforts in developing support tools, common library facilities, and training methods.
3. Language proliferation reduction fosters the creation of centralized centers of expertise which could provide a development and maintenance environment which could be shared by various tactical systems to reduce cost and increase reliability.

It should be pointed out, however, that despite the large positive step forward due to standardizing on a powerful software language and support environment, there remains other accompanying factors which must be addressed. The achievement of reliable embedded computer systems software without excessive cost is determined not only by the utilization of a particular language and its support environment, but also their utilization in conjunction with a system design methodology, and a commitment from management to the educational training and project time necessary to bring the components to bear. It is through employment of these evolving components that reliable and cost effective software systems can be realized (Figure 5).

This guide addresses the establishment of a system design methodology developed from a software engineering viewpoint. It has been designed to be compatible with both the Ada language and the demands of large scale software system construction.

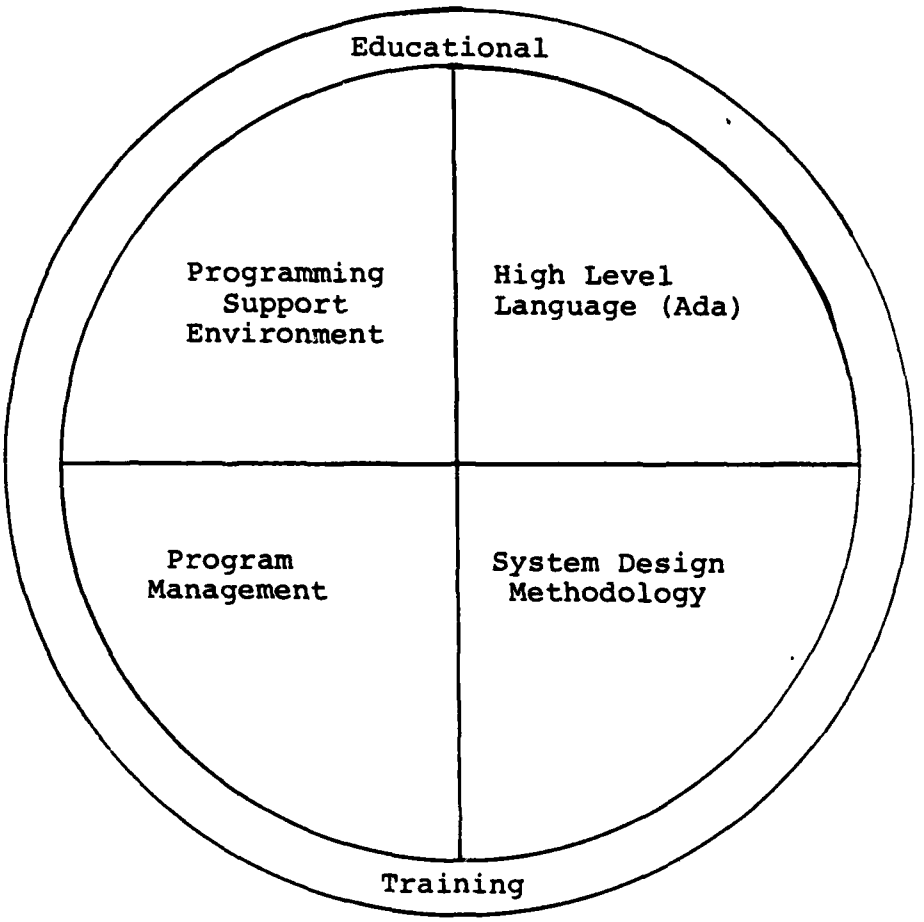


Figure 5 - Components of Reliable Software

Source: T. Walsh, P. Texel (17, 18)

RATIONALE FOR A SOFTWARE SYSTEM DESIGN METHODOLOGY

The difficulties associated with the process of systems/software analysis, development and maintenance, accompanied by the related side-effects of low reliability and excessive cost described in the previous section are ample incentives for a strong response in the problematic area of software design. The term software engineering is representative of the forum for responsiveness and mastery of this problematic area. Although software engineering's mission is quite broad, it includes the creation of a set of techniques sharing a natural affinity and linked by a common theme known here as a software system design methodology.

The importance and need for a software system design methodology has been thrust upon the computer world via the escalating software production and maintenance costs. Furthermore, since large software system development projects with significant numbers of embedded components cannot be comprehended by any one individual, it is essential to create a communicative framework as a common reference for all project members. Especially in the computer field it should be expected that personnel turnover will constantly result in new personnel who will become more productive sooner with a communicative framework. Lastly, cognizance of the numerous failures ad hoc design has produced in the past is ample incentive for seeking assistance from viable tools, techniques and methods.

There is a strong tendency in the data processing world to tout all new methods, tools, and techniques as the ultimate cure-all for computer ills. A corollary naturally being that all previous advances are rendered obsolete. The reader in the literature may also be troubled by semantics. Confusion is rendered by the constant introduction of new terms or words which possess multiple definitions. Comparisons of different methodologies and completeness checks become at best difficult. Finally, there are many questions concerning the elements of good system and software design that presently remain unanswered in software engineering.

At present, the pragmatic system builder is faced with the substantial challenge of modifying and extending existing tools and techniques in a unique combination for the specific project at hand. The pragmatic system builder should be comforted by the thought that it is not necessary to achieve perfection in the year 1982, but to synthesize the best collection of techniques as a practical baseline for present system development and future technical enhancement. This guide represents such an effort for the large scale software system design of the AN/TSQ-73 Missile Minder using the Ada programming language.

The approach to be taken for constructing a software system design methodology is based on lessons learned, analytical studies, and successful practices in the rapidly emerging field of software engineering. The following key principles which are somewhat interdependent, form the underpinning for Control Data's software system design methodology:

1. Life Cycle View - Software systems evolve through various stages of development. During the initial phase of requirements/specifications, the problem is defined and analyzed. The design phase follows and is responsible for translating the problem and its requirements into a blueprint of the actual solution. Once the design representation is complete, implementation begins and the conceptual solution from design is transformed into a computer-processable form. The testing phase follows. Using the requirements as a baseline for which errors are measured, a search is made for design errors and programming errors, and the program is put into final form. The final stage in the life cycle is maintenance. Empirical studies have shown the need and advantages for techniques in all stages especially the initial phases of requirements, specifications and design where surfacing errors will pay the richest dividends (19, 20, 21).

2. Levels of Abstraction - One of the most strategic mental tools for dealing effectively with complexity is the power of abstraction (22-26). The abstraction process describes or emphasizes some of the more important details, properties, or parts, while delegating others into a background that will be expressed later. When this approach is executed properly, the resulting structure has minimum complexity and highly independent levels. The first level of abstraction typically contains little or no detail, it is simply a capsule description of what the system is or does. At each successive level more detail is added to the abstraction of the previous level until ultimately, at the lowest level, all details of the system have been captured.

3. Isolation of Detail - The ability to isolate detail or perform information hiding (26) is related to levels of abstraction. Details or knowledge pertaining to data structures, algorithms, or resources can be hidden. Users desiring a high level view of the system can do so only if these levels exist. Otherwise, they would be distracted by the enormous amount of detail present at the lower level. By implementing a number of levels of abstraction, users with different needs can get the conceptual view of the system that reveals the level of detail they desire. Additionally, the levels of abstraction provide a vehicle to understand large scale systems in a manner compatible with human comprehension. One can view the forest before examining the trees.

4. Finite Human Capability - Human beings possess very limited processing skills of logical structures and need quantitative, graphical and automated assistance to increase precision. All techniques must aim to keep the design within the constraints of human understanding (27, 28).

5. Graphical Communications - The value of graphics should not be underestimated as a communication mechanism that may convey a great deal of information very quickly (29). Pattern recognition often reveals omissions, redundnacy, and awkwardness.

Program design when offered in graphical form has the following advantages over its textual equivalent:

- Interfaces between components enjoy higher visibility.
- Structural characteristics are expressed more concisely and are less ambiguous.
- Graphics are superior in expressing the flow oriented process.
- Psychological studies indicate that the graphical vehicle of communication is more readily comprehendible.

6. Automated Techniques - Recent software engineering advances indicate the strong viability of the automation of design tools sometimes known by the acronym CAD (Computer Aided Design) to support the human design process. These tools possess the capability to greatly boost design productivity and enhance the limited human design capability. A large benefit derived from automated design, via textual and graphical representations, is in the hours saved by letting the computer produce and record the readable documentation. The computer can historically save all the design documents and readily make them available during the entire life cycle (including maintenance). Software engineering studies would have a ready made data base and management would be provided with clear windows of visibility and accountability.

7. Human Communications - The major role of documentation is to foster human-to-human communications. The failure of humans to communicate properly is the source of many computer errors, especially on large systems. Written documentation is the primary means by which different project personnel communicate as the system evolves through time. Such documentation can be represented in either textual or graphic format.

8. User Friendliness - The psychology of the human user must be a consideration in the proposal of any design methodology techniques (30). The arrival at a certain design methodology technique should only come after an understanding of human perceptual skills, decision making ability, cognitive styles, and information processing capacity. The techniques should relieve the user of any of the unnecessary chores inherent in system design, should have ample provisions for corrections, and above all, should be easy to understand. The user should feel comfortable with their usage and be positively encouraged that the aids have made the job easier and more rewarding. User friendliness is an underlying principle in motivating computer personnel and increasing productivity.

9. Simplicity - Design methodology techniques should be based on "deep simplicities" so that they can be properly and effectively utilized. As is stated in Structured Programming (31), "Good program design finding deep simplicities in a complex logical task-leads to work reduction. It can reduce a 500-line program that makes sense line-by-line to 100 lines that make sense as a whole. Good design can reduce a 50,000 line program impossible to code correctly to a 20,000 line program that runs error free".

10. Software's Mathematical Nature - There has been an increasing awareness that software is basically mathematical in nature. As Dijkstra states (32), "I regard programming as one of the more creative branches of applied mathematics and the view of a program as an abstract mechanism makes it perfectly clear that designing will play an essential role in the activity of programming".

Thus, these key principles have shaped the software system design methodology which, in turn, determines the techniques which support the methodology. Figure 6 illustrates the adherence of the proposed techniques of the software system design methodology with the stated guiding principles.

BASIC PRINCIPLES \ TECHNIQUES	SYSTEM ENTITY DIAGRAM	SYSTEM DESIGN LANGUAGE	DATA FLOW DIAGRAM	DATA DICTIONARY	STRUCTURE CHARTS	PROGRAM DESIGN LANGUAGE	COMPLEXITY MEASURE	STRUCTURED PROGRAMMING	SOURCE CODE	
LIFE CYCLE VIEW	x	x	x	x	x	x	x	x	x	
LEVELS OF ABSTRACTION	x	x	x	x	x	x	x	x	x	
ISOLATION OF DETAIL	x	x	x	x	x	x	x	x	x	
FINITE HUMAN CAPABILITY	x	x	x	x	x	x	x	x		
GRAPHICAL COMMUNICATIONS	x		x		x		x			
AUTOMATED TECHNIQUES	x		x	x	x					
HUMAN COMMUNICATIONS	x	x	x	x	x	x	x	x	x	
USER FRIENDLINESS	x	x	x	x	x	x	x	x	x	
SIMPLICITY	x	x	x	x	x	x	x	x	x	
SOFTWARE'S MATHEMATICAL NATURE						x	x	x	x	

Figure 6 - Basic Principles/Design Technique Matrix

THE SOFTWARE SYSTEM DESIGN METHODOLOGY OVERVIEW

The guiding principles as set forth in the previous section form the foundation upon which the methodology rests. This overview briefly describes three phases of development, the eight techniques of the design methodology and the approach taken in selecting the methodology. Details of the individual techniques are discussed in the following sections of this plan.

The Ada compatible software system design methodology described in this document has been derived from the program design methodologies review whose contents are found in Appendix A and from the literature references found in Appendix B. Some of the objectives of this evolving and utilized design methodology are:

1. Supports the more difficult system architecture design process which include hardware design occuring concurrently with software design.
2. Utilizes the concept of levels of abstration within the system design phase and portions of the software design and coding phases to clearly view systems at different developmental levels and provides a mechanism for accountability and traceability of system requirements as they are mapped through various design representations to final implementation.
3. Supports clear and concise description and communication of the design between human beings by utilizing graphic tools where possible.
4. Supports automated tools to free human designers from tedious and painstaking efforts during the representation of their design constructs.

In order to facilitate further understanding of these objectives, they must be explained in more detail.

As the result of technological advances in recent years, it is now quite advantageous from a reliability and system cost point of view to take a significantly different approach to system construction.

These advances have reversed the hardware and software cost percentages of the total system costs. Figure 2 of this document illustrates this point nicely. Traditionally, system construction has been highly biased toward hardware. Hardware decisions relating to specifications and actual acquisition took place prior to software considerations (Figure 7). This approach was well suited for the time when hardware costs represented the overwhelming majority of the system costs. Today, however, there are many disadvantages of this approach.

This hardware approach is naturally biased toward hardware solutions. However, hardware medium is not as flexible as software to the inevitable changes that occur as systems evolve. Unnecessary system complexity is often built in by the hardware constraint. The end result of treating hardware and software independently increases the software difficulties, especially during the integration period, because the burden of compatibility is placed squarely on software.

It seems reasonable, especially in light of cost data, that an interdependent view of hardware/software should be taken when generating initial design in response to system requirement specifications. Thus, the major functions of a system should be identified and described independent of their eventual implementation in hardware or software. This can result in better system engineering, correctness, controls, disciplines, traceability and modularity. If such qualities are present, the accountability is simplified in tracing the technical links when the system requirements architectures become hardware and software requirements architecture. The dividends in the maintenance phase should become significant. Hence, a major goal of the methodology is to address the system architecture description process as well as the software program architecture process.

Another major goal or objective is to utilize the levels of abstraction concept within the methodology techniques, especially in the system design phase, to build and view systems at various logical levels. The abstraction process allows for clear communication and separation of functional facilities from the implementation of those features. This is accomplished by a series of abstract levels.

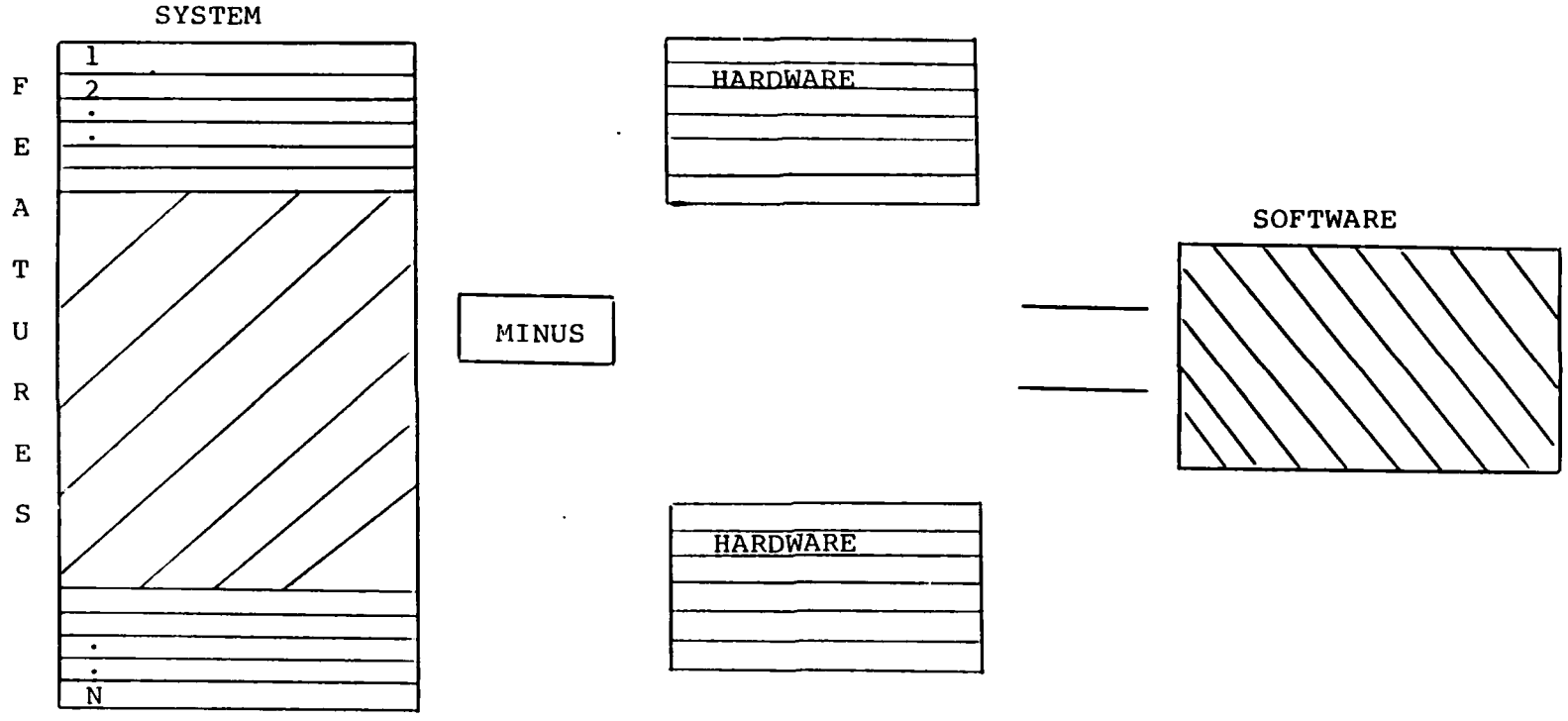


Figure 7 - Present Approach to System Building

Each level should be as highly independent from previous and succeeding levels as possible. Succeeding levels should add more details and preceeding levels should characterize information hiding. The motivation here is to minimize system complexity and facilitate human comprehension of the system's contents. The field of mathematics has long used this process to master complexity and seek more powerful and general consistency (33, 34). Levels of abstraction, of course, serve as a further description stratification of the proposed hardware/software solution. At the highest level of abstraction, the overall view of the system is simply stated. At each successive level, additional details become apparent until at the lowest level one finds all the details required for decision-making or implementation.

As computer aids in the form of graphic tools and storage mediums are available, their use is encouraged to provide information on the usefulness of computer automation for the production of both design and accompanying documentation. Automated representation at each methodology step/technique lends itself to greater control of maintenance changes which in turn could influence future configuration management techniques. The quantity of documentation associated with large system construction can best be handled by automated tools.

As stated previously, the approach to software system design advocated in this document is to generate hardware and software specifications in unison. This is accomplished by the usage of high level techniques known as System Entity Diagrams and a System Design Language within the System Design Phase. Once these techniques have described the system functions, empirical evaluation schemes can selectively choose which features may best be accomplished by both hardware and software (Figure 8). It is anticipated that such factors as cost, performance, and schedules would play a significant role in this selection activity when systems are new and in the conceptual stage.

The major advantage of describing hardware and software functions in unison is that it brings the software engineering perspective to the system level. This should serve to reduce system complexity by

CONTROL DATA SYSTEM DESIGN APPROACH

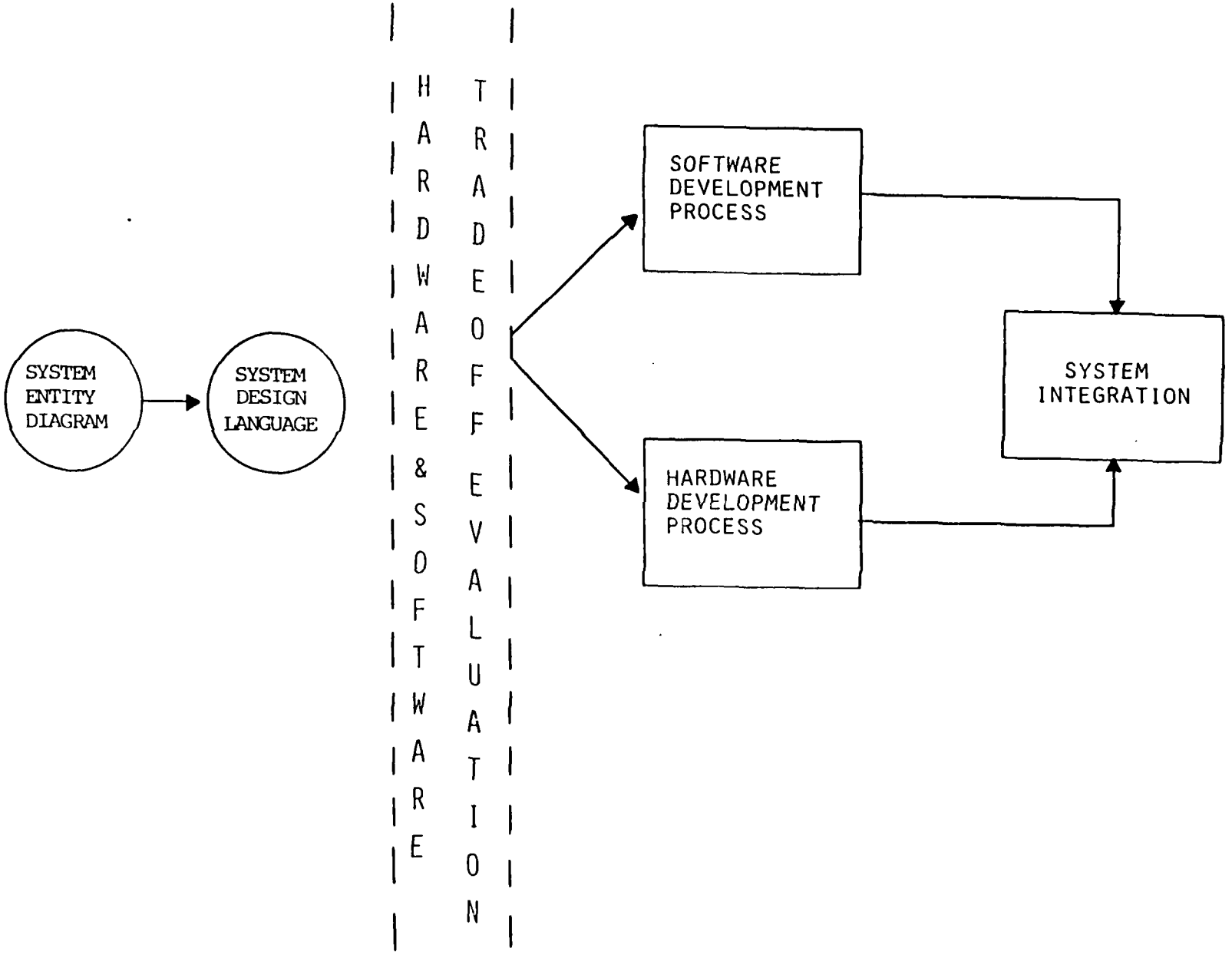


Figure 8 - Control Data System Design Approach

allowing for system solutions independent of eventual implementation mediums. It also establishes a framework for the levels of abstraction philosophy for the parallel development paths of hardware and software. Literature support for this innovative approach may be found in references 35 and 36. As Figure 9 indicates, the software development process then continues with unique steps composed of techniques until the actual code is fully described and integrated. Refer to Figure 10 for an illustration of development phases and steps.

The software system design of the TSQ-73 project will proceed in the manner as depicted in Figure 9. The design of large scale military tactical systems should begin with a conceptual view of the system as a whole remaining independent of hardware and software. This view is accomplished with the System Entity Diagram, the first step. Graphical in nature, the system entity diagram allows for the functional decomposition of the system at the highest level. In the automated form, it provides a user friendly method of generating the initial stages of the system design. The implementation of these processes may eventually be hardware, software, manual or some combination thereof. The SED function is supported using the System Design Language.

Although the SED could theoretically contain n levels of entities, it was decided that the TSQ-73 project could be limited to six. This decision was made for two reasons. First, the complexity of the missile minder system was intuitively felt to warrant a maximum of six levels. Second, the usage of an SDL technique is brand new, yet a high degree of confidence abounded that its Ada-based contructs could adequately support the continuation of the SED process and add additional details.

Supporting the system entity diagram is the system design language, the second step. The system design language (SDL) techniques explains the functions in processor-independent logic based on simple constructs and free-form textual English. The SDL for this project utilizes the Ada HOL with the exception of GOTO. Following this point in

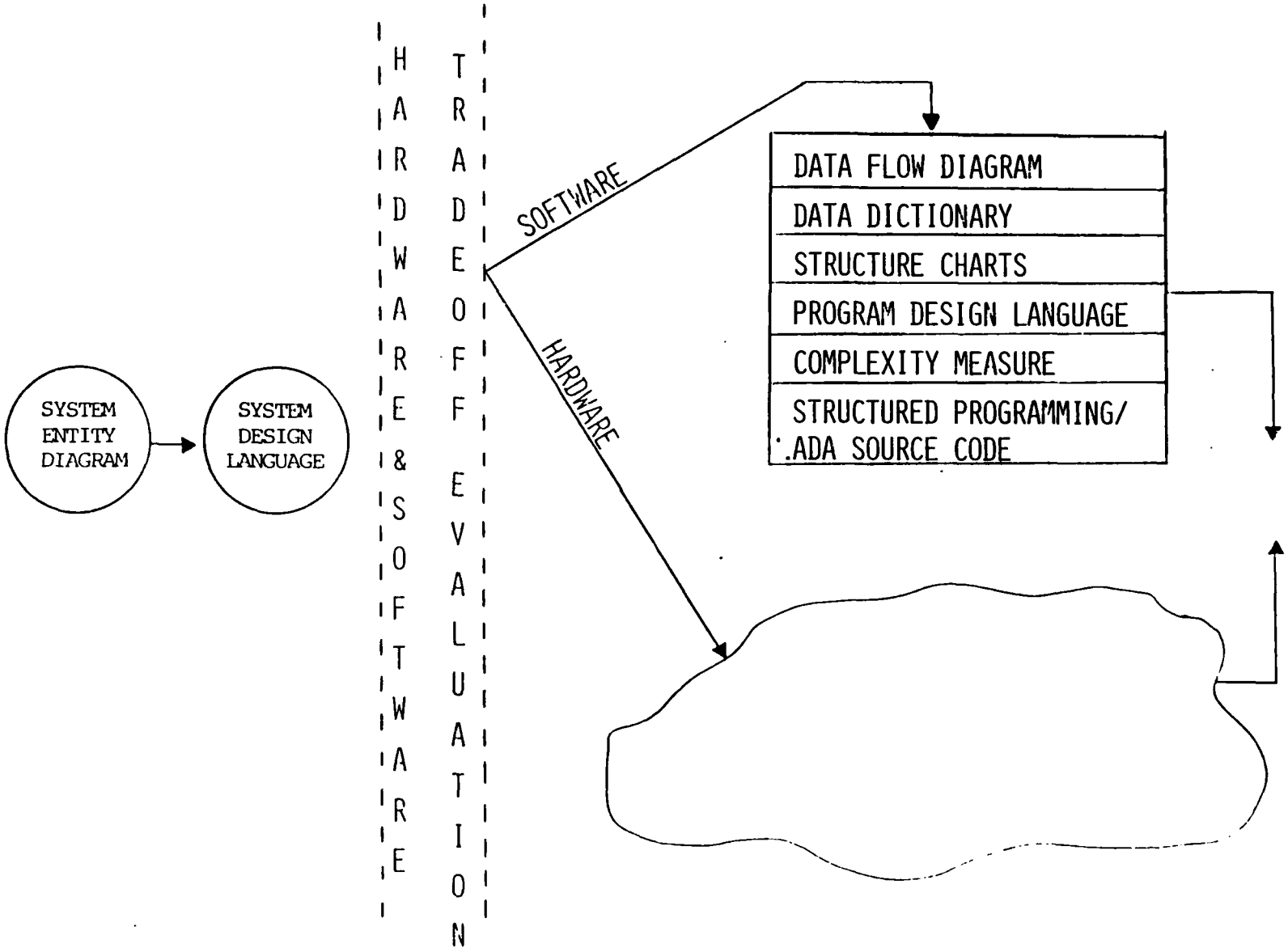


Figure 9 - Software System Methodology Techniques

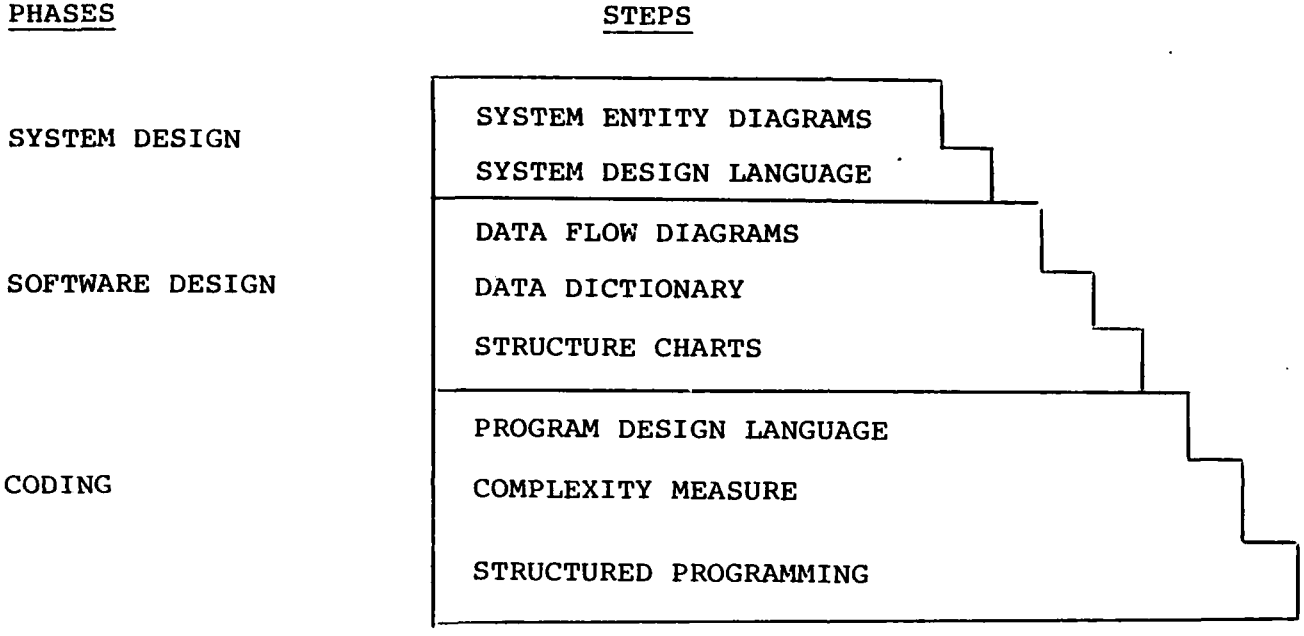


Figure 10 - Steps of Methodology Techniques

the design process, the hardware and software functions are split apart by empirical evaluation schemes, and separate development efforts proceed in parallel. After the completion of the SEDs, SDLs, and hardware/software trade-off analysis, it seems prudent to exhibit the results by a formal preliminary design review.

Since the scope of the contractual effort emphasizes software system re-design, the methodology continues only on the software side. A complete unified system design methodology could be constructed by creating analogous hardware techniques also based on the levels of abstraction techniques. The SDL descriptions which have at least some part software become bubbles on the data flow diagrams which lie within the bounds of software architecture. If hardware is also present, it is shown on the DFD as boundaries.

The first step in the software design phase is provided by the technique known as data flow diagrams (DFD). The data flow diagrams are graphical representations of the software system. They are used to model the system in terms of input data, output data, stored data, significant intermediate data forms and the transformation of data. Like the system entity diagram, they allow different users to view the system from various levels of abstraction, thereby allowing the isolation of detail as required by those users. The automation at this level allows for the production of both design and accompanying historical documentation.

In support of the data flow diagram is the data dictionary. It is here that the data from the DFDs is defined in terms of its component parts. Automation at this stage checks the entries in the dictionary for recursive definitions, aliases, and provides for the cross-referencing of data items. This last feature is very valuable when those inevitable changes occur, giving the user a roster of other data items referenced by a given data item as well as those which make reference to it. In this way, the impact of a change can be evaluated prior to the actual change. Also logged into the data dictionary are pseudo code descriptions of the data transformations or processes, aimed at revealing the nature of the transformation that might not be exposed by just a name.

After modelling the software system using the data flow diagrams, the methodology advances to the structure chart technique, the fifth step. Structure charts afford the user with a graphic representation of the system, but unlike the data flows, they model the system in terms of modules or program units, calls made between them and the data that is passed. Viewing the system from this vantage point is crucial since the "modules" suggest which Ada program units will be used. Like the data flows, structure charts are strongly linked to the guiding principles. The designers will utilize an automated tool to produce current documentation as well as historical documentation. Structure charts provide the link between the data flow level and the program design language (PDL) or program unit design.

The program design language may be viewed as a combination of formally defined constructs known as keywords which are a subset of the Ada language. The purpose of the PDL is to design the module or program unit. The result is a design which possesses accuracy and precision yet has allowed for a flexible range of expressiveness. As an additional gain, using a subset of Ada for the PDL will subtract from the time it takes to write the Ada code. The PDL constructs are a subset of the Ada language, and in turn, have the SDL constructs as a subset (Figure 11).

The concepts of structured design will be utilized to guide in the creation of reliable structures during PDL application. It is at this point that two quantitative measures based on mathematics are introduced. These are a logical complexity measure introduced by McCabe (37) and structured programming to two Italian mathematicians named Bohm and Jacopini (38).

The importance of limiting a module's logical complexity within some quantitative bound has recently gained recognition as a heuristic practice directly related to reliability. The metric utilized in the methodology is a derivative of a program's control flow and based on mathematical graph theory. All computer programs can be expressed as graphics where each node in the graph corresponds to a block of code in the program where the flow is sequential

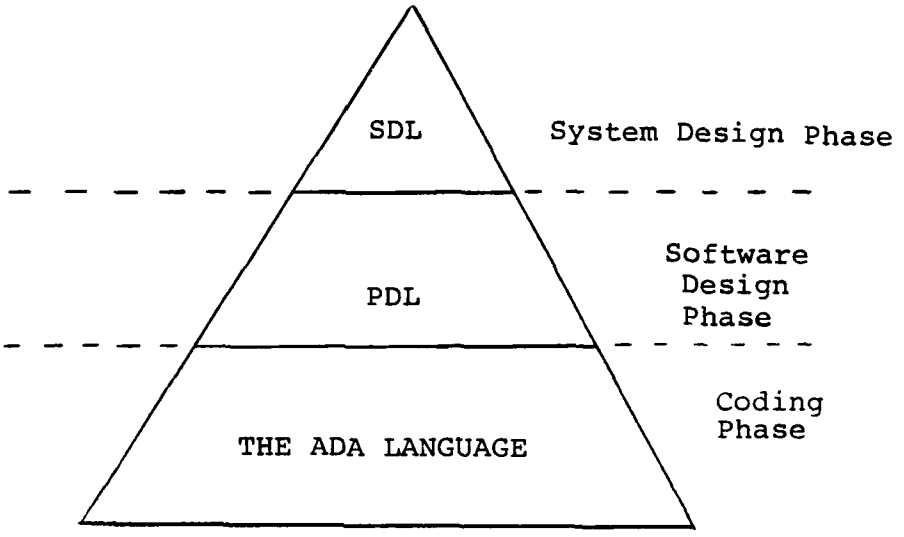


Figure 11 - SDL/PDL/Ada Language Relationship

and the area corresponds to branches taken in the program. An added advantage of this complexity measure is that it can be mathematically linked to structured programming.

Structured programming has its theoretical roots in a technical paper by Bohm and Jacopini who proved mathematically that any program could be constructed from three basic constructs. The importance of using a handful of constructs to create the most complex of human logical entities should not be underestimated. An analogy exists in electrical engineering where the most complicated logical circuit can be built by a combination of "and", "or", "not" gates. The lack of basic constructs enhances rather than limits the programming process by imposing a discipline at the code level. All the constructs have the desirable attribute of having a single entry point, and a single exit point. This avoids code from becoming unwieldly and enhances comprehendability and maintainability. These quantitative measures should enhance the Ada code which is the final product.

Using the eight described techniques in methodical steps to bring an analysis and design project from the requirements level to the code level is not merely a show of design technique agility. Each step requires the performance of its own validation function before the next step can proceed. This validation function within a step in the project is known as a walkthrough or design review.

When the DFDs are complete, a walkthrough will reveal any weakness in the initial requirements specification, and will point up interface criteria and exception criteria. Actual system scenarios may be walked through the DFDs to uncover any design omission. A design may address all the requirements, but validation will insure that all the requirements have been stated. The design should not proceed until the DFD milestone testing is complete.

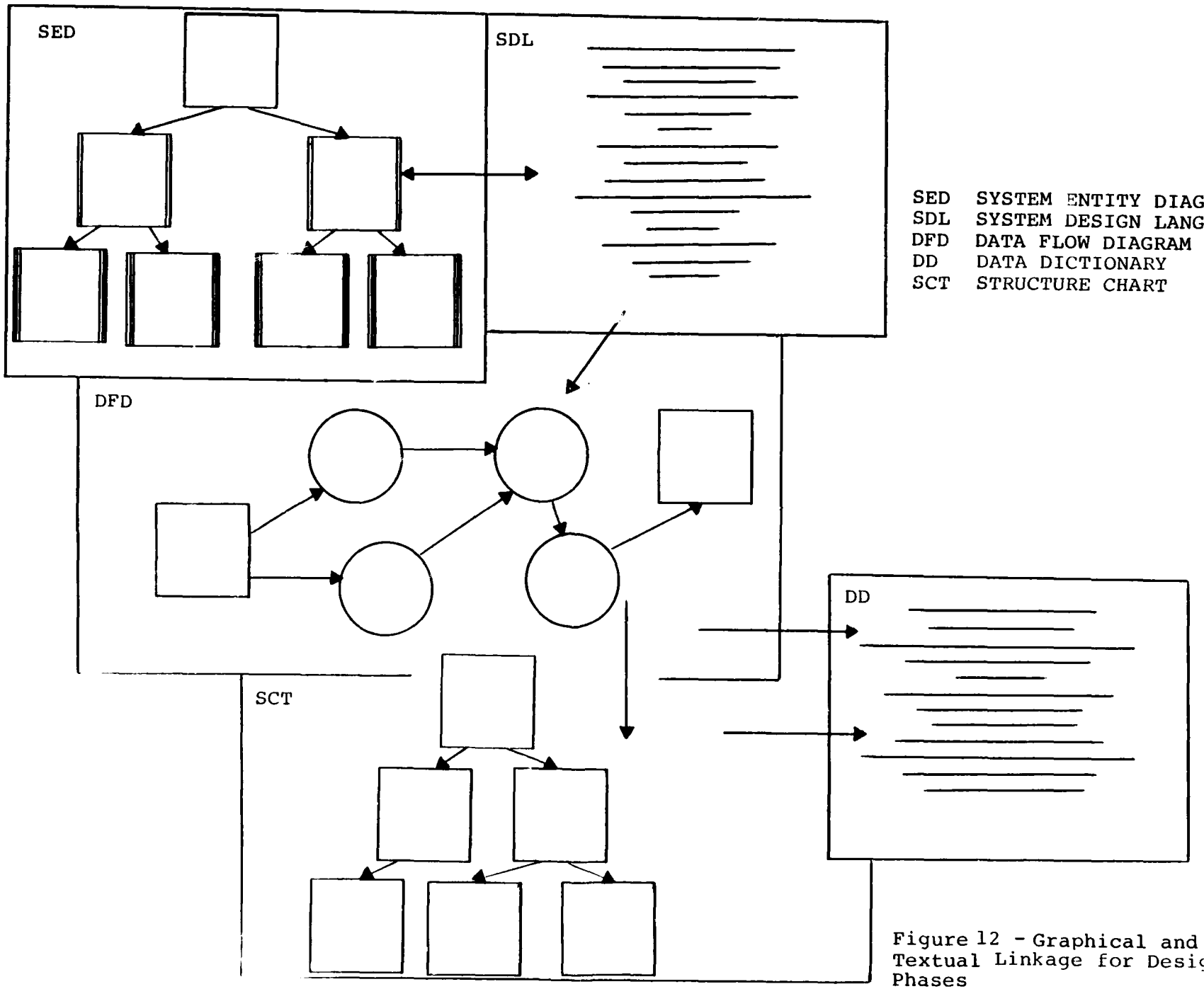
The SC validation is again a walkthrough, but this time a correct solution for program units is sought rather than a correct problem definition. Work should not proceed until the SC milestone validation is complete.

Figure 12 illustrates the graphical and textual linkage for design phases of the methodology.

Structure charts are followed by the use of a PDL which represents the logical structure of each eventual module in the system. The PDL in this software system design methodology is also a subset of the Ada language. Also, such attributes as coupling, cohesion, and complexity should be reviewed. It is often necessary to revise the SC based upon the anticipated program unit not meeting modular criteria.

The last step of the software development is coding and testing. Code should always be accompanied by appropriate walkthroughs. Several studies done on the usefulness of the walkthroughs outlined above have revealed that 50 to 70 percent of all errors are uncovered during the coding phase, and because of this early error detection, the errors found after software release were minor to correct. The most expensive errors are design errors and the most common design error is that of omission or some condition that was overlooked. Walkthrough at each step of analysis and design should minimize omissions.

The following sections in this designer's guide will fully describe each of the design methodology steps just outlined.

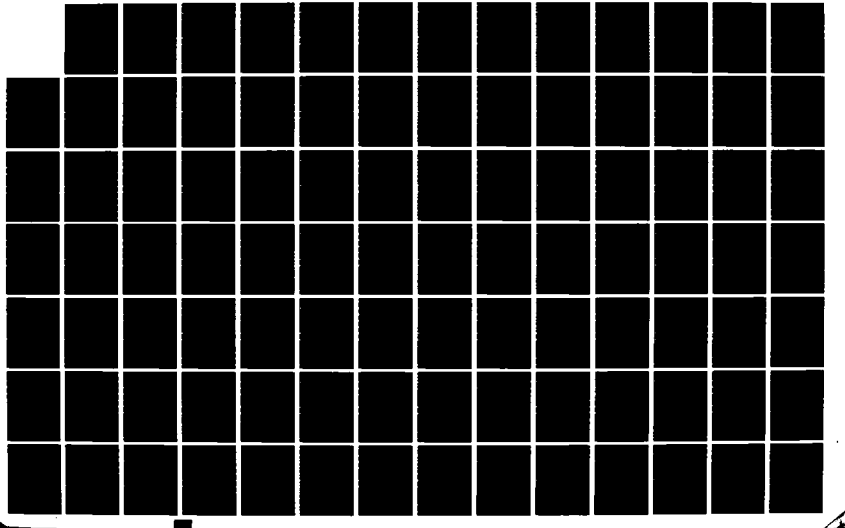


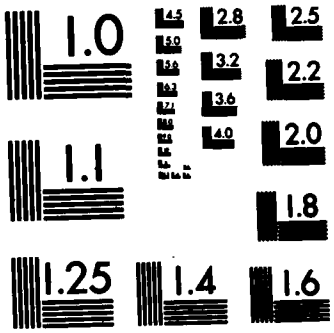
SED SYSTEM ENTITY DIAGRAM
SDL SYSTEM DESIGN LANGUAGE
DFD DATA FLOW DIAGRAM
DD DATA DICTIONARY
SCT STRUCTURE CHART

Figure 12 - Graphical and Textual Linkage for Design Phases

AD-A123 308 LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE MISSILE MINDER AN/TSQ-73 USING T..(U) CONTROL DATA CORP SHREWSBURY NJ GOVERNMENT SYSTEMS 09 NOV 82 3/6

UNCLASSIFIED DAAK80-81-C-0107 F/G 9/2 NL





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

SYSTEM ENTITY DIAGRAM

The System Entity Diagram is used to give a graphic representation of the hierarchical organization of system functions before hardware/ software trade-offs are made. The approach calls for the partitioning of the system's functions into respective levels of detail. In determining the delineation between these levels, the employed technique has utilized the concept of levels of abstraction popularized by Dijkstra (22,23). In employment of this idea, the system functions are partitioned into distinct "levels" that meet the design criteria. Each of the levels forms a group of related "entities", and is defined in terms of the preceeding levels. The purpose of this leveling is to manage system complexity by reducing the system to a group of smaller and more comprehendable pieces.

One of the immediate advantages of this technique is that it affords an early view of the composition of the system by the user. The SED technique has been specifically designed with the intention of revealing pertinent details without inundating the user with the technical mechanics. The SED should surface difficulties so they can be easily identified and corrected. Another purpose of the SED is to establish an accountability scheme for the components when they eventually find their implementation medium.

Each level depicted in the SED is defined in terms of the preceeding level. Conceptual terminology was developed to label each one of the levels with a unique title and graphical symbol. Accordingly, the first six levels have been entitled as facility, subfacility, component, subcomponent, function and subfunction. Each of these levels can be defined as being a discrete grouping of or subdivision of the system with the distinction being the level in which they exist. It should be noted that the concept of levels is far more important than the labels or symbols used for human communication purposes. The symbols used to represent the six levels are as follows:

Facility - The highest level of abstraction is represented as:

Subfacility - The second level is represented as:

Component - The third level is represented as:

Subcomponent - The fourth level is represented as:

Function - The fifth level is represented as:

Subfunction - The sixth level is represented as:

Connections between levels are accompanied by a vector drawn from the edge of one entity to the edge of an entity on the level directly below.

Since the object of the system entity diagram is to represent the system at the highest levels of abstraction before any hardware or software decision are made, care must be taken to maintain the proper relationships between the levels and that the designer is not overwhelmed by the tendency to think in terms of implementation details.

The graphic tools developed by Control Data Corporation lend themselves quite naturally to the idea of the system entity diagram. These automated techniques known as the Structured Analysis, Structured Design/Computer Aided Design of Software (SASD/CADS) tools support the creation and maintenance of data flow diagrams, data dictionaries and structure charts in computer graphic form. The concept of the Structured Analysis Methodology inherent in the SASD/CADS tool should enable designers to utilize the module structure capability to create and maintain the system entity diagram.

The system entity diagram (SED) uses different symbols to indicate different levels of abstraction. This reinforces the concept of levels which is so critical to the isolation of detail concept as well as enabling immediate identification of the level at which the system is being illustrated.

The use of automation of the design methodology techniques should free the designer from manual drawings, increase software visibility by letting the machine record all versions, and provide a complete data base for future software engineering studies into productivity. The following graphics (Figures 13 thru 15) represent automated print-outs of the SED technique generated during the AN/TSQ-73 re-design effort. They illustrate the decomposition necessary to architecturally grasp the elements of remote communications which is one of the four facilities of the AN/TSQ-73. For purposes of illustration, only a small piece of the actual TSQ-73 re-design SEDs is shown here.

As shown in Figure 13, each of the pictured four facilities has been assigned a descriptive title and a corresponding decimal number.

TITLE: AN/TSQ-73 SED		PAGE: 1
REVISION 3	AUTHOR: RAW/WAS	DATE: 10/13/81
NOTES:		

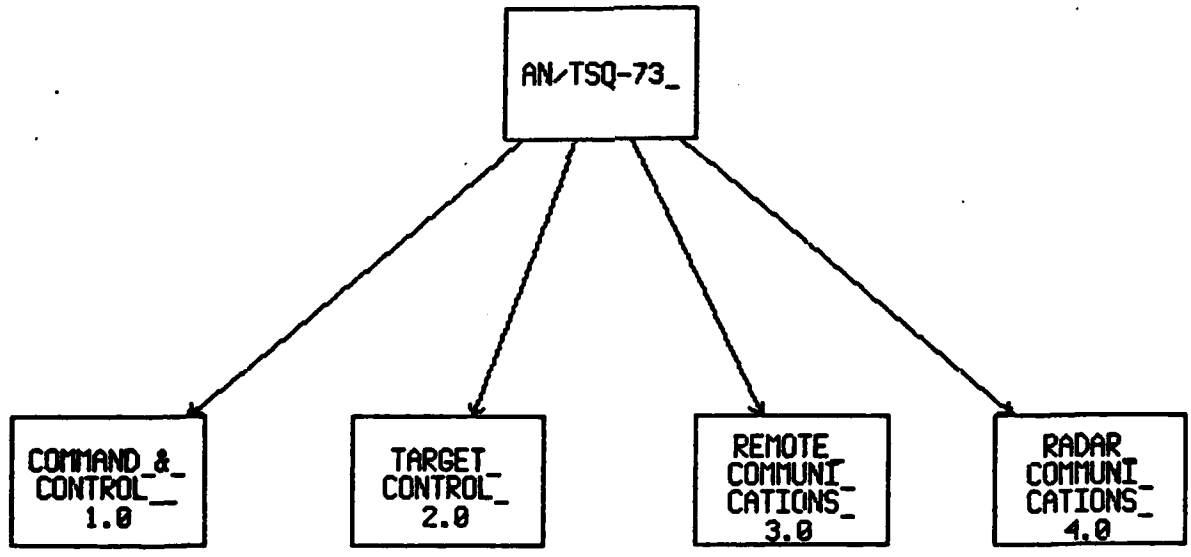


Figure 13 - System Entity Diagram Example

TITLE: 3.0 REMOTE COMMUNICATION SED		PAGE: 303
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 11/19/81

NOTES:
REVISION OF DIAGRAM 300

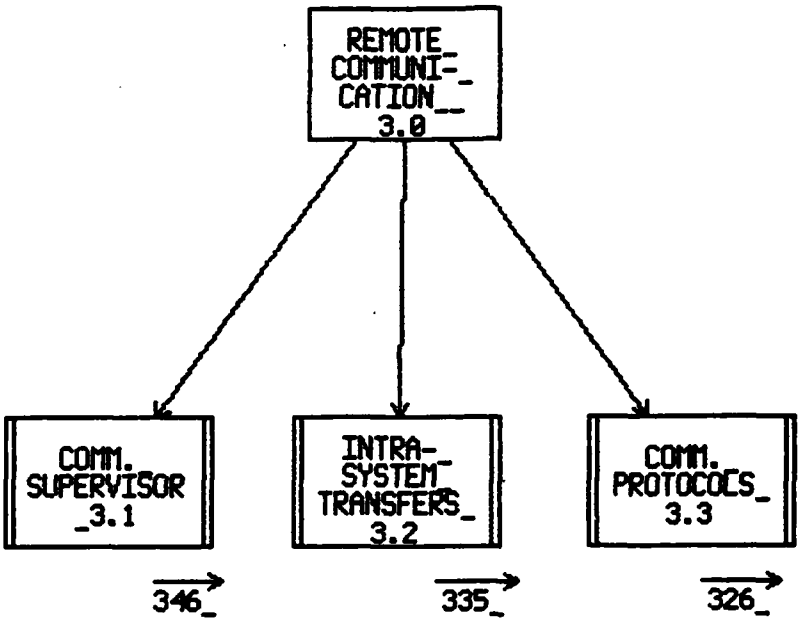


Figure 14 - System Entity Diagram Example

TITLE: 3.1 COMMUNICATION SUPERVISOR SED		PAGE: 346
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/18/82

NOTES:
REVISION OF DIAGRAM 304

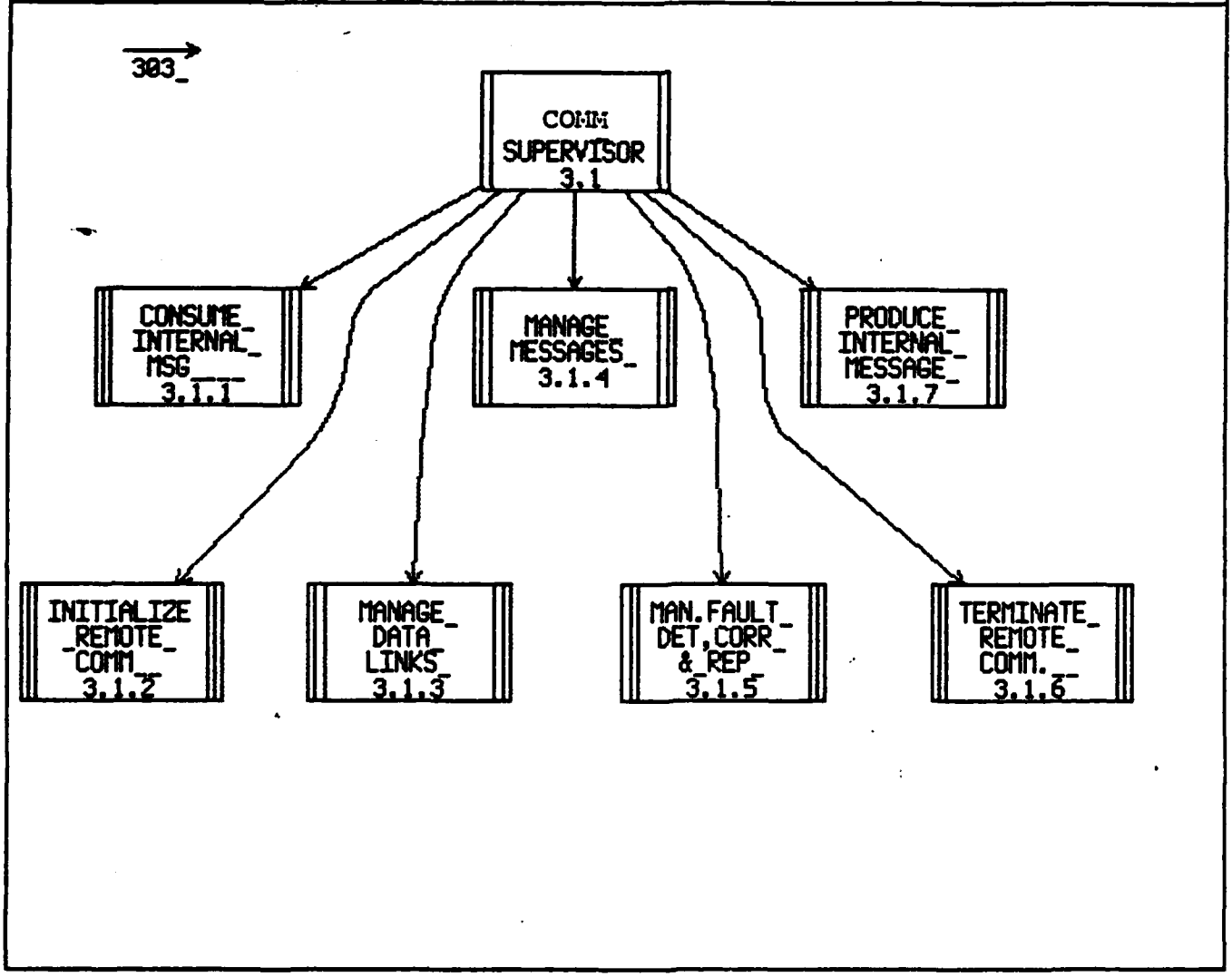


Figure 15 - System Entity Diagram Example

The numbering system employed allows traceability among functions at various levels of detail. The relationship between the decimal numbers which appear at different levels is such that the number for any given level is said to have a number which is the root of the numbers assigned on the levels below.

The small arrows under each entity (Figure 14) "point" to the page or record number which illustrates further decomposition of that particular entity. In Figure 15, the small arrow in the upper left hand corner indicates the page or record number which illustrates the level of abstraction directly above the one shown.

The number of system entity diagrams generated for a large system can be quite large and difficult to manage. The automated tool provided by Control Data Corporation automatically generates a listing of the titular (Figures 16-18) information contained in each diagram. This feature supports ease of diagram recall and maintenance as well as automatic generation and maintenance of a complete historical data base of all SED versions generated for a particular project.

The following guidelines are offered for system entity diagram users:

1. Apply the system entity diagram technique as a communication tool to represent the hierarchical organization of system function independent of implementation media.
2. Search the specification document to identify high level groupings of functional areas. The labels for these groups should appear as titles for the first level entities (facilities).
3. Decompose the system into functional components in an iterative fashion.
4. Continue the decomposition until a level of detail is revealed which will allow any given entity or functional area to be assigned to a hardware, software or manual implementation medium.

5. Use the system entity diagram in concert with the Ada-based textual system design language (see section immediately following) to identify and correct deficiencies in the system design until a satisfactory result is obtained.

```
RECORD NUMBER =    1
TITLE = AN/TSQ-73 SED
AUTHOR = RAW/WAS                                    REVISION = 3
TYPE = SCT     DATE = 10/13/81       ROOT =
NOTE1 =
NOTE2 =

RECORD NUMBER =  100
TITLE = 1.0 COMMAND & CONTROL SED
AUTHOR = RAW/WAS                                    REVISION = 3
TYPE = SCT     DATE = 10/13/80       ROOT = NONE
NOTE1 =
NOTE2 =

RECORD NUMBER =  101
TITLE = 1.1 OPERATING SYSTEM SED
AUTHOR = RAW/WAS                                    REVISION = 2
TYPE = SCT     DATE = 10/27/81       ROOT = NONE
NOTE1 =
NOTE2 =

RECORD NUMBER =  102
TITLE = 1.2 SUPPORT SOFTWARE RUNTIME SED
AUTHOR = RAW/WAS                                    REVISION = 1
TYPE = SCT     DATE = 10/2/81        ROOT = NONE
NOTE1 =
NOTE2 =

RECORD NUMBER =  103
TITLE = 1.3 RUNTIME INITIALIZATION SED
AUTHOR = RAW/WAS                                    REVISION = 3
TYPE = SCT     DATE = 10/14/81       ROOT = NONE
NOTE1 =
NOTE2 =

RECORD NUMBER =  104
TITLE = 1.4 USER APPLICATION LIBRARY SED
AUTHOR = RAW/WAS                                    REVISION = 1
TYPE = SCT     DATE = 10/12/80       ROOT = NONE
NOTE1 =
NOTE2 =

RECORD NUMBER =  105
TITLE = 1.5 I/O MESSAGE INTERPRETER SED
AUTHOR = RAW/WAS                                    REVISION = 1
TYPE = SCT     DATE = 10/14/81       ROOT = NONE
NOTE1 =
NOTE2 =

RECORD NUMBER =  106
TITLE = 1.6 DISPLAY REFRESH FILE MANAGER SED
AUTHOR = RAW/WAS                                    REVISION = 1
TYPE = SCT     DATE = 10/13/81       ROOT = NONE
NOTE1 =
```

Figure 16 - SED Automated Management Report Record

```
TYPE = SCT     DATE = 9/28/81          ROOT = NONE
NOTE1 = REVISED ON DIAGRAM 303
NOTE2 =

RECORD NUMBER =  301
TITLE = 3.2 TRANSMIT MESSAGE SED
AUTHOR = NAK/LWD/MJD                                REVISION = 0
TYPE = SCT     DATE = 9/28/81          ROOT = NONE
NOTE1 =                          ***DELETED***
NOTE2 =

RECORD NUMBER =  302
TITLE = 3.1 RECEIVE DDL MESSAGE SED
AUTHOR = NAK/LWD/MJD                                REVISION = 0
TYPE = SCT     DATE = 9/28/81          ROOT = NONE
NOTE1 =                          ***DELETED***
NOTE2 =

RECORD NUMBER =  303
TITLE = 3.0 REMOTE COMMUNICATION SED
AUTHOR = LWD/RAW/GIC                                REVISION = 1
TYPE = SCT     DATE = 11/19/81         ROOT = NONE
NOTE1 = REVISION OF DIAGRAM 300
NOTE2 =

RECORD NUMBER =  304
TITLE = 3.1 COMMUNICATION SUPERVISOR SED
AUTHOR = LWD/RAW/GIC                                REVISION = 0
TYPE = SCT     DATE = 11/19/81         ROOT = NONE
NOTE1 = REVISED ON DIAGRAM 346
NOTE2 =                               ***REVISED***

RECORD NUMBER =  305
TITLE = 3.1.1 INITIALIZE REMOTE COMMUNICATION SED
AUTHOR = LWD/RAW/GIC                                REVISION = 0
TYPE = SCT     DATE = 11/19/81         ROOT = NONE
NOTE1 =
NOTE2 =                          ***DELETED***

RECORD NUMBER =  306
TITLE = 3.1.1.1 HOUSEKEEPING FUNCTION SED
AUTHOR = LWD/RAW/GIC                                REVISION = 0
TYPE = SCT     DATE = 11/19/81         ROOT = NONE
NOTE1 =                           ***DELETED***
NOTE2 =

RECORD NUMBER =  307
TITLE = 3.1.1.2 INITIALIZE COMM SUBSYSTEM PARAMETERS SED
AUTHOR = LWD/RAW/GIC                                REVISION = 0
TYPE = SCT     DATE = 11/19/81         ROOT = NONE
NOTE1 =                           ***DELETED***
NOTE2 =

RECORD NUMBER =  308
TITLE = 3.1.1.2.2 INITIALIZE COMMUNICATION TABLES SED
```

Figure 17 - SED Automated Management Report Record

```
NOTE2 =

RECORD NUMBER =  340
TITLE = 3.3.3.3 TRANSMIT MESSAGE
AUTHOR = LWD/RAW/GIC                              REVISION = 0
TYPE = SCT     DATE = 1/15/82        ROOT =
NOTE1 =
NOTE2 =

RECORD NUMBER =  341
TITLE = 3.3.3.3.1 ENCODE MESSAGE TO PROTOCOL
AUTHOR = LWD/RAW/GIC                              REVISION = 0
TYPE = SCT     DATE = 1/15/82        ROOT =
NOTE1 =
NOTE2 =

RECORD NUMBER =  342
TITLE = 3.3.3.3.2 SEND MESSAGE SED
AUTHOR = LWD/RAW/GIC                              REVISION = 0
TYPE = SCT     DATE = 1/15/82        ROOT =
NOTE1 =
NOTE2 =

RECORD NUMBER =  343
TITLE = 3.3.3.5 PROTOCOL UTILITIES SED
AUTHOR = LWD/RAW/GIC                              REVISION = 0
TYPE = SCT     DATE = 1/15/82        ROOT =
NOTE1 =
NOTE2 =

RECORD NUMBER =  344
TITLE = 3.3.3.6 TERMINATE DATA LINK SED
AUTHOR = LWD/RAW/GIC                              REVISION = 0
TYPE = SCT     DATE = 1/15/82        ROOT =
NOTE1 =
NOTE2 =

RECORD NUMBER =  345
TITLE = 3.3.3.7 PRODUCE COMM. SUPERVISOR MESSAGE SED
AUTHOR = LWD/RAW/GIC                              REVISION = 0
TYPE = SCT     DATE = 1/15/82        ROOT =
NOTE1 =
NOTE2 =

RECORD NUMBER =  346
TITLE = 3.1 COMMUNICATION SUPERVISOR SED
AUTHOR = LWD/RAW/GIC                              REVISION = 1
TYPE = SCT     DATE = 1/18/82        ROOT =
NOTE1 = REVISION OF DIAGRAM 304
NOTE2 =

RECORD NUMBER =  348
TITLE = 3.1.2.1 HOUSEKEEPING
AUTHOR = LWD/RAW/GIC                              REVISION = 0
TYPE = SCT     DATE = 1/15/82        ROOT =
```

Figure 18 - SED Automated Management Report Record

SYSTEM DESIGN LANGUAGE

The purpose of the System Design Language technique is to provide an Ada-based textual representation of system functions. In order to meet this objective the System Design Language (SDL) technique must represent the following features of the system in question:

- System Objective(s)
- System Requirements and Capabilities
- System Capacities
- System Constraints
- System Entities, Subentities, and their relationships
- System Control
- System Data Flow
- System Exception Handling

In the process of including the above features in the SDL it is important to keep in mind that the combined purpose of these two steps (SED & SDL) is ultimately to specify the functions of the system at a level where the systems engineer may separate hardware functions from software functions. Thus, the system design language maps the information presented graphically by the SED technique into more detailed language-oriented text to describe system processes and provide information for hardware/software evaluations.

The following principles were selected as guiding criteria in choosing the mechanisms of the system design language. The include:

1) Clarity - The SDL must be able to clearly apply all concepts and constraints that the chosen design allows.

2) Conciseness - The SDL should be concise allowing designers and reviewers to obtain a maximum amount of information in a minimum amount of time. This usually implies that the fewer ways to say the same thing the better, and suggests special cases are to be avoided.

3) Lack of Ambiguity - The meaning of any SDL statement or construct must be clear and unambiguous. Any SDL construct or statement should have only a single interpretation.

4) Completeness - It is improtant to be able to express any concept or constraint the system requires. Any aid the SDL gives in ensuring all cases are covered is a plus.

5) Logic - The SDL should be capable of describing problems independent of solutions and solutions independent of low-level details like hardware without masking the requirements on the lower levels.

6) Ability to Express Relationships - The SDL should allow one to easily relate various levels of problem description and solution to higher and lower levels, ultimately including the code.

7) Ease of Modification - The SDL must allow easy modification of designs both during the design process as problems are uncovered and in development as changes are required.

Since the System Design Language is Ada-based, the use of Ada program units will be employed to accurately describe the structure of the system. To this end, the following guidelines in the use of the Ada program units at the system level are offered:

- Utilize packages to represent collections of logically related system activities. The items identified in any given package should be logically independent from those represented in other packages.
- Utilize procedures and functions to represent system activities which appear to be sufficiently distinct to act as stand-alone units but at the same time, dependent upon other stand-alone units to accomplish a more global activity.
- Utilize tasks to represent activities which can provide services to other program units in a concurrent fashion.

Based on these guidelines, it appears that the overall structure of the SDL should take advantage of Ada's "packaging" feature with the titles and numbers of the packages being identical to those found on the first level of the SED.

An example of the use of Ada's packaging feature applied to the SED shown in Figure 13 would result in a primitive SDL with the following structure.

```
procedure TSQ-73 is
 •••
   package COMMAND AND CONTROL (1.0) is
   •••
   end COMMAND AND CONTROL;
   package TARGET CONTROL (2.0) is
   •••
   end TARGET CONTROL;
   package REMOTE COMMUNICTION (3.0) is
   •••
   end REMOTE COMMUNICTION;
   package RADAR COMMUNICTION (4.0) is
   •••
   end RADAR COMMUNICATION;
end TSQ-73;
```

The package concept as used in the above example shows that four highly independent system activities have been identified. This is accomplished by searching the specification document to identify high-level groupings of functional areas. If at this point the designer feels that the functional areas which have been identified are not independent, other groupings should be investigated to achieve the highest degree of independence possible. When this result is achieved, the designer has used Ada program units (in this case packages) to describe the relationship among the highest level entities in the system.

In this form the SDL does not add any new information to that shown in graphical form in Figure 13. To overcome this limitation, Ada's comment feature will be employed. Comments will provide a description of each high level program unit which has been identified for system functions. The description should include the system requirements, capabilities, capacities, and constraints that are addressed by that particular program unit.

To illustrate the use of the comment feature a small portion of the comments for the remote communication package are given below.

```
package REMOTE COMMUNICATION (3.0) is
-- The objective of the REMOTE COMMUNICTION SYSTEM (3.0) is
--to enable the TSQ-73 to communicate digitally with various
--remote sites by means of the previously defined military
--protocols, TADIL-B, ATDL-1, and modified MBDL.  The remote
--communication subsystem shall provide for:
--     -2 group data links (ATDL-1)
--     -2 battalion data links (ATDL-1)
--     -2 remote radar data links (ATDL-1)
--     -1 tactical operations system data link (ATDL-1)
--     -1 air traffic management system data link (ATDL-1)
--     -1 inter-service communication data link (ATDL-1)
--     -4 fire unit data links (MBDL)
end REMOTE COMMUNICATION;
```

Comments appropriate to the functional nature of the other packages which have been identified would also be provided at this time.

It is important to reemphasize that the principles of the SED and SDL techniques are communicative in nature. Thus, the information presented at each level of each technique must be consistent and state precisely what the design requires.

The SDL generation proceeds to the next level of abstraction. At this level the functionality required to satisfy the objectives of each package are detailed as shown in Figure 14. For example, the functional areas identified for the Remote Communication (3.0) package are Communication Supervisor (3.1), Intra System Transfers (3.2), and Communication Protocols (3.3).

```
package REMOTE COMMUNICATION (3.0) is
--  The objective of the REMOTE COMMUNICATION SUBSYSTEM (3.0) is
--to enable the TSQ-73 to communicate digitally with various
--remote sites by means of the previously defined military
```

```
    --protocols, TADIL-B, ATDL-1, and modified MBDL.  The remote
    --communication subsystem shall provide for:
    --     -2 group data links (ATDL-1)
    --     -2 battalion data links (ATDL-1)
    --     -2 remote radar data links (ATDL-1)
    --     -1 tactical operations system data link (ATDL-1)
    --     -1 air traffic management system data link (ATDL-1)
    --     -1 inter-service communication data link (ATDL-1)
    --     -4 fire unit data links (MBDL)

    package COMMUNICATION SUPERVISOR (3.0) is
    --  The objectives of the Communication Supervisor (3.1) are:
    --1.  To provide a controlling interface between the command
    --     and control 1.0 package [which in turn handles communication
    --     between Target Control (2.0) and radar communcation (4.0)]
         and the remote communication package.
    --2.  To provide for the management of the remote communication
         package.
    end;

    package INTRASYSTEM TRANSFERS (3.2) is
    --
    --
    end;
    package COMMUNICATION PROTOCOLS (3.3) is
    --
    --
    end;
end REMOTE COMMUNICATION (3.0);
```

To complete the example of the specification portion for the communication supervisor package the items identified in the third level of abstraction are included [See Figure (15) for the corresponding SED].

```
package COMMUNICATION SUPERVISOR (3.1) is
-- The objectives of the COMMUNICATION SUPERVISOR (3.1) are:
--1. To provide a controlling interface between the command
     and control 1.0 package [which in turn handles commu-
     nication between TARGET CONTROL (2.0) and radar commu-
     nication (4.0)] and the remote communication package.
--2. To provide for the management of the remote communication
     package.

     procedure Consume Internal Message (3.1.1);
     procedure Initialize Remote Communication (3.1.2);
     procedure Manage Data Links (3.1.3);
     procedure Manage Messages (3.1.4);
     procedure Manage Fault Detection, Correction & Reporting
               (3.1.5);
     procedure Terminate Remote Communication (3.1.6);
     procedure Produce Internal Message (3.1.7);
end;
```

Based on the guidelines presented previously, the procedures identified in the communication supervisor package are sufficiently distinct to be stand-alone units but are dependent upon the other procedures in that package to satisfy the objective of the more global communication supervisor functional area. In contrast, the communication supervisor is independent of the intrasystem transfers and communication protocol activities (hence utilization of the package constructs) while all three are needed to satisfy the Remote Communication objective. The Remote Communication package is also independent of Command and Control, Target Control and Radar Communication, the combination of which meet the mission of the A/N (TSQ-73).

If more information is required to detail the design concepts, the option of providing the package body is available. The package body provides the logic required to describe (not implement) the procedures identified in the specification portion as well as

other local procedures, functions, or tasks as necessary to meet the requirements of those procedures.

An example of this guideline is shown below using the portion of the package body for communication supervisor which pertains to the procedure CONSUME INTERNAL MESSAGE (3.1.1) follows.

```
procedure CONSUME INTERNAL MESSAGE (3.1.1) is
begin
      if Command and Control Message or Communication
           Protocol Message = then
           case Message Type is
               when Initialize Remote Communication =>
                 Queue Message to Initialize Remote
                 Communication (3.1.2);
               when Manage Data Links =>
                 Queue Message to Manage Data Links (3.1.3);
               when Manage Messages =>
                 Queue Message to Manage Messages (3.1.4);
               when Others =>
                 set error code for invalid message type;
                 queue to produce Internal Message (3.1.7);
           end case;
      else
           set error code for invalid message originator;
           queue to produce Internal Message (3.1.7);
      end if;
end CONSUME INTERNAL MESSAGE;
```

The previous example illustrates the structural features of the SDL and, in addition shows that system control, system data flow, and system exception handling can be illustrated as well.

Further description of more specific items found at lower levels of abstraction can be described using the strong typing features of Ada.

The example shown in Figures 19 and 20 represent how system requirements, capabilities, capacities, and constraints can be described in the SDL using these strong typing features. A package of system level data types can serve as a collection mechanism for these items.

To aid in the use of the SDL described here the following guidelines are offered.

1. Utilize all features of the Ada language as necessary to provide the required system descriptions, while maintaining a structured approach. Avoid the use of the GO TO construct.
2. Use the Ada program units according to the following criteria:
   - Utilize packages to represent collections of logically related system activities. The items identified in any given package should be logically independent from those represented in other packages.
   - Utilize procedures and functions to represent system activities which appear to be sufficiently distinct to act as stand-alone units but at the same time, dependent upon other stand-alone units to accomplish a more global activity.
   - Utilize tasks to represent activities which can provide services to other program units in a concurrent fashion.
3. Proceed to a level of detail which allows hardware/software trade-offs to be made.
4. Provide appropriate comments for each high level program unit.
5. Generate the SDL simultaneously with the SED using each technique to check the other.
6. Maintain consistency with numbering conventions.
7. Keep in mind that the SDL is a communicative device, the purpose of which is to describe system functionality.
8. Do not attempt to write executable source code.

```
--Support will be maintained for 13 active data links
--plus an on-line redundant data transmission link
--for each active link.  Thus, a total of 26 active
--links with a minimum of two spare links will be re-
--quired.  Each link will be capable of utilizing baud
--rates of 600 or 1200 bps, and will support the pre-
--defined TADIL-B, ATDL-1, and MBDL military protocols.

type PROTOCOL CHOICE is (TADIL-B, ATDL-1, MBDL);
     BAUD RATE CHOICE:constant array (PROTOCOL CHOICE) of
     integer := (1200, 1200, 600);
type ORIENTATION CHOICE is (BIT ORIENT, CHAR ORIENT);
type TRANSMISSION CHOICE is (PT TO PT, MULTI DROP);
type SYNCH CHOICE is (SYNCHRONOUS, ASYNCHRONOUS);
type MODE CHOICE is (PRIMARY, REDUNDANT);

type PRI MBDL LINK SPEC is
     record
         PROTOCOL : PROTOCOL CHOICE := MBDL;
         BAUD RATE : INTEGER := BAUD RATE CHOICE
           (PROTOCOL);
         ORIENTATION : ORIENTATION CHOICE := CHAR ORIENT;
         TRANSMISSION : TRANSMISSION CHOICE := PT TO PT;
         SYNCH : SYNCH CHOICE := SYNCHRONOUS;
         MODE : MODE CHOICE := PRIMARY;
     end record;
```

Figure 19

```
type RED MBDL LINK SPEC Is
    record
       PROTOCOL : PROTOCOL CHOICE := MBDL;
       BAUD RATE : INTEGER := BAUD RATE CHOICE
         (PROTOCOL);
       ORIENTATION : ORIENTATION CHOICE := CHAR ORIENT;
       TRANSMISSION : TRANSMISSION CHOICE := PT TO PT;
       SYNCH : SYNCH CHOICE := SYNCHRONOUS;
       MODE : MODE CHOICE := REDUNDANT;
    end record;



PRIMARY MBDL LINK : PRI MBDL LINK SPEC range 1...4;

REDUNDANT MBDL LINK : RED MBDL LINK SPEC range 1...4;
```

Figure 20

DATA FLOW DIAGRAM (DFD)

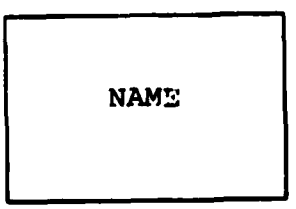
In order to obtain a more distinct picture of the system's functions and logical flow of data, a graphical abstraction system flowchart known as a data flow diagram is utilized (39,40,41,42). Data flow diagrams attempt to display compact generalized pictures of functional requirements and their accompanying data transformations. This technique, also known as a "bubble chart", models the system in terms of the following:

- Input Data
- Output Data
- Stored Data
- Significant Intermediate Data Forms
- Data Transformations

Although DFDs generally occur on many conceptual levels, the initial level attempts to show on one page the major processes required in the system and is known as the context diagram. Consequently, only those details which are relevent to DFDs should be present at this level of abstraction. The emphasis should be to identify and isolate major functions and resultant data transformations. Each transformation may eventually become one or more modules at a lower level, but the DFD should not provide details of procedural processes or of implementation.

Certain basic graphic symbols are used to identify components of the DFD. These are described as follows:

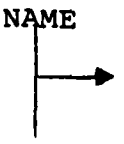
The Source or Sink is the interface to the external world through which data enters or leaves the system. This is required by a labeled rectangle. Example:



The Data Flow identifies information moving from one part of the system to another. It should not "pass through" intermediate processes. The data flow acts in a manner similar to a queue in that information first in is first out. Care should be taken to choose descriptive data flow names. The data flow is represented by a labeled straight line with an arrowhead on at least one end which indicates the direction of the flow. Example:

A File is a place where data resides for possible future use.

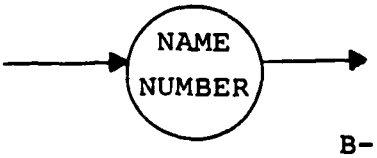
A file is shown as a labeled line connected by a perpendicular arrow. Example:



A Transformation is the most basic symbol of the data flow diagram. It identifies a function that transforms data. It can represent changes in any of the following:

- Value
- Format
- Meaning
- Residence

Transformations are stated in terms of simple sentences indicating input and output. The basic form is an imperative sentence consisting of a "meaningful" verb and a direct object. It is graphically represented by a circle with arrows indicating input and output. Example:



As mentioned earlier, data flow diagrams can have different levels of abstraction. Thus, data flow diagrams can be expanded by choosing one transformation and developing another diagram using the same process.

In summary, the following guidelines are to be used when creating DFDs and representing different levels of abstraction:

1. Utilize simple sentences in transformation bubble.
2. Number transformation bubble reflecting logical level.
3. Capture all types of input and output from transformation bubble.
4. Avoid detailed control logic representation.
5. Choose data flow names that are descriptive.
6. Attempt to limit diagrams to seven transformations.
7. Seek a balanced workload for transformations.
8. Break transformations down to simplest components by additional levels.
9. Illustrate boundaries of the system when applicable.

An illustrative data flow diagram is shown in Figures 21-29 and is an extension of the remote communications example begun at the SED and SDL level.

TITLE: 3.0 REMOTE COMMUNICATIONS DFD		PAGE: 300
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/14/82
NOTES:		ROOT: 3.0

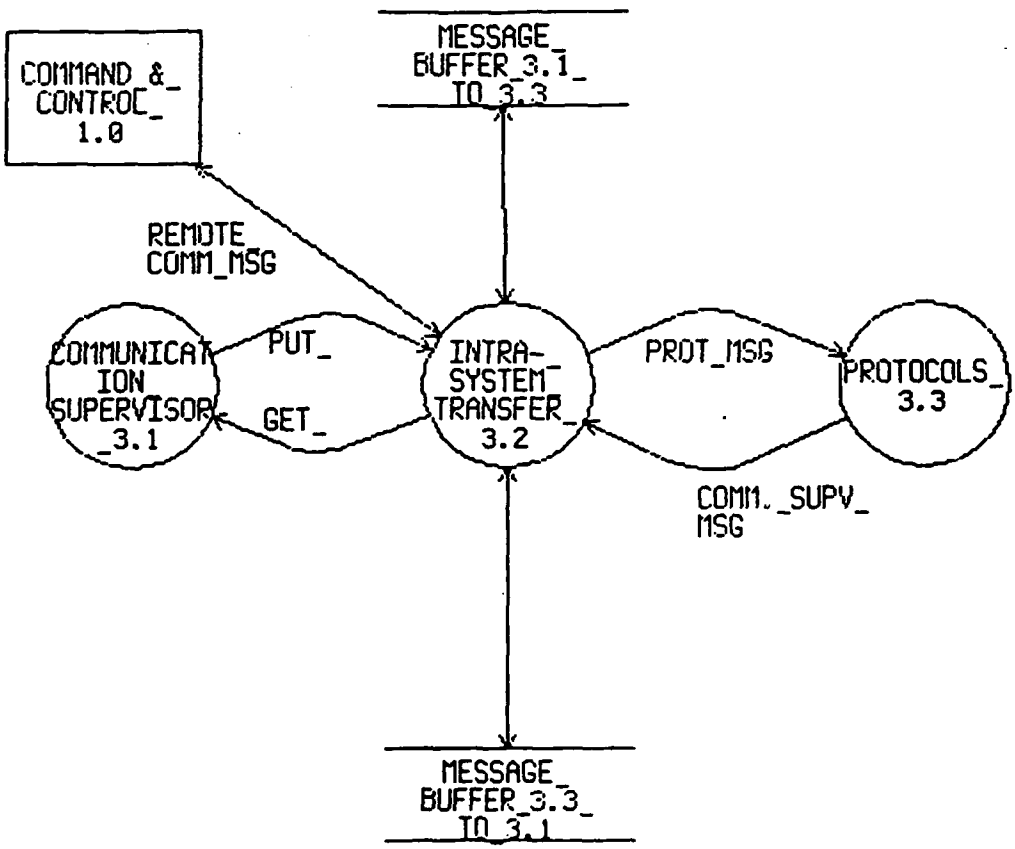


Figure 21 - Data Flow Diagram Example

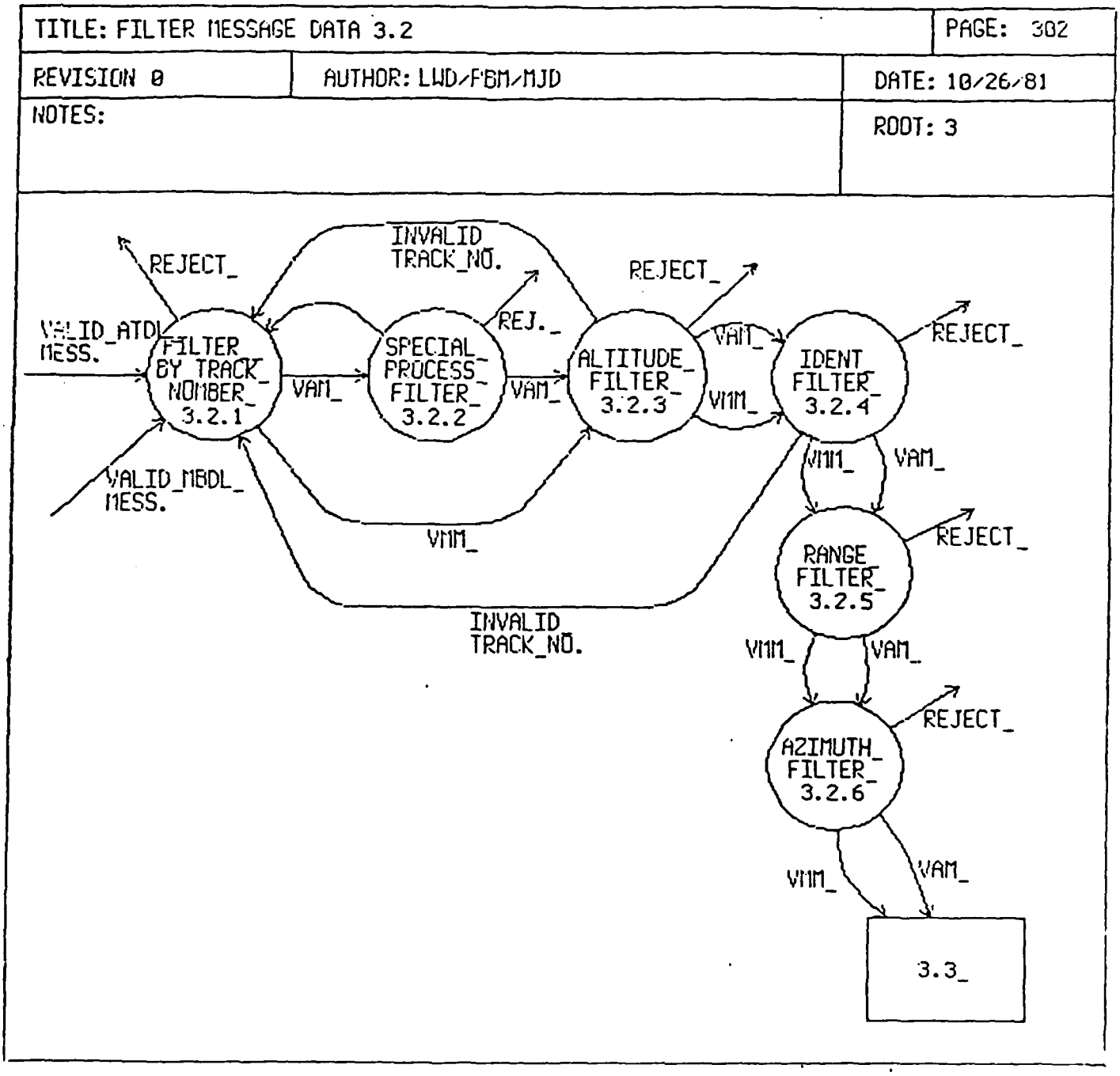


Figure 22 - Data Flow Diagram Example

TITLE: 3.3 PROTOCOLS DFD		PAGE: 302
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/28/82
NOTES:		ROOT: 3.0

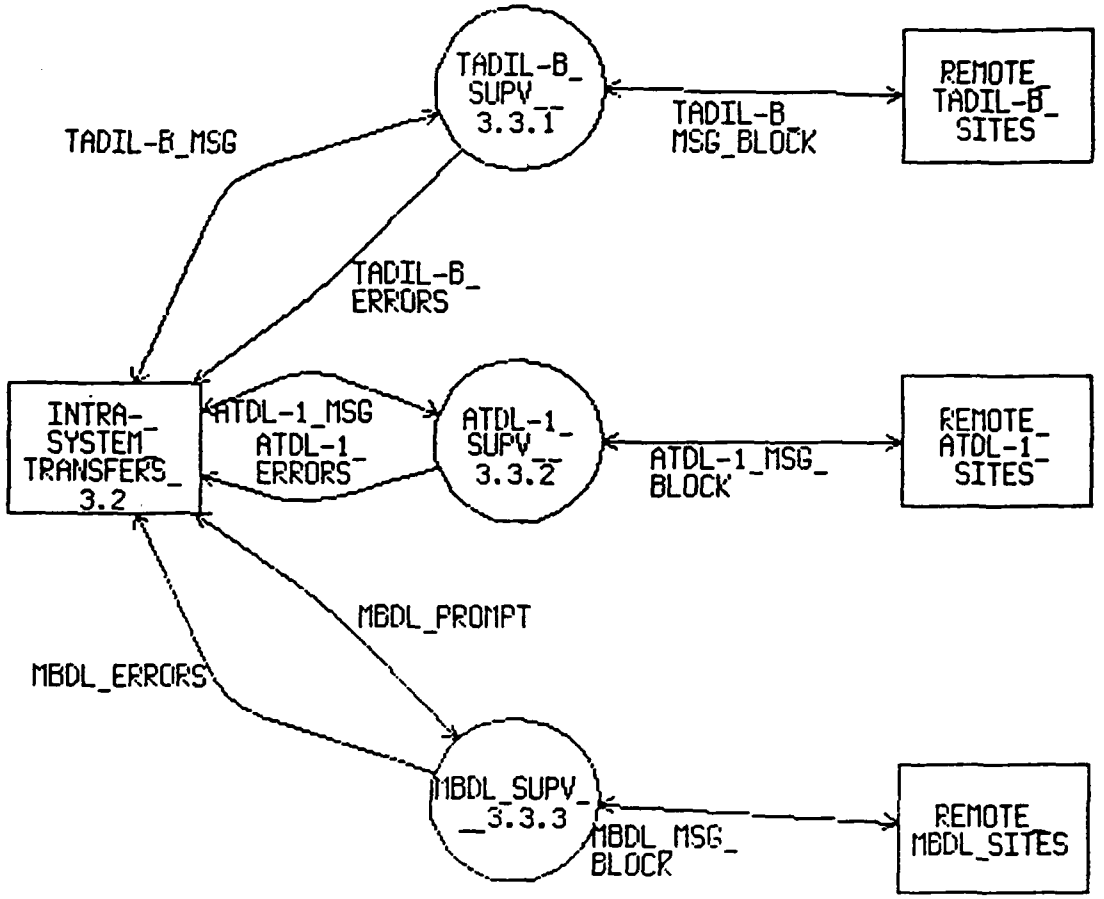


Figure 23 - Data Flow Diagram Example

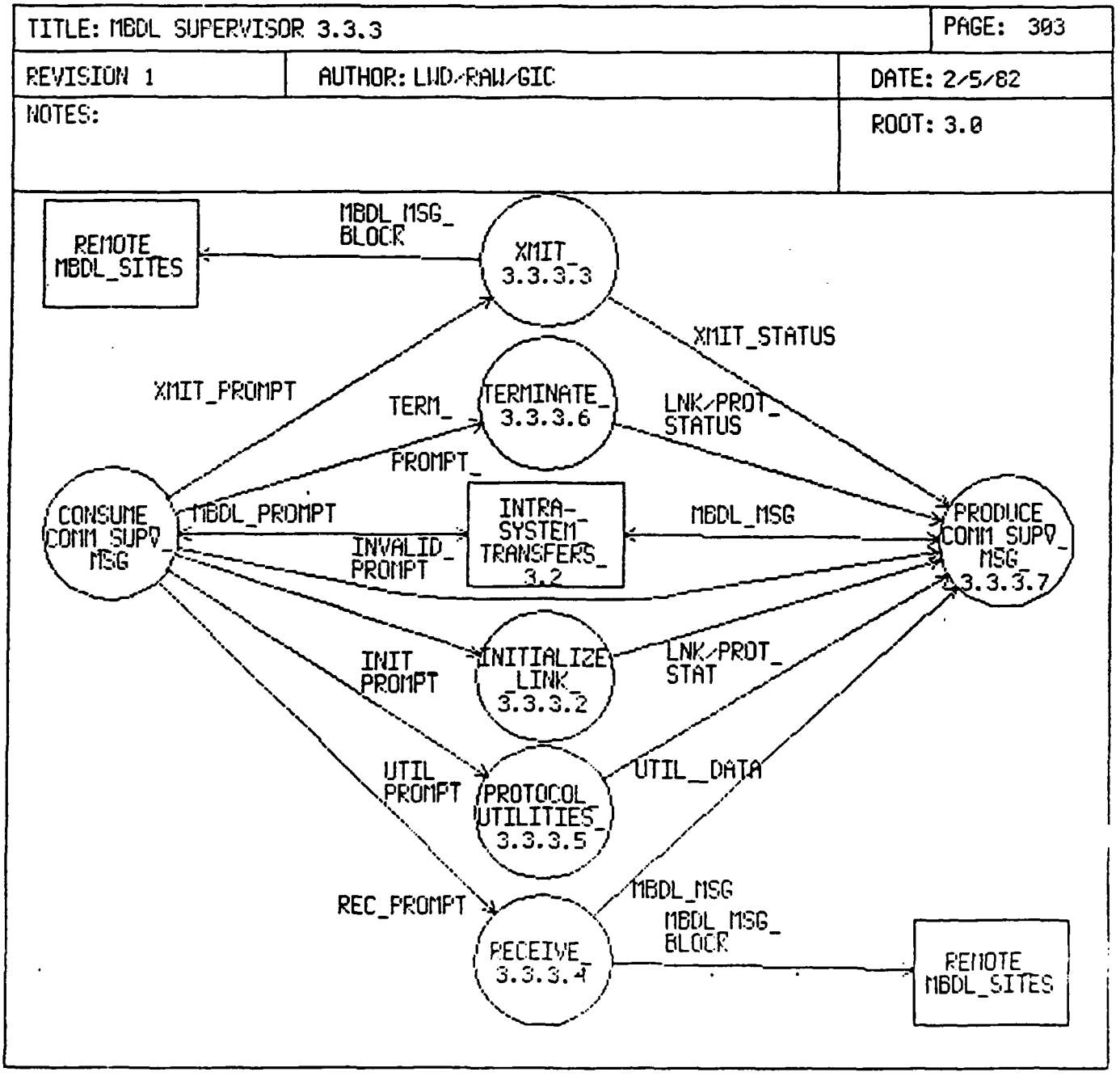


Figure 24 - Data Flow Diagram Example

TITLE: 3.3.3.3 XMIT MESSAGE DFD		PAGE: 304
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 2/5/82
NOTES:		ROOT: 3.0

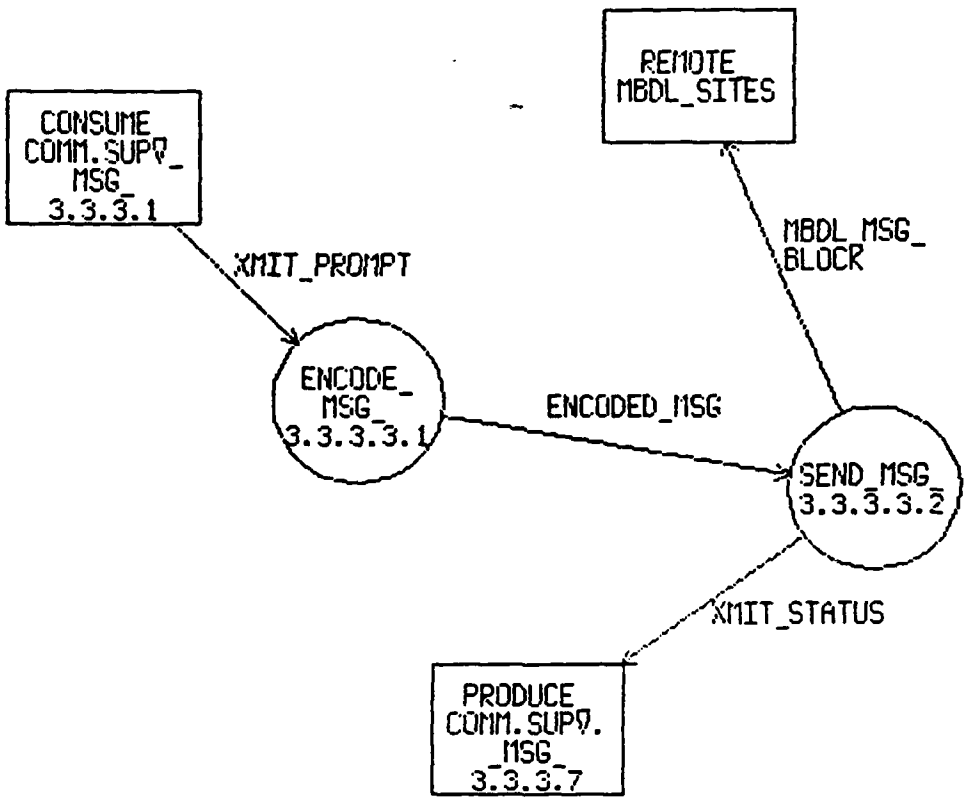


Figure 25 - Data Flow Diagram Example

TITLE: 3.3.3.3.2 SEND MESSAGE DFD		PAGE: 305
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 2/5/82
NOTES:		ROOT: 3.0

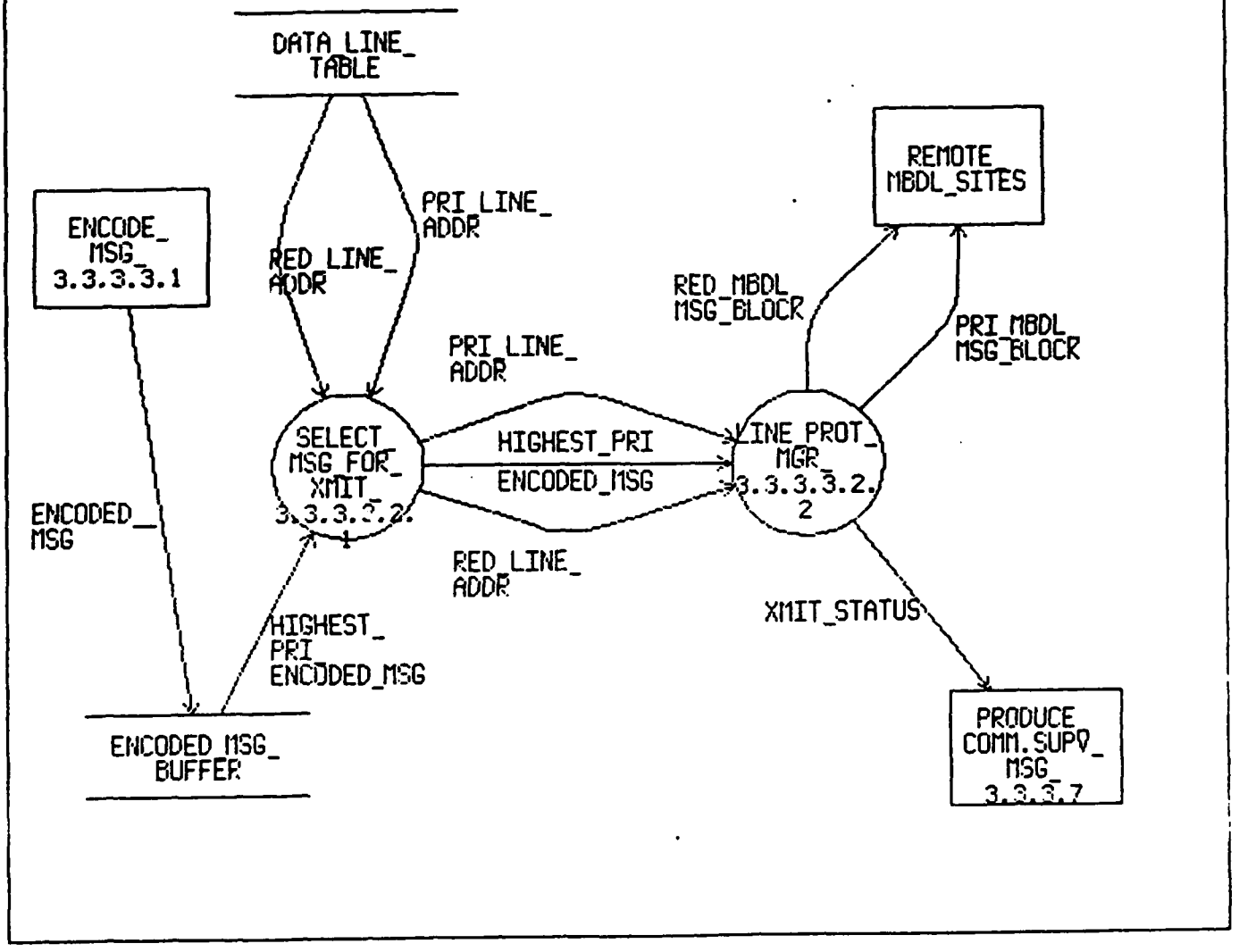


Figure 26 - Data Flow Diagram Example

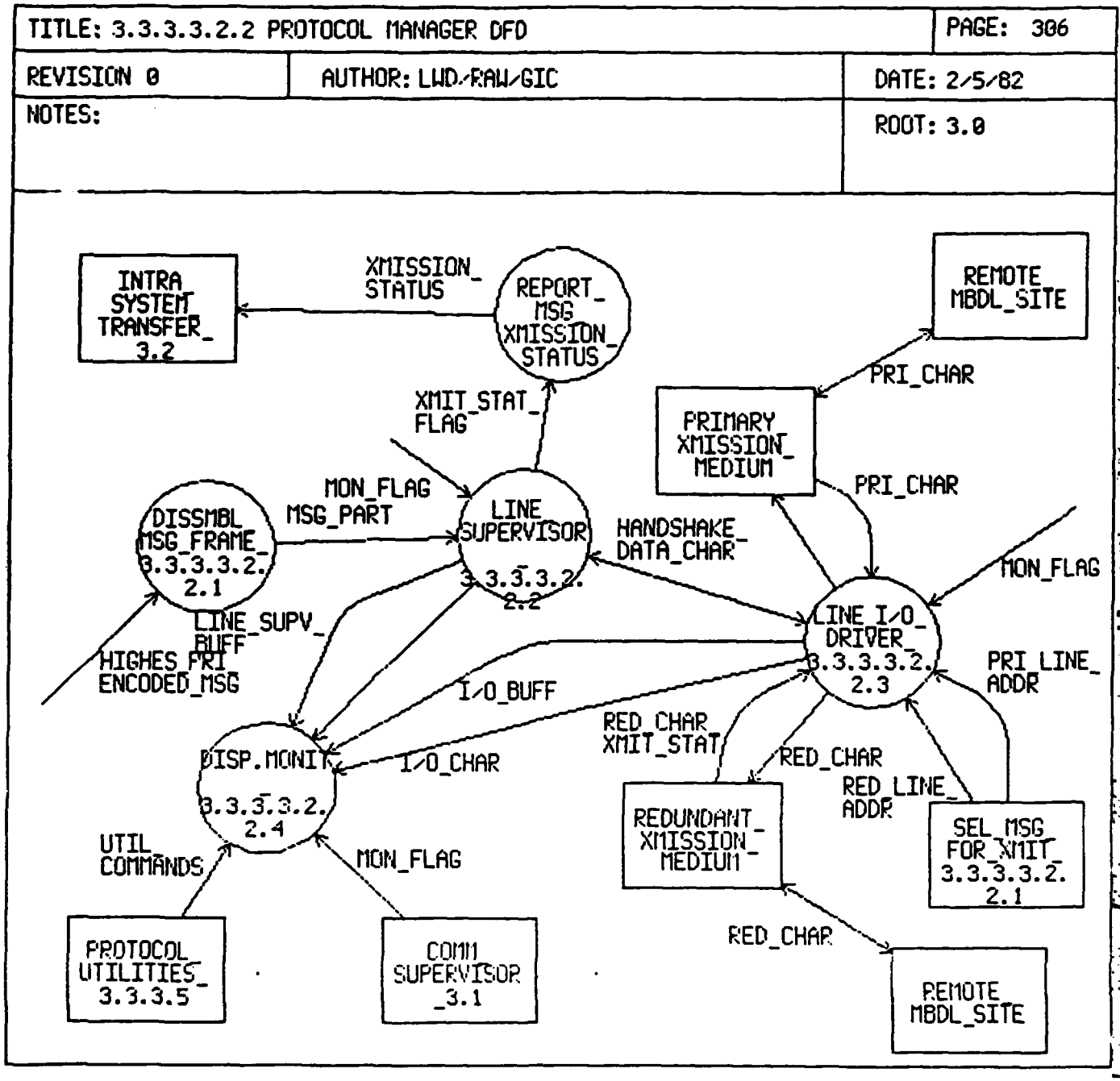


Figure 27 - Data Flow Diagram Example

TITLE: 3.3.3.3.2.2.4 DISPLAY MONITOR DFD		PAGE: 307
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 2/5/82
NOTES:		ROOT: 3.0

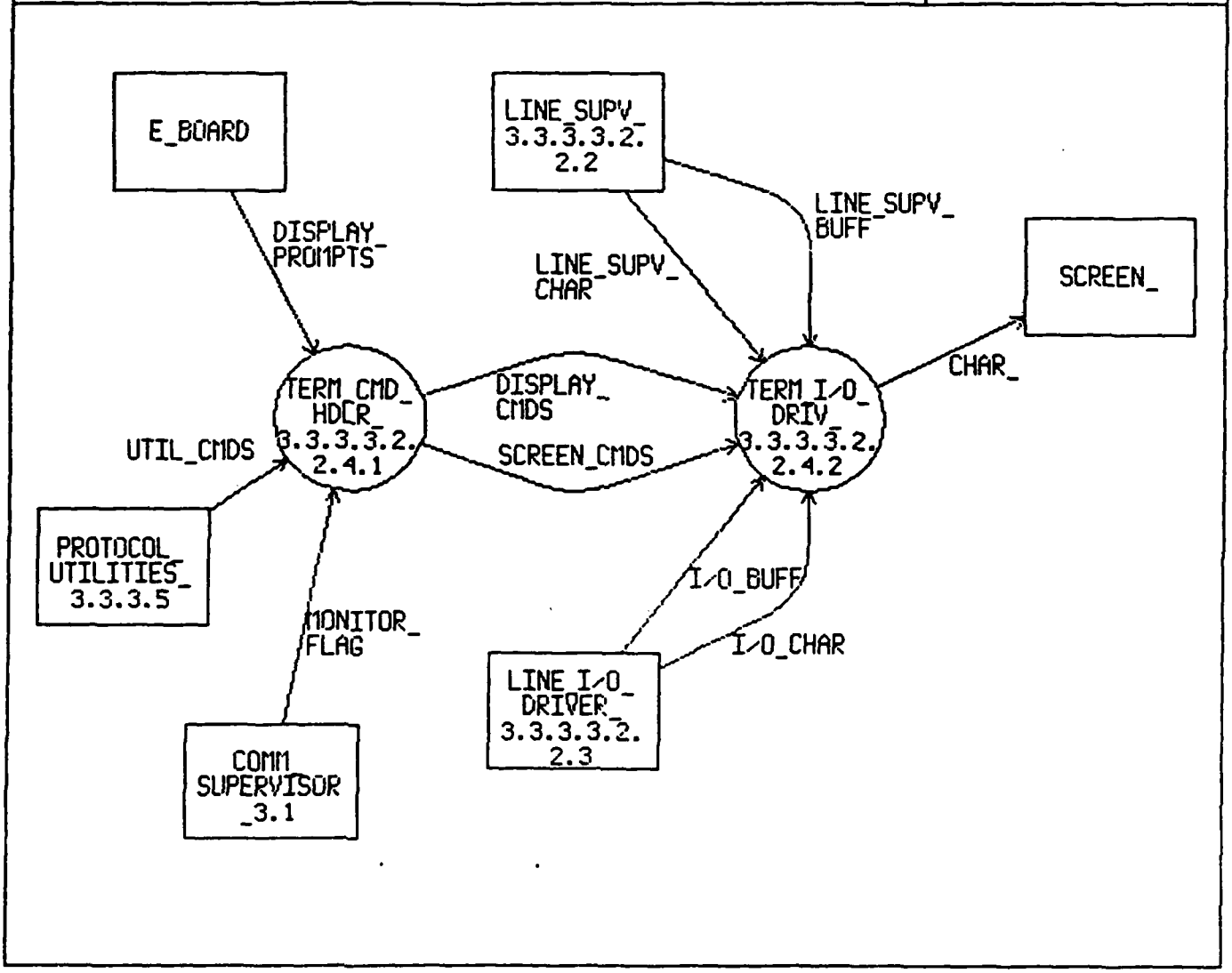


Figure 28 - Data Flow Diagram Example

TITLE: 3.3.3.3.2.2.4.2 TERMINAL I/O DRIVER DFD		PAGE: 308
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 2/8/82
NOTES:		ROOT: 3.0

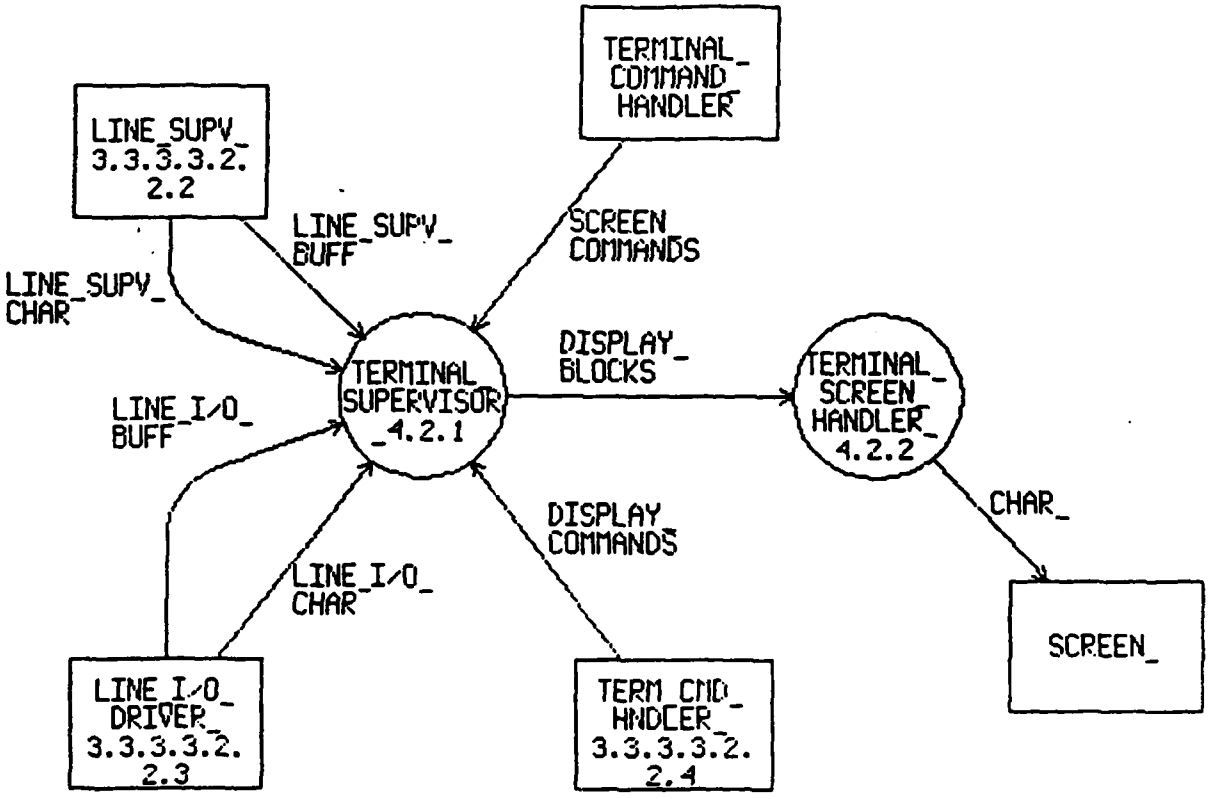


Figure 29 - Data Flow Diagram Example

DATA DICTIONARY

In utilizing data flow diagrams and subsequently, structure charts as software design techniques, the system designer generates a vast number of terms associated with data or processes. Some words by their nature are ambiguous while others are esoteric. The use of acronyms or aliases does little to reveal complete understanding of a data flow, yet is commonplace at this design stage. It is, therefore, essential to have documentation that addresses the problems arising when multiple interpretations are given to a single term and to add awareness of how apparently minor changes can impact the system.

The data dictionary is a design technique which helps to meet these objectives. Used in parallel with data flow diagrams and updated as required when structure charts are created, the data dictionary serves as a central repository for data definitions and process descriptions. All data flows, files and transformations from the data flow diagrams will be defined in the dictionary, thus ensuring a complete glossary of data items and transformation descriptions. Data items on the structure chart will also appear in the dictionary if different from data on the flow diagrams.

The automated data dictionary tool developed by Control Data will be utilized for this application. Entries are made in a prescribed format. The automated tool eliminates the drudgery involved in finding aliases, cross-referencing modules impacted by changes in data items and locating recursive definitions. In particular, the cross-referencing capability helps to ease the ripple-effect of changes made at a later date. Below are the formats of the entries and some general guidelines for creating the dictionary.

Entries can be of two types: Data Definitions and Action Definitions.

DATA DEFINITIONS' FORMAT

```
DATA HEADER:
--DATA ATTRIBUTES;
DATA COMPOSITION;
```

The DATA HEADER contains the name and type. The type may be either FLOW, FILE, or (all others) ELEMENT.

The DATA ATTRIBUTES are comments written in English to explain the usage or to clarify an item. They are optional.

The DATA COMPOSITION contains COMPONENTS and META-SYMBOLS. COMPONENTS are FLOW, ELEMENTS or FILES.

The META-SYMBOLS are:			
	CONJUNCTION	&	(and)
	ALTERNATION	|	(or)
	ITERATION	{ }	(repeat)
	OPTIONAL	[ ]	

Example:

```
OPERATOR_MESSAGE (FLOW):                                        HEADER
-THESE MESSAGES ARE MANUALLY GENERATED BY THE OPERATOR(S)  ATTRIBUTE
{COMMAND|REQUEST}!                                          COMPOSITION
```

OPERATOR_MESSAGE here is defined to be a series of commands or requests. The exclamation point indicates the end of the definition. The terms COMMAND and REQUEST would also be entries in the data dictionary. All FLOW and FILE types would eventually be defined in terms of ELEMENT types. ELEMENT types should be reduced to self-defining ELEMENT types and then should be elaborated on with the attribute or comment section of the definition. In this way an individual unfamiliar with the definition of a term can trace its meaning through successive levels to its exact composition if necessary.

Some data definitions' guidelines follow:

- Words are ambiguous, select the best one that can be found and define it precisely.
- Avoid circular definitions.
- Reduce everything to self defining terms.
- Define self defining terms with comments.
- Entries are defined in terms of composition; to show usage use comments.
- Upon completion, walkthrough.

ACTION DEFINITIONS' FORMAT

```
ACTION HEADER:
--ACTION ATTRIBUTES;
ACTION FLOW;
```

The ACTION HEADER contains name and type and number corresponding to the "bubble" (the type is process).
-- A transformation is a process on the DFD level.
-- A procedure is an example of a module on SC level.

The ACTION ATTRIBUTES are comments and are used like the DATA ATTRIBUTES.

The ACTION FLOW contains flow descriptions and comments written in a pseudo code.

Example:
VALIDATE_MESSAGE is a bubble from a data flow diagram and is enumerated as 3.1.

```
VALIDATE_MESSAGE (PROCESS 3.1)
IF MESSAGE IS MBDL
THEN
  PERFORM PARITY_CHECK
ELSE
  PERFORM CHECKSUM;
```

```
UPDATE TABLE;
END IF;
IF MESSAGE_COUNT IS >N THEN
-- N IS NUMBER OF ACCEPTABLE FAILURES
   SEND_ALERT;
END IF;!
```

Some action flow guidelines:

- Use nouns from data dictionary.
- Use specific descriptive verbs (not do, process, handle).
- Avoid adjectives and adverbs.
- Tell policy not procedure.
- Refer to input and output.
- Remember rejects.

In summary, the data dictionary helps to achieve the following objectives:

1. Establish a glossary of terms.
2. Provide standard terminology.
3. Define all data on data flow diagrams.
4. Define all transformations on data flow diagrams.
5. Provide the capability for cross-referencing.
6. Resolve the problems of aliases and acronyms.
7. Help reduce maintenance costs.

The Data flow diagram and associated data dictionary entries shown in Figures 30 thru 34 serve to illustrate the data dictionary tools (flow, process, file descriptions, cross-referencing and cross-checking) as they apply to the given data flow.

```
MBDL_MSG_BLOCK (FLOW):

        'FORMAT FOR MBDL BLOCK'

        ; -CHARACTERS; -MESSAGE LENGTH = 128; (SYNC_ &
        SYNC_ &
        S_O_H &
        DESTINATION_STATION_ID &
        DESTINATION_PERIPHERAL_ID &
        ORIGINATION_STATION_ID ORIGINATION_PERIPHERAL_ID &
        ORIGINATION_TIME &
        TRANSMISSION_TIME &
        MESSAGE_SECURITY MESSAGE_PRIORITY &
        MESSAGE_TYPE &
        ORIGINATOR_SERIAL_NUMBER &
        S_T_X &
        TEXT_LENGTH &
        TRANSACTION_COMMAND &
        TARGET_IDENTITY &
        HEIGHT_DATA &
        X_COORDINATE_DATA &
        Y_COORDINATE_DATA &
        ALERT_STATUS &
        E_T_X &
        CHECK_SUM &
        F_F &
        E_O_T &
        RED_LINE_ADDR &
        PRI_LINE_ADDR)!

MESSAGE_PRIORITY (ELEMENT):

        - CHARACTER POSITION = 19;
        - INTEGER;
        'INDICATES PRIORITY OF MESSAGE IN
            ENCODED_MESSAGE_BUFFER'

        ; (1|
        2|
        3|
        4|
        5|
        6|
        7|
        8|
        9);!
```

Figure 30 - Data Dictionary Example (Flow)

```
  3.1 LINE_PROTOCOL_MANAGER (PROCESS 3.3.3.3.2.2):

    GET PRI_LINE_ADDR;
    GET RED_LINE_ADDR;
    GET HIGHEST_PRIORITY_ENCODED_MSG;
    PUT MSG BLOCK TO DESTINATION ON PRIMARY LINE;
    PUT MSG BLOCK TO DESTINATION ON REDUNDANT LINE;
    POST XMIT_STATUS!


  3.2 SELECT_MESSAGE_FOR_XMIT (PROCESS 3.3.3.3.2.1):

    - MESSAGE COUNT IN BUFFER (0..N);

    IF MESSAGE COUNT IN BUFFER =/ 0 THEN:
      FOR EACH MESSAGE LOOP:
        DETERMINE PRIORITY;
        STORE PRIORITY.
      END LOOP.
    ELSE:
      DO NOTHING..
    END IF;
    RETURN MESSAGE WITH HIGHEST PRIORITY;!
EOI ENCOUNTERED.
```

Figure 31 - Data Dictionary Example (Process)

```
ACK_0 (ELEMENT):

        (ASCII ACK &
        0); 'PRIMARY LINE ACKNOWLEDGEMENT'!


ACK_1 (ELEMENT):

        (ASCII ACK &
        1); 'REDUNDANT LINE ACKNOWLEDGEMENT'!


ALERT_STATUS (ELEMENT):

        - CHARACTER POSITION = 46;
        - INTEGER;

        (DIGIT)!


BAUD_RATE (ELEMENT):

        {300|
        600|
        750|
        1200|
        2400|
        4800}!


CHECK_SUM (ELEMENT):

        - CHARACTER POSITION = (48+N);
        - INTEGER;

        (DIGIT)!


DATA_LINE_TABLE (FILE):

        DESTINATION_STATION_ID &
        (PROTOCOL_ASSIGNMENT &
        BAUD_RATE &
        PRI_LINE_ADDR &
        RED_LINE_ADDR)!


DESTINATION_PERIPHERAL_ID (ELEMENT):

        - CHARACTER POSITION = (6..7);
        - INTEGER;

        (DIGIT &
        DIGIT)!
```

Figure 32 - Data Dictionary Example (File Description)

```
MBDL_MSG_BLOCK (FLOW):

                ~~ Makes References To ~~
        ALERT_STATUS . . . . . (ELEMENT)
        CHECK_SUM . . . . . (ELEMENT)
        DESTINATION_PERIPHERAL_ID . . . (ELEMENT)
        DESTINATION_STATION_ID . . . . (ELEMENT)
        E_O_T . . . . . (ELEMENT)
        E_T_X . . . . . (ELEMENT)
        F_F . . . . . (ELEMENT)
        HEIGHT_DATA . . . . . (ELEMENT)
        MESSAGE_PRIORITY . . . . . (ELEMENT)
        MESSAGE_SECURITY . . . . . (ELEMENT)
        MESSAGE_TYPE . . . . . (ELEMENT)
        ORIGINATION_PERIPHERAL_ID . . . (ELEMENT)
        ORIGINATION_STATION_ID . . . . (ELEMENT)
        ORIGINATION_TIME . . . . . (ELEMENT)
        ORIGINATOR_SERIAL_NUMBER . . . (ELEMENT)
        PRI_LINE_ADDR . . . . . (ELEMENT)
        RED_LINE_ADDR . . . . . (ELEMENT)
        SYNC_ . . . . . (ELEMENT)
        S_O_H . . . . . (ELEMENT)
        S_T_X . . . . . (ELEMENT)
        TARGET_IDENTITY . . . . . (ELEMENT)
        TEXT_LENGTH . . . . . (ELEMENT)
        TRANSACTION_COMMAND . . . . . (ELEMENT)
        TRANSMISSION_TIME . . . . . (ELEMENT)
        X_COORDINATE_DATA . . . . . (ELEMENT)


                ~~ Is Referenced By ~~
        HIGHEST_PRIORITY_ENCODED_MSG . (FLOW)
        PRI_MBDL_MSG_BLOCK . . . . . (FLOW)
        RED_MBDL_MSG_BLOCK . . . . . (FLOW)
```

Figure 33 - Data Dictionary Example (Cross Reference)

```
~~ Undefined Names ~~


                NO UNDEFINED ENTRIES FOUND
            CROSS CHECK REPORT FOR DD 'DRAFT'

~~ Entries Which Are Not Referenced By Any Other Entry ~~


        DATA_LINE_TABLE . . . . . (FILE)
        ENCODED_MESSAGE_BUFFER  . . . . (FILE)
        LINE_PROTOCOL_MANAGER . . . . . (PROCESS 3.3.3.3.2.2)
        PRI_MBDL_MSG_BLOCK  . . . . . (FLOW)
        RED_MBDL_MSG_BLOCK  . . . . . (FLOW)
        SELECT_MESSAGE_FOR_XMIT . . . . (PROCESS 3.3.3.3.2.1)

            6 UNREFERENCED ENTRIES FOUND
            CROSS CHECK REPORT FOR DD 'DRAFT'

~~ Entries Which Do Not Make Reference To Any Other Entry ~~


        ENCODED_MESSAGE_BUFFER  . . . . (FILE)
        SELECT_MESSAGE_FOR_XMIT . . . . (PROCESS 3.3.3.3.2.1)

            2 NOREF ENTRIES FOUND
            CROSS CHECK REPORT FOR DD 'DRAFT'

            ~~ Recursively Defined Entries ~~


            NO RECURSIVELY DEFINED ENTRIES FOUND
EOI ENCOUNTERED.
```

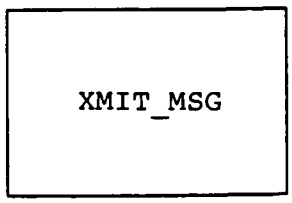
Figure 34 - Data Dictionary Example (Cross Checking)

STRUCTURE CHART

As its name implies, the structure chart is a software design technique that graphically presents data, control communication and modules required for program units to work. The structure chart accomplishes two major functions: (1) it partitions the implied software described by the data flow diagrams and data dictionary into a hierarchy of program units, each of which is viewed as performing a well defined function in the program, and (2) it establishes the means and direction of communication between program units.

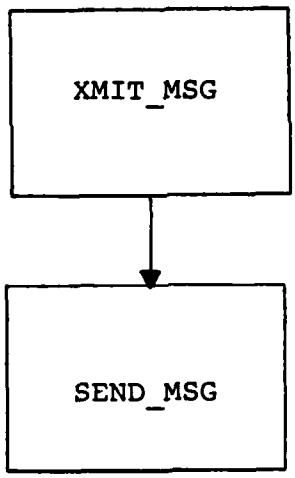
In keeping with the overall goal of step-wise refinement, the structure chart takes the design one step closer to a coding solution of the original problem by logically viewing each module in its proper place in the emerging software. The logical view is preserved because, in order to create the structure chart, a designer must decide what functions are needed to accomplish the transformations specified in the data flow diagram. These functions were not viewed as any particular physical entity, e.g., a subprogram, they are simply logical activities that must take place in order to take the system from one data flow bubble to another. These functions, and whatever subfunctions they may in turn be composed of, become the module or program unit upon which the structure is based.

The structure chart uses a symbology that is restricted to representation of program units, the logical relationship among program units and the communication between them. The symbol for a program unit is a rectangle identified by a functional name.

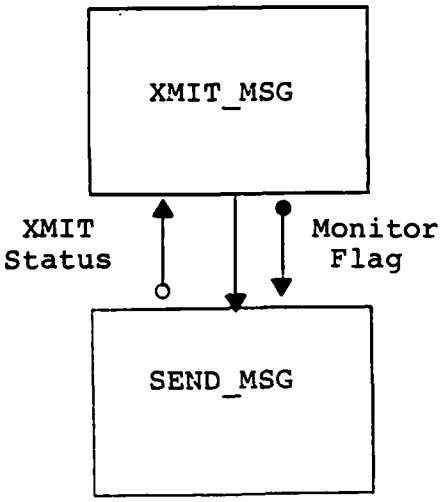


The program unit name should be comprised of a descriptive verb and ideally, a single, nonplural object identifying the total function of the module or program unit. Use of vague, general terms like "process" should be avoided in program unit names. At the structure chart level a specific, non-general, function is desired. The use of vague terms in a program unit name is a sign that the actual functional make-up of the transform has not been clearly understood and the named program unit just parrots the data flow diagram transformation.

Connectivity between program units is indicated by an arrow that is drawn from the edge of one program unit to the edge of the other. Even though the arrow head points in only one direction, from the calling to the called program unit, it is understood that the called program unit returns to the calling program unit.



When one program unit calls another, the calling program unit may send data or control information. The called program unit may produce data and/or control information which in turn is passed back to the calling program unit. For example, the following two program units pass information in the following manner:



The solid circle tail of the arrow corresponds to control communications and data communications is shown by an open circle tail.

A structure chart is produced by decomposing a data flow diagram/ data dictionary into its functional parts, deciding on relationships and communication among parts and finally graphically representing the resulting information.

To initiate this technique, the data flow diagram is analyzed to identify the central transformation or transformations, or in other words, where the data is at its most abstract or highly processed form. This section becomes the second level of the structure chart. The first level is completed by adding a control module. Once these upper levels of the chart have been achieved, the lower levels can be derived by top-down refinement.

The process of decomposing a data flow diagram transform into its equivalent set of structure chart boxes is of singular importance at this state of design. The functions, relationships, and communication produce the software tree structure identifying program units that will be designed and coded. Scheduling and management of program units can begin at this point.

The following guidelines are recommended in the usage of the structure chart design tool:

- Unless otherwise indicated a program unit will be assumed to have a single entry point and a single exit point.
- Program units will be represented by a rectangle.
- Program units should be identified by a functional name comprised of a descriptive verb.
- Representation of program units will be in a hierarchical order. The high level modules provide data to and activate the low level program units.
- Depending on the programming language, program units can appear as:
  - Packages
  - Tasks
  - Procedures
  - Functions
  - Subroutines
  - Macros
  - Programs
  - Subprograms
- Graphics should show:
  - Program units
  - Calls between these units
  - Data shared between them

As shown in Figures 35 thru 37, an automated structure chart tool is available for use on the project to support this software technique.

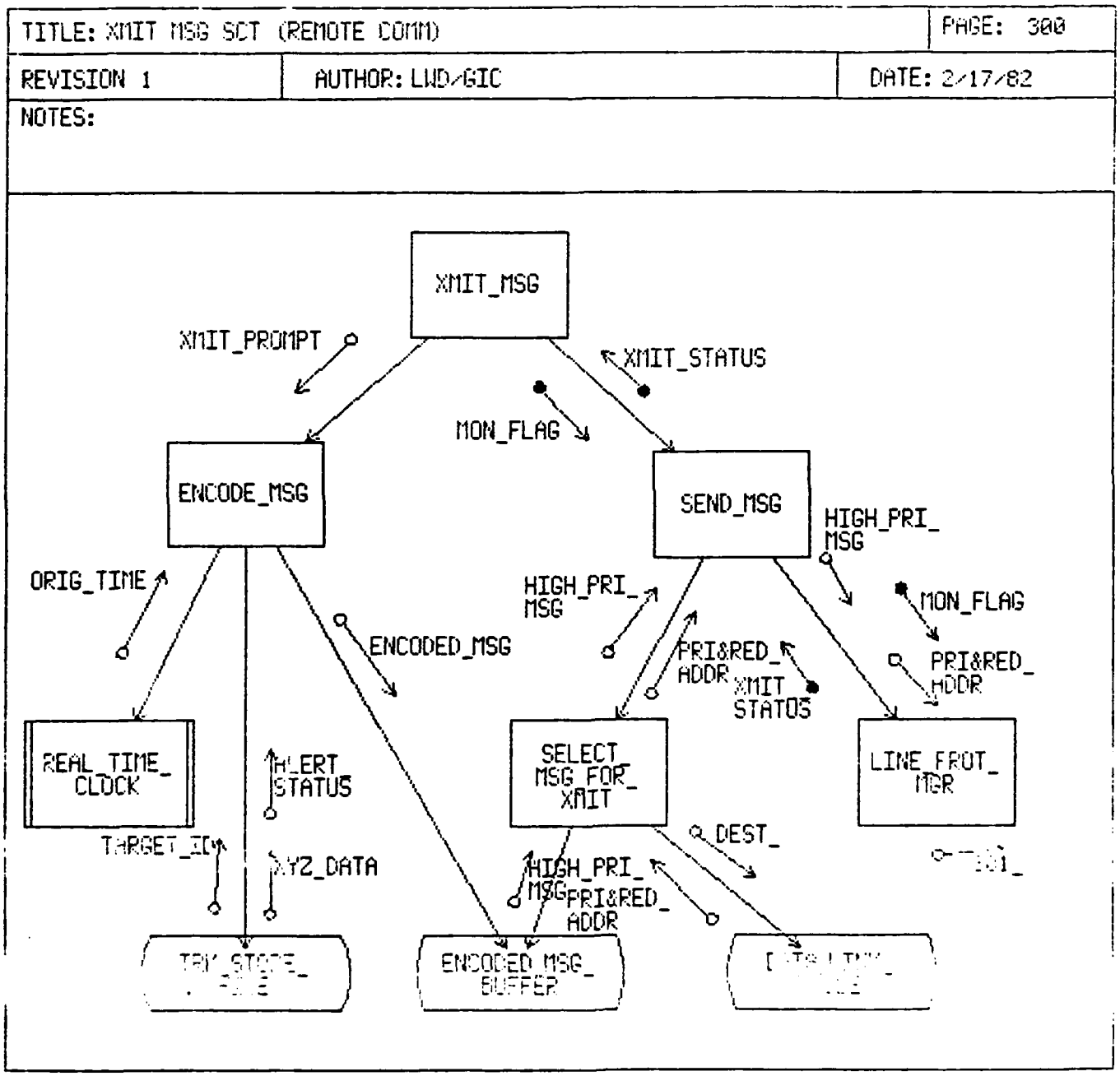


Figure 35 - Structure Chart Example

TITLE: "LINE PROTOCOL MANAGER SCT (REMOTE COMM)		PAGE: 301
REVISION 1	AUTHOR: LWD/GIC	DATE: 2/19/82
NOTES:		

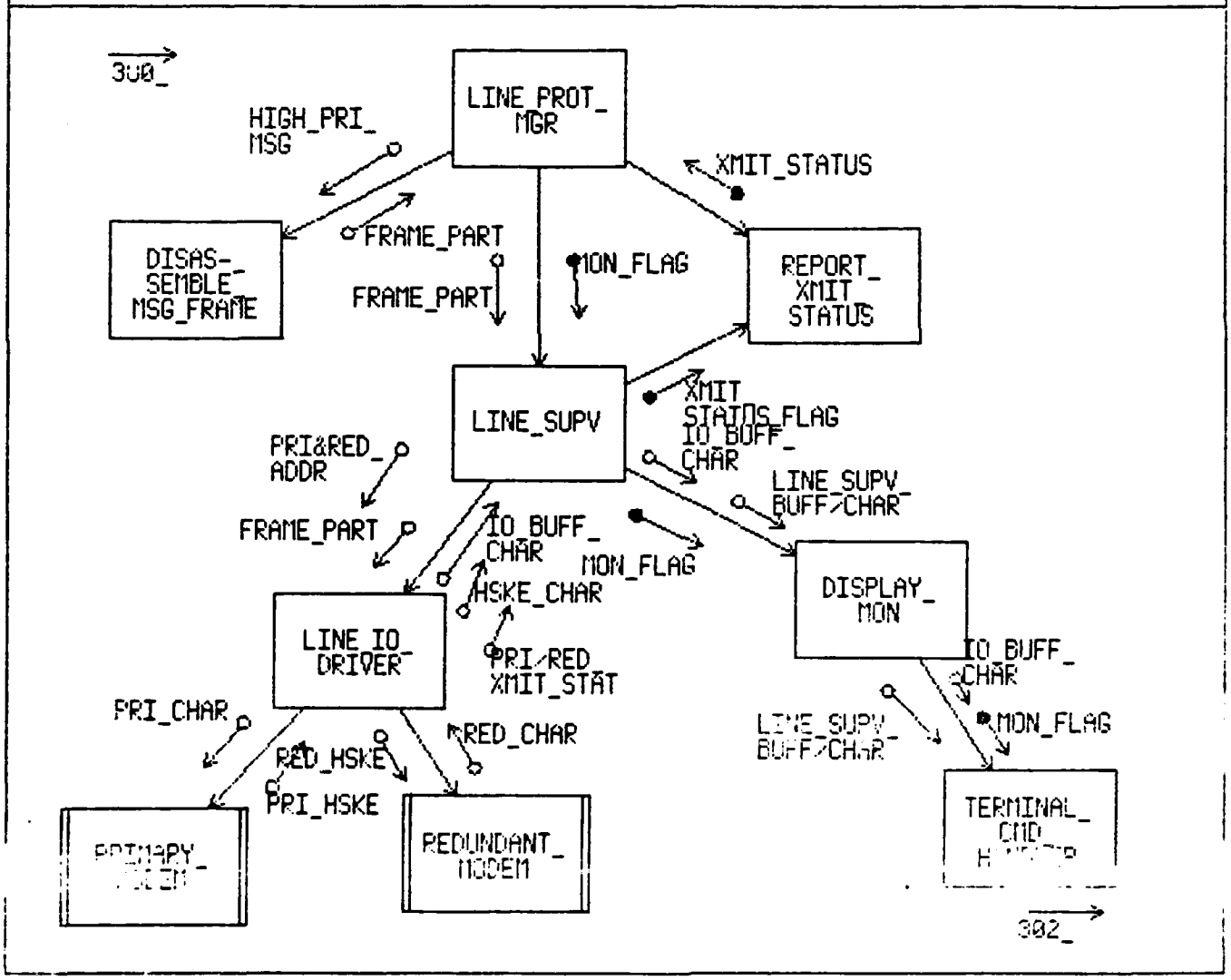


Figure 36 - Structure Chart Example

TITLE: TERMINAL COMMAND HANDLER SCT (REMOTE COMM)		PAGE: 302
REVISION 1	AUTHOR: LWD/GIC	DATE: 2/18/82
NOTES:		

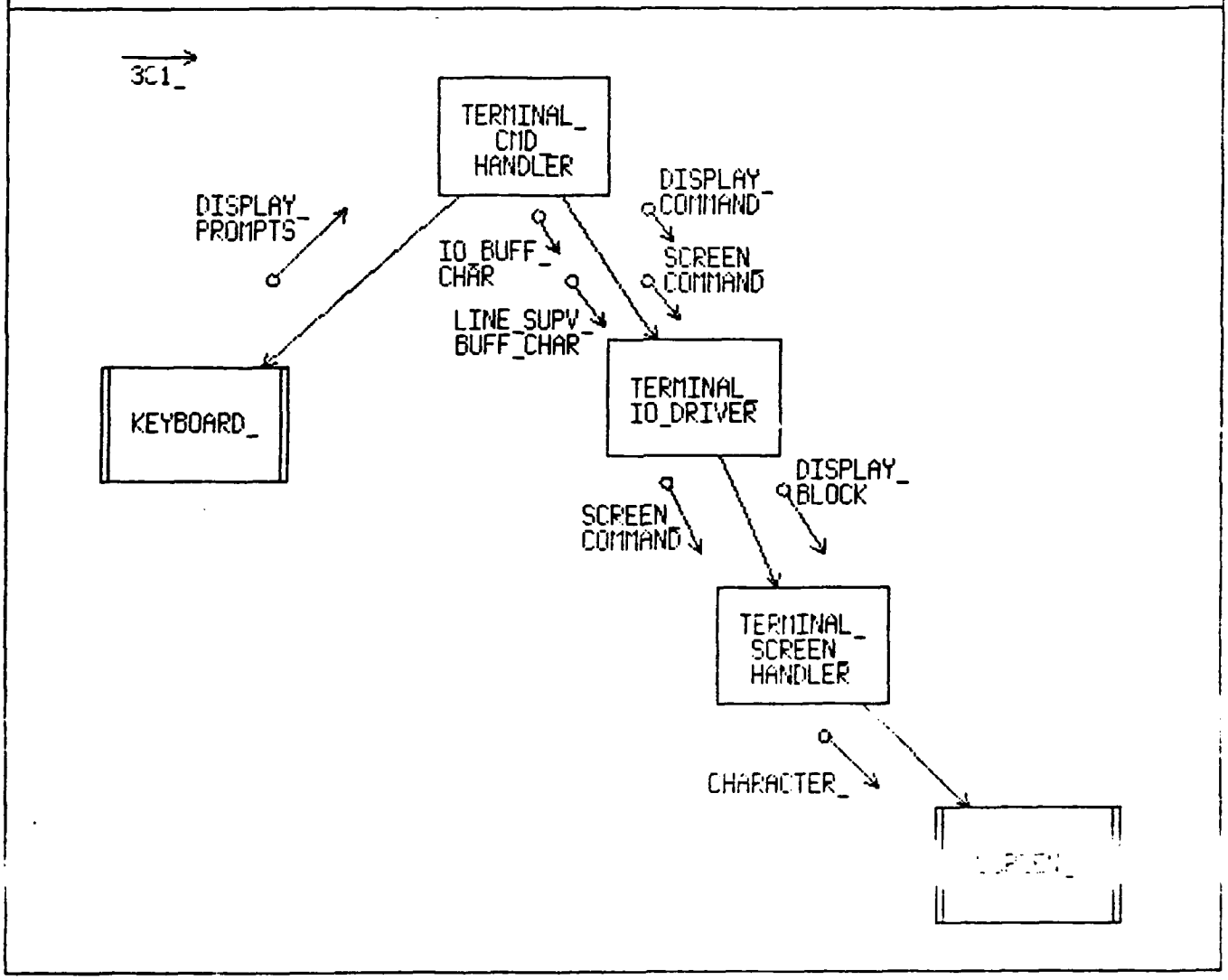


Figure 37 - Structure Chart Example

ADA AS A PROGRAM DESIGN LANGUAGE

Coding in a given language causes one to seek problem solutions by utilizing the features available in that language. Similarly, when one uses a design criteria, technique, or tool, the resulting design is biased toward the features available. By using a high powered design method and a low level programming language, a direct mapping will occur but the code may be inefficient because some high level programming language features were not available to the designer. Therefore a compatible high powered program design language should be utilized with the Ada high level language.

A Program Design Language (PDL) can be viewed as a combination of formally defined constructs known as keywords derived from a given programming language accompanied by a set of informal rules. An early attempt at such a mechanism was set forth by Caine and Gordon (43). The purpose is to capture the essential structure at a level higher than the implementation language itself. The resulting design should possess accuracy and precision, yet allow for a flexible range of expressiveness. In general, PDLs are not intended to be machine executable. Figure 38 which was excerpted from a recent technical article (44) differentiates nicely the expectations of high order languages and design languages.

This designer's guide will use an Ada-like program design language so that the program solution will be better geared toward an Ada implementation. The term Ada-like by definition deviates from Ada-exact. The use of the GO TO and the exit from a loop will not be allowed in program design. Any other Ada construct or reserved word is acceptable and will ease the transformation from design to code. The appropriateness of utilizing a subset stems from the ease of mapping the PDL into the target language.

High Level Language	Design Language
1) Must be executable (via compiler or interpreter)	1) Is generally not executable
2) Is fully analyzable by computer	2) May be only partly analyzable by computer
3) Must be rigorously defined	3) Can (and might) be loosely defined
4) Usually supports only low levels of constructs and abstractions	4) Supports varying levels of abstraction
5) Normally is subject to configuration management	5) Easily changed (because typically is not under configuration control unless in a contract situation)
6) Intended for machine communication	6) Intended for human communication

Source: Sammet, J.E., et. al. (23)

Figure 38 - Differences Between High Level Language and Design Language

As the PDL will be utilized to design program units shown on the Structure Charts, the following guidelines will be used:

- Program units will be designed to solve a particular problem and have logic which is easy to describe.
- Simple solutions, designs and interfaces will be utilized.
- Program units will be enclosed by identifiable boundaries.
- Program units can only be referenced from other parts of the program by its name.
- It is desireable for program units to have only a single, common entry and a single, common exit. With tasks and task types this may not be possible.
- Program units will be designed using the top-down concept.
- Logic flow will begin at the top and flow to one bottom exit point except for "Exceptions".
- Logically related program units will be identified for possible implementation as Ada packages.
- The comment mechanism is utilized where information can be best transmitted for human communication purposes by English text.

The largest difficulty anticipated is the tendency to elevate all the details of the implementation language to the PDL level of abstraction. For example, Figure 39 is an illustration of a Program Design Language based on Ada constructs and Figure 40 is the actual Ada language source code. This example is adapted from a book by Pyle (14). Figure 41 lists some of the comparative differences between Ada/PDL and Ada code.

```
generic
  type DATA_ITEM is private
  type ITEM_KEY is -- not decided as yet

package DATA_MODULE is -- outside world view of DATA_MODULE
    procedure READ (KEY : ITEM_KEY; DATA : out DATA_ITEM)
    procedure WRITE (KEY : ITEM_KEY; DATA : in DATA_ITEM)
    procedure UPDATE (KEY : ITEM_KEY; NEW : in DATA_ITEM;
                      OLD : out DATA_ITEM)
    procedure DELETE (KEY : ITEM_KEY)
    function FIND (KEY : ITEM_KEY) return BOOLEAN
    NO_DATA_FOR_KEY; OVERWRITE_DUPLICATE_KEY : exception
end DATA_MODULE

package body DATA_MODULE is
    task DATA_MGR is -- this task will do all of the work
        entry GET (
        entry PUT (
        entry CHANGE (
        entry REMOVE (
        entry IS_THERE (
    end DATA_MGR

    procedure READ (  ) is

            DATA_MGR.GET (  )
            -- if no data item for the key raise appropriate exception
    end READ

    procedure WRITE (  ) is
          DATA_MGR.PUT (  )
          -- if key is already in use raise appropriate exception
    end WRITE
```

Figure 39 - Program Design Language (1 of 3)

```
     procedure UPDATE is
            DATA_MGR.CHANGE
            -- if no data item for the key raise appropriate exception
     end UPDATE

     procedure DELETE (  ) is
            DATA_MGR.REMOVE
            -- if no data item for the key raise appropriate exception
     end DELETE

     function FIND IS
          DATA_MGR.IS_THERE (KEY, FOUND : out BOOLEAN)
          return FOUND
     end FIND

task body DATA.MGR is
   DATA_SET : -- the physical representation of
              -- DATA_SET has not been decided
   KEY_IN_USE : -- will depend on choice for DATA_SET

-- Note that all entries will have parameters for the
-- random access key and for a Boolean data object to indicate
-- the success or failure of the operation.  In addition, GET,
-- PUT and CHANGE will have parameters for the data items being
-- passed.
begin
   loop
      select
         accept GET (PARAMETER LIST)
               if DATA_SET HAS AN ENTRY CORRESPONDING TO KEY THEN
                   SET BOOLEAN PARAMETER TO TRUE
                   ASSIGN DATA ITEM TO DATA PARAMETER
               else
                   SET BOOLEAN PARAMETER TO FALSE
               end if
         end GET
      or
```

Figure 39 - Program Design Language (2 of 3)

```
            accept PUT (PARAMETER LIST)
                  if DATA_SET HAS NO ENTRY CORRESPONDING TO KEY THEN
                      ADD DATA ITEM TO DATA_SET
                      SET BOOLEAN PARAMETER TO TRUE
                  else
                      SET BOOLEAN PARAMETER TO FALSE
                  end if
            end PUT
         or
            accept DELETE (PARAMETER LIST)
                  if KEY MATCHES THEN
                      SET BOOLEAN TO TRUE
                  else
                      SET BOOLEAN PARAMETER TO FALSE
                  end if
                  INDICATE ENTRY CORRESPONDING TO KEY IS AVAILABLE
            END DELETE
         or
            accept CHANGE (PARAMETER LIST)
                  if KEY IS IN USE THEN
                      RETRIEVE EXISTING VALUE TO RETURN
                  end if
                  INDICATE THAT ENTRY CORRESPONDING TO KEY IS IN USE
                  STORE NEW VALUE
            end CHANGE
         or
            accept IS_THERE (PARAMETER LIST)
                  RETURN KEY_IN_USE STATUS
            end IS_THERE
         or
            terminate
         end select
      end loop
   end DATA_MGR
end DATA_MODULE
```

Figure 39 - Program Design Language (3 of 3)

ADA SOURCE CODE

```
--Indexed data set.  Random access  Data module
--Accessable to several tasks with protection against
--interference during read/write.
--The type of data item and the key should be
--fixed for any one data set but several different
--data items and keys will be needed.
--The data module has its own storage to hold
--the current data items.  Memory (storage) elements
--are flagged as in use or available.

generic --  making package generic allows any type for key items
  type DATA_ITEM is private; -- make up of type is not necessary
  type ITEM_KEY is (< >); -- key will be a discrete type

package DATA_MODULE is -- operations allowed on data
  procedure READ (KEY : ITEM_KEY; DATA : out DATA_ITEM);
  procedure WRITE (KEY : ITEM_KEY; DATA : in DATA_ITEM);
  procedure UPDATE (KEY : ITEM_KEY;
                    NEW_DATA_ITEM : in DATA_ITEM;
                    OLD_DATA_ITEM : out DATA_ITEM);
  procedure DELETE (KEY : ITEM_KEY);
  function FIND (KEY : ITEM_KEY) return BOOLEAN;
 NO_DATA_FOR_KEY, DUPLICATE_KEY_OVERITE : exception;
end DATA_MODULE;

package body DATA_MODULE is
  task DATA_MANAGER is --The entries provide the only
                       --interface to the actual
                       --data. Making this access
                       --module a task provides inter-
                       --ference protection and
                       --queueing of requests.
    entry GET (KEY : ITEM_KEY; DATA : out DATA_ITEM;
               FOUND : out BOOLEAN);
```

Figure 40 - Ada Language Source Code (1 of 4)

```
    entry PUT (KEY : ITEM_KEY; DATA : in DATA_ITEM;
               FOUND : out BOOLEAN);
    entry CHANGE (KEY : ITEM_KEY; NEW : in ITEM;
                  OLD : out ITEM; FOUND : out BOOLEAN);
    entry IS_THERE (KEY : ITEM_KEY; FOUND : out BOOLEAN);
    entry REMOVE (KEY : ITEM_KEY; FOUND : out BOOLEAN);
  END DATA_MANAGER;

--The remainder of the package body is the
--implementation of the specification section.

procedure READ (KEY : ITEM_KEY; DATA : out DATA_ITEM) is
         FOUND : BOOLEAN;
    begin
        DATA_MANAGER.GET (KEY, DATA, FOUND);
        if not FOUND then
           raise NO_DATA_FOR_KEY;
        end if;
end READ;

procedure WRITE (KEY : ITEM_KEY; DATA : in DATA_ITEM) is
      KEY_IN_USE : BOOLEAN;
      begin
          DATA_MANAGER.PUT (KEY, DATA, KEY_IN_USE);
             if KEY_IN_USE then
                raise DUPLICATE_KEY_OVERITE;
             end if;
end WRITE;

procedure UPDATE (KEY : ITEM_KEY; NEW_DATA_ITEM : in DATA_ITEM);
                  OLD_DATE_ITEM : out DATA_ITEM) is
          FOUND : BOOLEAN;
     DATA_MANAGER.CHANGE (KEY, NEW_DATA_ITEM, OLD_DATA_ITEM,
                               FOUND);
     if not FOUND then
        raise NO_DATA_FOR_KEY;
     end if
end UPDATE;
```

Figure 40 - Ada Language Source Code (2 of 4)

```
procedure DELETE (KEY : ITEM_KEY) is
         FOUND : BOOLEAN;
    begin
        DATA_MANAGER.REMOVE (KEY, FOUND);
        if not FOUND then
          raise NO_DATA_FOR_KEY;
        end if;
end DELETE;

function FIND (KEY : ITEM_KEY) return BOOLEAN is
        FOUND : BOOLEAN;
        begin
            DATA_MANAGER.IS_THERE (KEY, FOUND);
            return FOUND;
end FIND;

task body DATA_MANAGER is
     DATA_SET : array (KEY) of DATA_ITEM;
     KEY_IN_USE : array (KEY) of BOOLEAN := (others => FALSE);

     begin
        loop
           select
              accept GET (KEY : ITEM_KEY; DATA : out DATA_ITEM:
                     FOUND : out BOOLEAN);
                  do
                     FOUND := KEY_IN_USE (KEY);
                     if FOUND then
                        DATA := DATA_SET (KEY);
                     end if;
              end GET;
        or
              accept PUT (KEY : ITEM_KEY; DATA : in DATA_ITEM
                     FOUND : out BOOLEAN)
                  do
```

Figure 40 - Ada Language Source Code (3 of 4)

```
                FOUND := KEY_IN_USE (KEY);
                DATA_SET (KEY) := DATA;
            IN_USE (KEY) := TRUE;
        end PUT
    or
        accept CHANGE (KEY : DATA_KEY; NEW : in DATA_MIND);
            OLD : out DATA_ITEM; FOUND : out BOOLEAN);
        do

            FOUND := KEY_IN_USE (KEY);
            if FOUND then
                OLD := DATA_SET (KEY);
            end if
            DATA_SET (KEY) := NEW;
            KEY_IN_USE (KEY) := TRUE;
        end CHANGE;
    or
        accept DELETE (KEY : ITEM_KEY; FOUND : out BOOLEAN);
        do
            FOUND := KEY_IN_USE (KEY);
            KEY_IN_USE (KEY) := FALSE;
        end DELETE;
    or
        accept IS_THERE (KEY : ITEM_KEY; FOUND : out BOOLEAN);
        do
            FOUND := KEY_IN_USE (K);
        end IS_THERE;
    or
        terminate
  end loop;
 end DATA_MANAGER;
end DATA_MODULE;
```

Figure 40 - Ada Language Source Code (4 of 4)

Ada/PDL Compared to Ada Code

1. The type of key to be used to access the data set has not been decided.

2. The specification section of the package DATA_MODULE is identical in both PDL and code. This is necessary because this specification is the view of this package from the outside world. At the PDL stage this view must be layed out in enough detail to allow other parts of the system to be written. Any lower level detail is invisible to the rest of the system.

3. The entry points listed in the task specification for task DATA_MANAGER will do the work of dealing with the actual data. At the PDL stage the arguments for these entries have not been decided upon. Design can continue however, because the PDL just wants to show how these entries are used. Exactly how the entries work can be left until code time.

4. The procedure "bodies" make use of the parameterless entry points to show the connection between the logical and physical levels of data management.

5. Comments in the procedure "bodies" explain some things that will need to be coded during coding, but are of passing interest only at this stage.

6. A major difference between the PDL and code is the fact that the actual organization of the data structure has not been decided at the PDL level. This fact points out a great strength in an Ada/PDL. The PDL can continue with the task description without needing to know what it is really dealing with. The PDL simply uses the abstract idea of a data pool, the decision as to what this data pool will be (e.g., an array, a linked list, a stack, etc.), can be put off until the code section. Since the actual data structure is confined to the task body for DATA_MANAGER it is invisible to all higher level program units so any change in data structure will not effect anything but the task body.

7. The entries for task body DATA_MANAGER are completely described because they deal with the abstract representation of the data. The code for these entries is not complete because without knowing what type of data structure you are dealing with you cannot write code to manipulate it physically.

Figure 41 - PDL and Ada Code Differences

When the PDL for a program unit is finished, the following criteria characterizing good design should be represented in the resulting modular or program units. The term module is used in the description as it is generally understood.

Coupling is a measure of module interdependence; i.e., the extent to which an error or change in one module will cause an error or necessitate a change in another module. It should be apparent that in a system with a great deal of module interdependence, called tightly coupled, errors or changes in any part of the system will resonate or ripple through the entire system causing more errors and changes. Coupling is probably the most important of the design qualities because of the far reaching consequences if done poorly. Because of this importance, the coupling resulting from Data Flow Diagram decomposition should be closely reviewed to create maximum looseness before the design process moves to the next stage.

Cohesion is a measure of the single mindedness of a module. A highly cohesive module concerns itself with accomplishing one function only. As with coupling, it should be apparent that a module with low cohesiveness can propagate an error or change in one of its functions to all of its other functions. Ideally a highly cohesive module should appear like a black box to any other module that wishes to use it.

State Memory is global data. Because shared data can act as a conduit through which errors and/or changes can propagate from one module to another, state memory undesireably increases coupling. Ideally, there should be no global data, but if its use cannot be avoided it should be restricted to read-only use to minimize the coupling effect.

Module Size, it has long been noted that limiting module source code size helps insure module independence, readability, testability, and maintainability. IBM, on the New York Times Project limited modules to 50 lines of source code and TRW has recommended modules be no longer than two (2) print pages in length. This quantitative limit is especially important in the maintenance phase where human beings are directly concerned with comprehendability. In order to change code without some inadvertant consequences taking place, the effected sections must be precisely comprehended. Therefore, guidelines for module design will be, not only designed with loose coupling and high cohesion, but limited to between ten and fifty high order language statements.

Measure of Complexity, the technique used to measure the complexity of a program unit and explained in the succeeding section, will be used to assist designing uncomplicated program units.

When the structure charts have been developed initially, the probability of exact correctness is extremely low. It is through the generation of the PDL in the design of the module/program unit that the correctness will be determined. Hence the structure chart will have to be updated due to units being too large, too small, tight coupling or low cohesion. Some of the terms and action that can result from PDL generation causing the structure charts to be updated are:

Factoring is the decomposition of a large, low cohesive module into submodules. Factoring is often all that is needed to increase cohesion because one module with poor cohesion, as a result of trying to do too many things, can be factored into several modules that each perform one specific function.

Decision Splitting is the separation of a decision from the action, resulting from that decision. Decision splitting reduced cohesion and increases coupling, but is easily remedied by simply unsplitting the decision.

Fan/Out, or the number of submodules any one module has should be restricted to a maximum of seven. If a preliminary design yields a higher fan/out some factoring to yield at lease one more level is necessary.

Fan/In, or the number of masters that use any one slave module, should be as high as possible. High fan/in results from effectively factoring a common function out of all the modules in which it appears.

Error Reporting should be done at the point of error discovery not at a centralized error handling location. The Ada language provides the Exception facility for both predefined and user defined Exceptions which may be raised during the execution of statements.

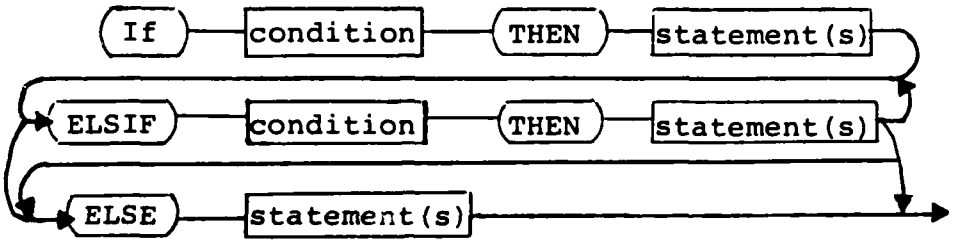
Complexity Measure is an indication of the program unit's simple or complex design and is explained in the next section. Calculations of this measure should produce an indication as low as possible.

ADA/PDL SYNTAX DIAGRAMS

IF-THEN-ELSIF-ELSE

The If-Then-Elsif-Else statement causes control to be transferred to one or none of a number of sequences of statements based on the evaluation of the truth of a conditional statement.

Syntax Diagram



Layout

```
if    (p)    then
 English language
elsif  (q)  then
 English language
       .
       .
       .
elsif (r) then
 English language
else
 English language
end if;
```

LOOP STATEMENT

The loop statement specifies a sequence of statements to be executed repeatedly zero or more times.

APPENDIX C

SYSTEM ENTITY DIAGRAMS

APPENDIX C

SYSTEM ENTITY DIAGRAM

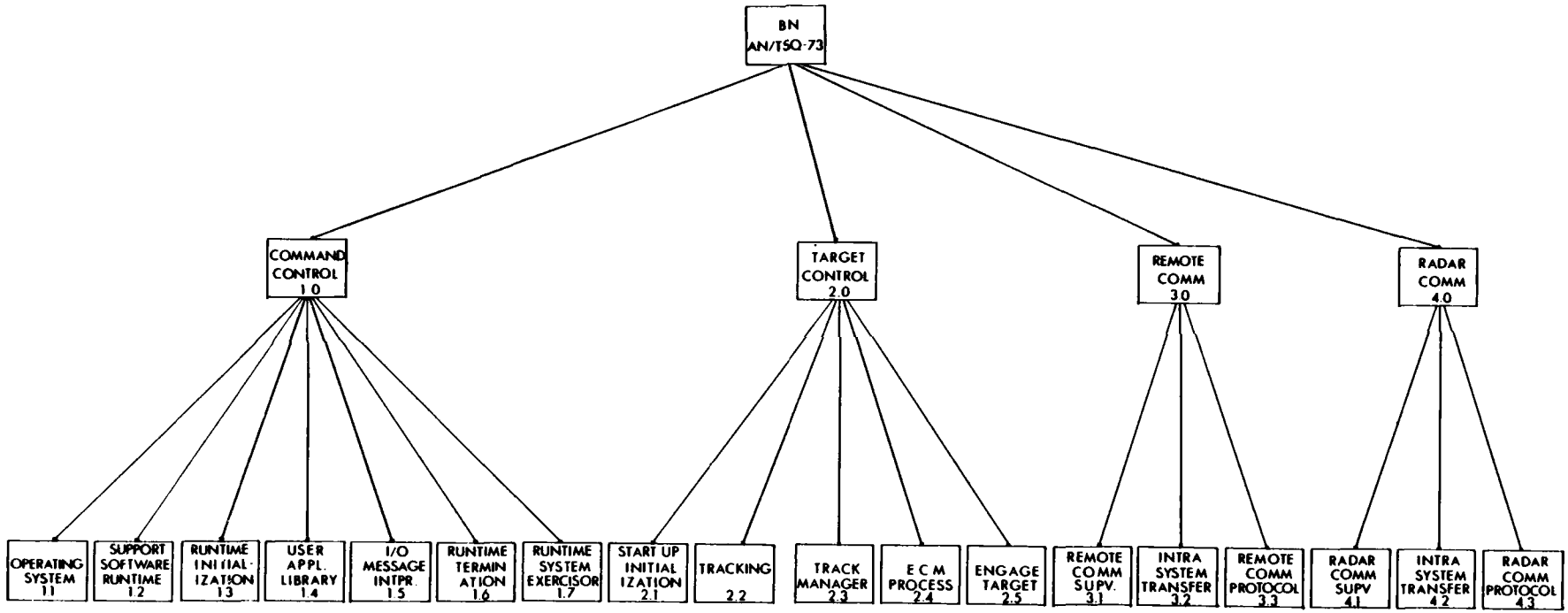
FINAL REPORT

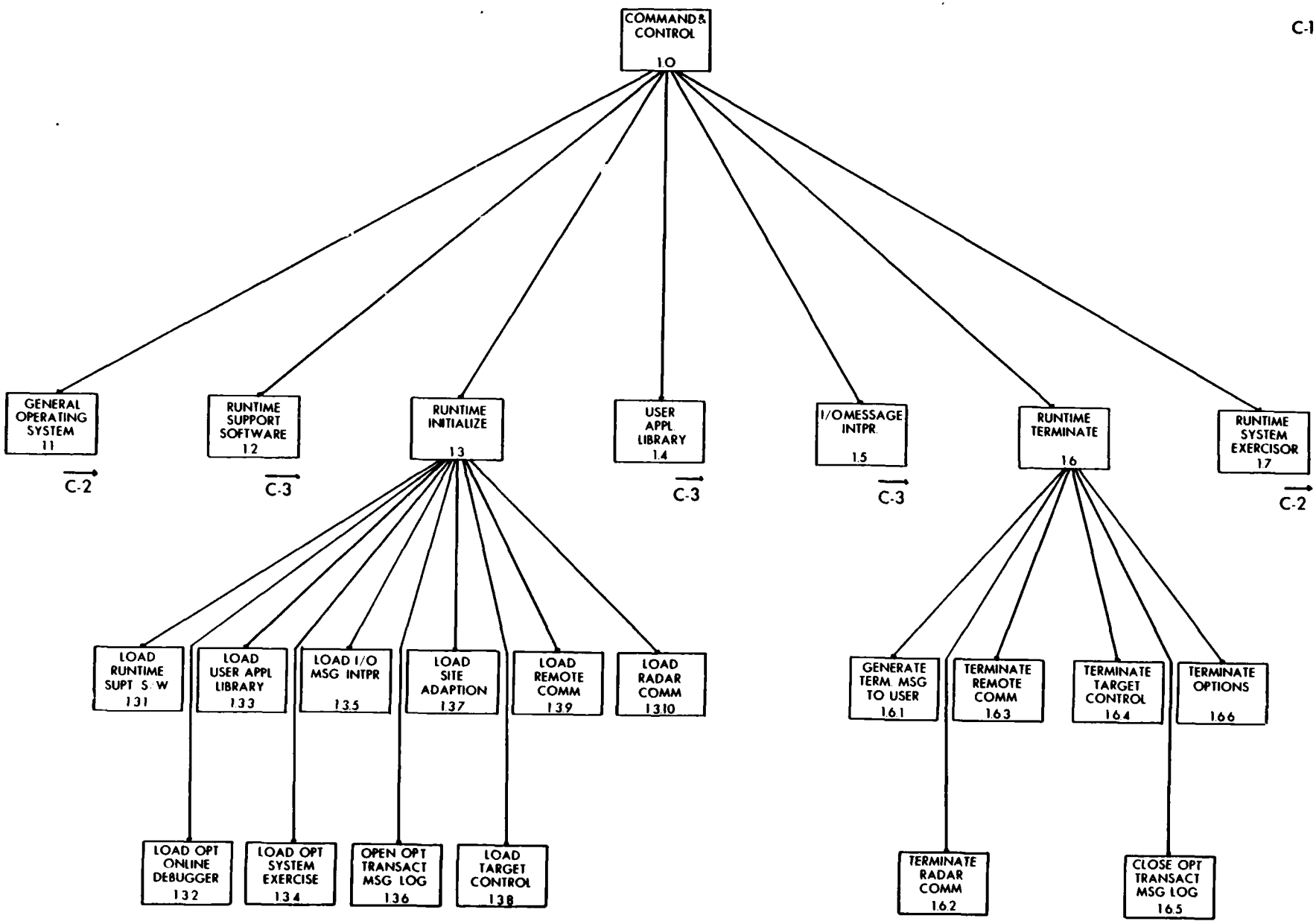
LARGE SCALE SOFTWARE SYSTEM DESIGN FOR THE MISSILE MINDER AN/TSQ-73 USING THE ADA PROGRAMMING LANGUAGE

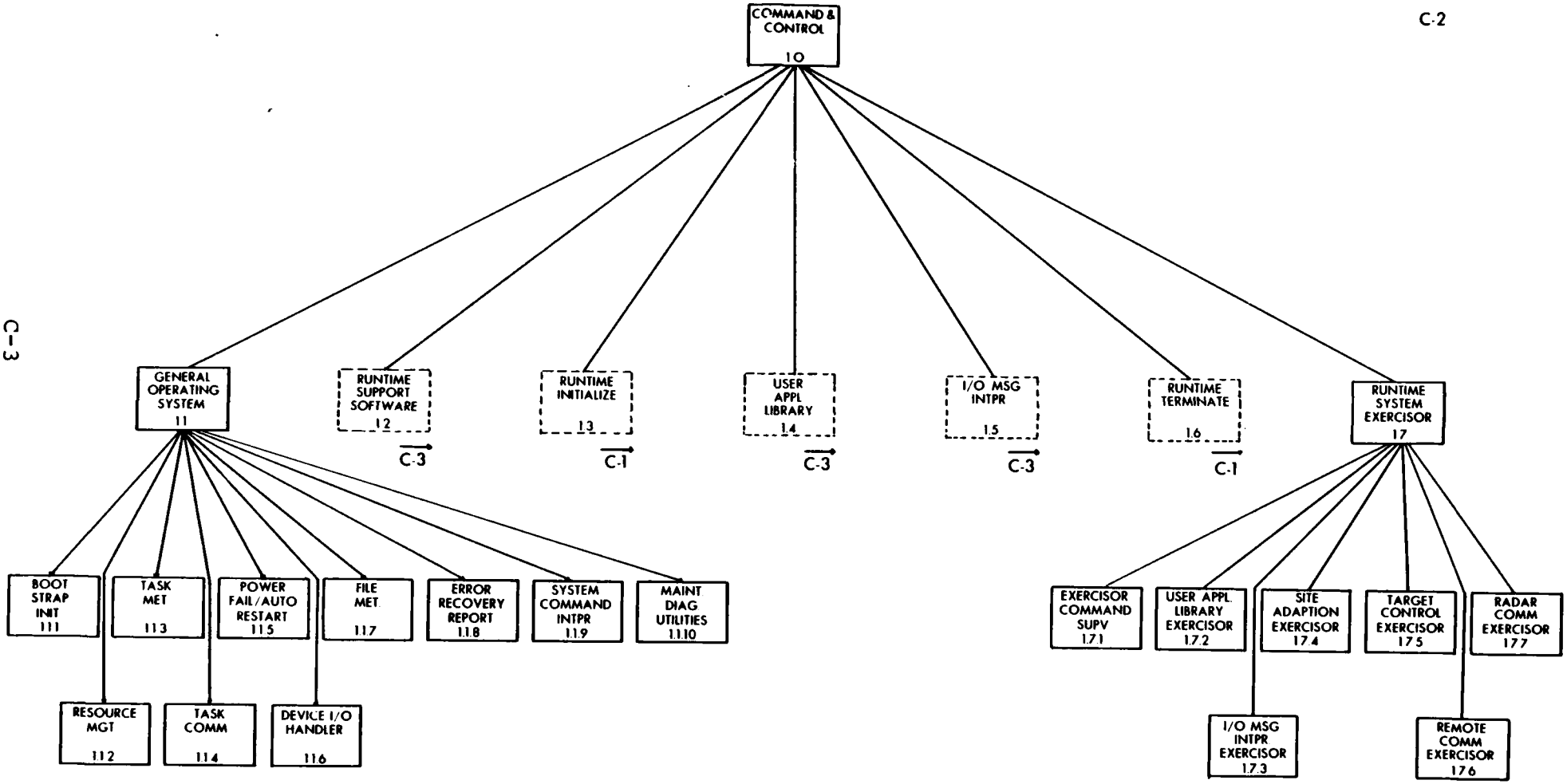
PREPARED FOR

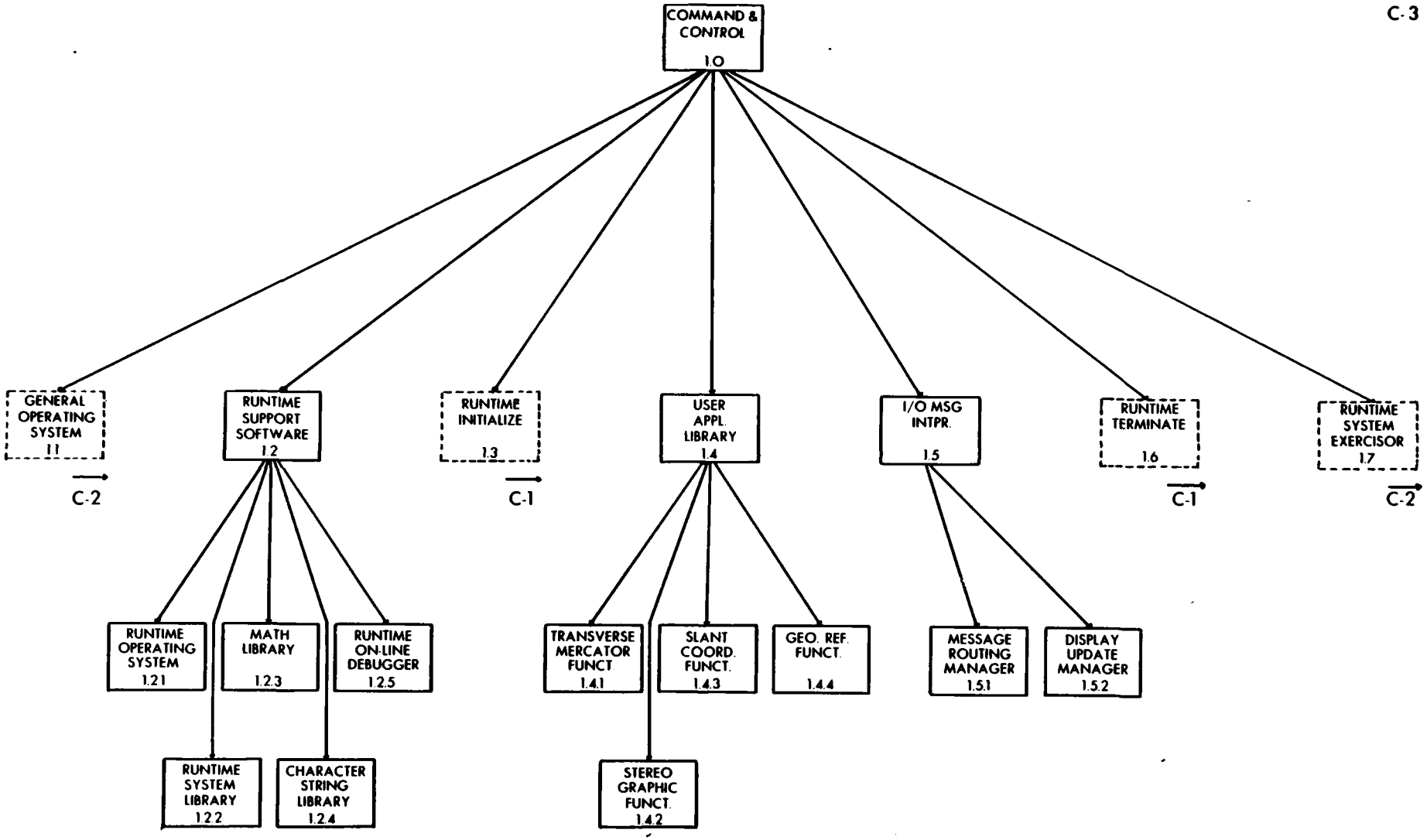
U.S. ARMY COMMUNICATIONS ELECTRONICS COMMAND

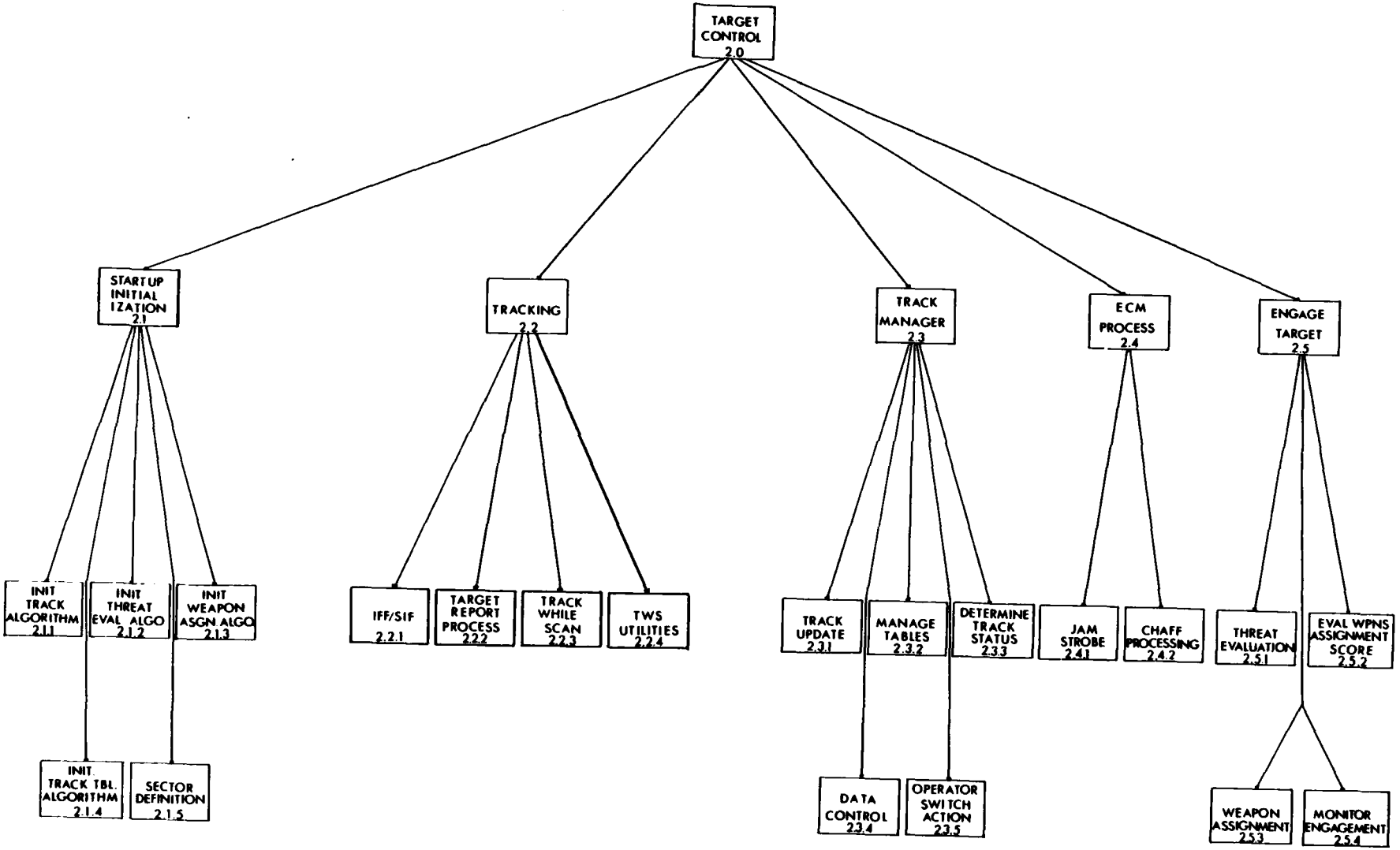
FORT MONMOUTH, NEW JERSEY 07703

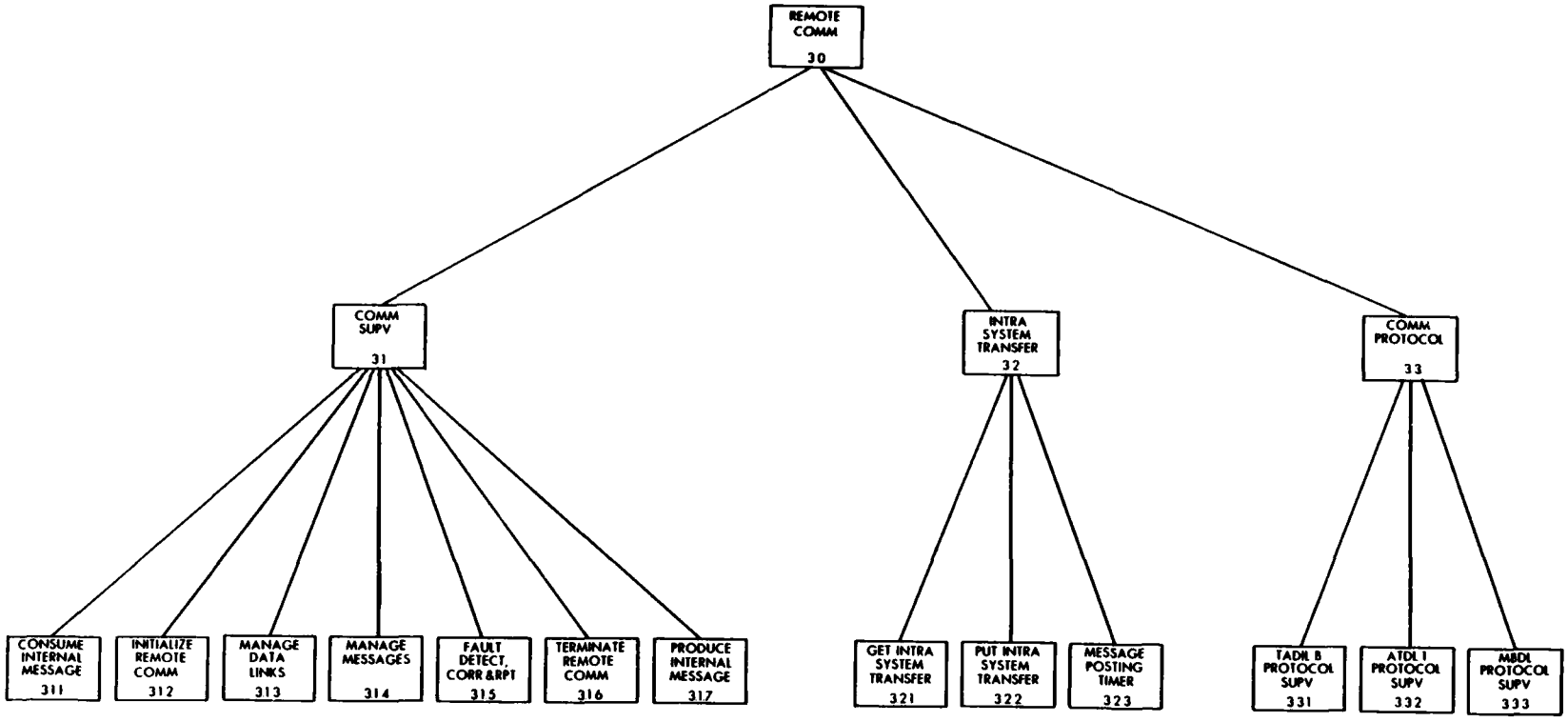

BN
AN/TSQ-73
COMMAND CONTROL 1.0
TARGET CONTROL 2.0
REMOTE COMM 3.0
RADAR COMM 4.0
OPERATING SYSTEM 1.1
SUPPORT SOFTWARE RUNTIME 1.2
RUNTIME INITIAL-IZATION 1.3
USER APPL. LIBRARY 1.4
I/O MESSAGE INTPR. 1.5
RUNTIME TERMIN ATION 1.6
RUNTIME SYSTEM EXERCISOR 1.7
START UP INITIAL IZATION 2.1
TRACKING 2.2
TRACK MANAGER 2.3
E C M PROCESS 2.4
ENGAGE TARGET 2.5
REMOTE COMM SUPV. 3.1
INTRA SYSTEM TRANSFER 3.2
REMOTE COMM PROTOCOL 3.3
RADAR COMM SUPV 4.1
INTRA SYSTEM TRANSFER 4.2
RADAR COMM PROTOCOL 4.3

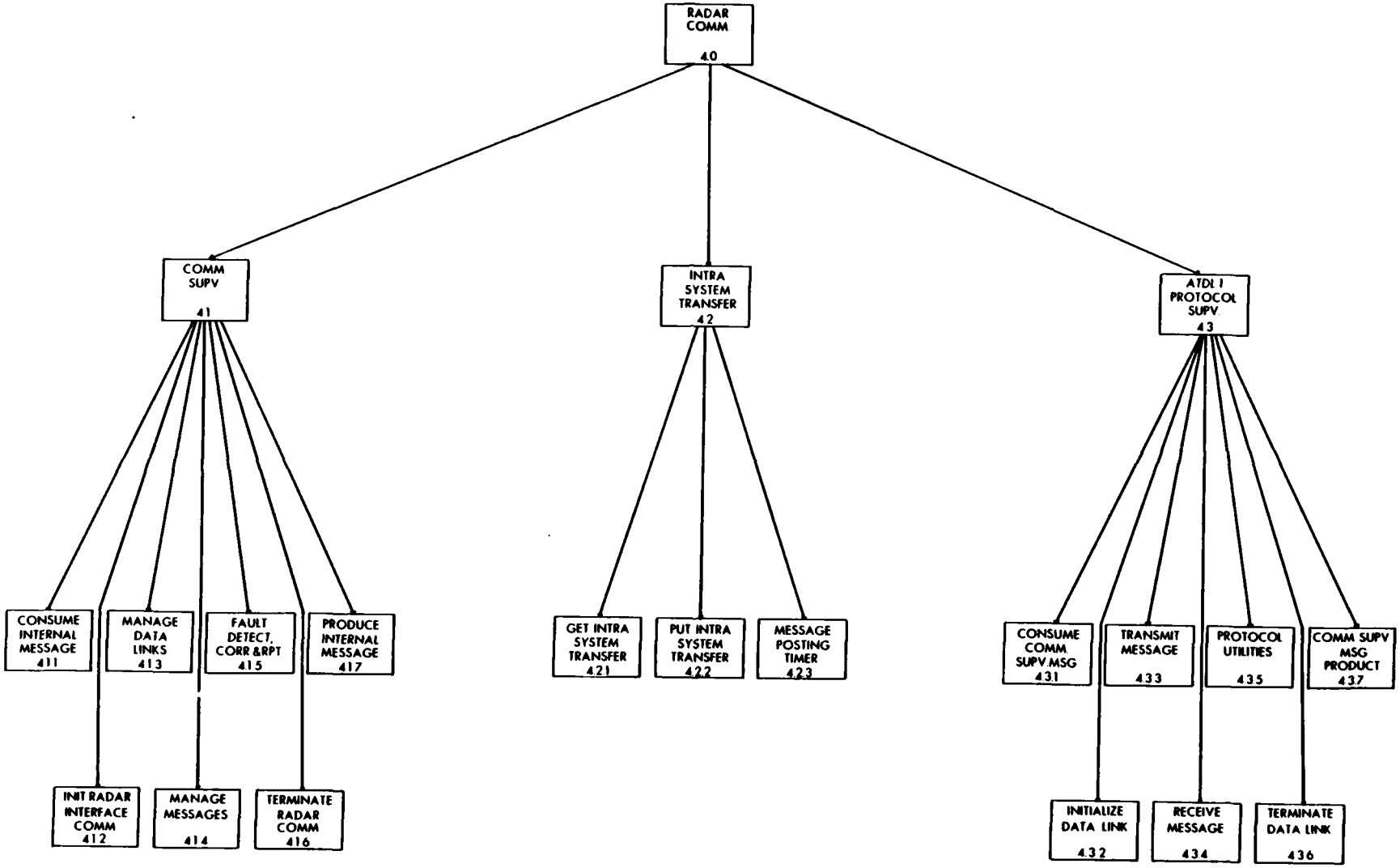

COMMAND & CONTROL
1.0
GENERAL OPERATING SYSTEM
1.1
C-2
RUNTIME SUPPORT SOFTWARE
1.2
C-3
RUNTIME INITIALIZE
1.3
USER APPL LIBRARY
1.4
C-3
I/O MESSAGE INTPR
1.5
C-3
RUNTIME TERMINATE
1.6
RUNTIME SYSTEM EXERCISOR
1.7
C-2
LOAD RUNTIME SUPT S/W
1.3.1
LOAD USER APPL LIBRARY
1.3.3
LOAD I/O MSG INTPR
1.3.5
LOAD SITE ADAPTION
1.3.7
LOAD REMOTE COMM
1.3.9
LOAD RADAR COMM
1.3.10
LOAD OPT ONLINE DEBUGGER
1.3.2
LOAD OPT SYSTEM EXERCISE
1.3.4
OPEN OPT TRANSACT MSG LOG
1.3.6
LOAD TARGET CONTROL
1.3.8
GENERATE TERM MSG TO USER
1.6.1
TERMINATE REMOTE COMM
1.6.3
TERMINATE TARGET CONTROL
1.6.4
TERMINATE OPTIONS
1.6.6
TERMINATE RADAR COMM
1.6.2
CLOSE OPT TRANSACT MSG LOG
1.6.5


COMMAND & CONTROL
1.0
GENERAL OPERATING SYSTEM
1.1
RUNTIME SUPPORT SOFTWARE
1.2
C-3
RUNTIME INITIALIZE
1.3
C-1
USER APPL LIBRARY
1.4
C-3
I/O MSG INTPR
1.5
C-3
RUNTIME TERMINATE
1.6
C-1
RUNTIME SYSTEM EXERCISOR
1.7
BOOT STRAP INIT
1.1.1
TASK MET
1.1.3
POWER FAIL/AUTO RESTART
1.1.5
FILE MET
1.1.7
ERROR RECOVERY REPORT
1.1.8
SYSTEM COMMAND INTPR
1.1.9
MAINT DIAG UTILITIES
1.1.10
RESOURCE MGT
1.1.2
TASK COMM
1.1.4
DEVICE I/O HANDLER
1.1.6
EXERCISOR COMMAND SUPV
1.7.1
USER APPL. LIBRARY EXERCISOR
1.7.2
SITE ADAPTION EXERCISOR
1.7.4
TARGET CONTROL EXERCISOR
1.7.5
RADAR COMM EXERCISOR
1.7.7
I/O MSG INTPR EXERCISOR
1.7.3
REMOTE COMM EXERCISOR
1.7.6

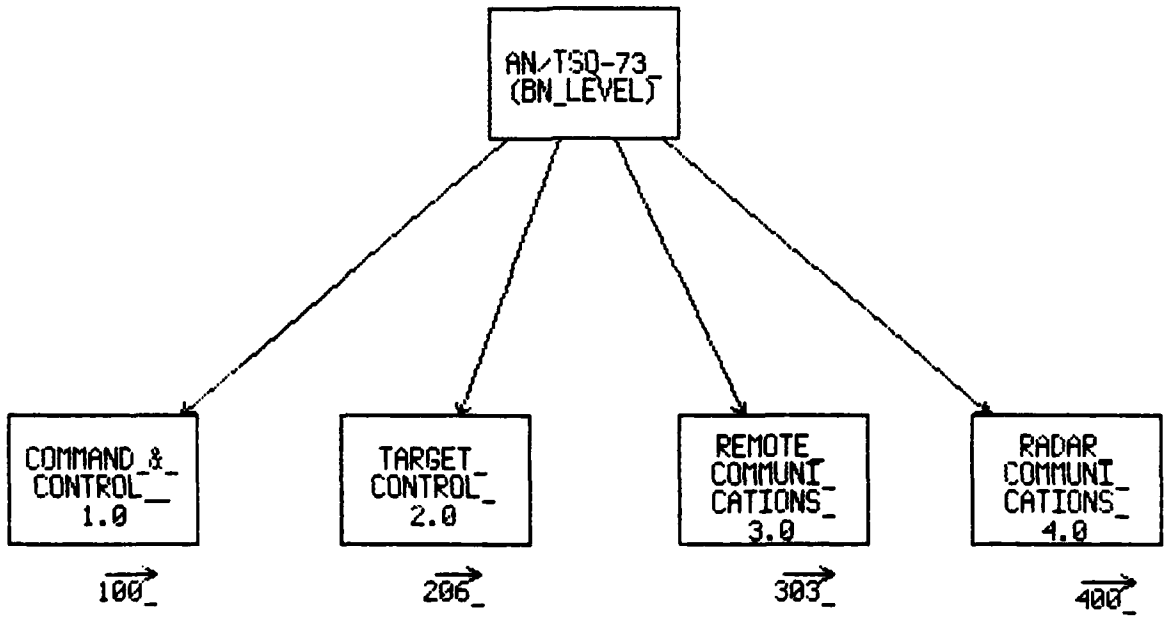

C-3
COMMAND & CONTROL
1.0
GENERAL OPERATING SYSTEM
1.1
C-2
RUNTIME SUPPORT SOFTWARE
1.2
RUNTIME OPERATING SYSTEM
1.2.1
RUNTIME SYSTEM LIBRARY
1.2.2
MATH LIBRARY
1.2.3
CHARACTER STRING LIBRARY
1.2.4
RUNTIME ON-LINE DEBUGGER
1.2.5
RUNTIME INITIALIZE
1.3
C-1
USER APPL. LIBRARY
1.4
TRANSVERSE MERCATOR FUNCT.
1.4.1
STEREO GRAPHIC FUNCT.
1.4.2
SLANT COORD. FUNCT.
1.4.3
GEO. REF. FUNCT.
1.4.4
I/O MSG INTPR.
1.5
MESSAGE ROUTING MANAGER
1.5.1
DISPLAY UPDATE MANAGER
1.5.2
RUNTIME TERMINATE
1.6
C-1
RUNTIME SYSTEM EXERCISOR
1.7
C-2
C-4


TARGET CONTROL 2.0
STARTUP INITIAL IZATION 2.1
TRACKING 2.2
TRACK MANAGER 2.3
ECM PROCESS 2.4
ENGAGE TARGET 2.5
INIT TRACK ALGORITHM 2.1.1
INIT THREAT EVAL ALGO 2.1.2
INIT WEAPON ASGN ALGO 2.1.3
INIT TRACK TBL ALGORITHM 2.1.4
SECTOR DEFINITION 2.1.5
IFF/SIF 2.2.1
TARGET REPORT PROCESS 2.2.2
TRACK WHILE SCAN 2.2.3
TWS UTILITIES 2.2.4
TRACK UPDATE 2.3.1
MANAGE TABLES 2.3.2
DETERMINE TRACK STATUS 2.3.3
DATA CONTROL 2.3.4
OPERATOR SWITCH ACTION 2.3.5
JAM STROBE 2.4.1
CHAFF PROCESSING 2.4.2
THREAT EVALUATION 2.5.1
EVAL WPNS ASSIGNMENT SCORE 2.5.2
WEAPON ASSIGNMENT 2.5.3
MONITOR ENGAGEMENT 2.5.4

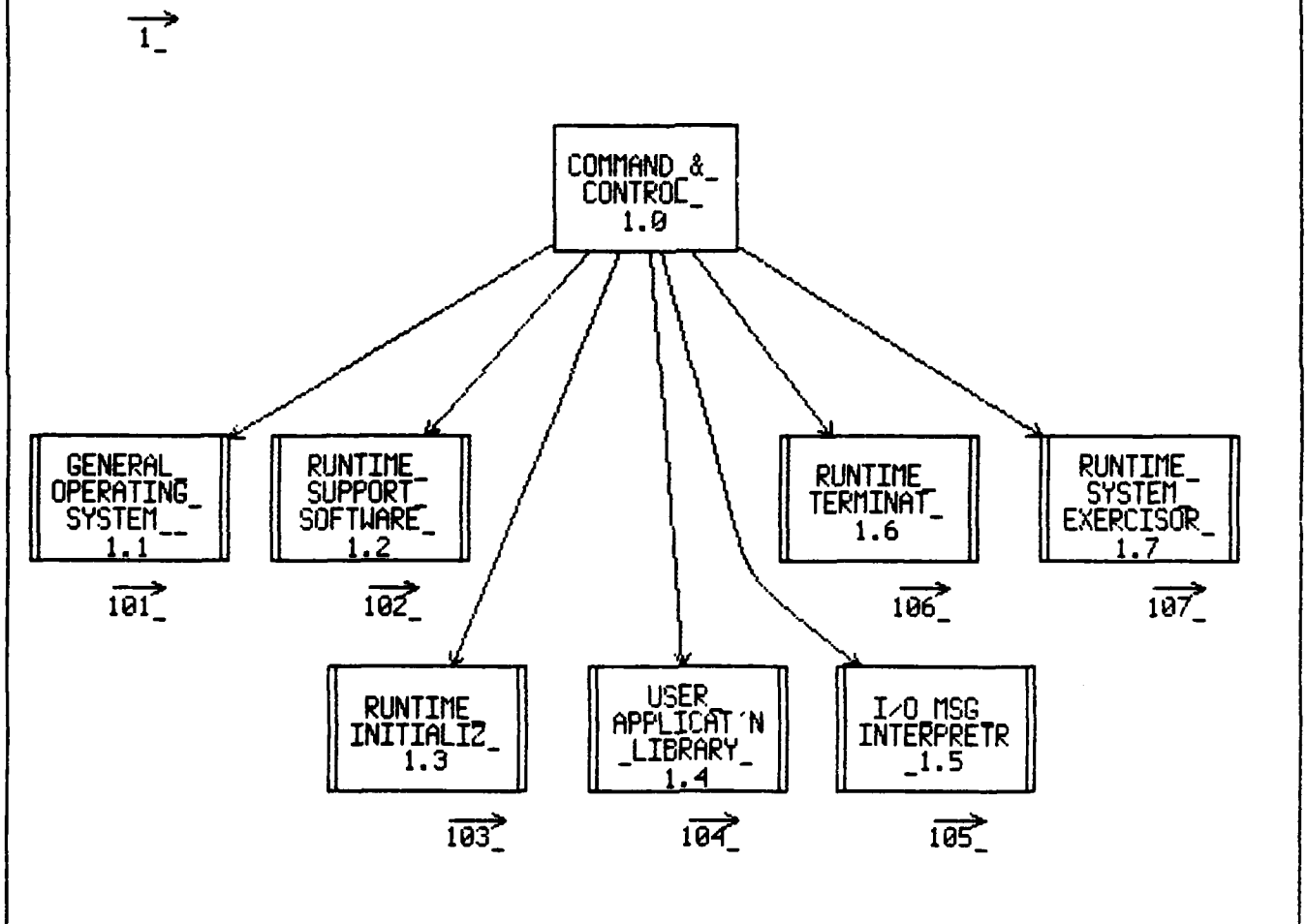

REMOTE COMM 30
COMM SUPV 31
INTRA SYSTEM TRANSFER 32
COMM PROTOCOL 33
CONSUME INTERNAL MESSAGE 311
INITIALIZE REMOTE COMM 312
MANAGE DATA LINKS 313
MANAGE MESSAGES 314
FAULT DETECT, CORR &RPT 315
TERMINATE REMOTE COMM 316
PRODUCE INTERNAL MESSAGE 317
GET INTRA SYSTEM TRANSFER 321
PUT INTRA SYSTEM TRANSFER 322
MESSAGE POSTING TIMER 323
TADIL B PROTOCOL SUPV 331
ATDL I PROTOCOL SUPV 332
MBDL PROTOCOL SUPV 333


RADAR COMM 4.0
COMM SUPV 41
INTRA SYSTEM TRANSFER 42
ATDL I PROTOCOL SUPV 43
CONSUME INTERNAL MESSAGE 411
MANAGE DATA LINKS 413
FAULT DETECT, CORR &RPT 415
PRODUCE INTERNAL MESSAGE 417
INIT RADAR INTERFACE COMM 412
MANAGE MESSAGES 414
TERMINATE RADAR COMM 416
GET INTRA SYSTEM TRANSFER 421
PUT INTRA SYSTEM TRANSFER 422
MESSAGE POSTING TIMER 423
CONSUME COMM SUPV MSG 431
TRANSMIT MESSAGE 433
PROTOCOL UTILITIES 435
COMM SUPV MSG PRODUCT 437
INITIALIZE DATA LINK 432
RECEIVE MESSAGE 434
TERMINATE DATA LINK 436

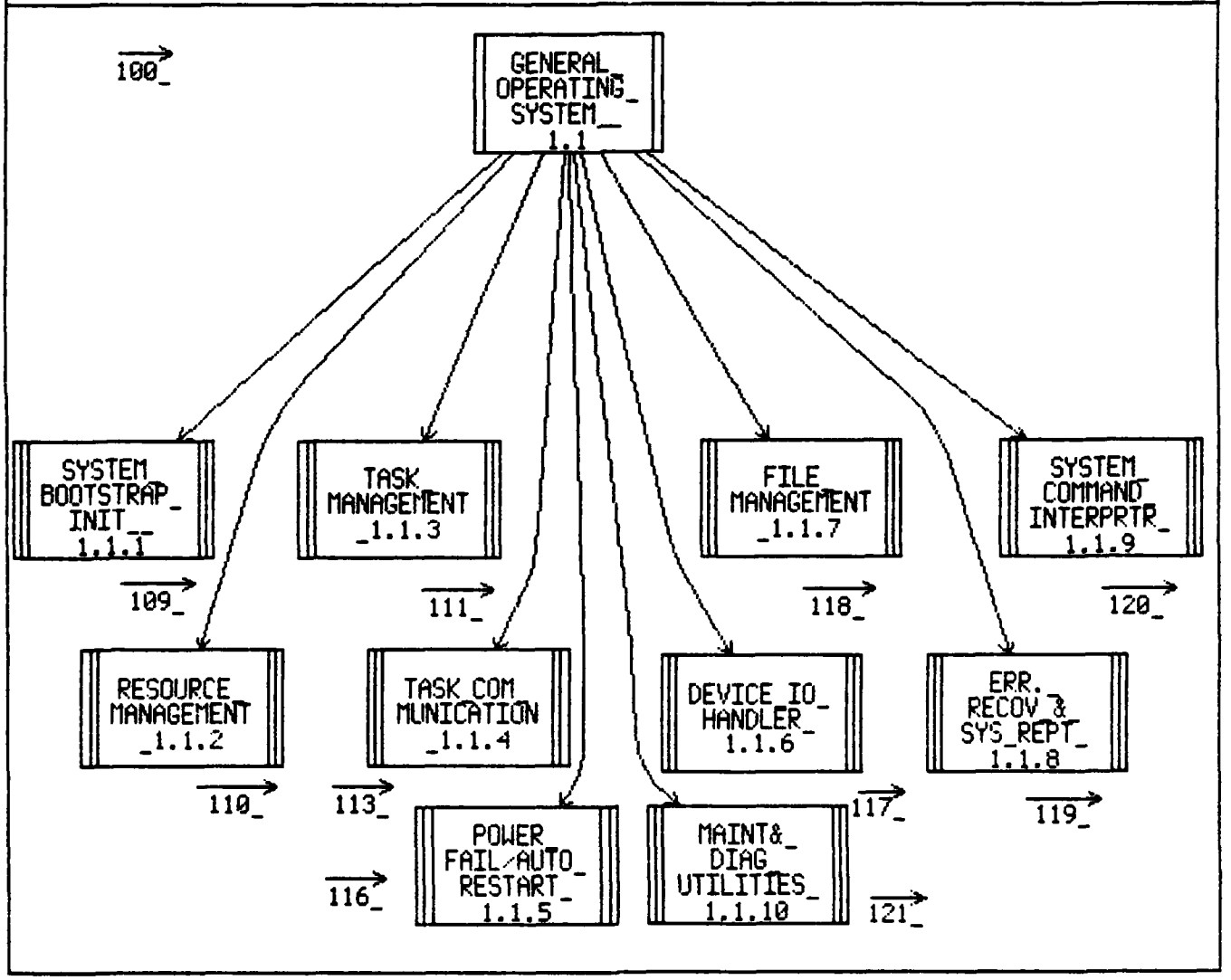
TITLE: AN/TSQ-73 SED		PAGE: 1
REVISION 3	AUTHOR: RAW/WAS	DATE: 10/13/81
NOTES:		



TITLE: 1.0 COMMAND & CONTROL SED		PAGE: 100
REVISION 4	AUTHOR: RAW/WAS	DATE: 3/24/82
NOTES:		

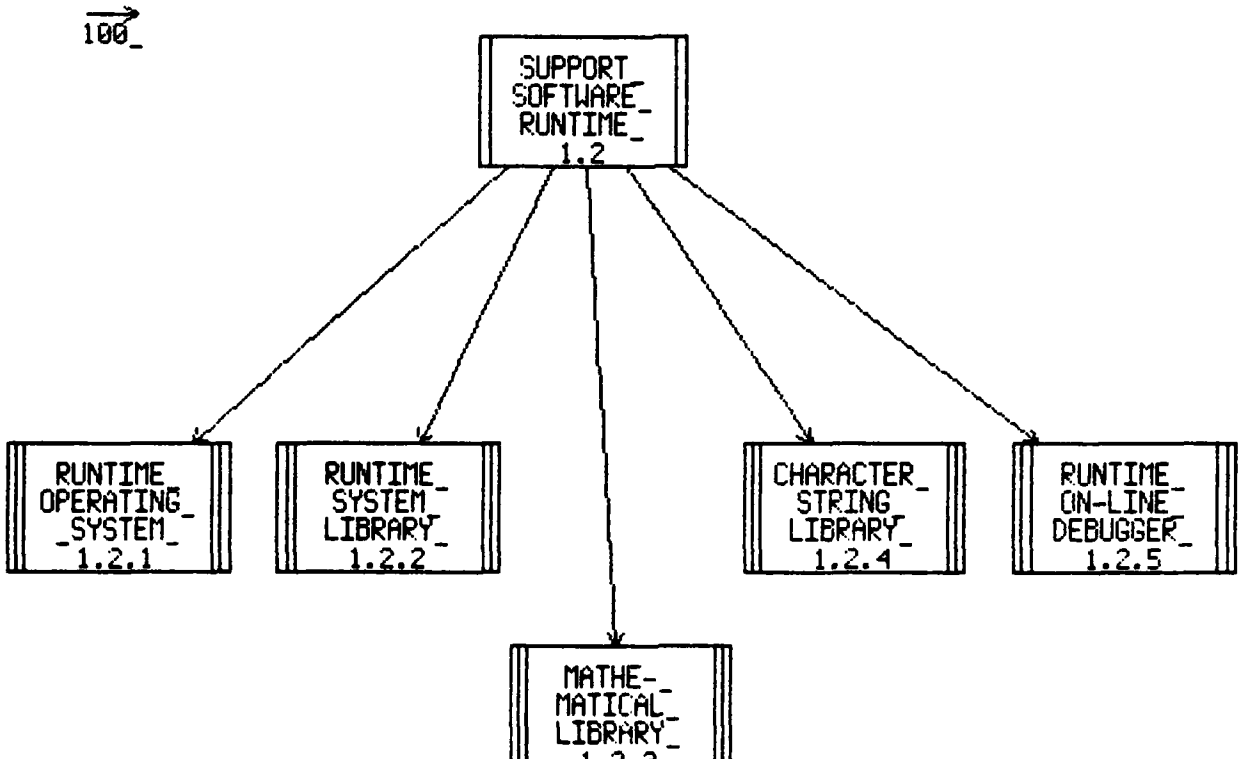


TITLE: 1.1 GENERAL OPERATING SYSTEM SED		PAGE: 101
REVISION 3	AUTHOR: RAW	DATE: 3/24/82
NOTES:		

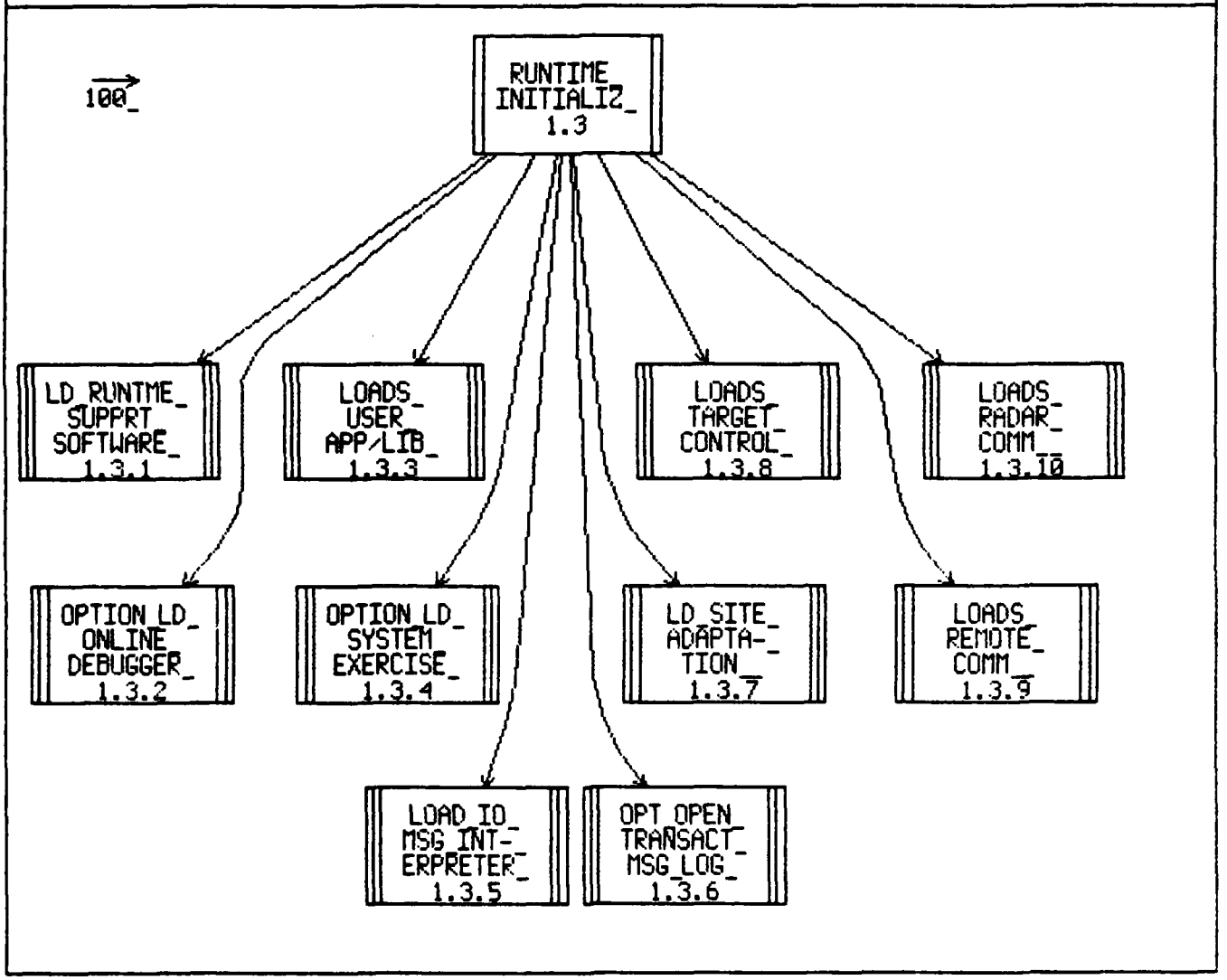


TITLE: 1.2 RUNTIME SUPPORT SOFTWARE SED		PAGE: 102
REVISION 2	AUTHOR: RAW	DATE: 3/24/82

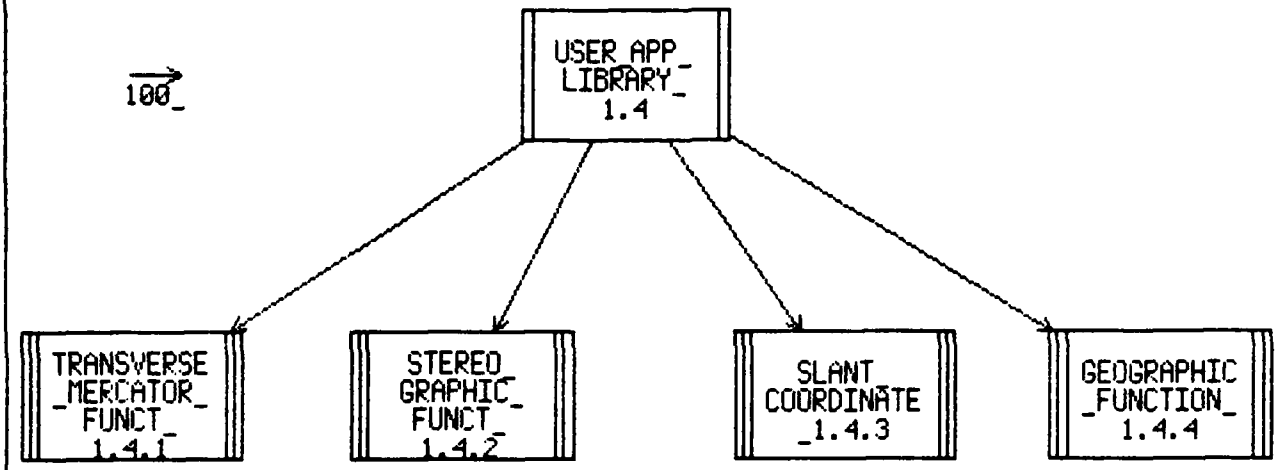
NOTES:



TITLE: 1.3 RUNTIME INITIALIZATION SED		PAGE: 103
REVISION 4	AUTHOR: RAW/WAS	DATE: 3/24/82
NOTES:		

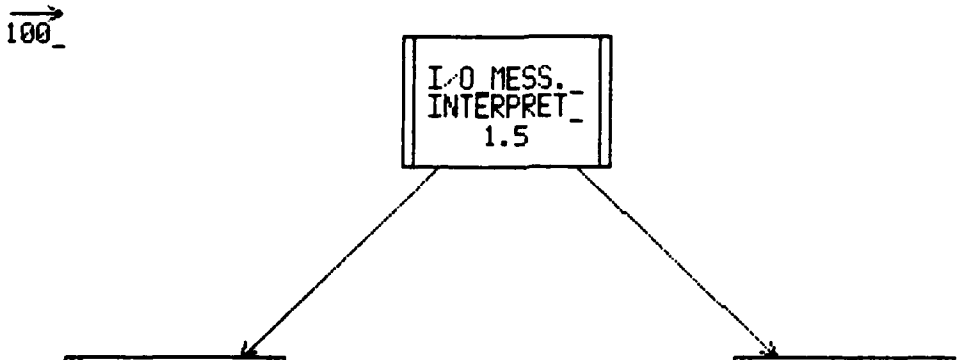


TITLE: 1.4 USER APPLICATION LIBRARY SED		PAGE: 104
REVISION 1	AUTHOR: RAW/WAS	DATE: 10/12/80
NOTES:		



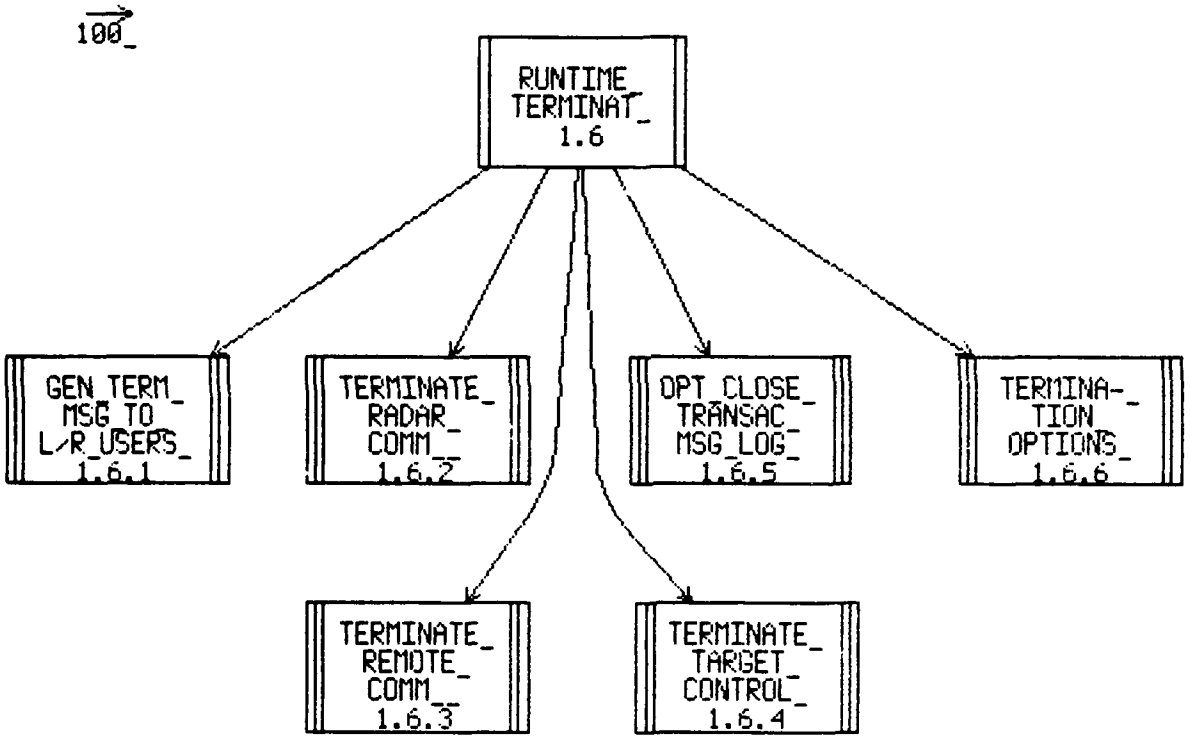
TITLE: 1.5 I/O MESSAGE INTERPRETER SED		PAGE: 105
REVISION 2	AUTHOR: RAW	DATE: 3/24/82

NOTES:

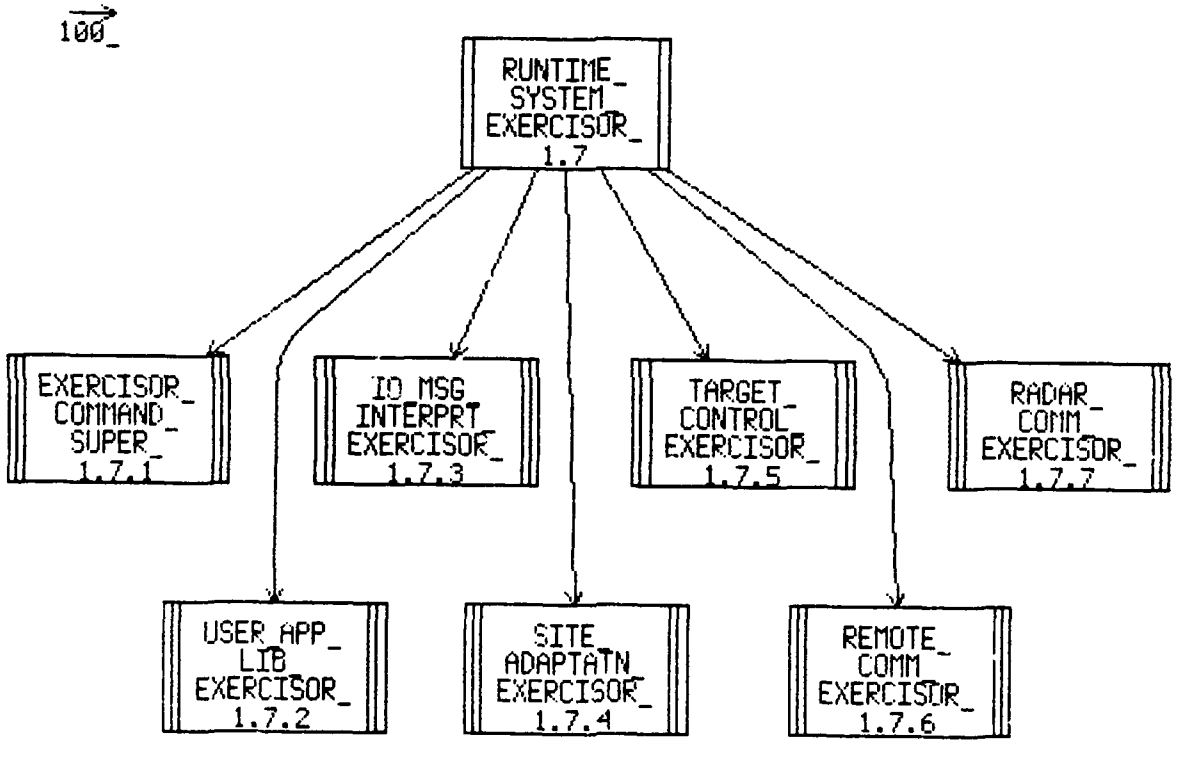


TITLE: 1.6 RUNTIME TERMINATION SED		PAGE: 106
REVISION 2	AUTHOR: RAW	DATE: 3/24/82

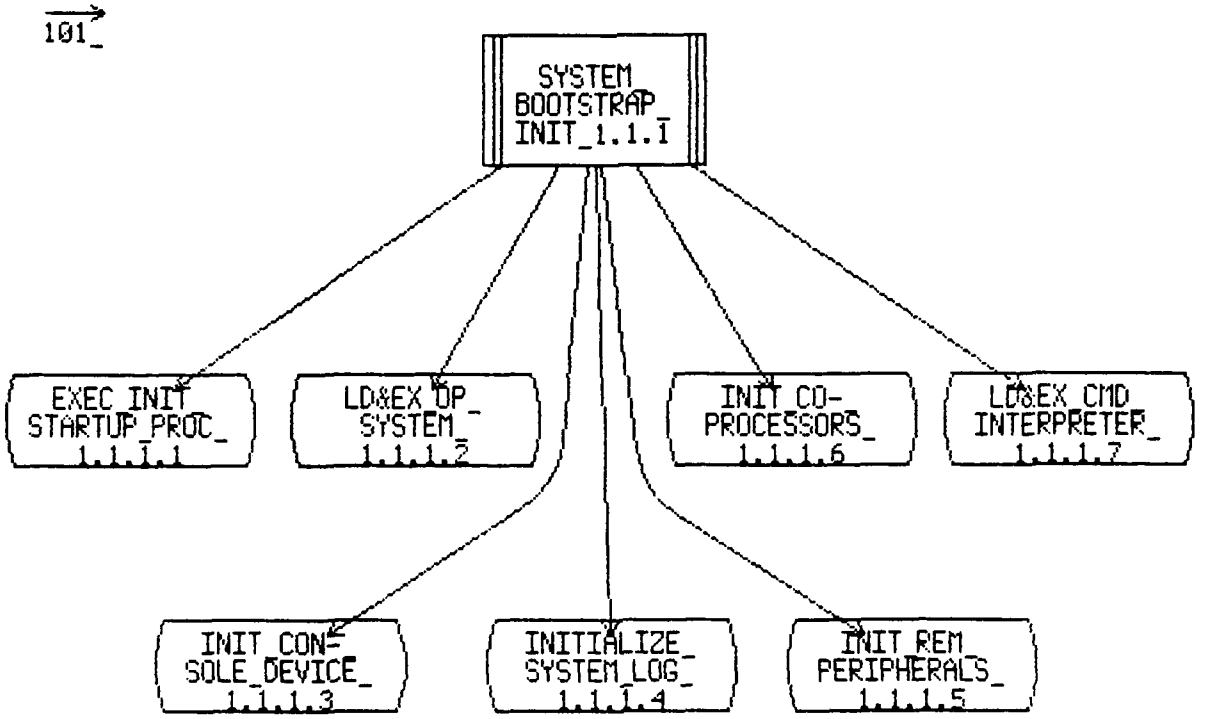
NOTES:



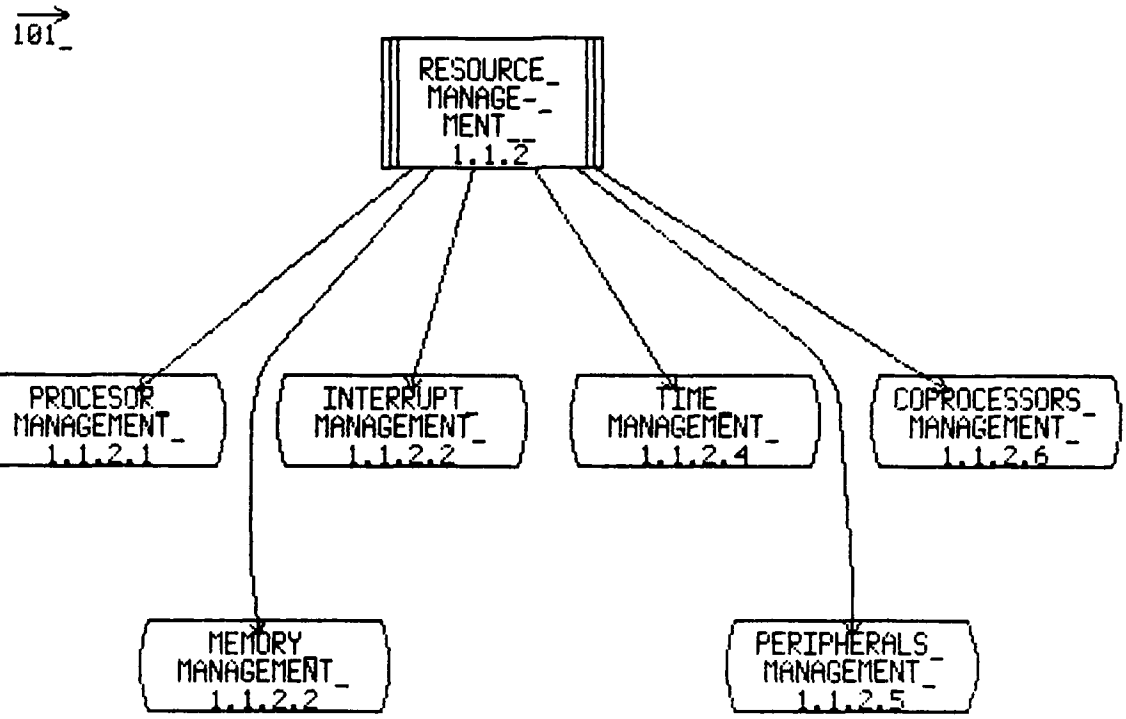
TITLE: 1.7 RUNTIME SYSTEM EXERCISOR SED		PAGE: 107
REVISION 2	AUTHOR: RAW	DATE: 3/24/82
NOTES:		



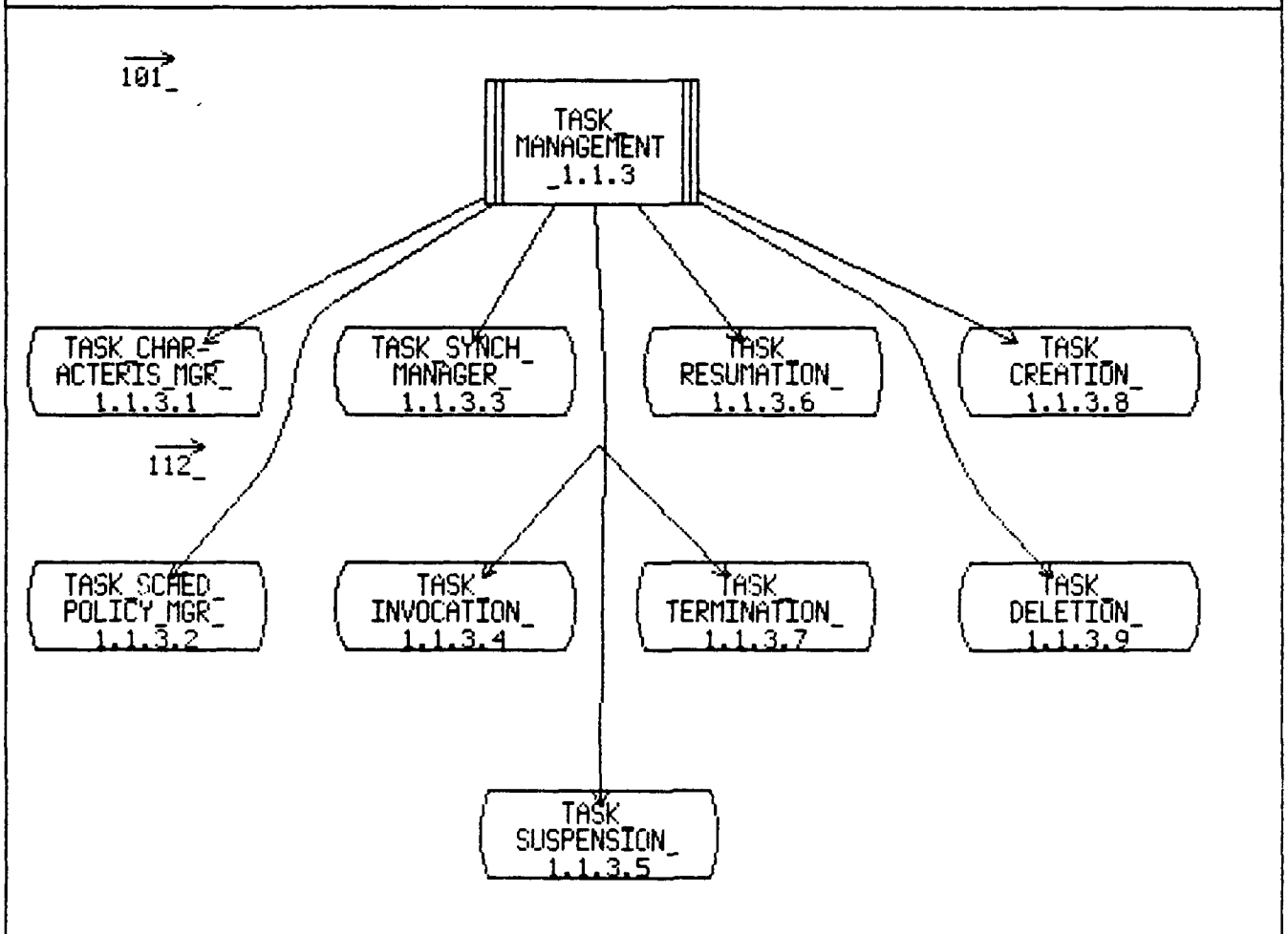
TITLE: 1.1.1 SYSTEM BOOTSTRAP INITIALIZATION SED		PAGE: 109
REVISION 0	AUTHOR: RAW	DATE: 3/24/82
NOTES:		



TITLE: 1.1.2 RESOURCE MANAGEMENT SED		PAGE: 110
REVISION 1	AUTHOR: RAW	DATE: 3/25/82
NOTES:		

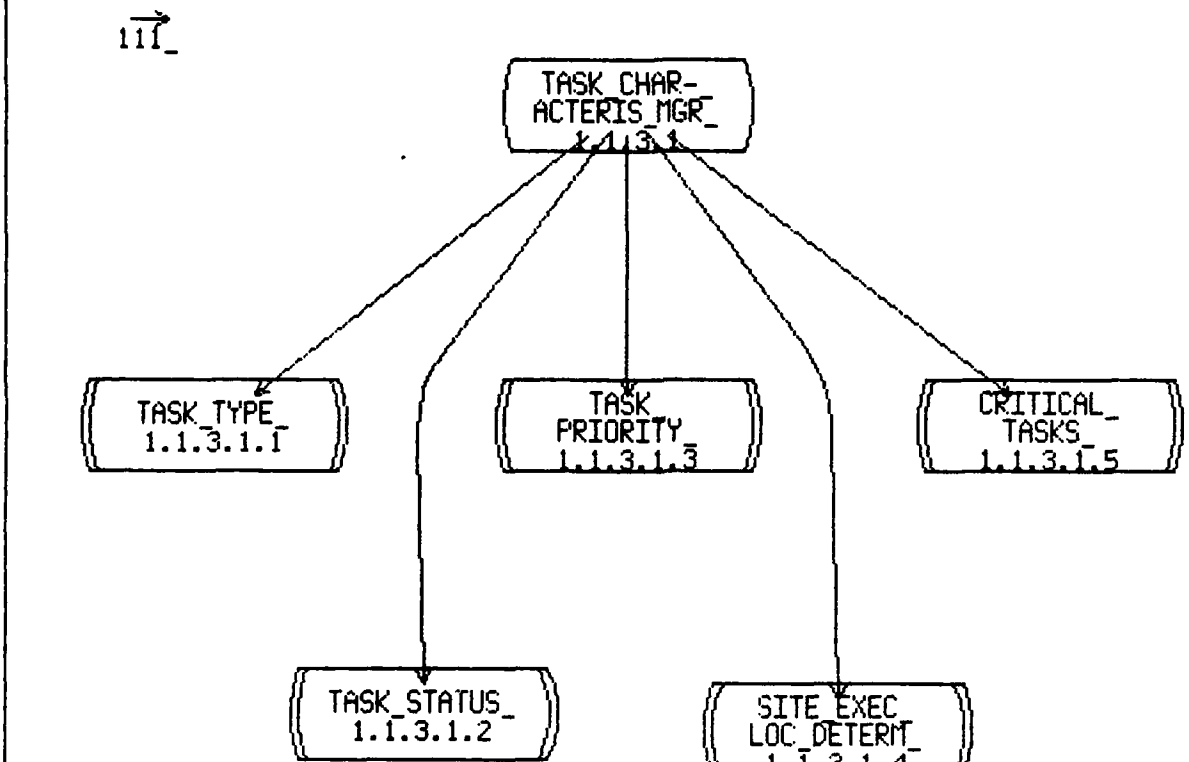


TITLE: 1.1.3 TASK MANAGEMENT SED		PAGE: 111
REVISION 1	AUTHOR: RAW	DATE: 3/26/82
NOTES:		



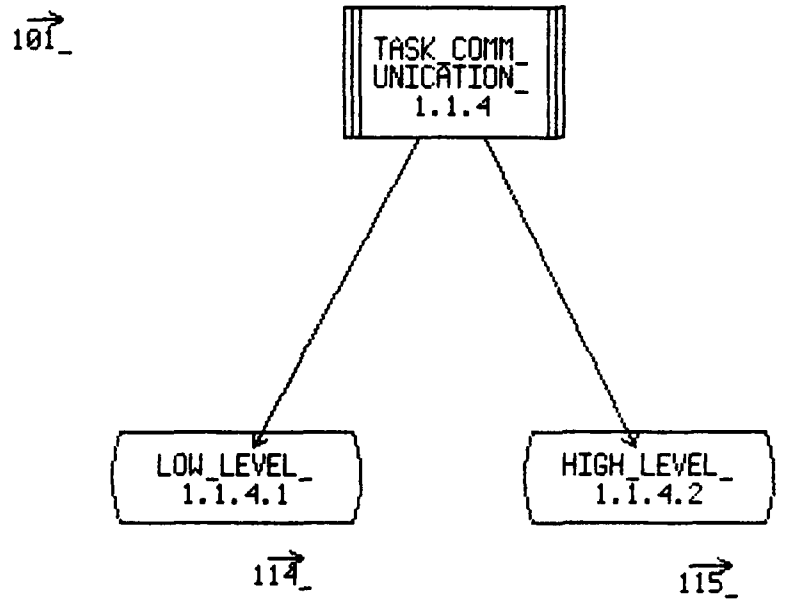
TITLE: 1.1.3.1 TASK CHARACTERISTICS MANAGER SED		PAGE: 112
REVISION 1	AUTHOR: RAW	DATE: 82/03/26.

NOTES:
3/26/82

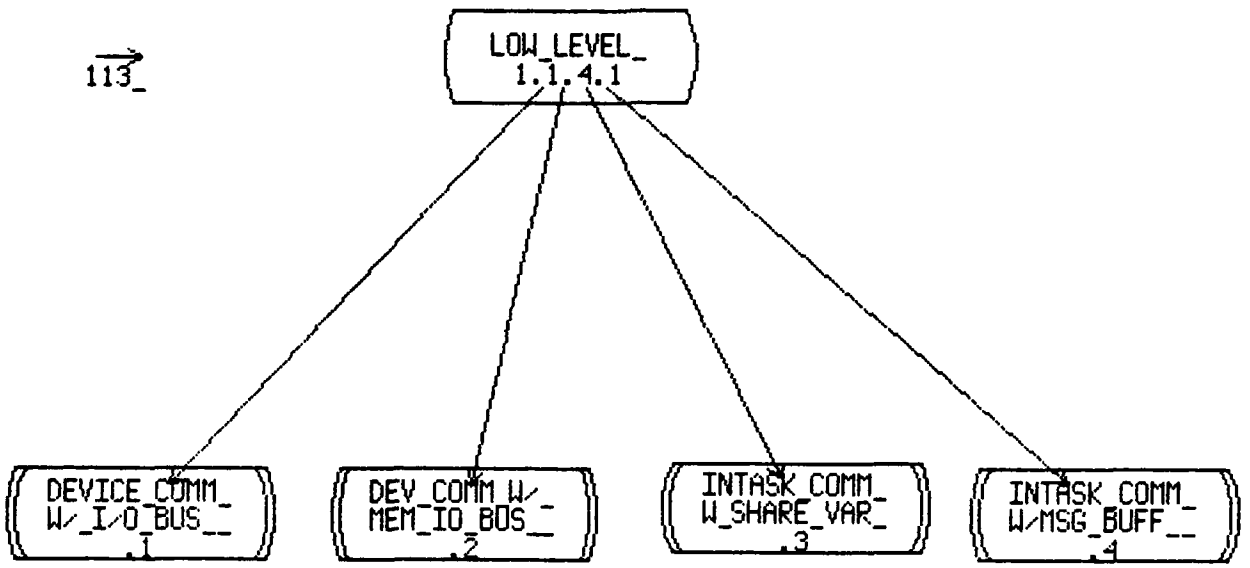


TITLE: 1.1.4 TASK COMMUNICATIONS SED		PAGE: 113
REVISION 0	AUTHOR: RAW	DATE: 3/26/82

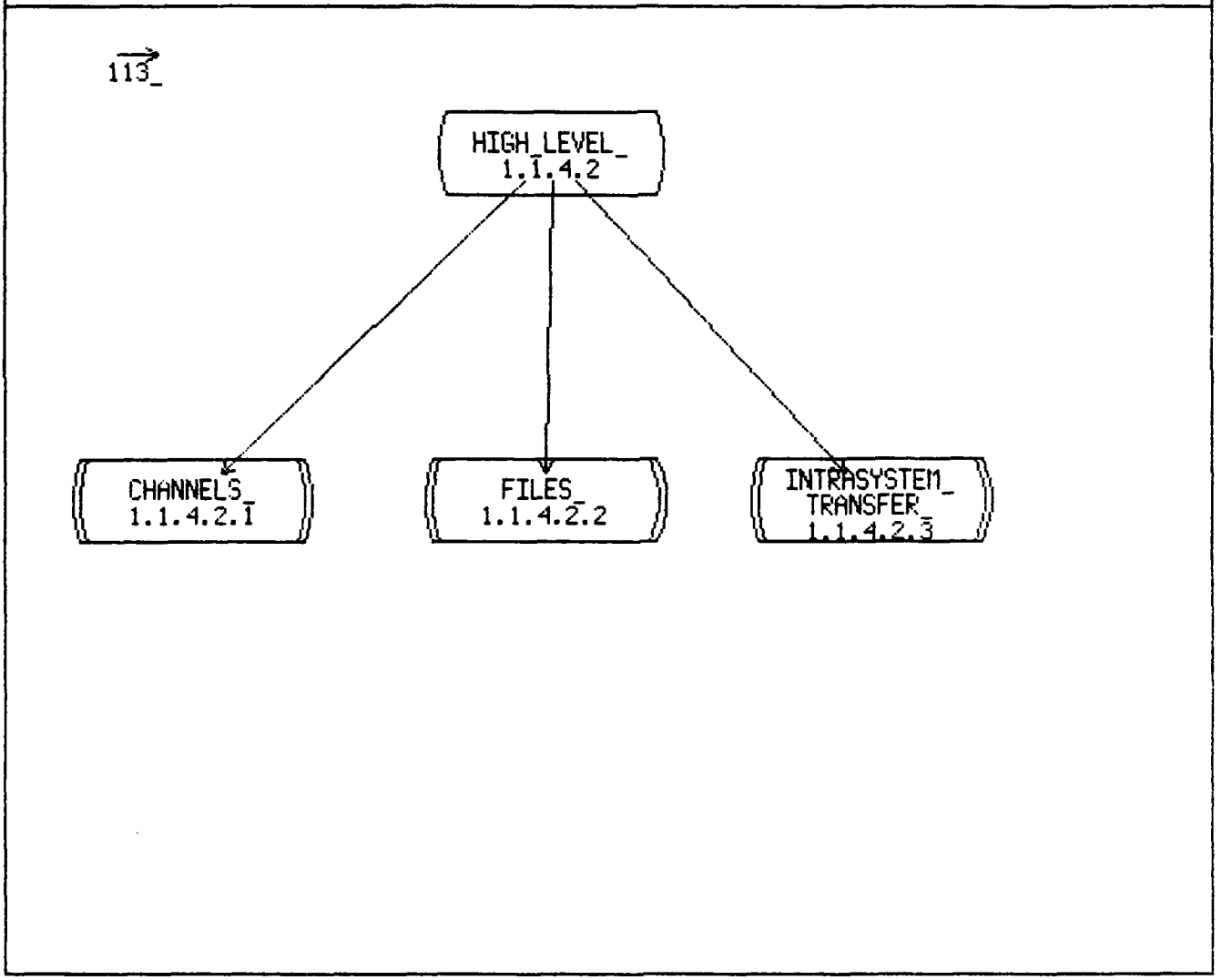
NOTES:



TITLE: 1.1.4.1 LOW LEVEL SED		PAGE: 114
REVISION 1	AUTHOR: RAW	DATE: 3/26/82
NOTES:		

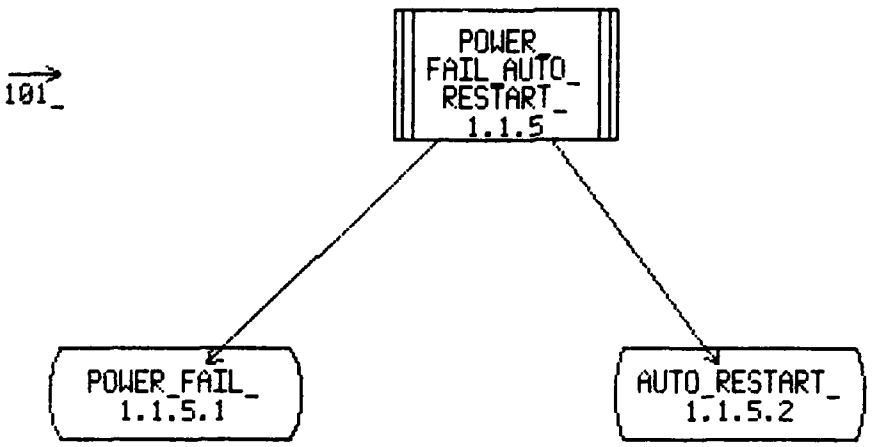


TITLE: 1.1.4.2 HIGH LEVEL SED		PAGE: 115
REVISION 1	AUTHOR: RAW	DATE: 3/26/82
NOTES:		

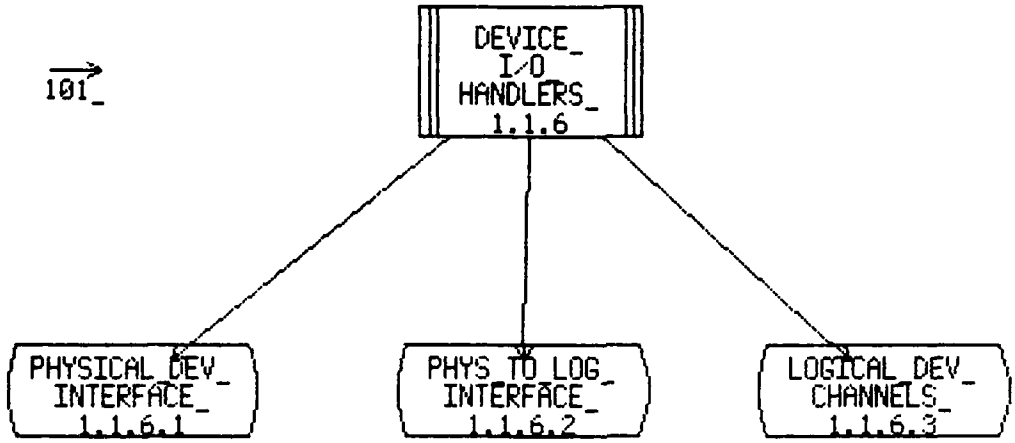


TITLE: 1.1.5 POWER FAIL/AUTOMATIC RESTART SED		PAGE: 116
REVISION 1	AUTHOR: RAW	DATE: 82/03/26.

NOTES:

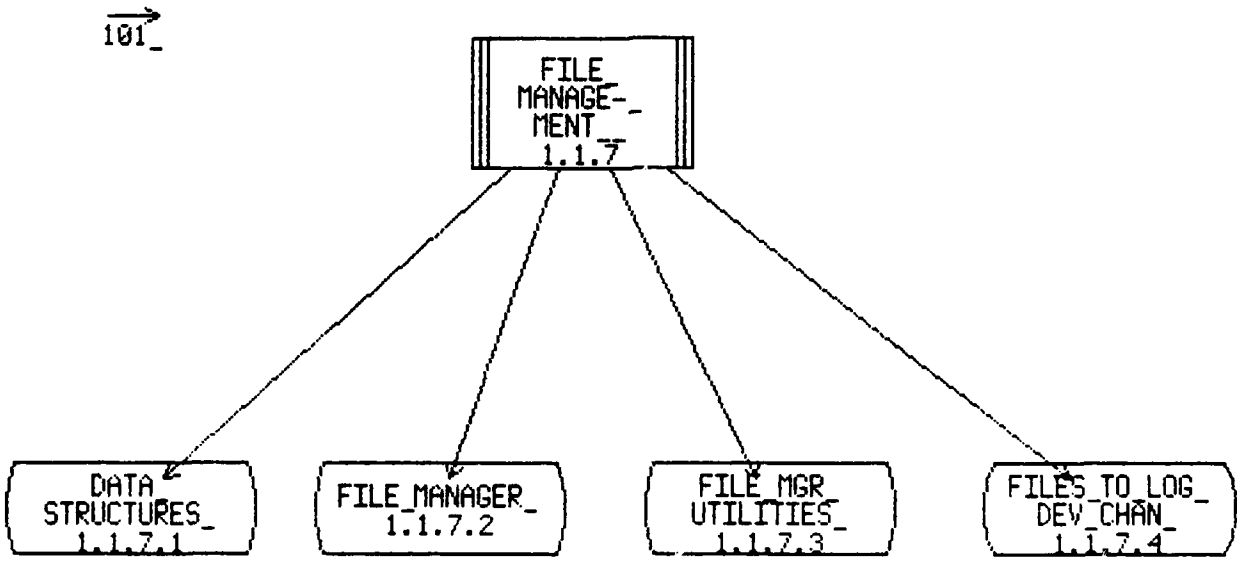


TITLE: 1.1.6 DEVICE I/O HANDLERS SED		PAGE: 117
REVISION 1	AUTHOR: DRAW	DATE: 3/26/82
NOTES:		

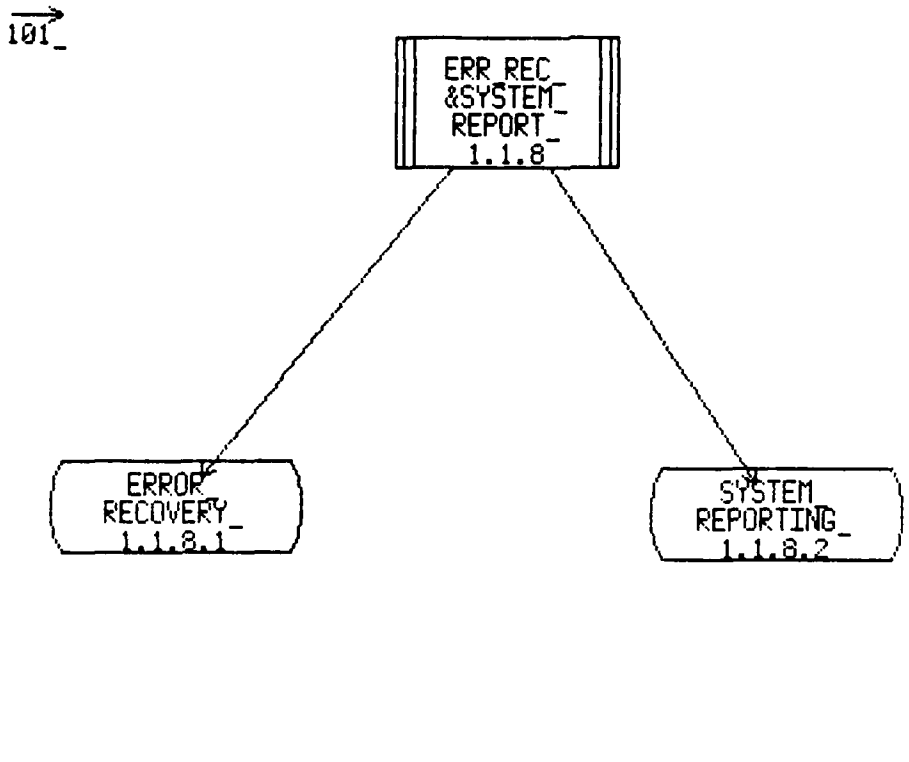


TITLE: 1.1.7 FILE MANAGEMENT SED		PAGE: 118
REVISION 1	AUTHOR: RAW	DATE: 3/26/82

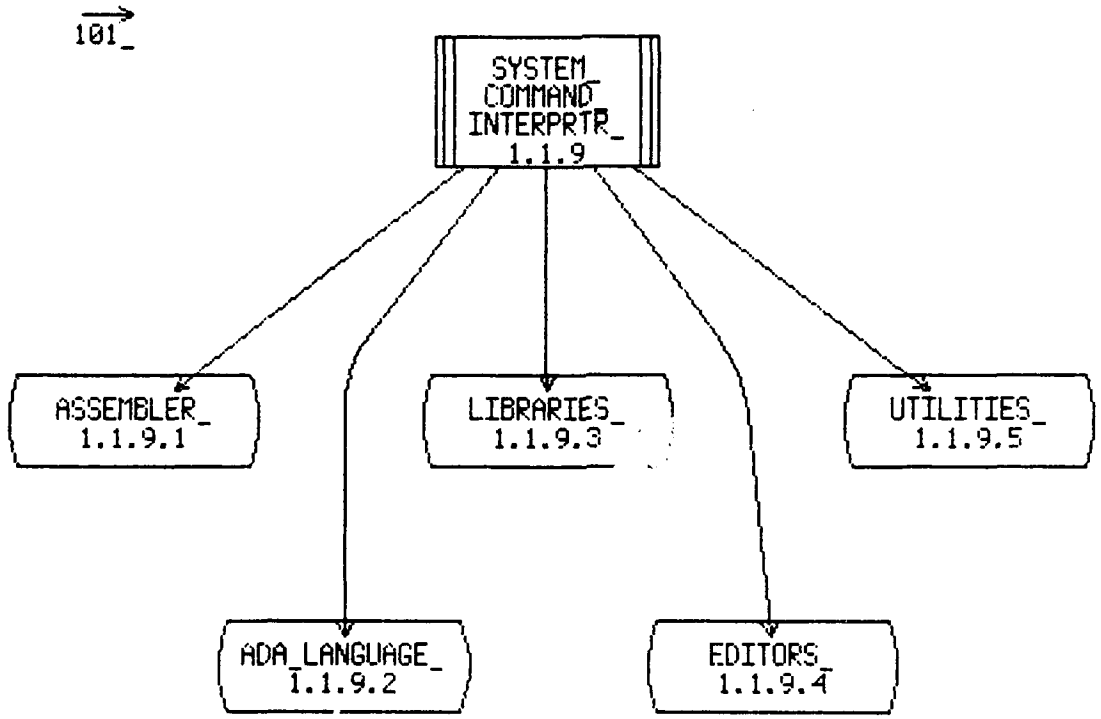
NOTES:



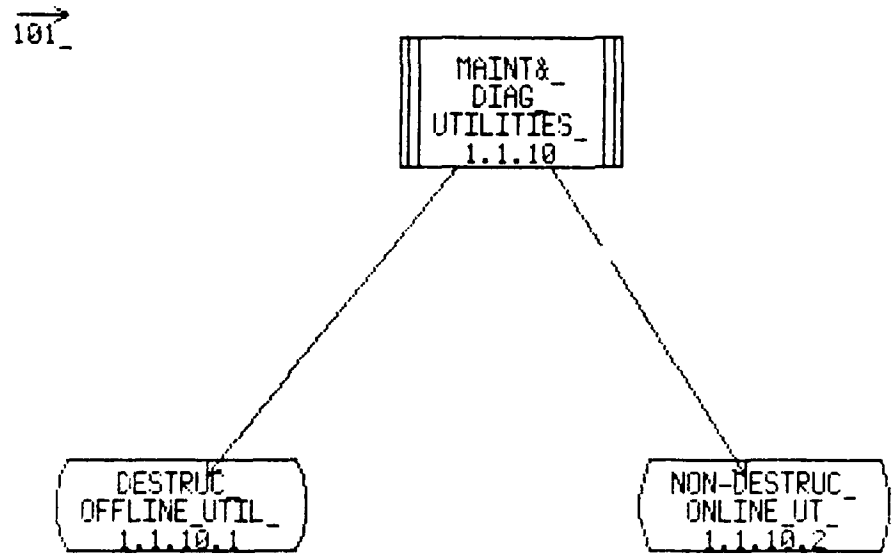
TITLE: 1.1.8 ERROR RECOVERY & SYSTEM REPORTING SED		PAGE: 119
REVISION 1	AUTHOR: RAW	DATE: 2/26/82
NOTES:		



TITLE: 1.1.9 SYSTEM COMMAND INTERPRETER SED		PAGE: 120
REVISION 1	AUTHOR: PAW	DATE: 3/29/82
NOTES:		

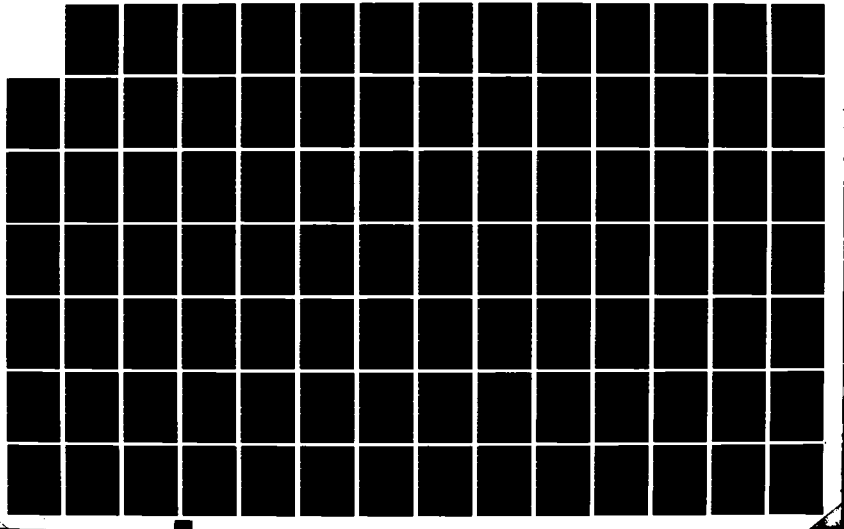


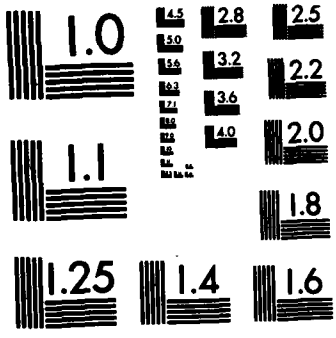
TITLE: 1.1.10 MAINTENANCE & DIAGNOSTIC UTILITIES SED		PAGE: 121
REVISION 1	AUTHOR: RAW	DATE: 3/29/82
NOTES:		



AD-A123 308 LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE MISSILE MINDER AN/TSQ-73 USING T..(U) CONTROL DATA CORP SHREWSBURY NJ GOVERNMENT SYSTEMS 09 NOV 82 4/6

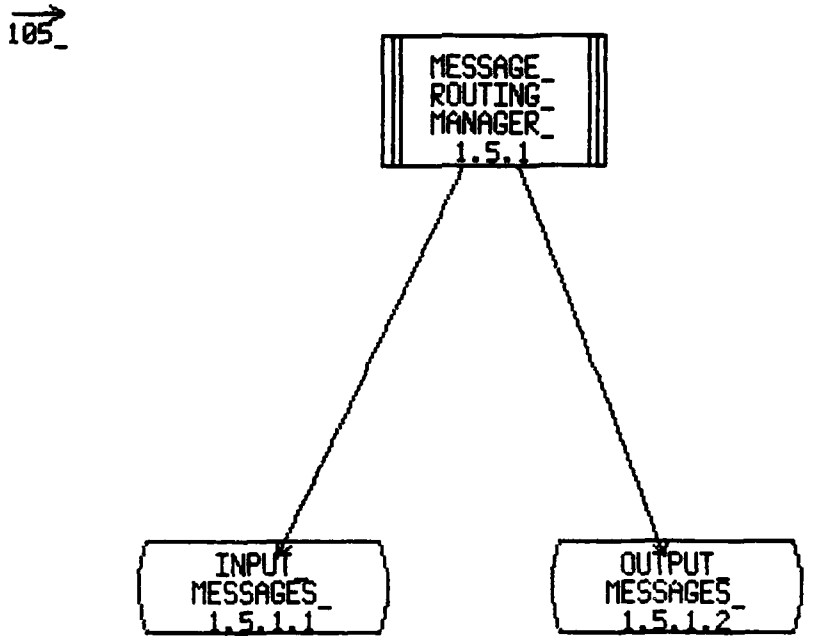
UNCLASSIFIED DAAK80-81-C-0107 F/G 9/2 NL





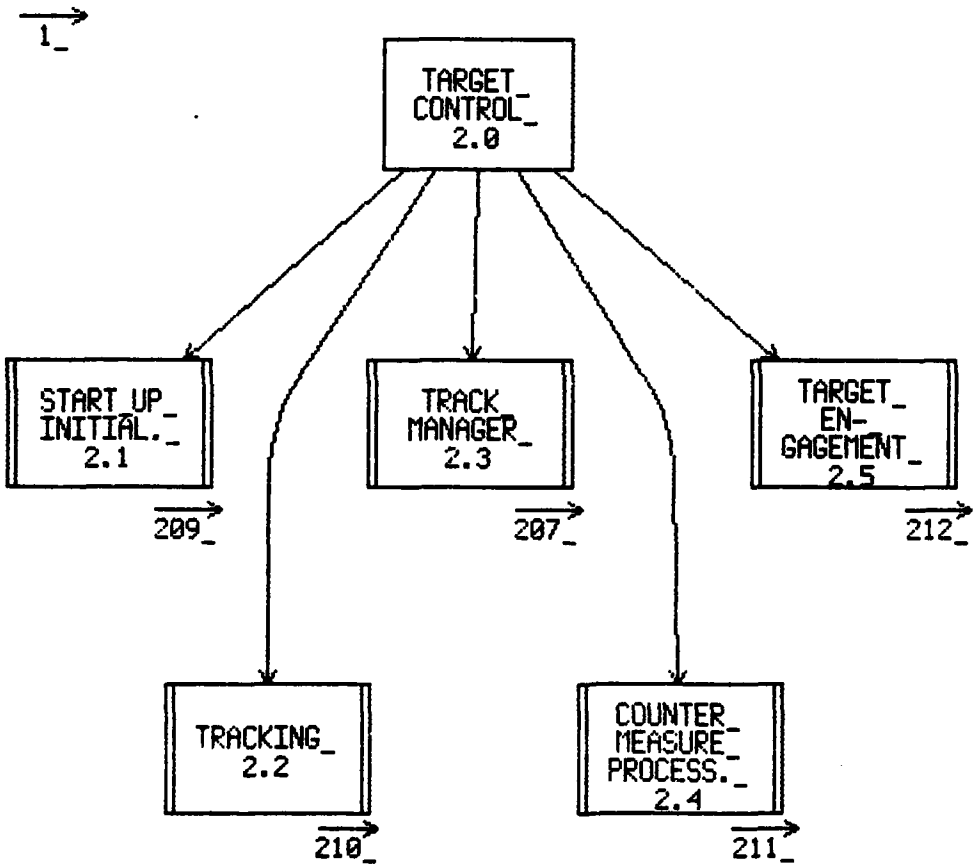
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

TITLE: 1.5.1 MESSAGE ROUTING MANAGER SED		PAGE: 122
REVISION 1	AUTHOR: RAW	DATE: 3/29/82
NOTES:		

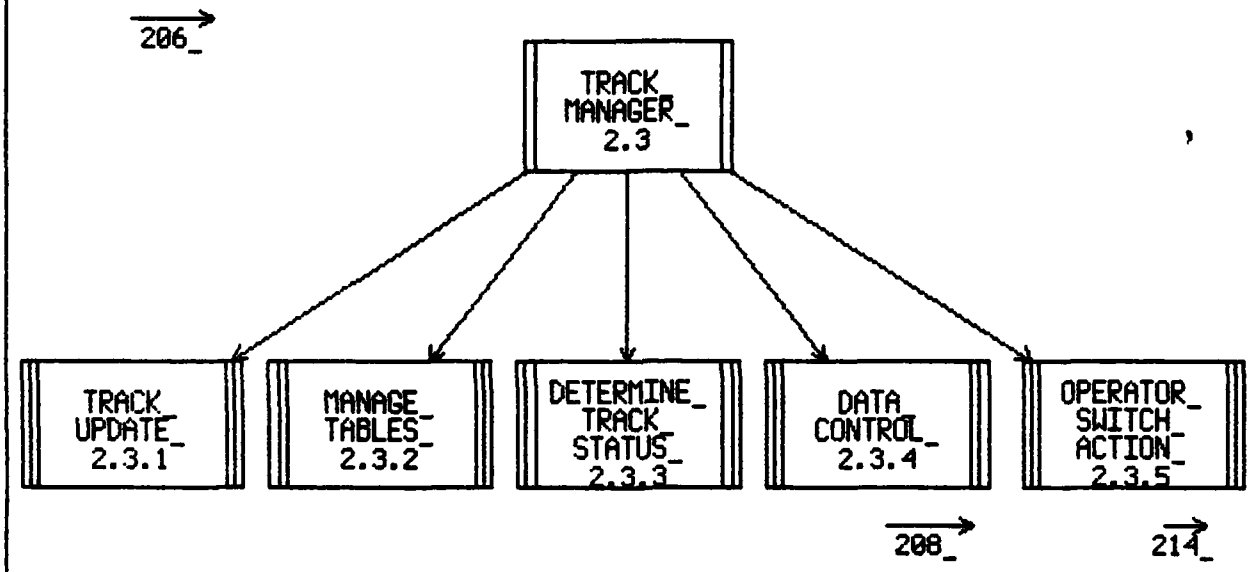


TITLE: 2.0 TARGET CONTROL SED		PAGE: 206
REVISION 5	AUTHOR: MJD/PBM/WAS	DATE: 3/24/82

NOTES:
REVISION OF DIAGRAM 205

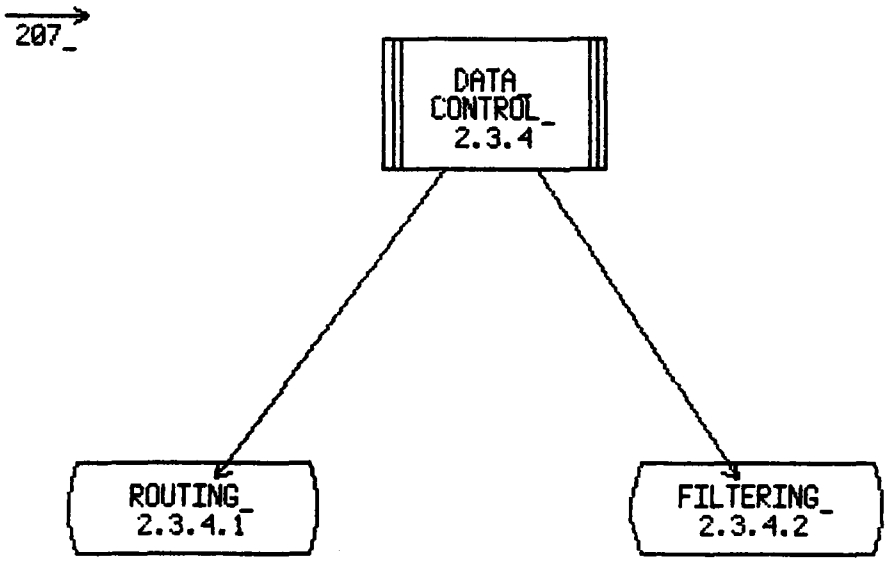


TITLE: 2.3 TRACK MANAGER SED		PAGE: 207
REVISION 1	AUTHOR: MJD/PBM/WAS	DATE: 3/24/82
NOTES:		



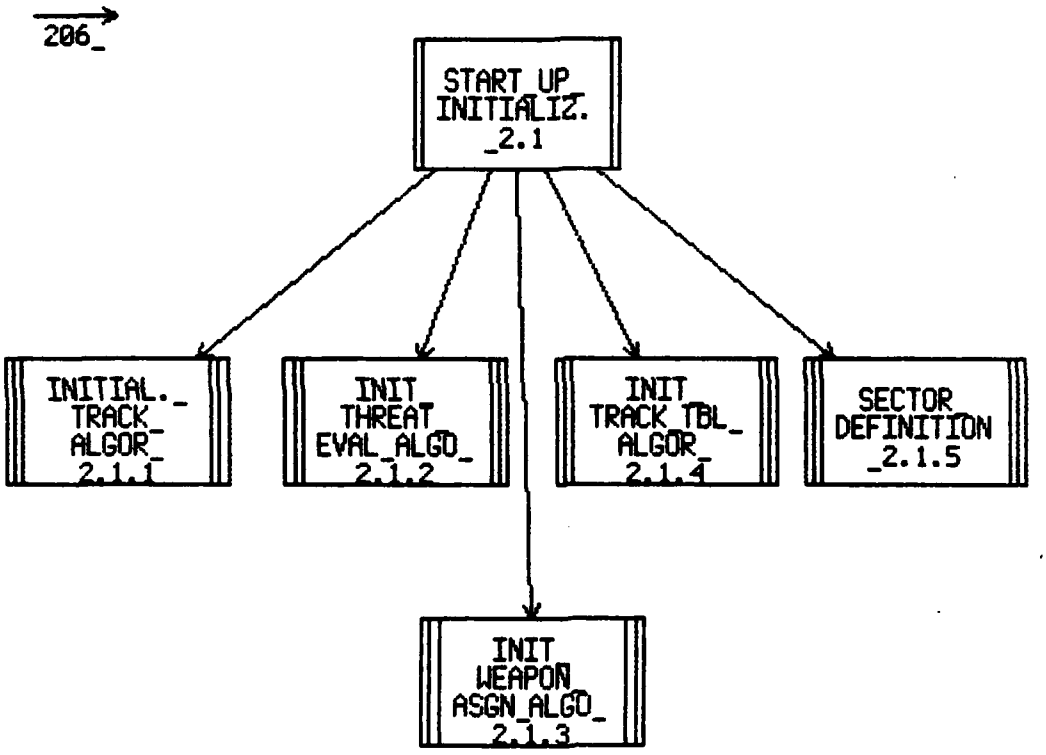
TITLE: 2.3.4 DATA CONTROL SED		PAGE: 208
REVISION 0	AUTHOR: MJD/PBM/WAS	DATE: 1/5/82

NOTES:

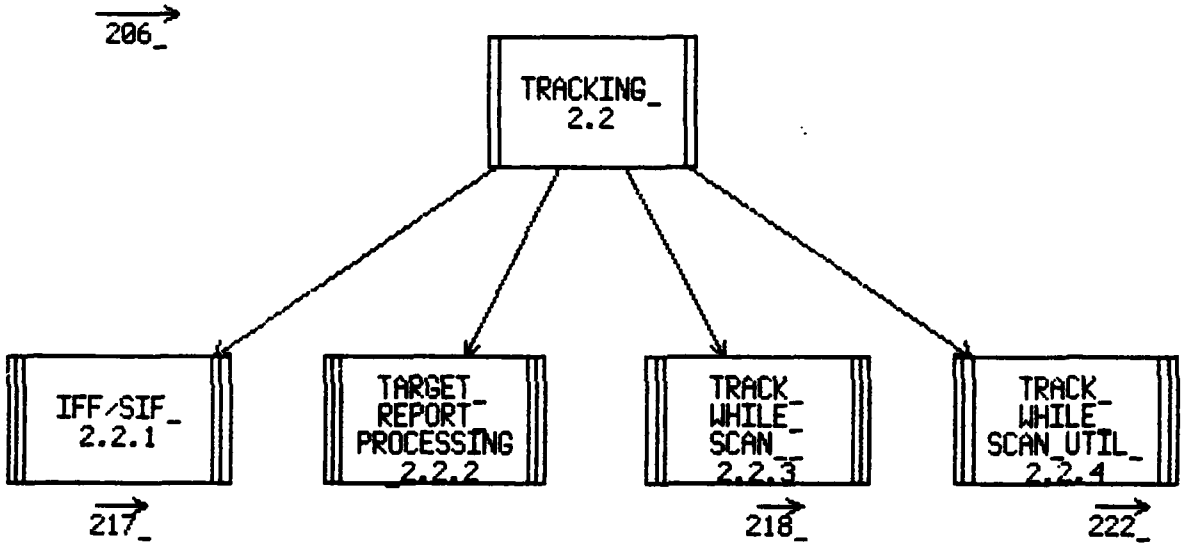


TITLE: 2.1 START UP INITIALIZATION SED		PAGE: 209
REVISION 0	AUTHOR: MJD/PBM/WAS	DATE: 1/5/82

NOTES:

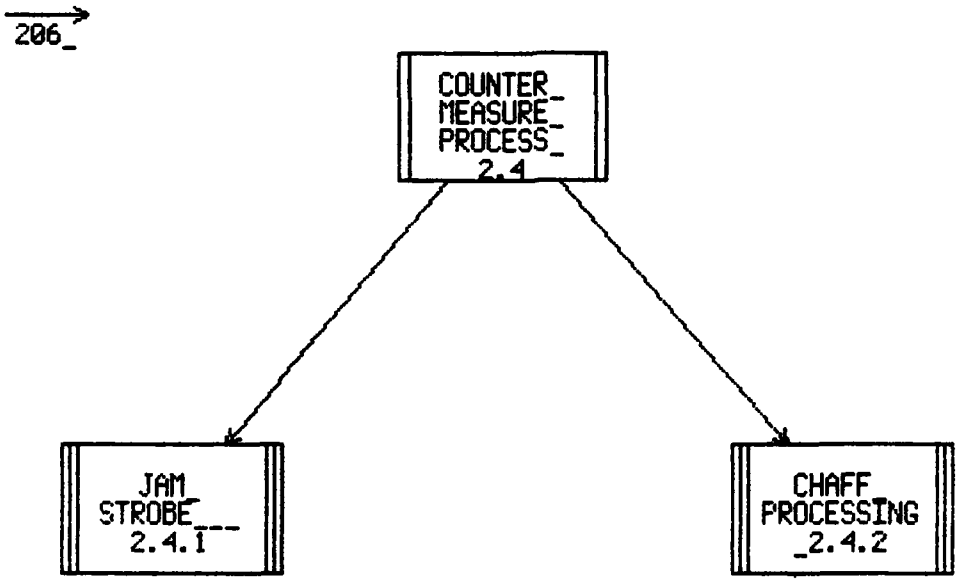


TITLE: 2.2 TRACKING SED		PAGE: 210
REVISION 2	AUTHOR: LWD	DATE: 5/14/82
NOTES:		

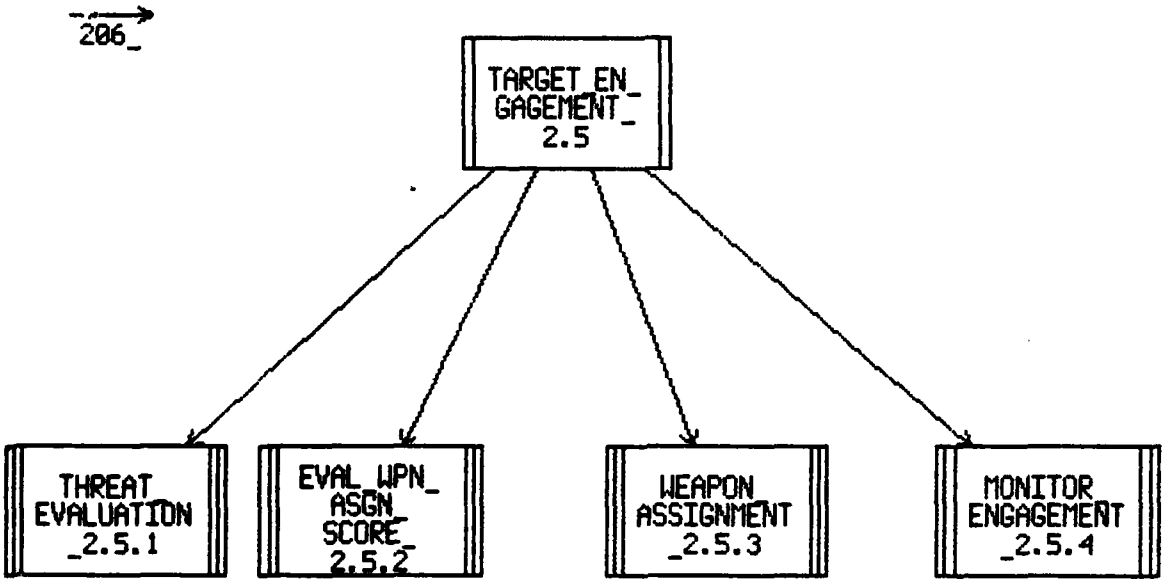


TITLE: 2.4 COUNTER MEASURE PROCESSING SED		PAGE: 211
REVISION 0	AUTHOR: MJD/PBM/WAS	DATE: 1/5/82

NOTES:

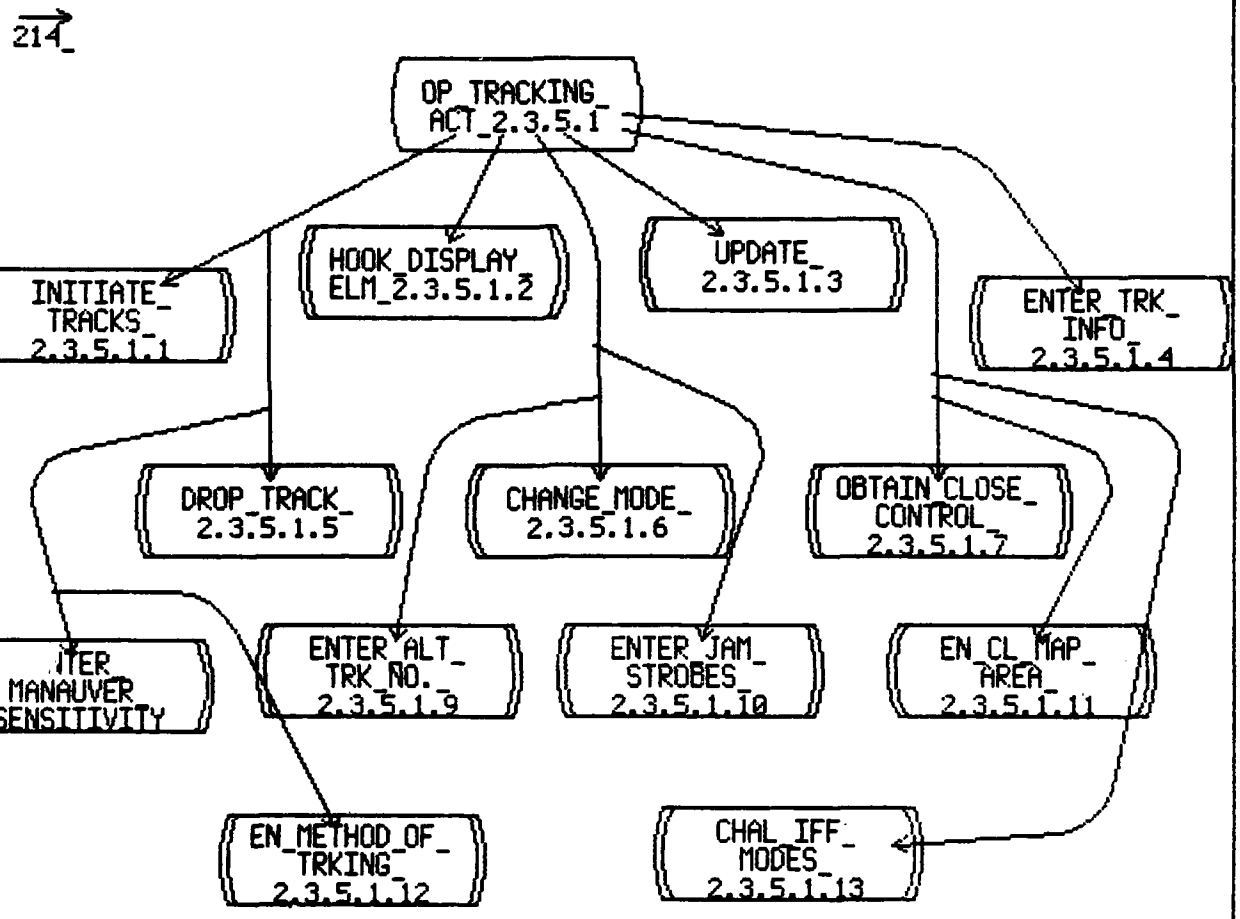


TITLE: 2.5 TARGET ENGAGEMENT SED		PAGE: 212
REVISION 0	AUTHOR: MJD/PBM/WAS	DATE: 1/5/82
NOTES:		

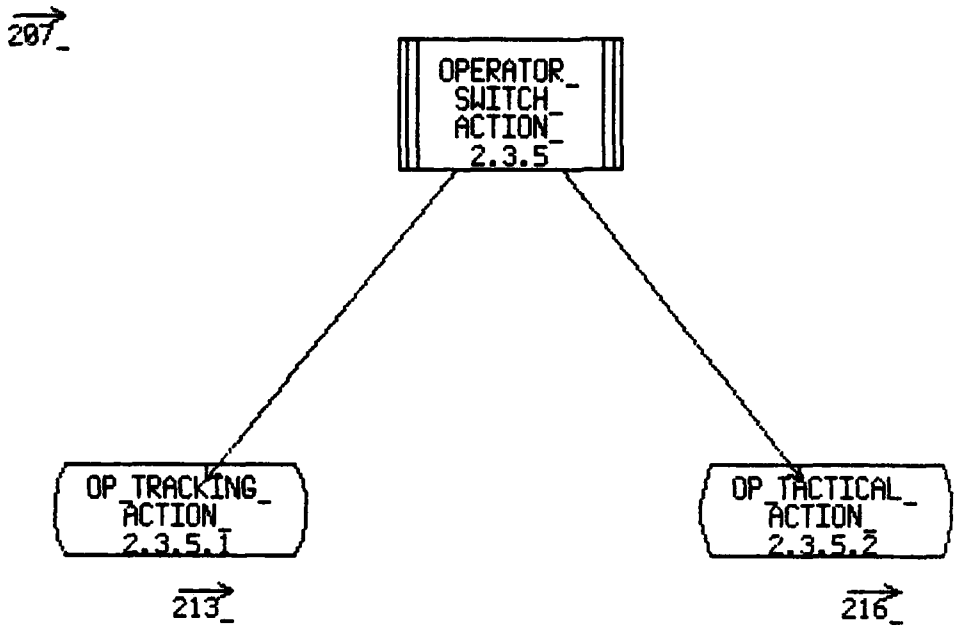


TITLE: 2.3.5.1 OPERATOR TRACKING ACTIONS SED		PAGE: 213
REVISION 1	AUTHOR: MJD/WAS/PBM	DATE: 3/25/82

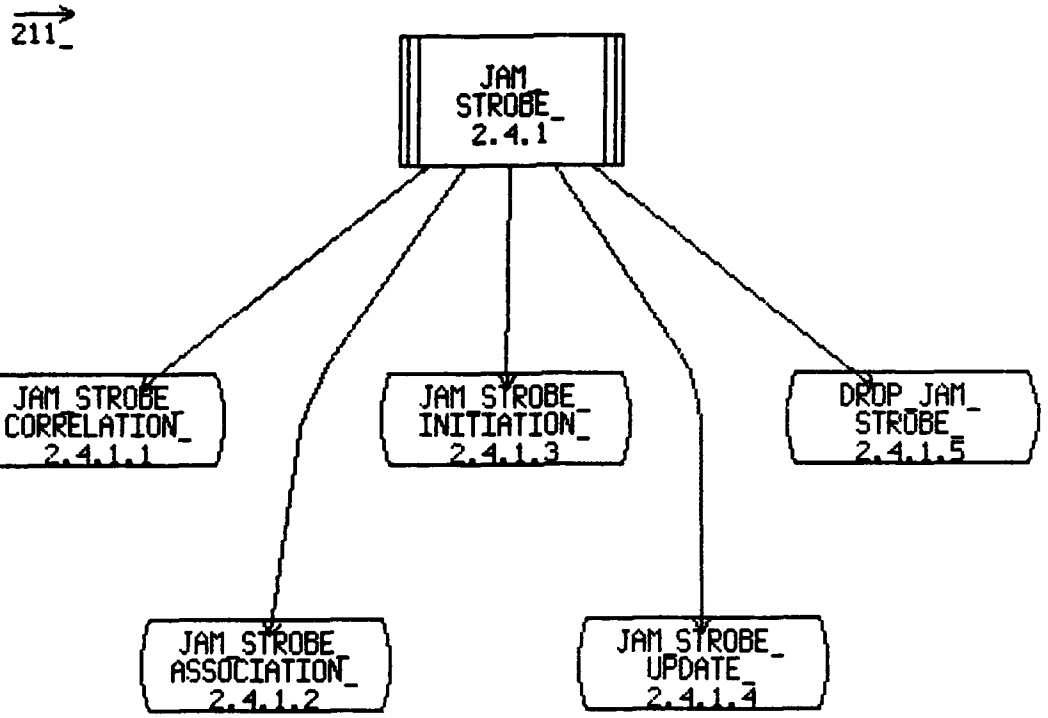
NOTES:
2.3.5.1.8 IS ENTER MANEUVER SENSITIVITY



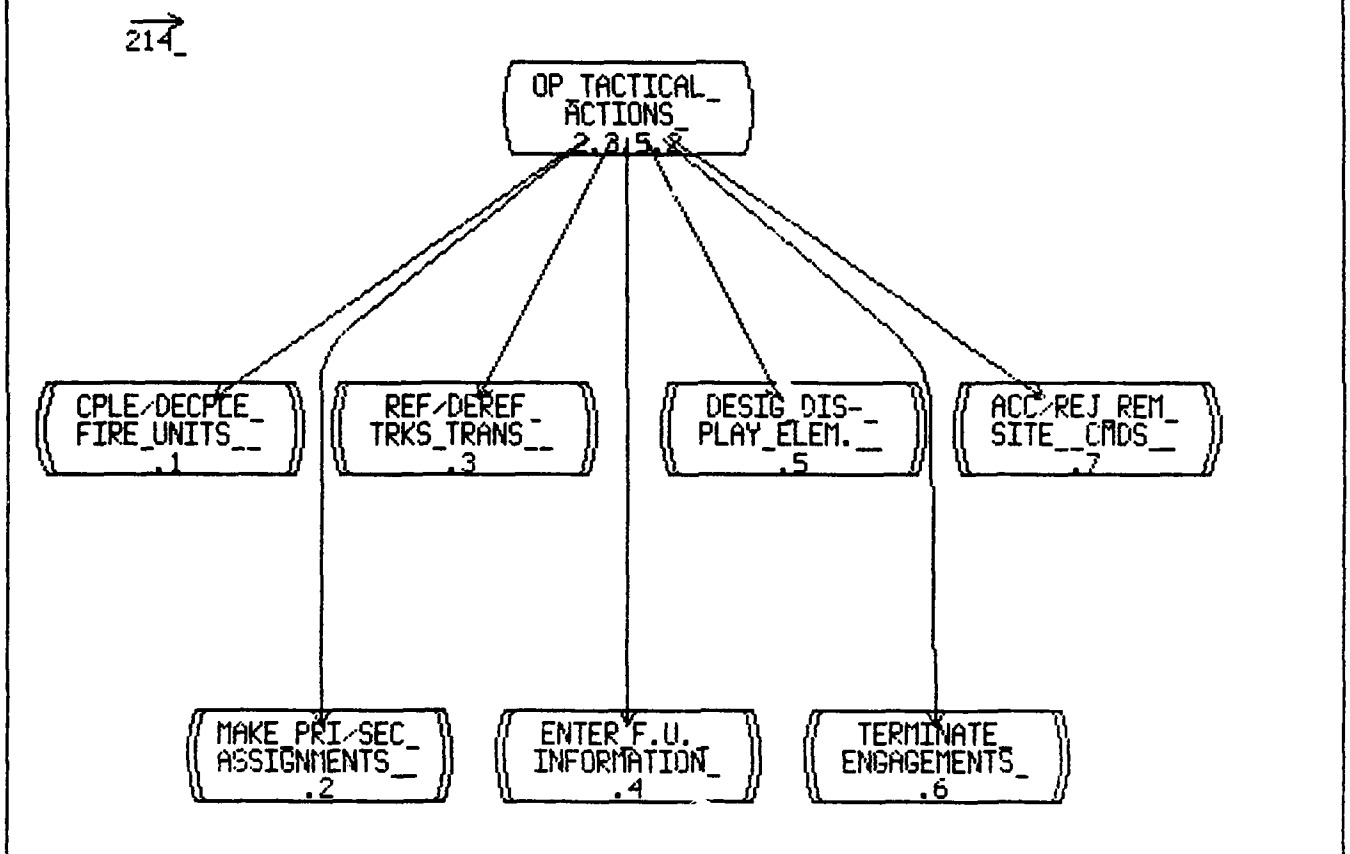
TITLE: 2.3.5 OPERATOR SWITCH ACTION SED		PAGE: 214
REVISION 1	AUTHOR: RAW	DATE: 3/29/82
NOTES:		



TITLE: 2.4.1 JAM STROBE SED		PAGE: 215
REVISION 1	AUTHOR: MJD/PBM	DATE: 3/26/82
NOTES:		

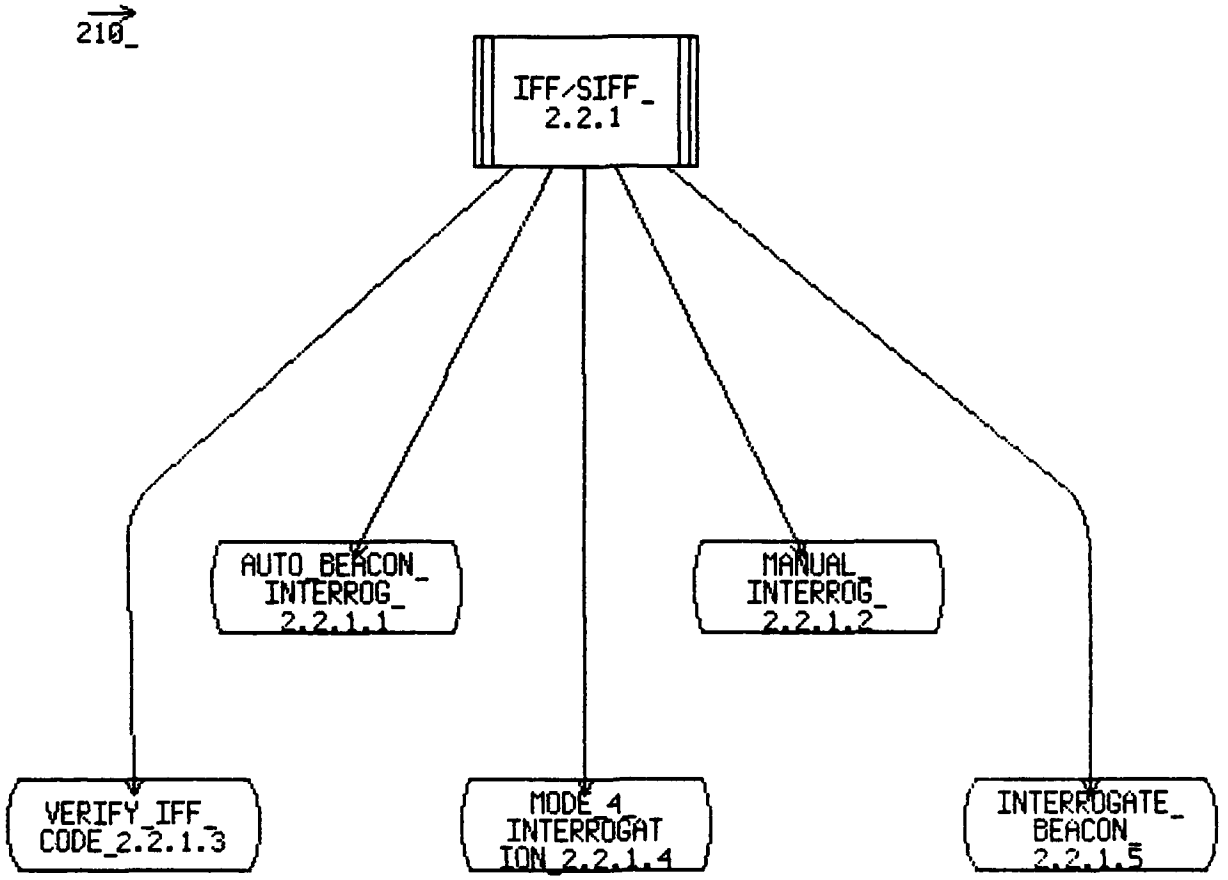


TITLE: 2.3.5.2 OPERATOR TACTICAL ACTIONS SED		PAGE: 216
REVISION 1	AUTHOR: MJD/WAS	DATE: 3/29/82
NOTES:		

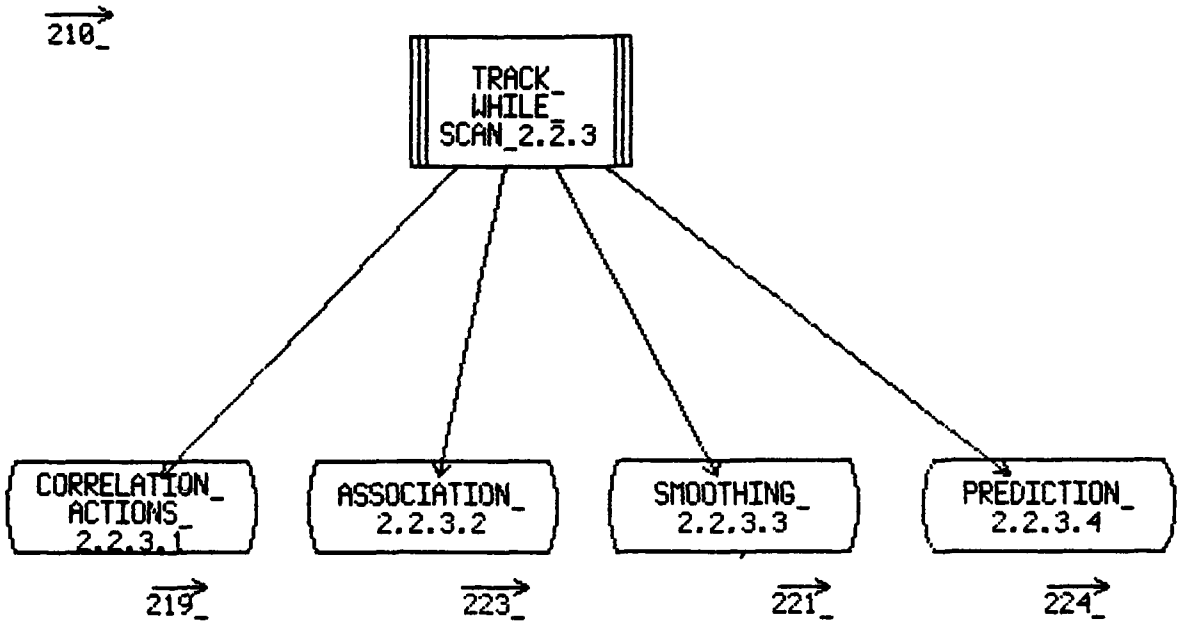


TITLE: 2.2.1 IFF/SIFF SED		PAGE: 217
REVISION 1	AUTHOR: PBM/MJD	DATE: 3/29/82

NOTES:

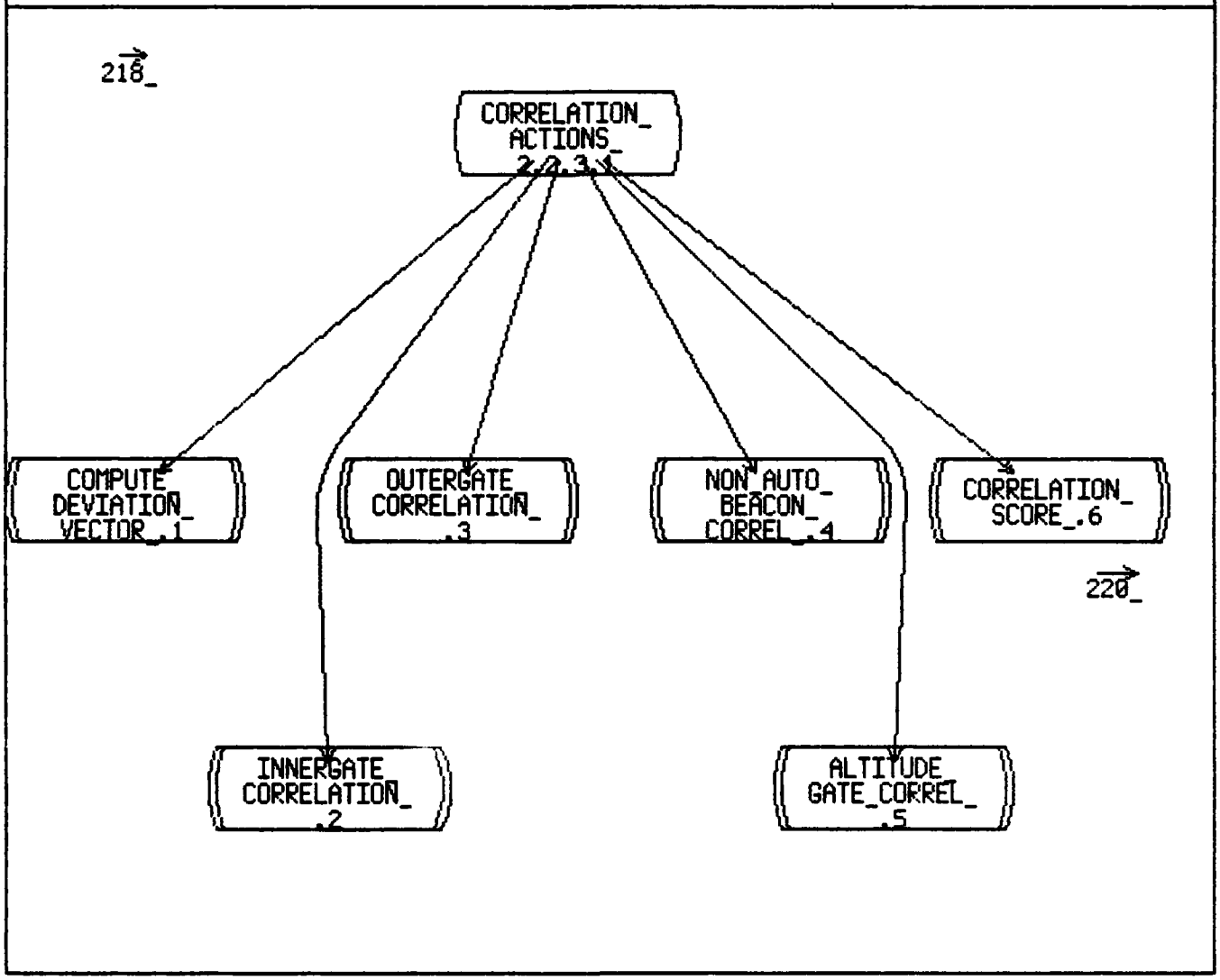


TITLE: 2.2.3 TRACK WHILE SCAN SED		PAGE: 218
REVISION 1	AUTHOR: LWD	DATE: 5/12/82
NOTES:		



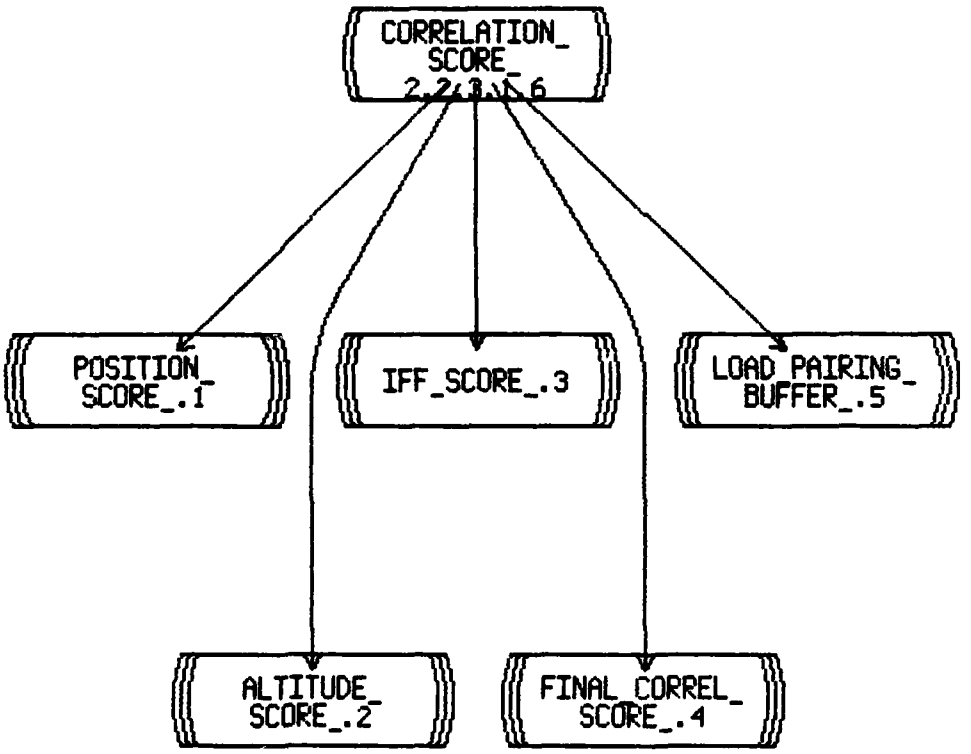
TITLE: 2.2.3.1 CORRELATION ACTIONS SED		PAGE: 219
REVISION 1	AUTHOR: LWD	DATE: 5/12/82

NOTES:

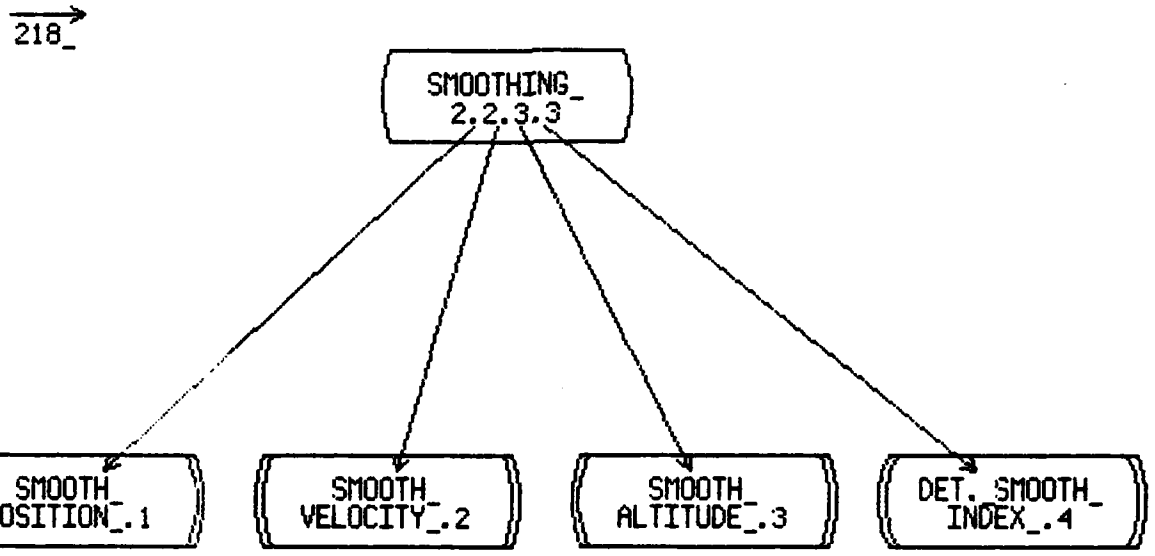


TITLE: 2.2.3.1.6 CORRELATION SCORE SED		PAGE: 220
REVISION 1	AUTHOR: LWD	DATE: 5/14/82
NOTES:		

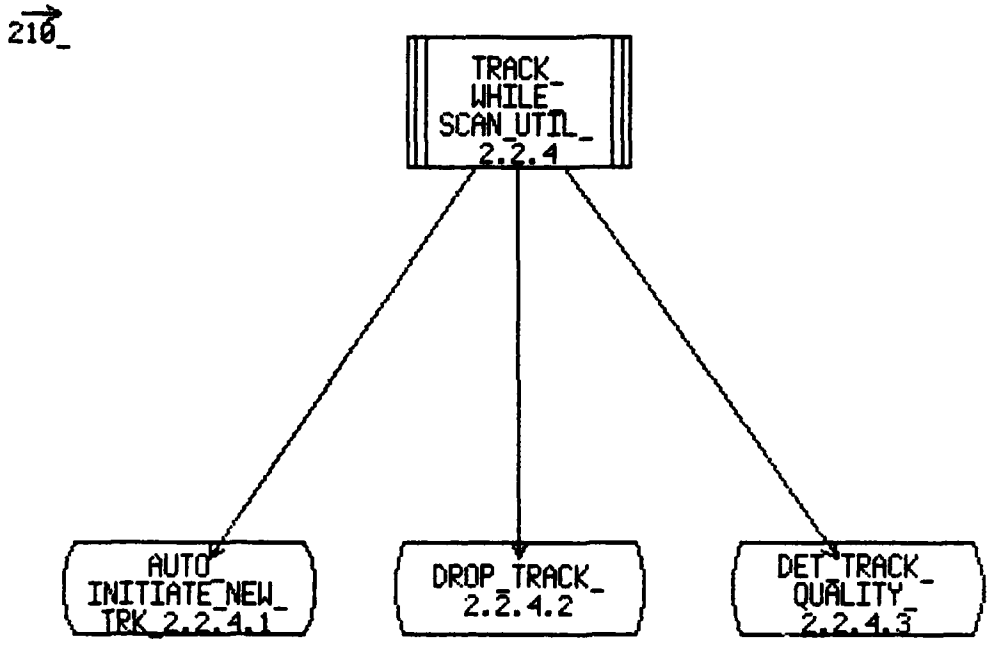
219



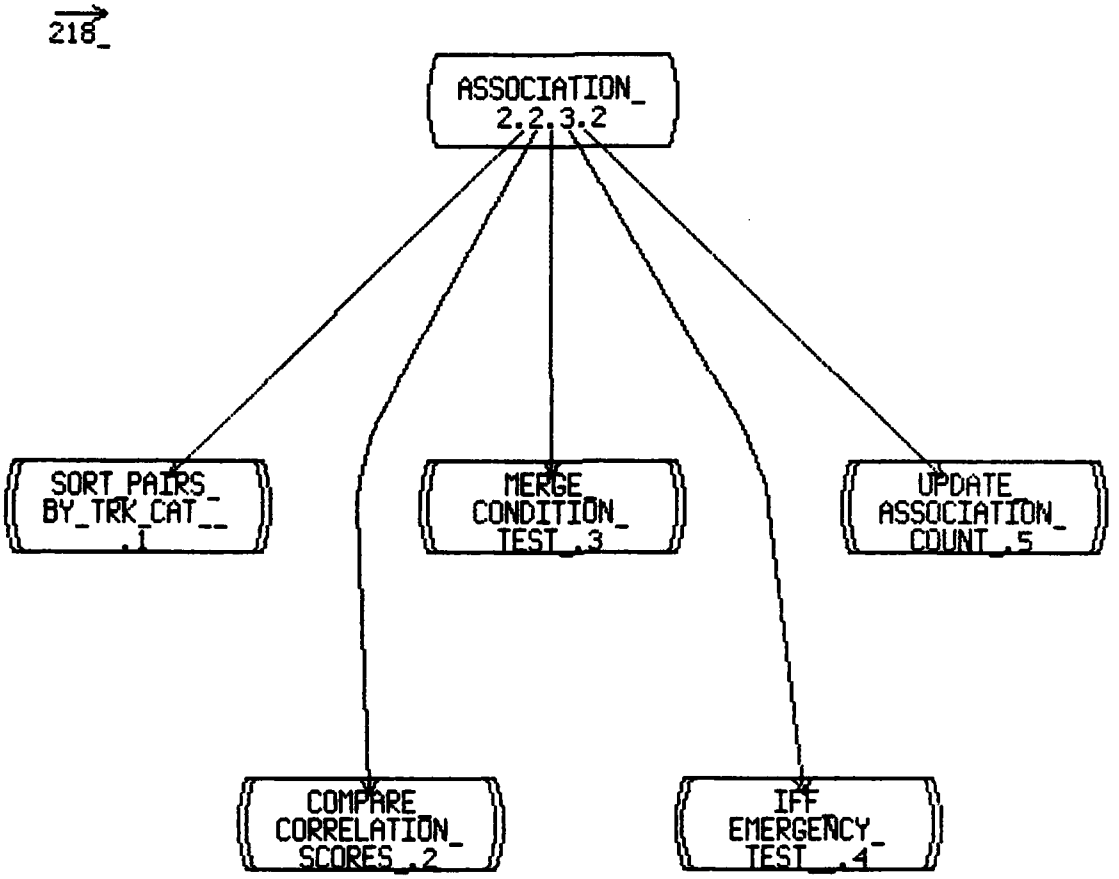
TITLE: 2.2.3.3 SMOOTHING SED		PAGE: 221
REVISION 1	AUTHOR: LWD	DATE: 5/14/82
NOTES:		



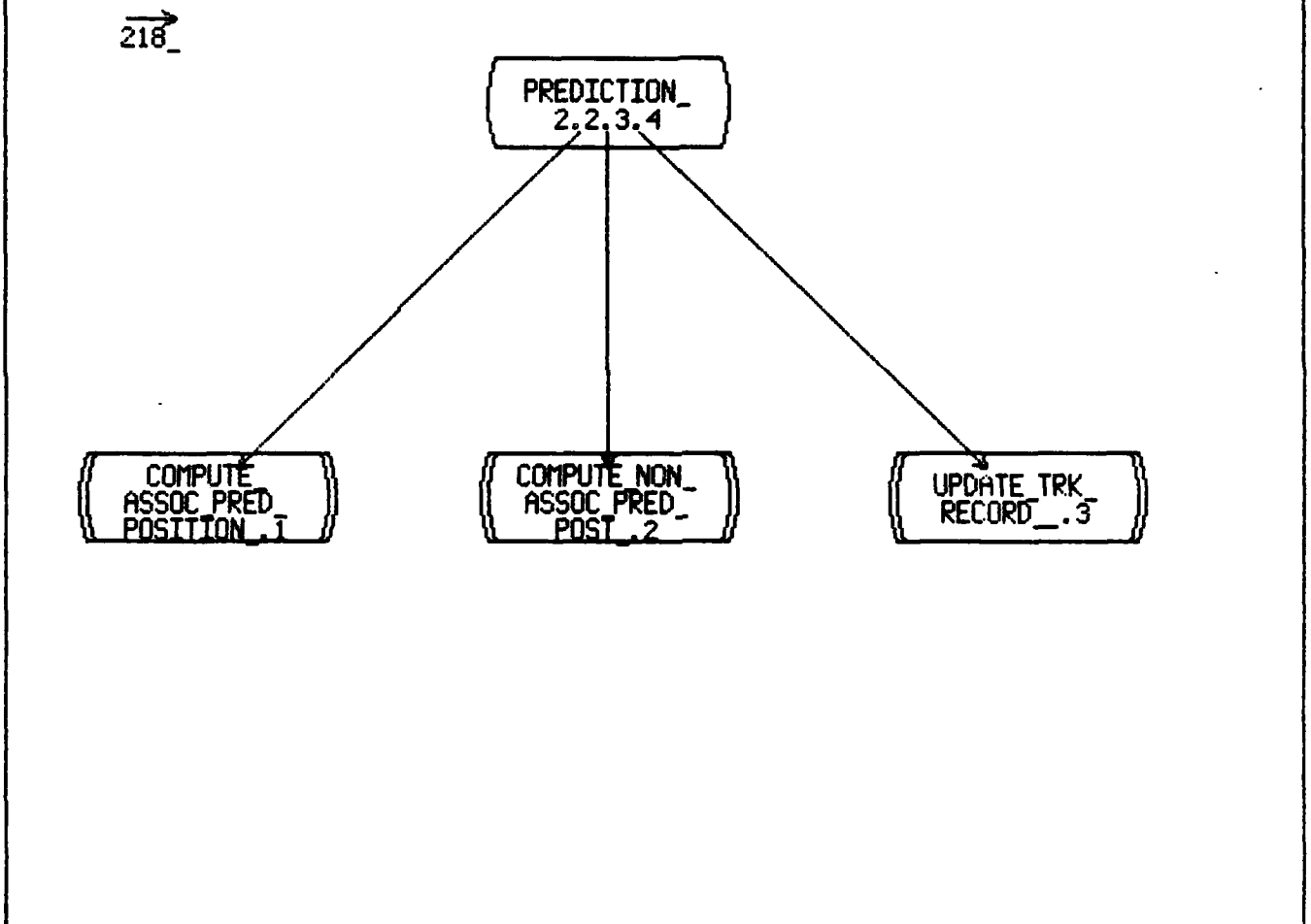
TITLE: 2.2.4 TRACK WHILE SCAN UTILITIES SED		PAGE: 222
REVISION 1	AUTHOR: LWED	DATE: 5/14/82
NOTES:		



TITLE: 2.2.3.2 ASSOCIATION SED		PAGE: 223
REVISION 1	AUTHOR: LWD	DATE: 5/14/82
NOTES:		

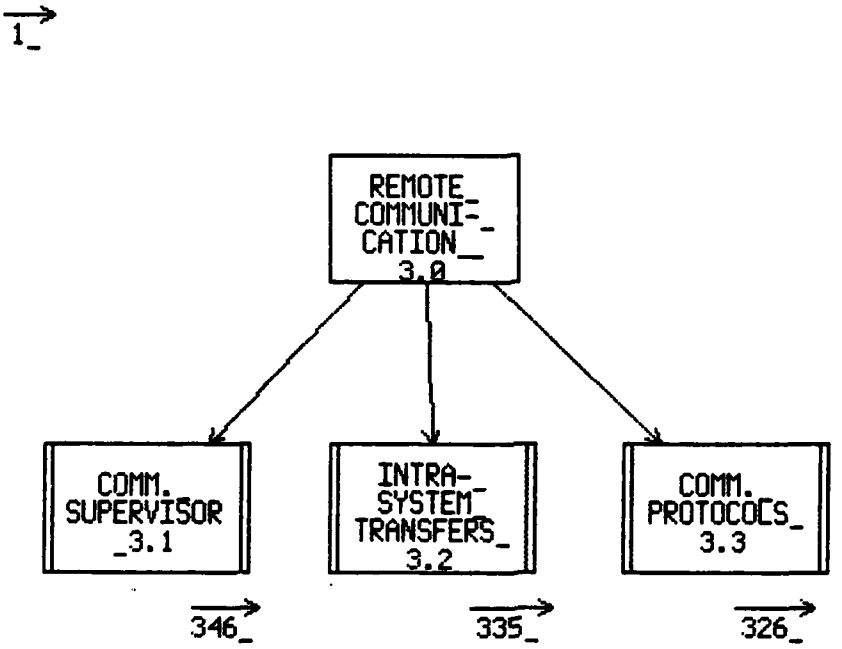


TITLE: 2.2.3.4 PREDICTION SED		PAGE: 224
REVISION 1	AUTHOR: LWD	DATE: 5/14/82
NOTES:		

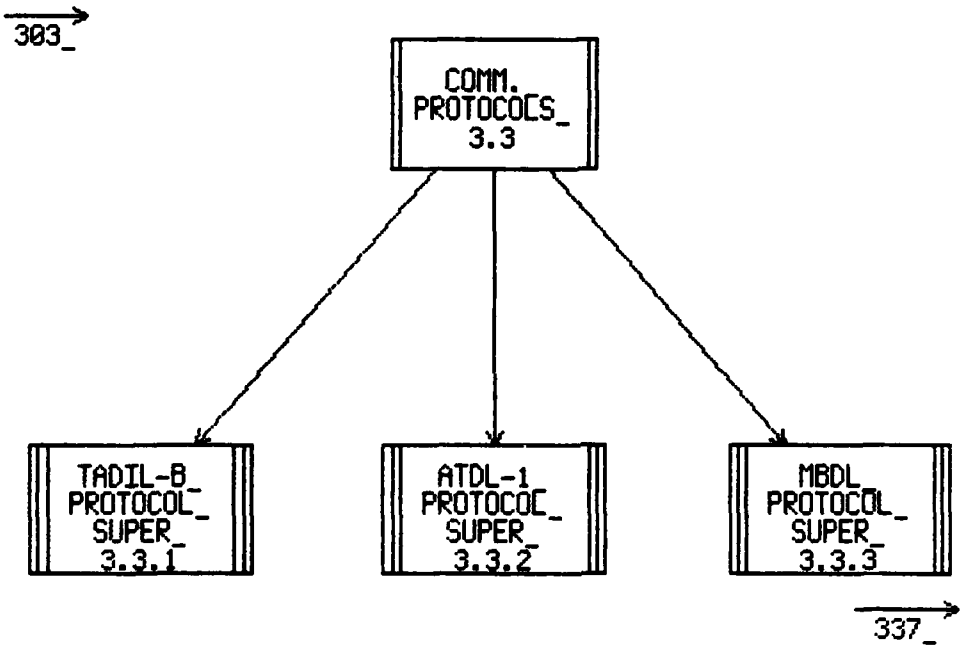


TITLE: 3.0 REMOTE COMMUNICATION SED		PAGE: 303
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 11/19/81

NOTES:
REVISION OF DIAGRAM 300

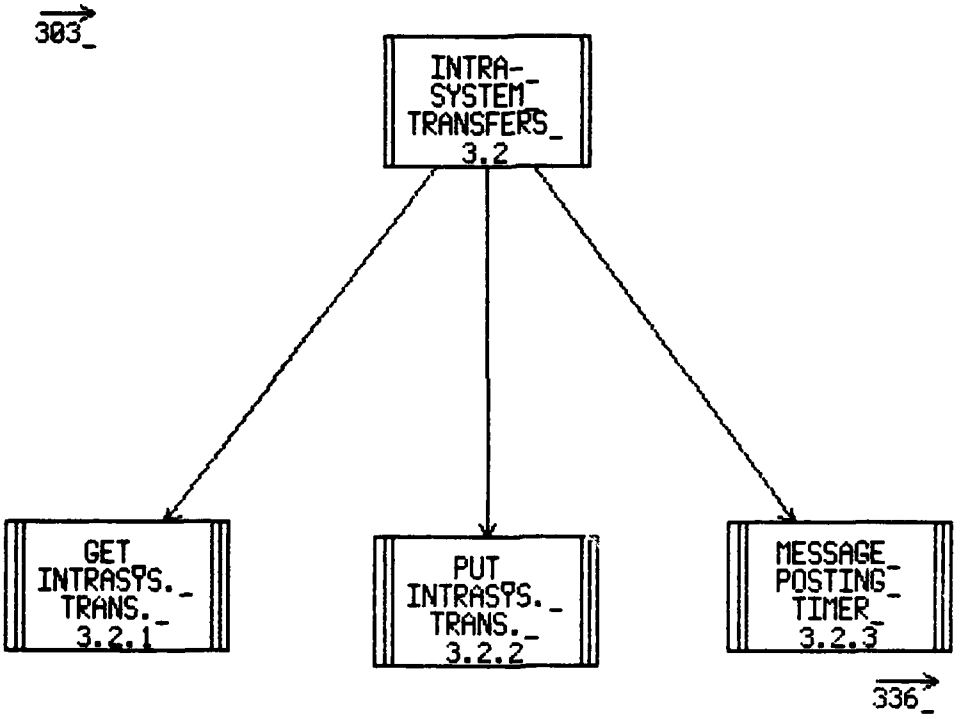


TITLE: 3.3 COMMUNICATION PROTOCOLS SED		PAGE: 326
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 12/4/81
NOTES:		



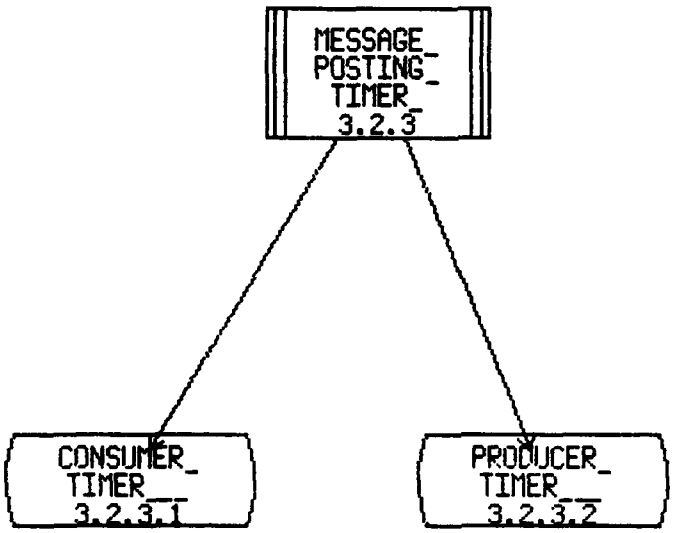
TITLE: 3.2 INTRASYSTEM TRANSFERS SED		PAGE: 335
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/14/82

NOTES:
REVISION OF DIAGRAM 323



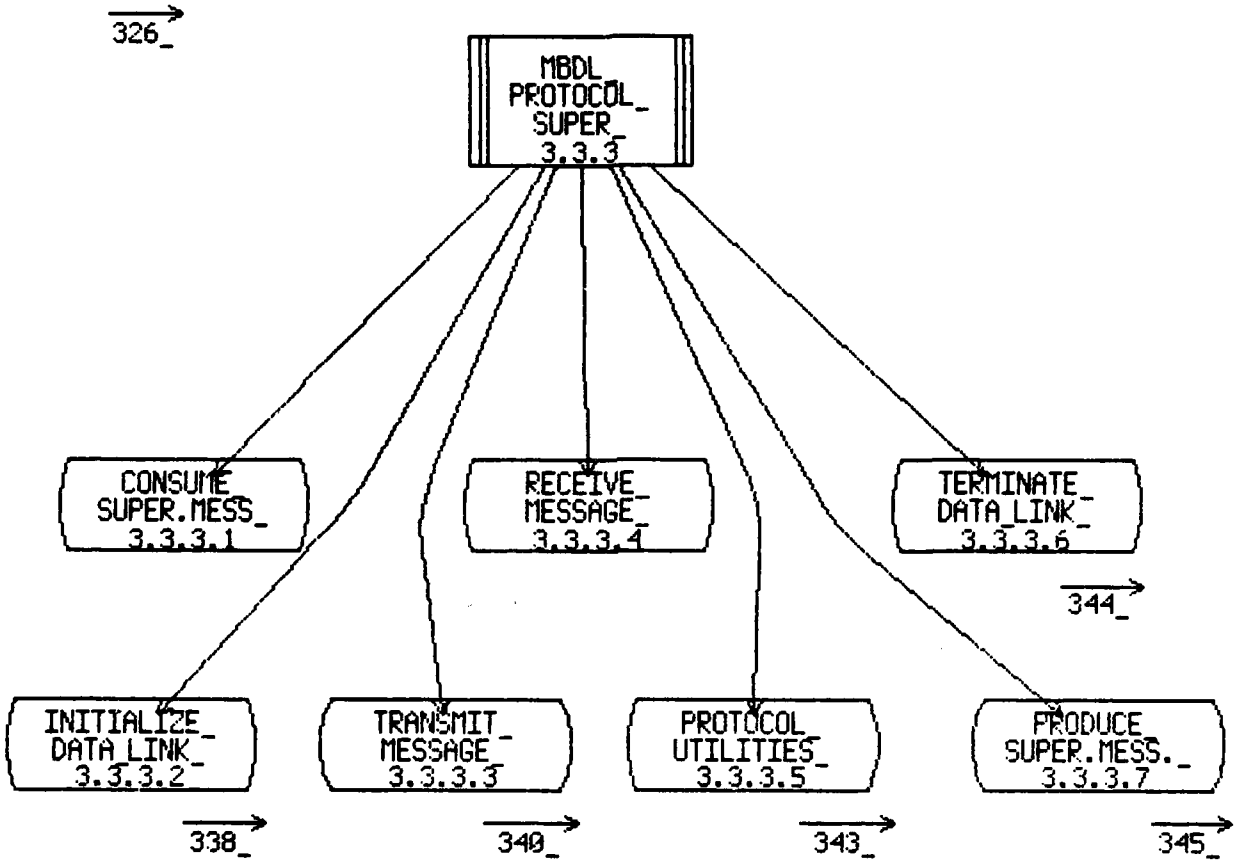
TITLE: 3.2.3 MESSAGE POSTING TIMER SED		PAGE: 336
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/14/82
NOTES:		

335

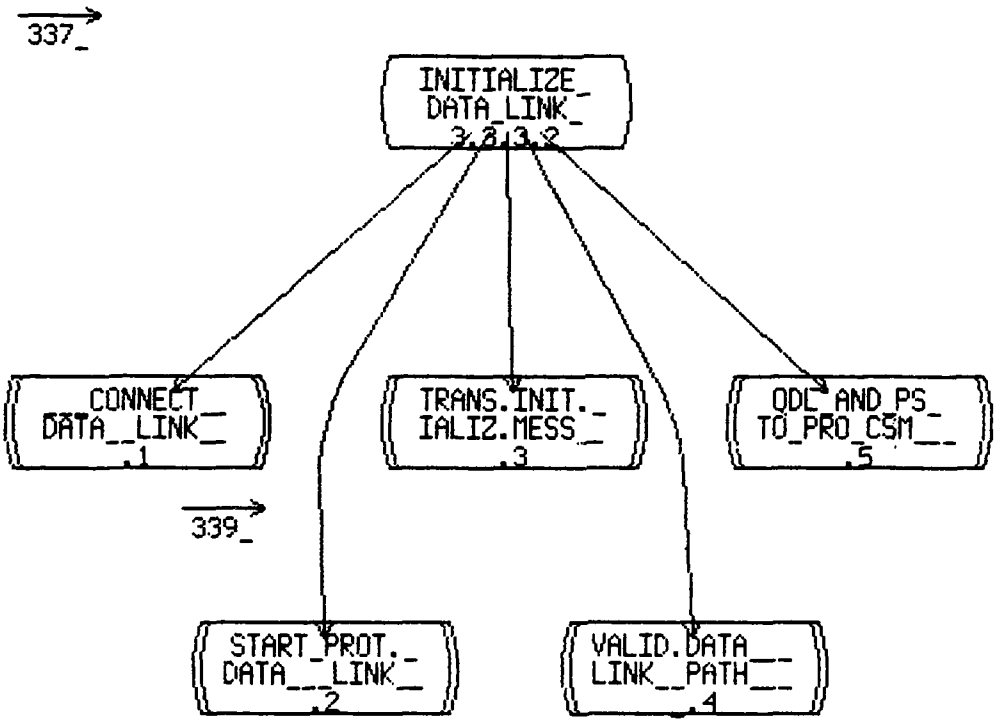


TITLE: 3.3.3 MBDL PROTOCOL SUPERVISOR SED		PAGE: 337
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82

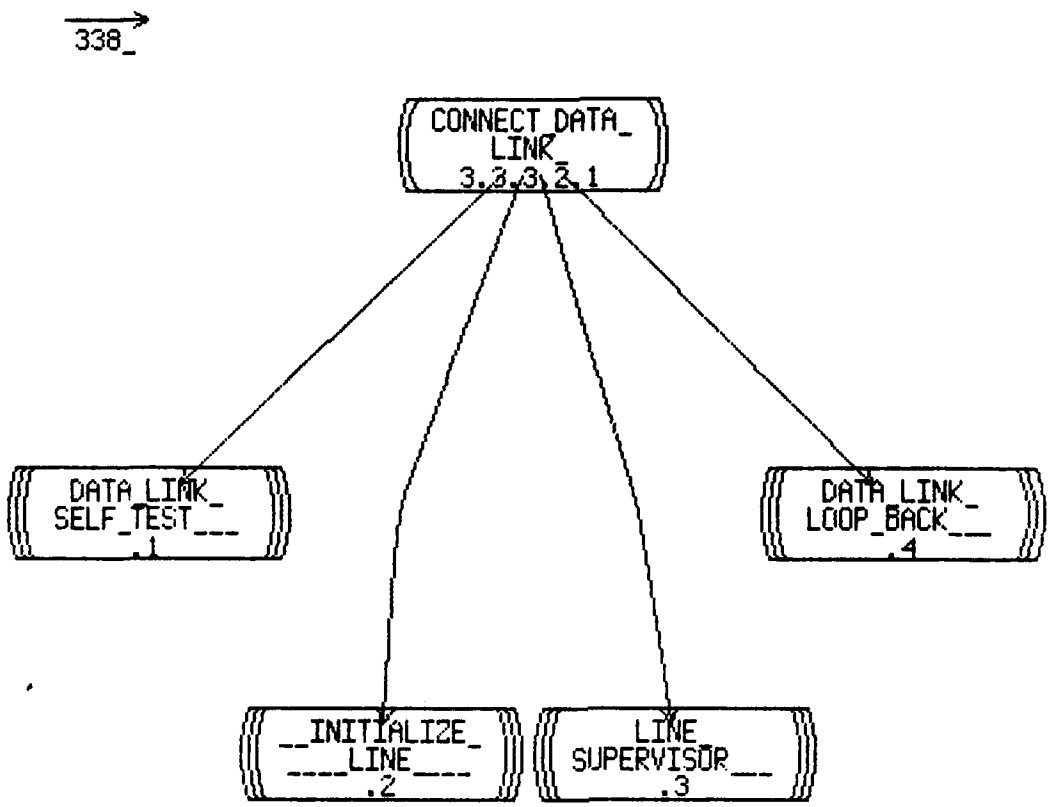
NOTES:
REVISION OF DIAGRAM 327



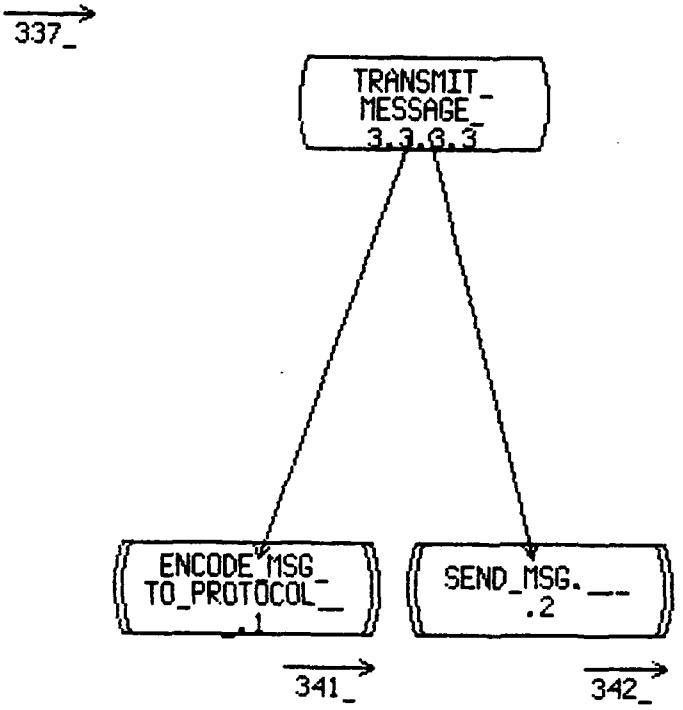
TITLE: 3.3.3.2 INITIALIZE DATA LINK SED		PAGE: 338
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82
NOTES:		



TITLE: 3.3.3.2.1 CONNECT DATA LINK SED		PAGE: 339
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82
NOTES:		

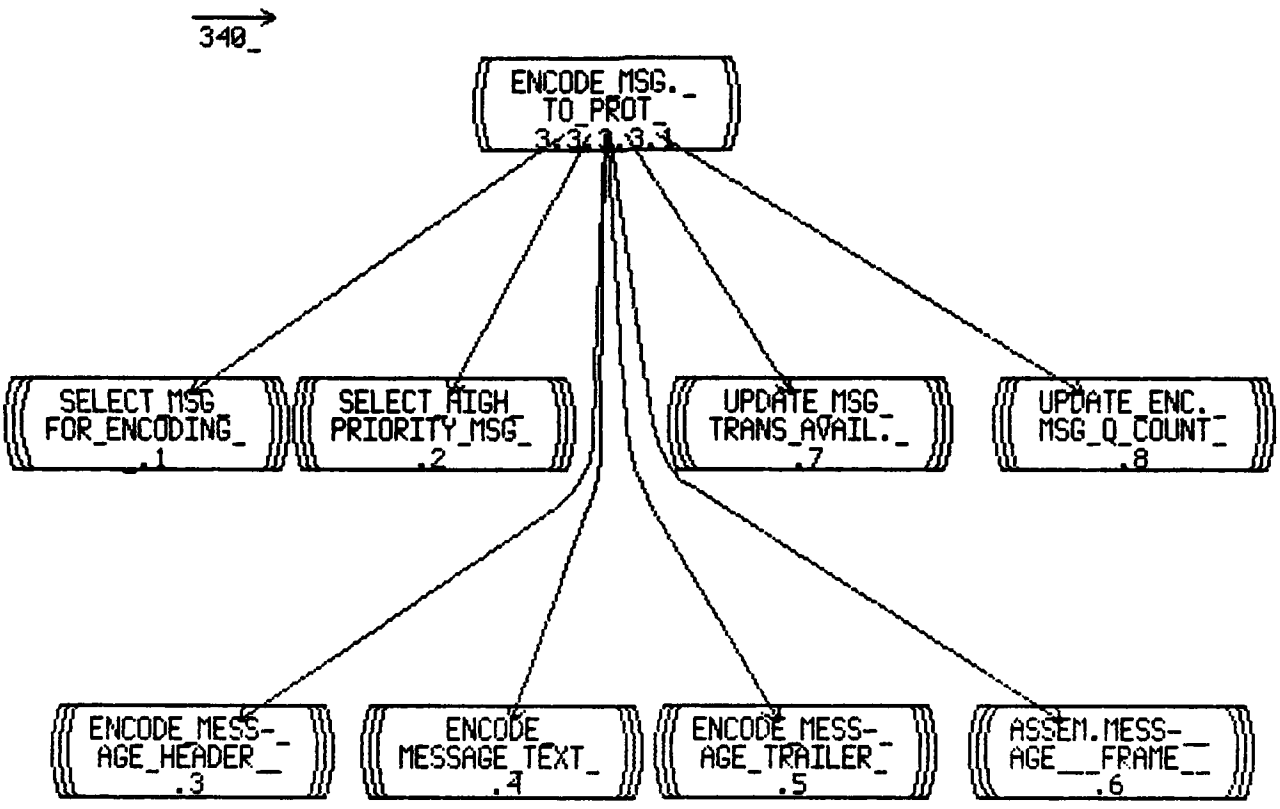


TITLE: 3.3.3.3 TRANSMIT MESSAGE SED		PAGE: 340
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82
NOTES:		



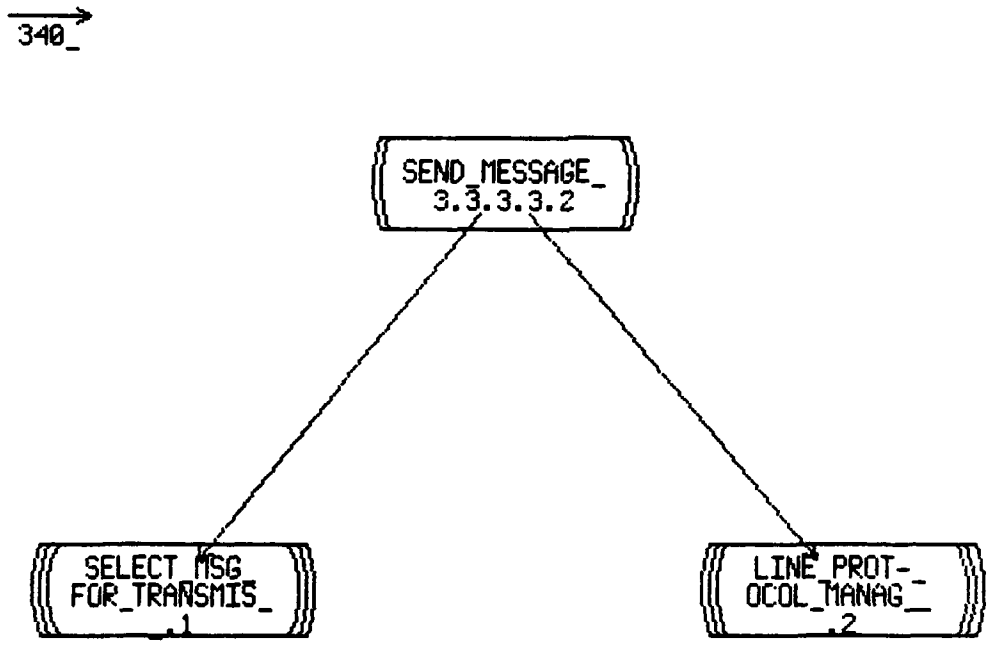
TITLE: 3.3.3.3.1 ENCODE MESSAGE TO PROTOCOL SED		PAGE: 341
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82

NOTES:

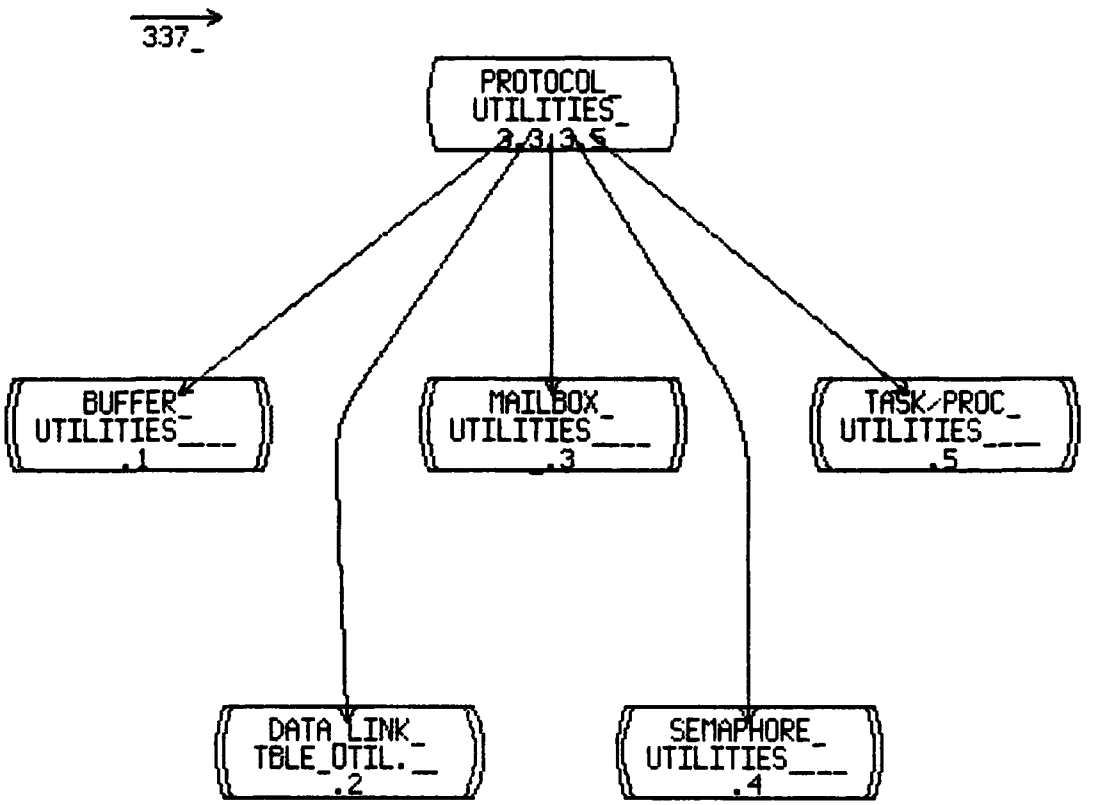


TITLE: 3.3.3.3.2 SEND MESSAGE SED		PAGE: 342
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82

NOTES:

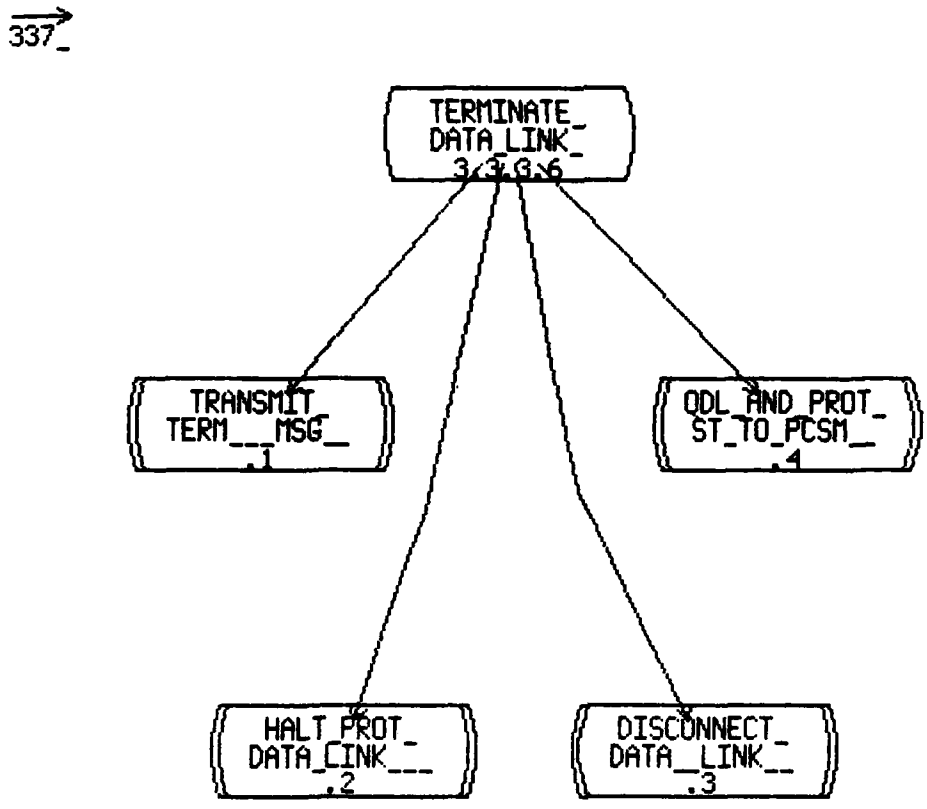


TITLE: 3.3.3.5 PROTOCOL UTILITIES SED		PAGE: 343
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82
NOTES:		



TITLE: 3.3.3.6 TERMINATE DATA LINK SED		PAGE: 344
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82

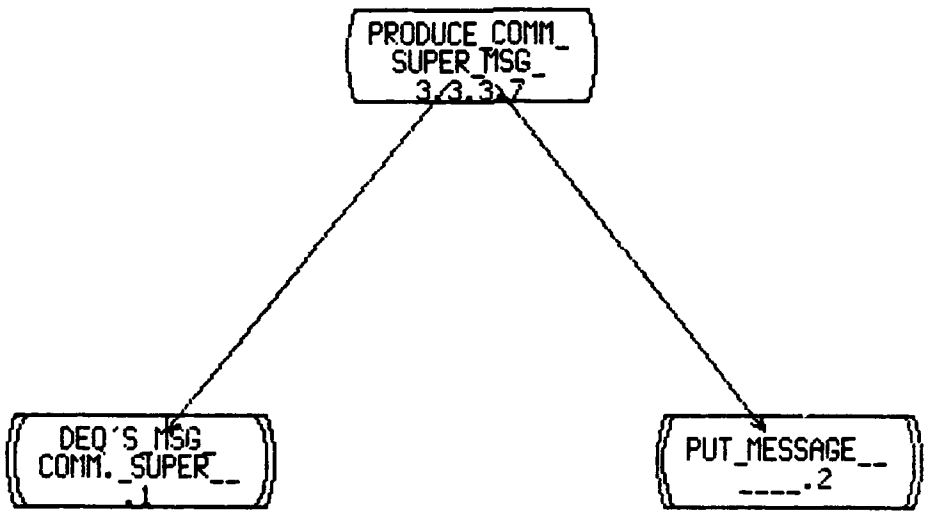
NOTES:



TITLE: 3.3.3.7 PRODUCE COMM. SUPERVISOR MESSAGE SED		PAGE: 345
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/15/82

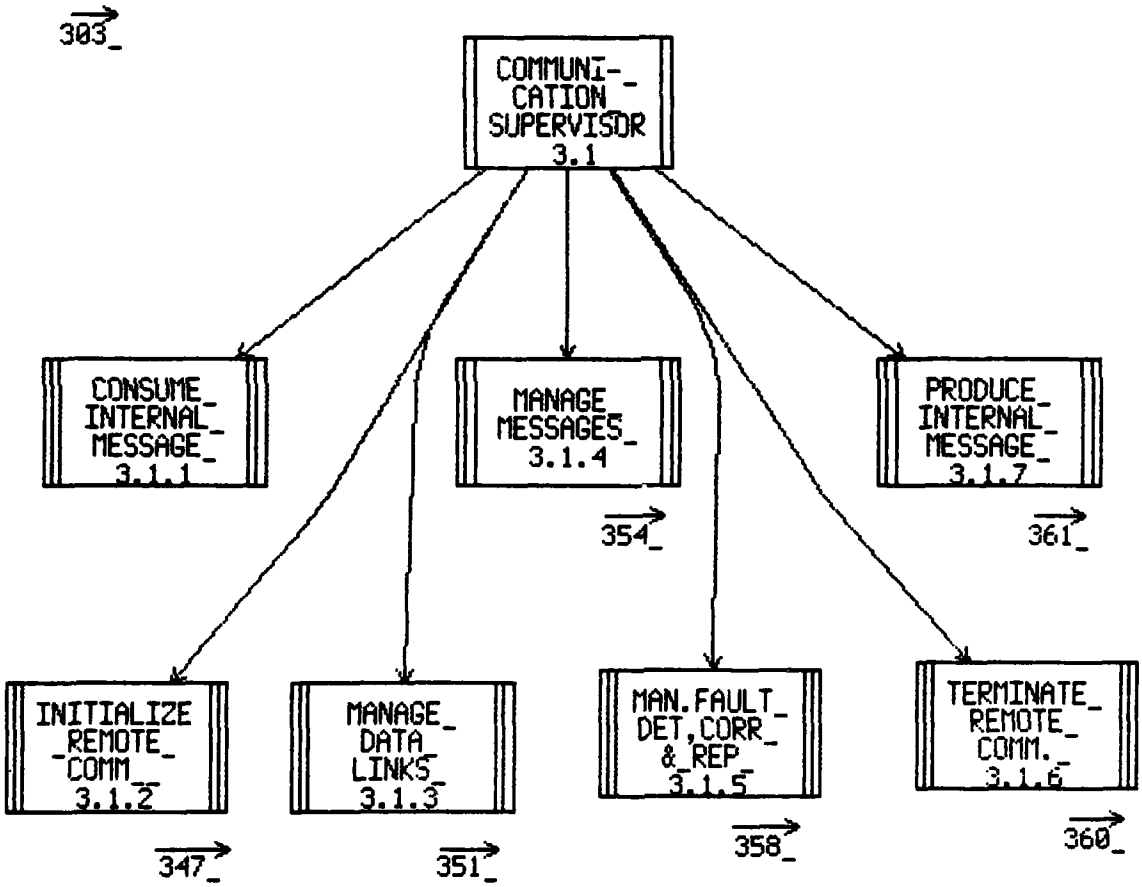
NOTES:

337

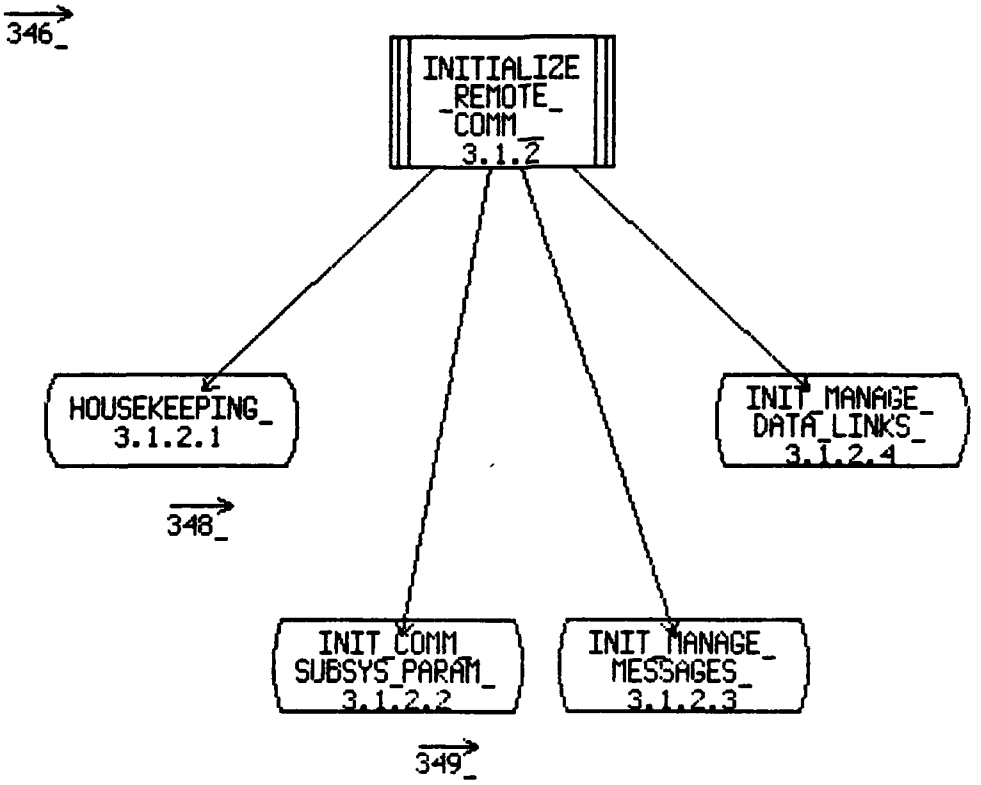


TITLE: 3.1 COMMUNICATION SUPERVISOR SED		PAGE: 346
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/18/82

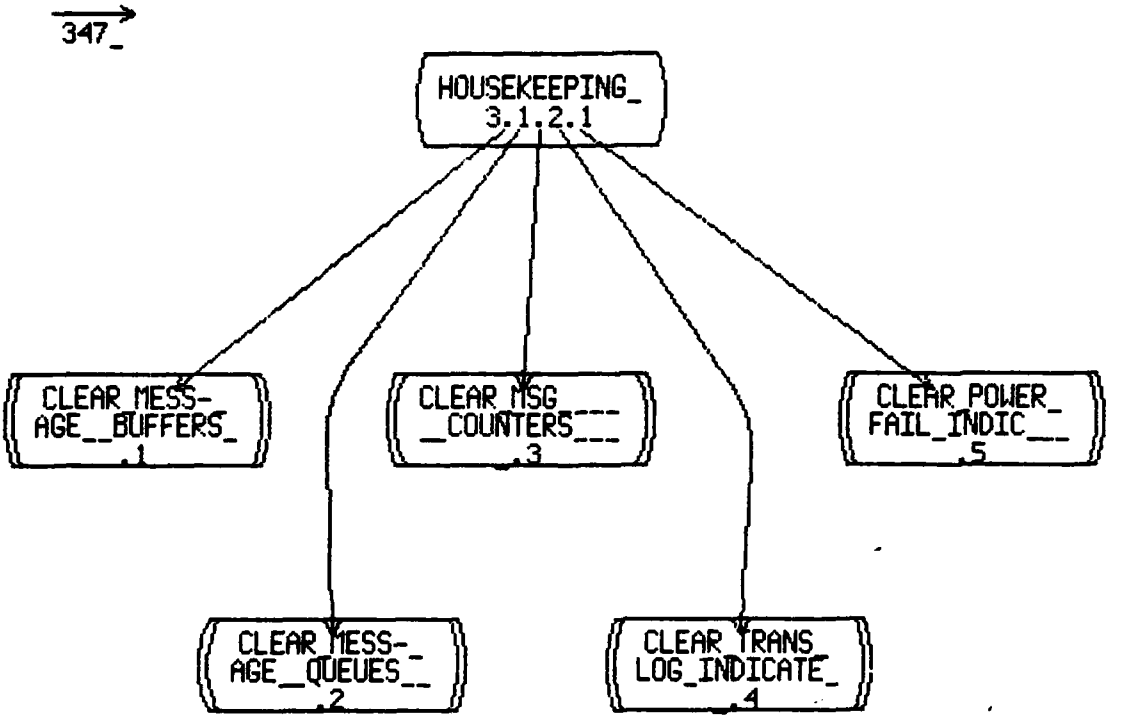
NOTES:
REVISION OF DIAGRAM 304



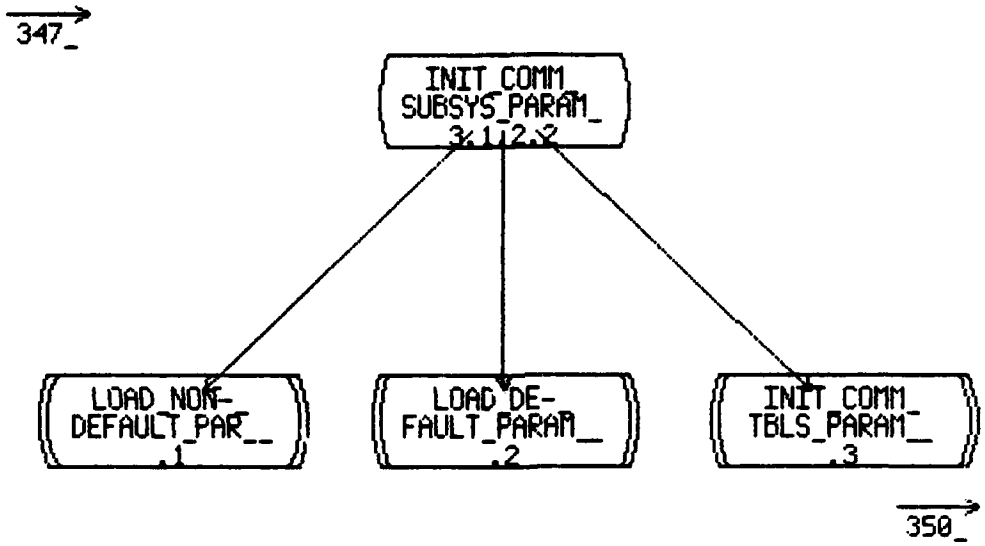
TITLE: 3.1.2 INITIALIZE REMOTE COMMUNICATION SED		PAGE: 347
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		



TITLE: 3.1.2.1 HOUSEKEEPING SED		PAGE: 348
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		

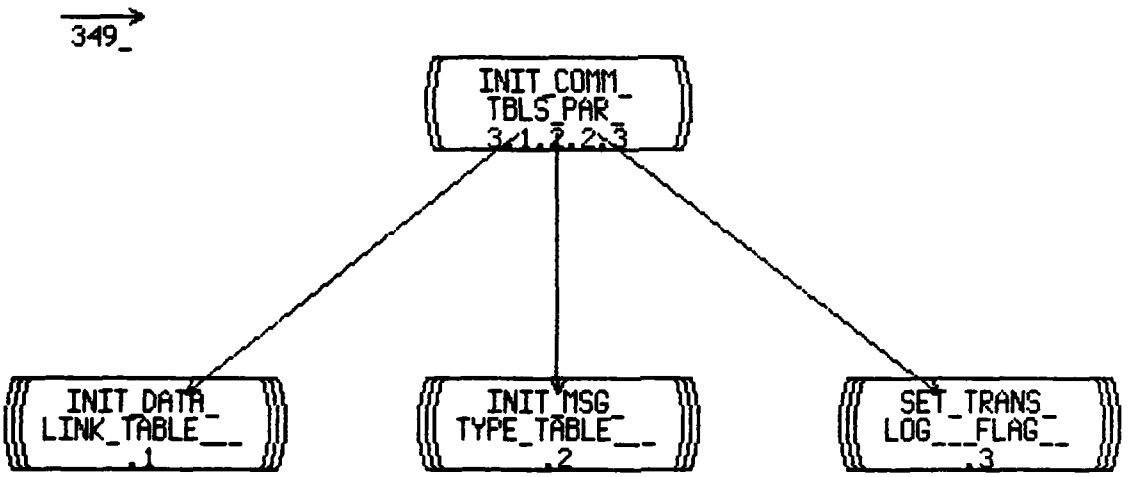


TITLE: 3.1.2.2 INITIALIZE COMM SUBSYSTEM PARAMETERS SED		PAGE: 349
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		



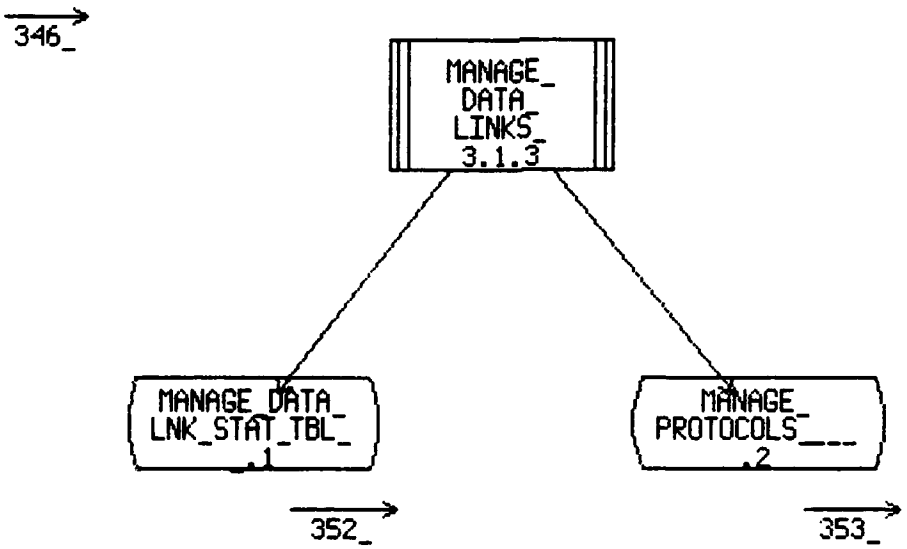
TITLE: 3.1.2.2.3 INITIALIZE COMM TABLES USING PARAMETERS SED		PAGE: 350
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82

NOTES:

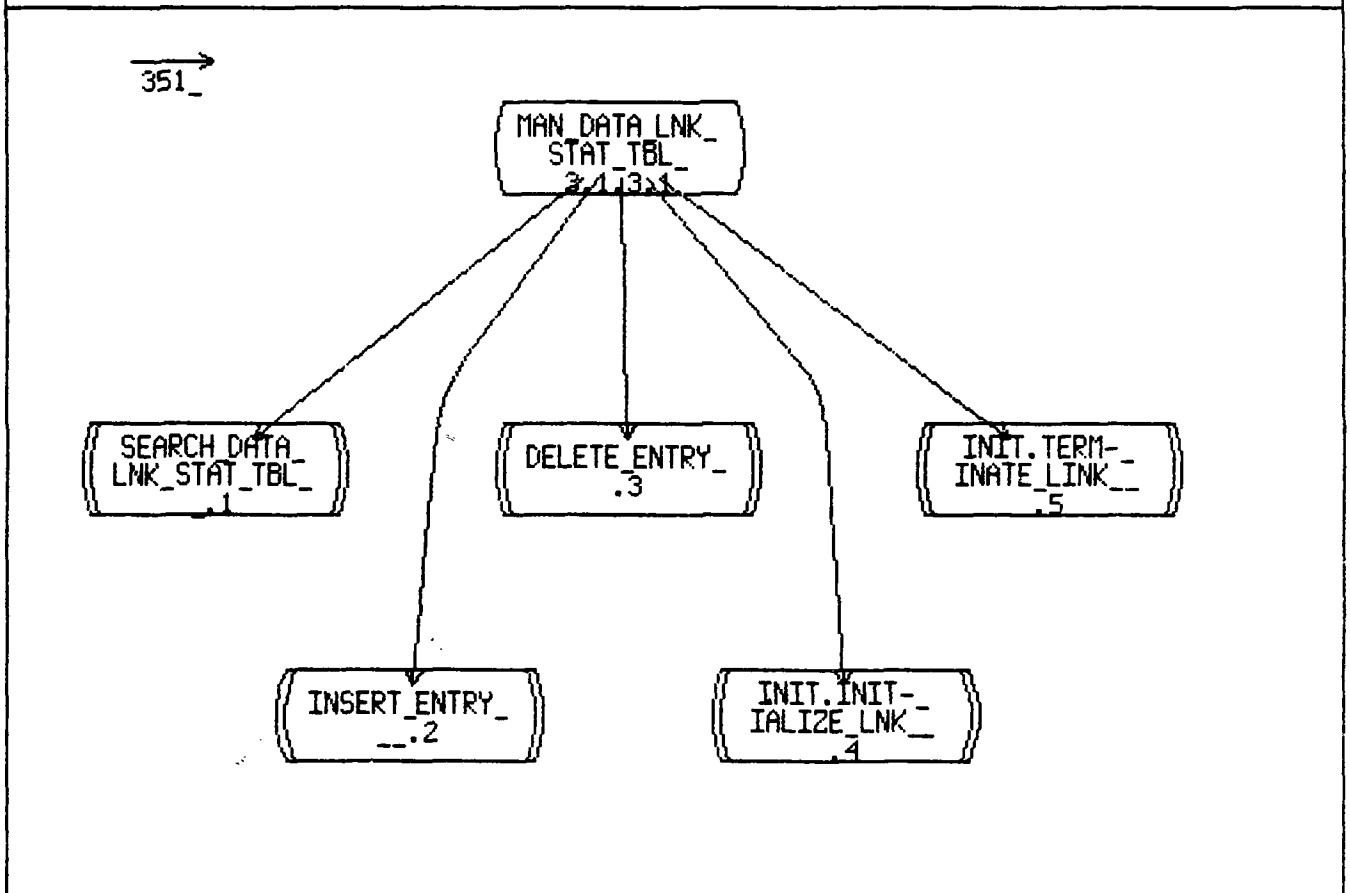


TITLE: 3.1.3 MANAGE DATA LINKS SED		PAGE: 351
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82

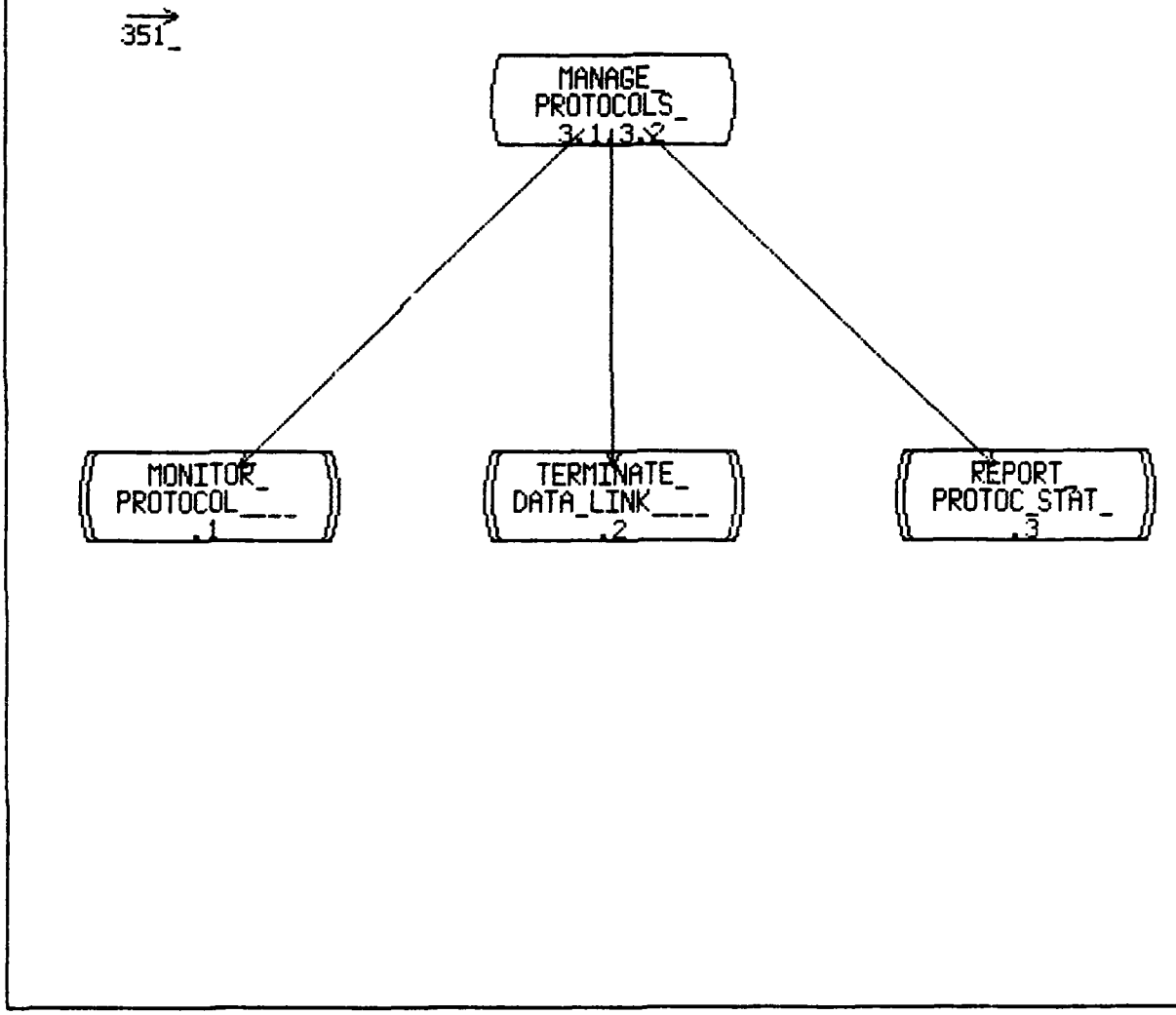
NOTES:



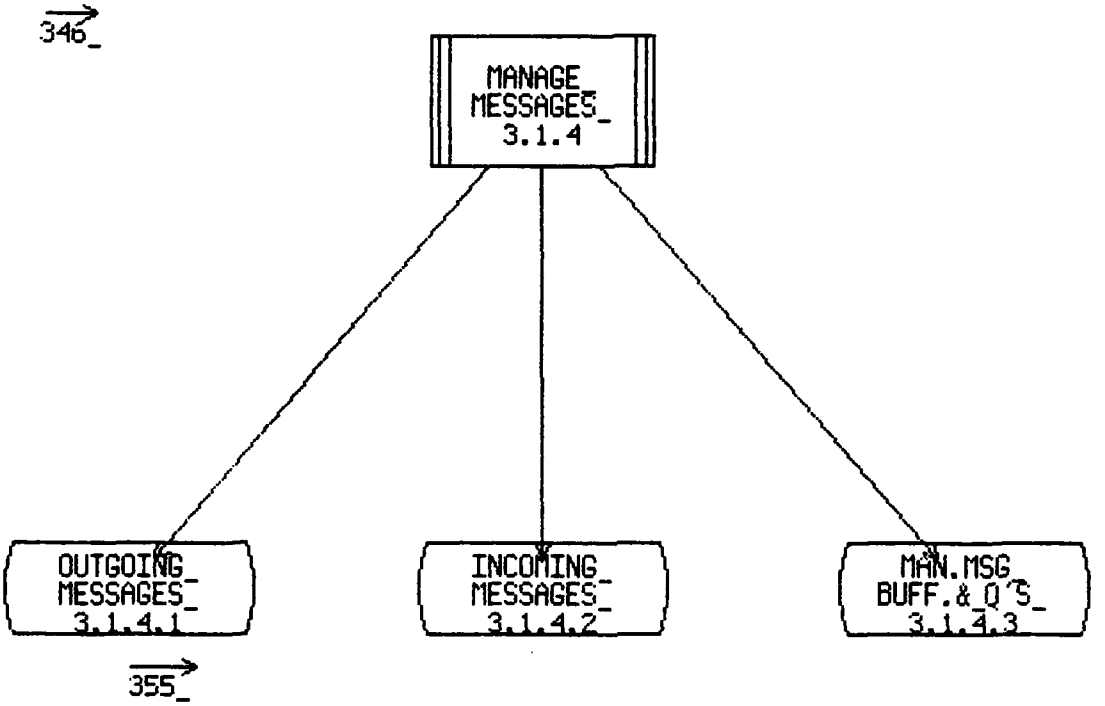
TITLE: 3.1.3.1 MANAGE DATA LINK STATUS TABLE SED		PAGE: 352
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		



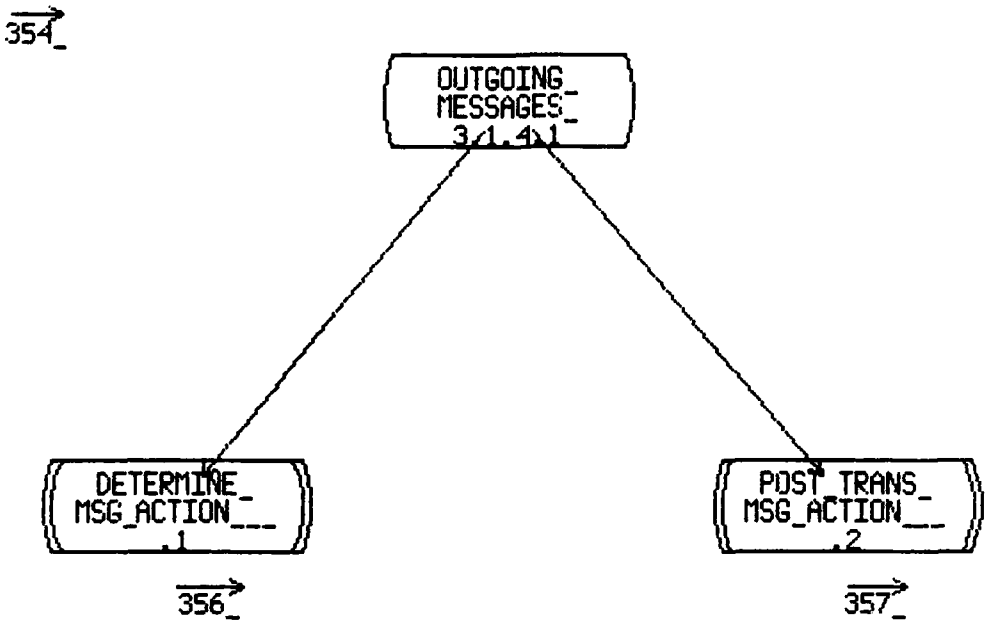
TITLE: 3.1.3.2 MANAGE PROTOCOLS SED		PAGE: 353
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		



TITLE: 3.1.4 MANAGE MESSAGES SED		PAGE: 354
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		

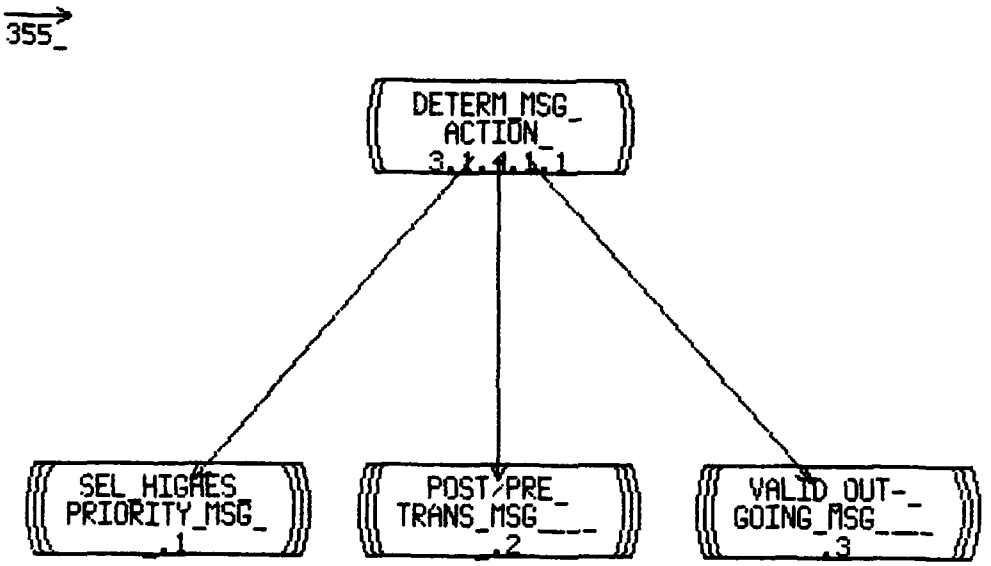


TITLE: 3.1.4.1 OUTGOING MESSAGES SED		PAGE: 355
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		

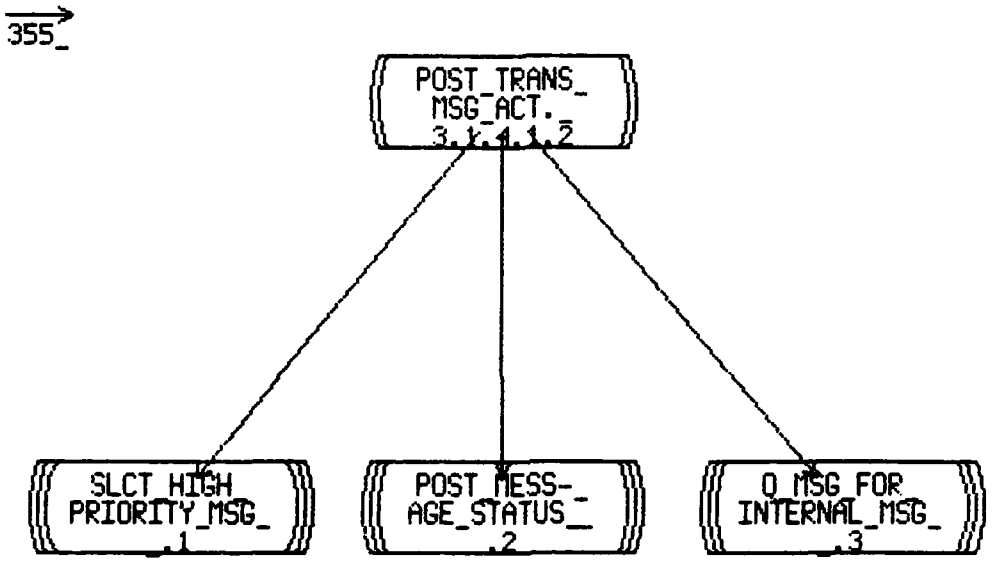


TITLE: 3.1.4.1.1 DETERMINE MESSAGE ACTION SED		PAGE: 356
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82

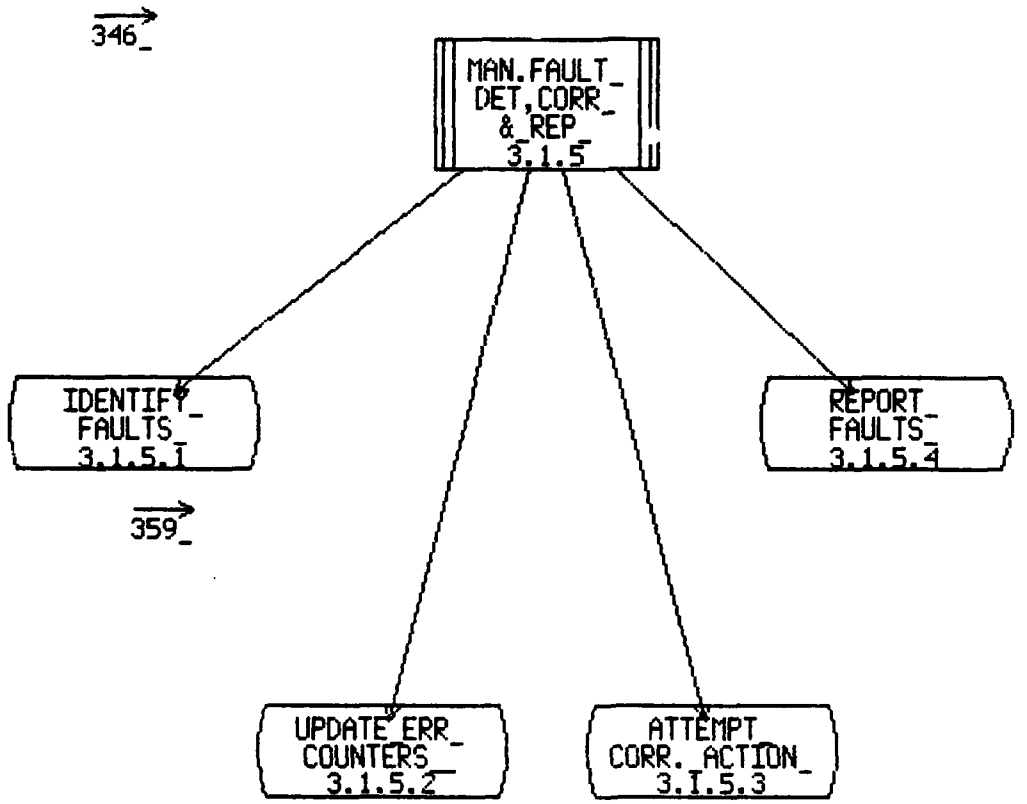
NOTES:



TITLE: 3.1.4.1.2 POST TRANSMISSION MESSAGE ACTION SED		PAGE: 357
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		

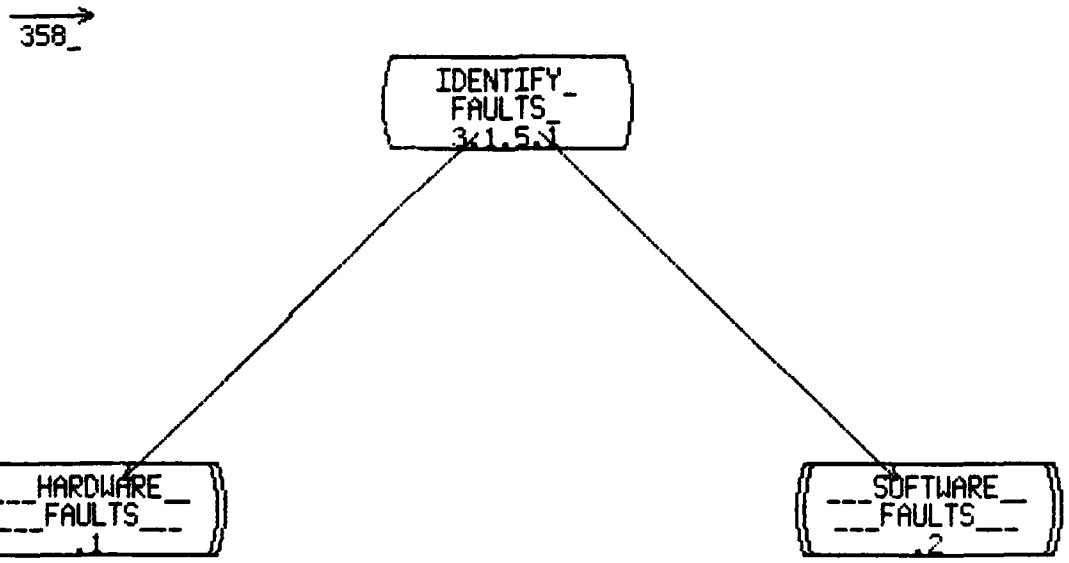


TITLE: 3.1.5. MANAGE FAULT DETECTION, CORRECTION AND REPORTING SED		PAGE: 358
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES: SAVE		



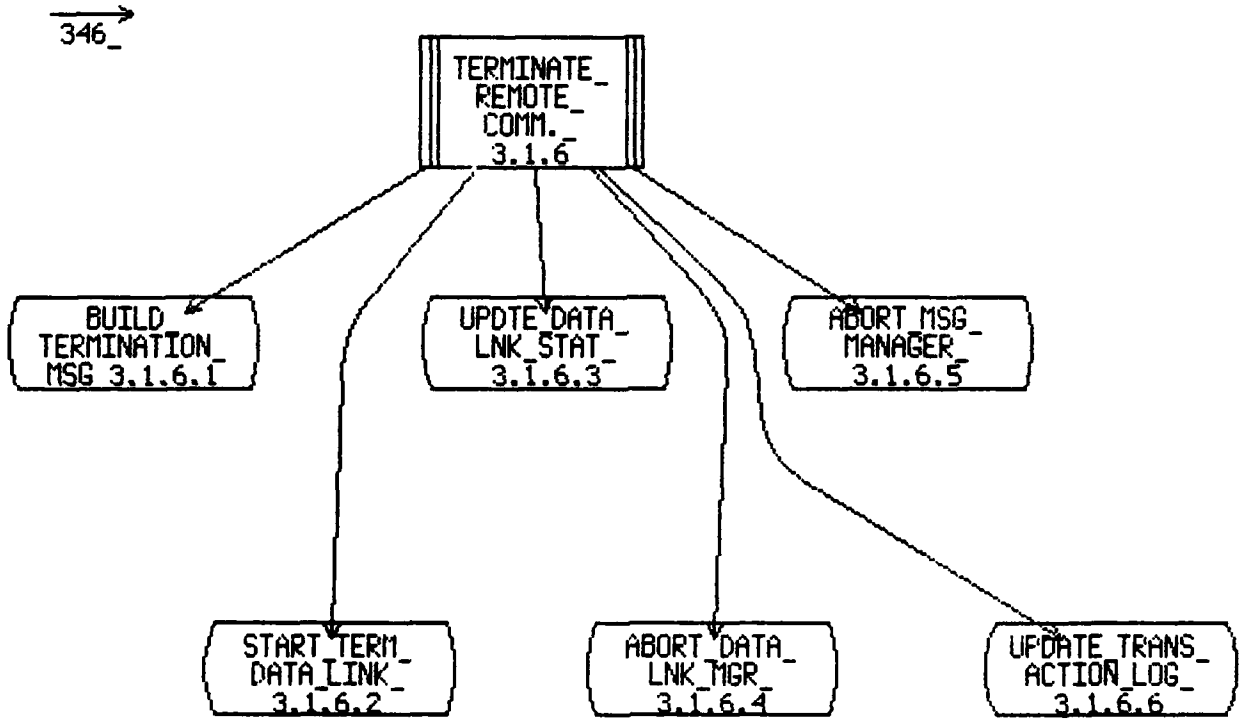
TITLE: 3.1.5.1 IDENTIFY FAULTS SED		PAGE: 359
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82

NOTES:

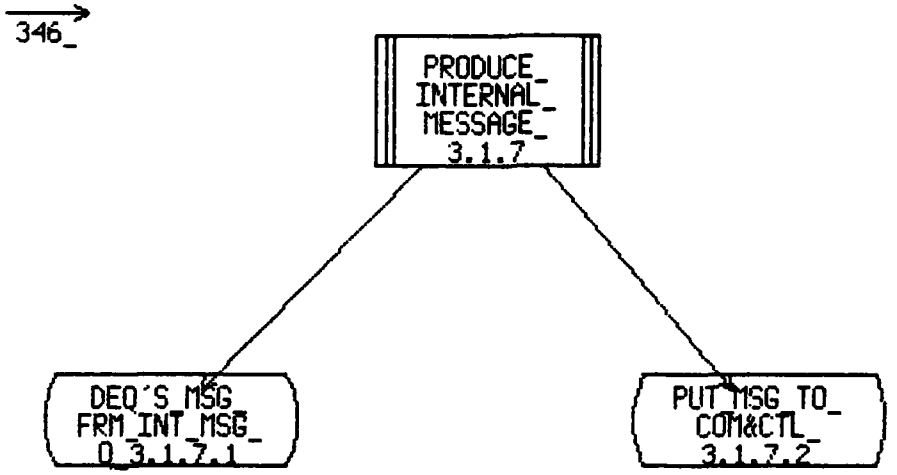


TITLE: 3.1.6 TERMINATE REMOTE COMMUNICATION SED		PAGE: 360
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82

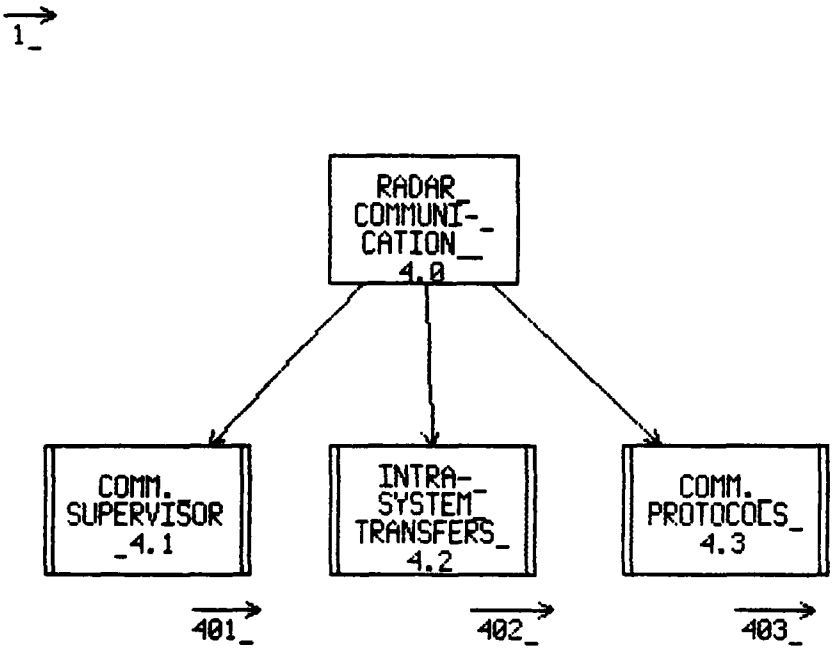
NOTES:



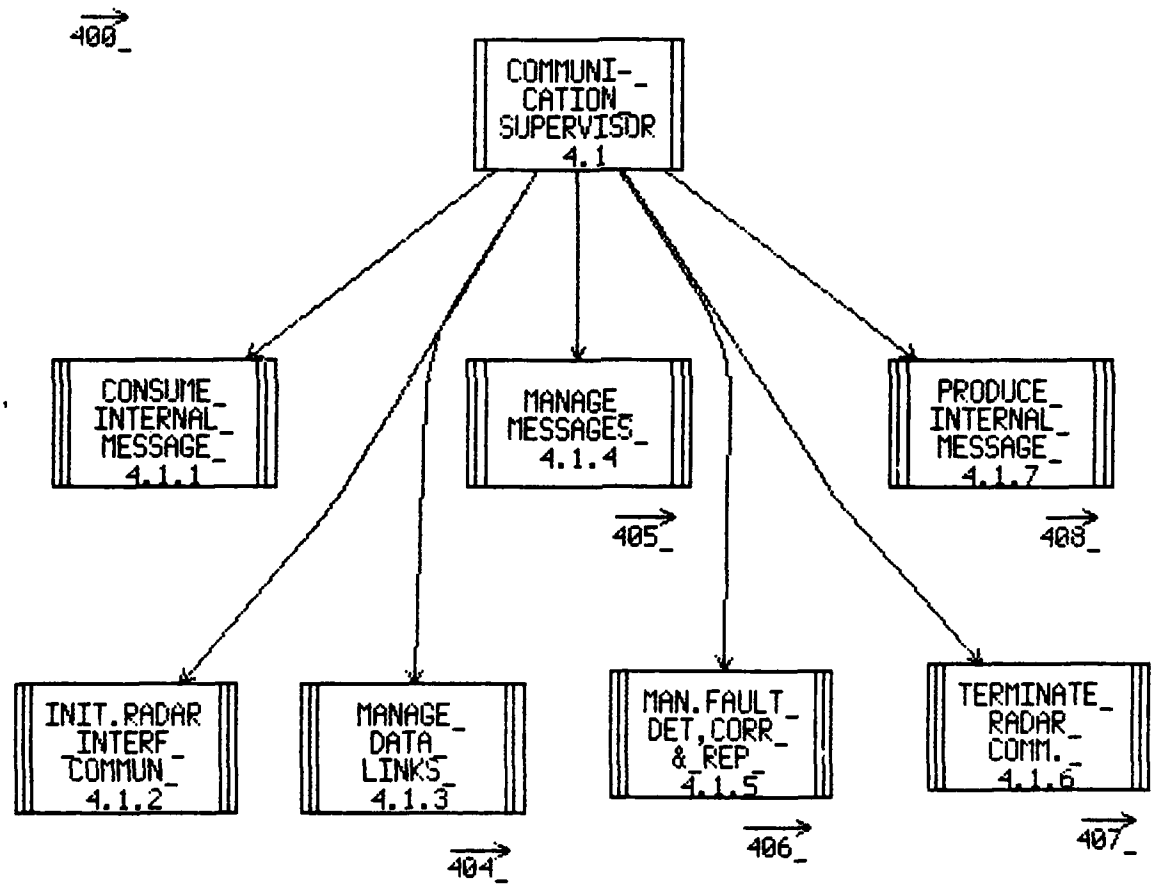
TITLE: 3.1.7 PRODUCE INTERNAL MESSAGE SED		PAGE: 361
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/19/82
NOTES:		



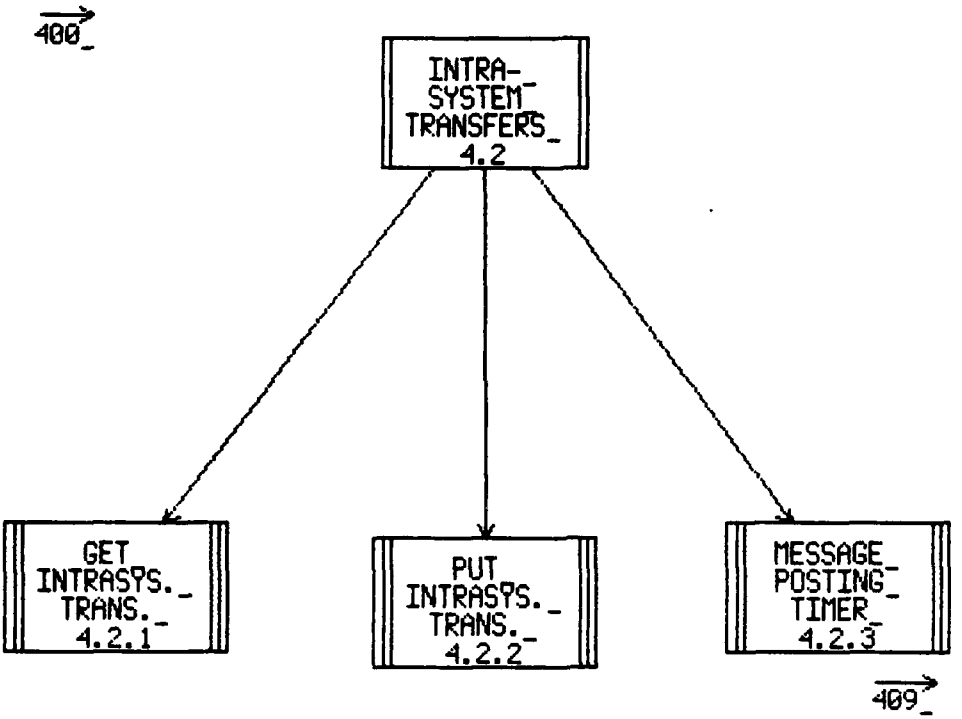
TITLE: 4.0 RADAR COMMUNICATION SED		PAGE: 400
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		



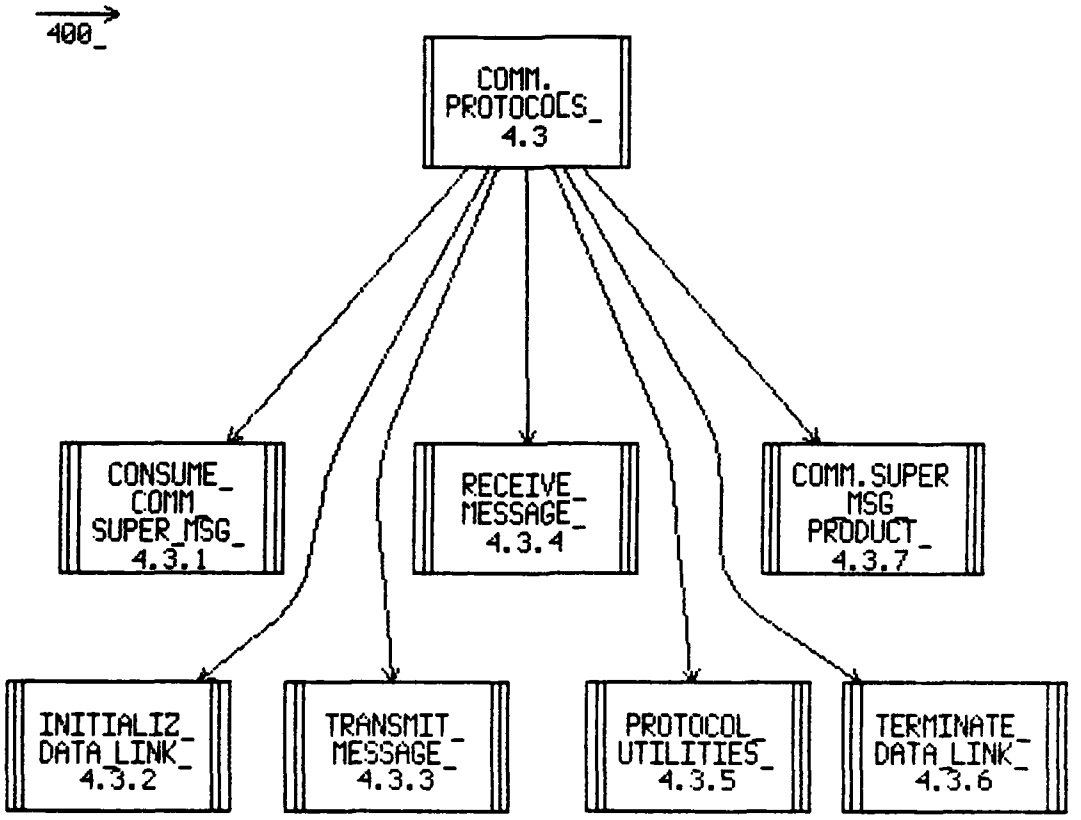
TITLE: 4.1 COMMUNICATION SUPERVISOR SED		PAGE: 401
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		



TITLE: 4.2 INTRASYSTEM TRANSFERS SED		PAGE: 402
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		

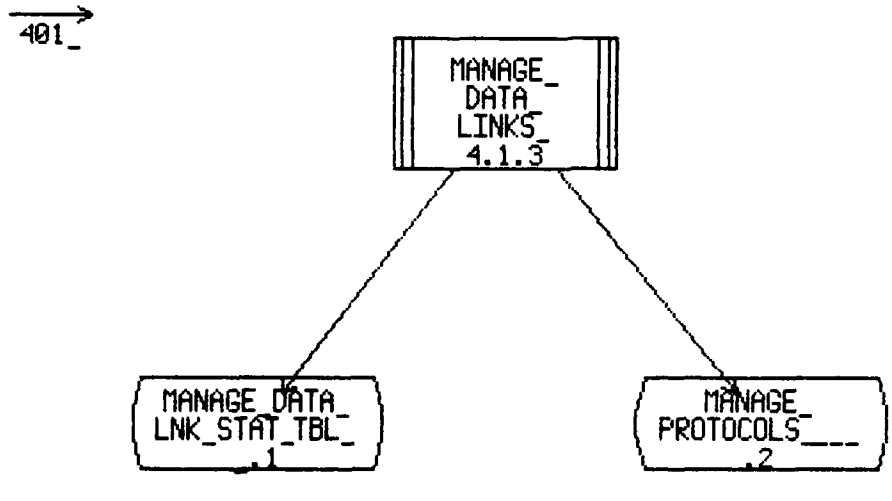


TITLE: 4.3 COMMUNICATION PROTOCOLS SED		PAGE: 403
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		



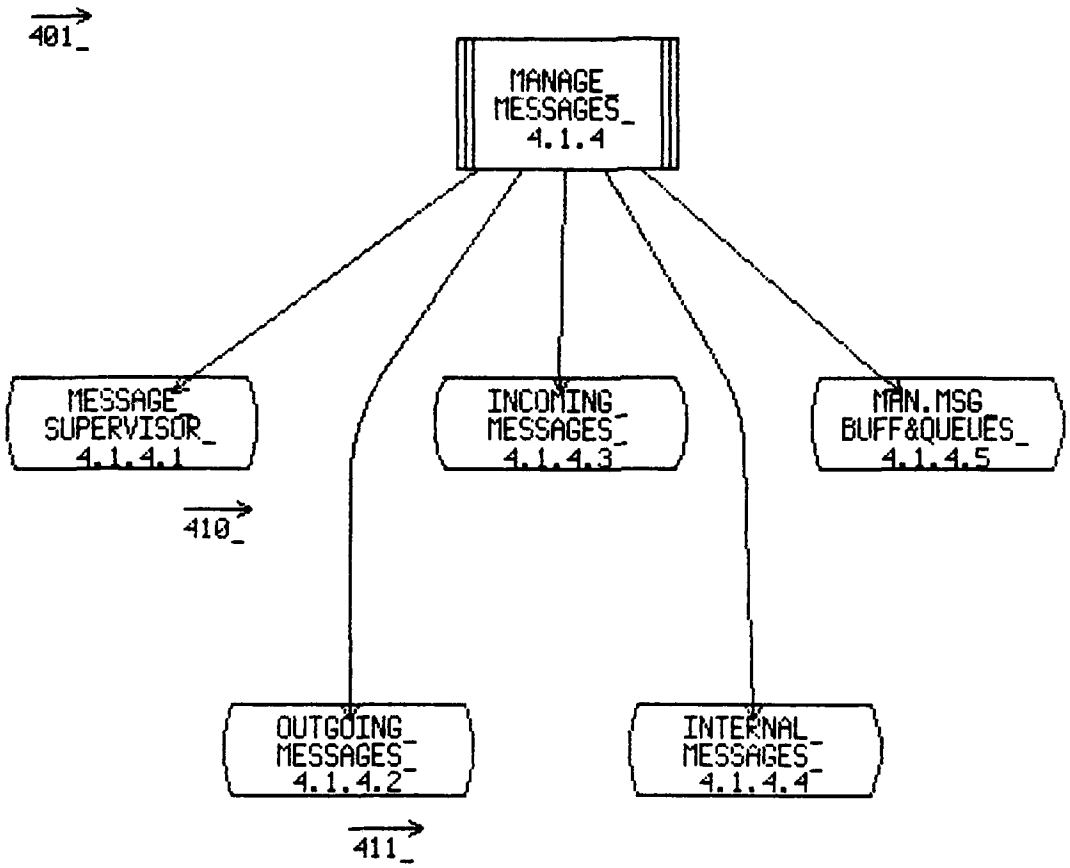
TITLE: 4.1.3 MANAGE DATA LINKS SED		PAGE: 404
REVISION 0	AUTHOR: RAW/GIC	DATE: 4/9/82

NOTES:

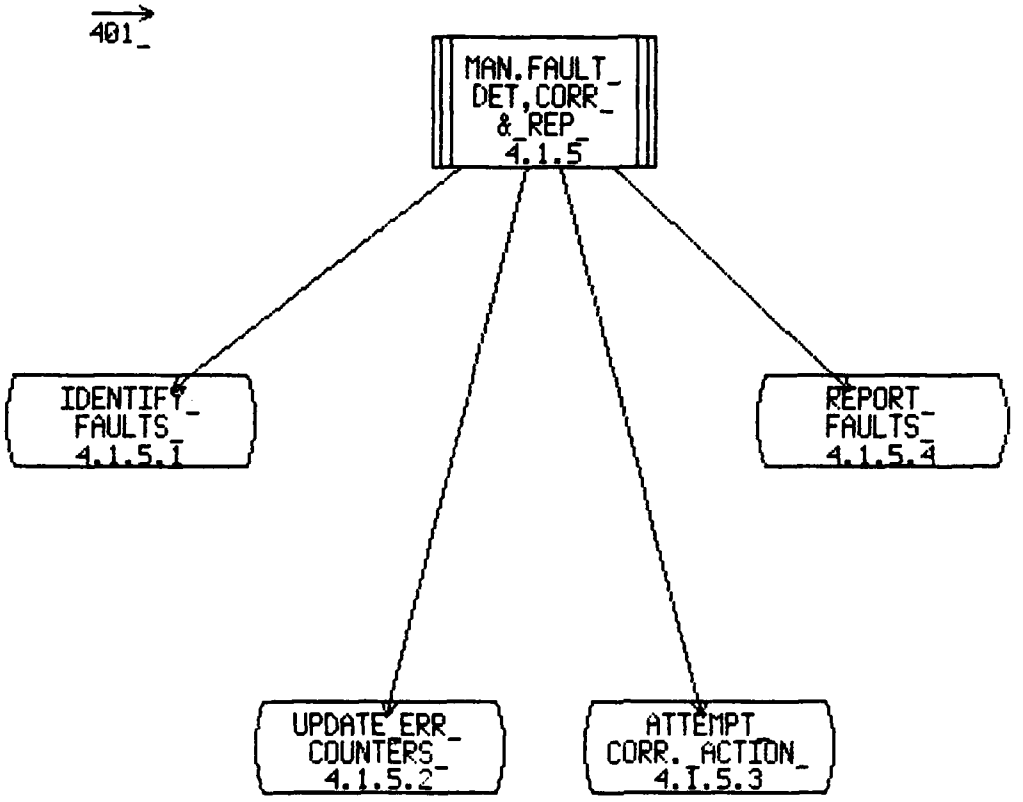


TITLE: 4.1.4 MANAGE MESSAGES SED		PAGE: 405
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82

NOTES:

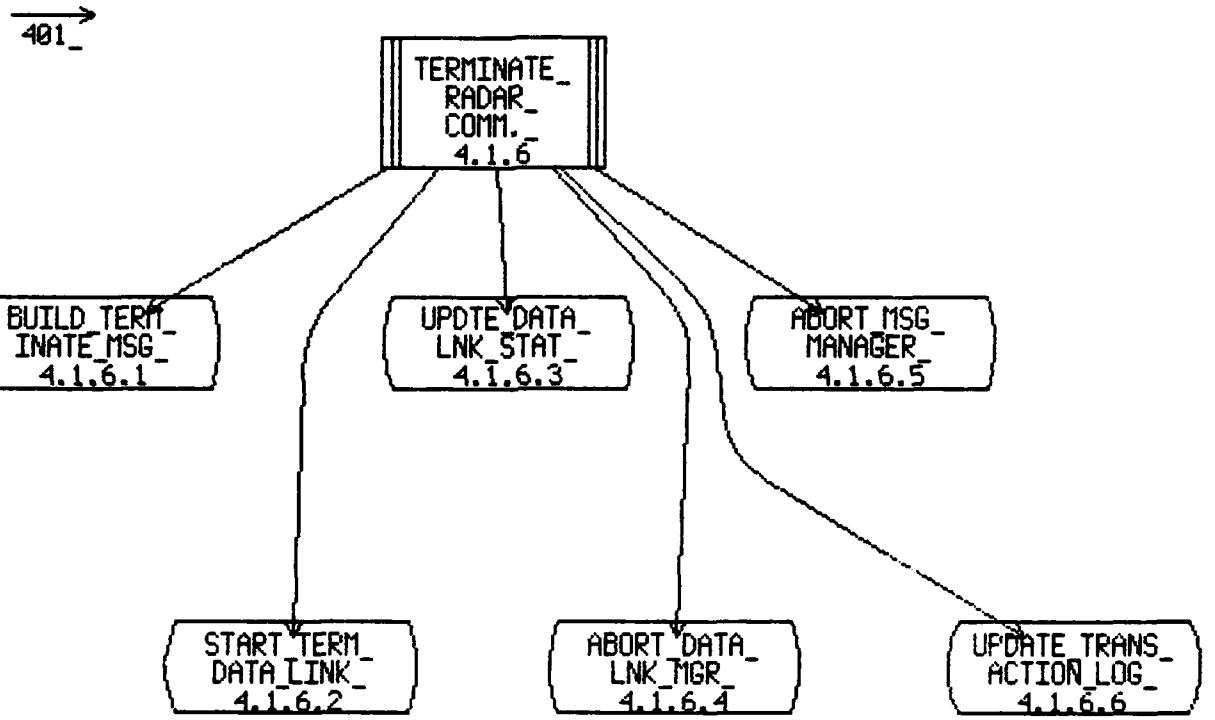


TITLE: 4.1.5 MANAGE FAULT DETECTION, CORRECTION AND REPORTING SED		PAGE: 406
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		

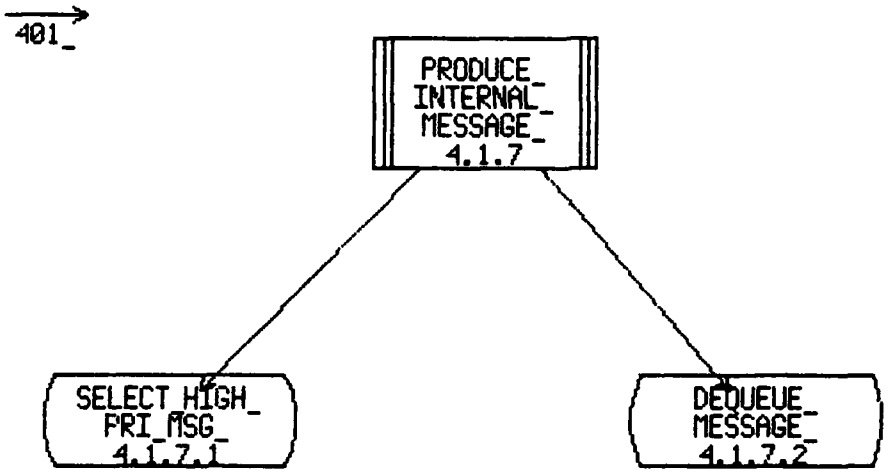


TITLE: 4.1.6 TERMINATE RADAR COMMUNICATION SED		PAGE: 407
REVISION 0	AUTHOR: RAW/GIC	DATE: 4/9/82

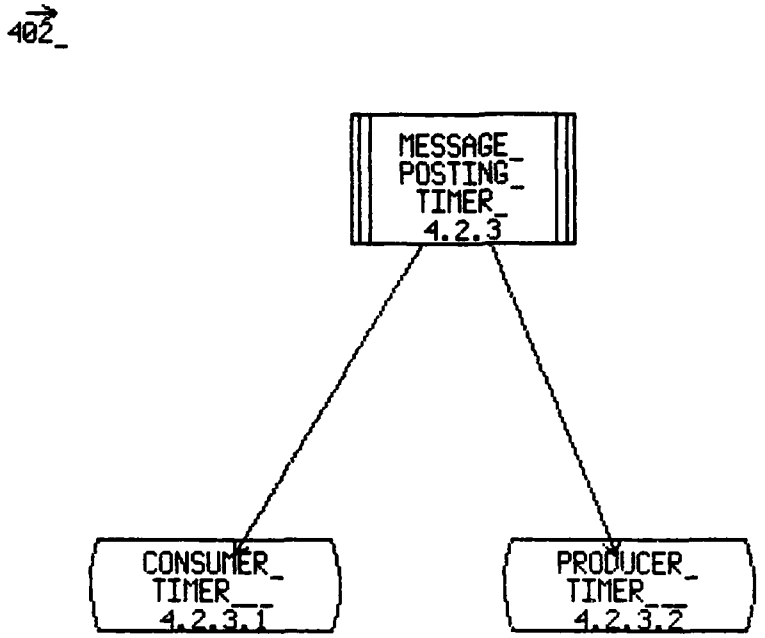
NOTES:



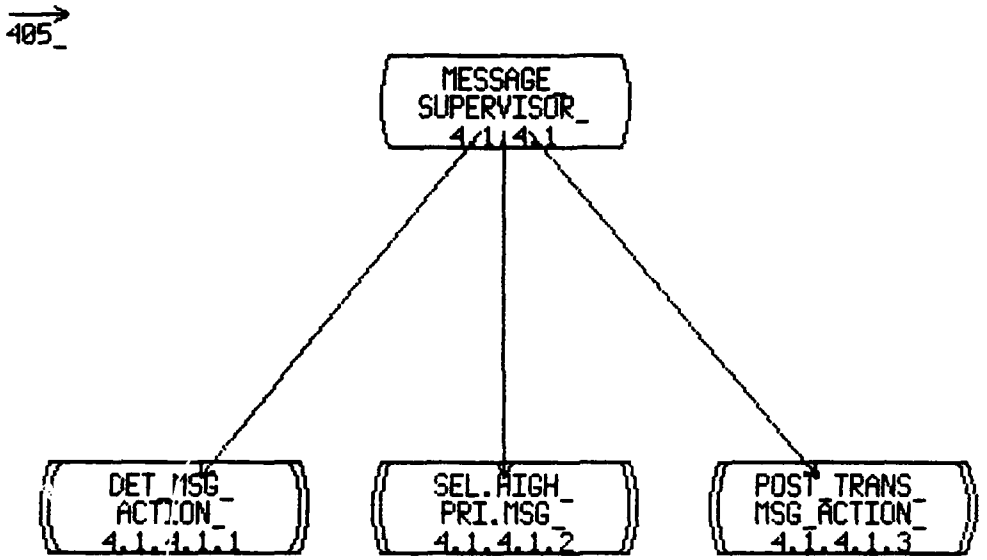
TITLE: 4.1.7 PRODUCE INTERNAL MESSAGE SED		PAGE: 408
REVISION 0	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		



TITLE: 4.2.3 MESSAGE POSTING TIMER SED		PAGE: 409
REVISION 0	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		

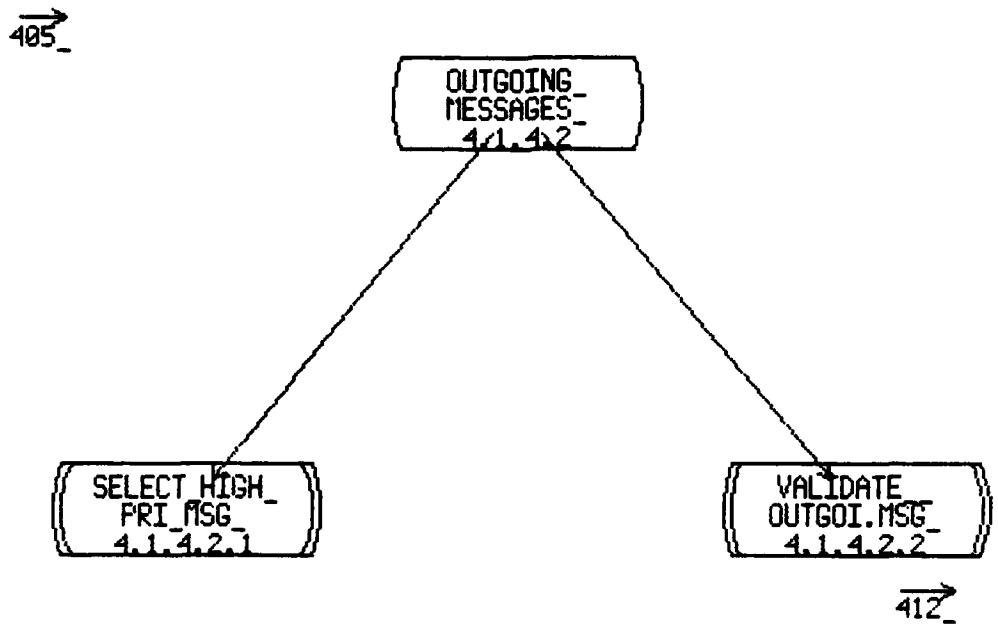


TITLE: 4.1.4.1 MESSAGE SUPERVISOR SED		PAGE: 410
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		

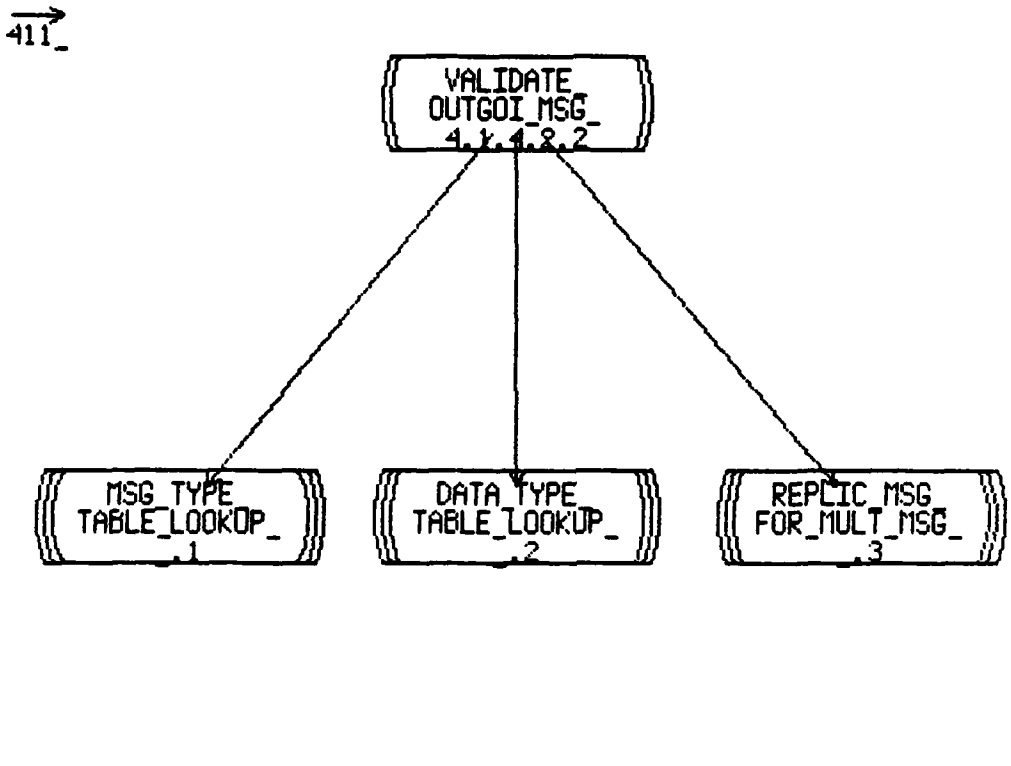


TITLE: 4.1.4.2 OUTGOING MESSAGES SED		PAGE: 411
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82

NOTES:



TITLE: 4.1.4.2.2 VALIDATE OUTGOING MESSAGES SED		PAGE: 412
REVISION 1	AUTHOR: RAW/GIC	DATE: 4/9/82
NOTES:		



APPENDIX D

ADA-BASED SYSTEM DESIGN LANGUAGE

APPENDIX D

ADA-BASED SYSTEM DESIGN LANGUAGE

FINAL REPORT

LARGE SCALE SOFTWARE SYSTEM
FOR THE
MISSILE MINDER AN/TSQ-73
USING
THE ADA PROGRAMMING LANGUAGE

PREPARED FOR

U.S. ARMY COMMUNICATIONS ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703

```
$ TYPE Q73.SDL
procedure TSQ-73 is
--  The objective of the Missile Minder (AN/TSQ-73)
--system is the Air Defense command and control
--system designed to coordinate the actions of
--HAWK,IMPROVED HAWK, and NIKE/HERCULES surface
--to air missiles against hostile air targets.
--The system is fielded in two configurations, the
--GROUP/BDE-level system and the BATALLION/BN-level
--system.
--  The mission of the GROUP/BDE-level system
--concentrates on the monitoring of Air Defense
--activities of subordinate BN-level systems,
--allocation and asignment of Air Defense resources,
--planning Air Defense operations, supervising the
--effectiveness of real-time Air Defense resources,
--serving as the tactical operations center for group
--operations,and interfacing with other GROUP-level
--MISSILE MINDER systems, other Army tactical command
--and control systems, and tactical command and
--control systems of other military services.
--  The mission of the BATTALION/BN-level configuration
--is to control the actual direction of the air battle
--with its execution being conducted at the fire unit
--level.  The BN-level system acquires,tracks, and
--identifies aircraft,provides real-time threat
--evaluation and weapons assignment for hostile
--aircraft engagement, and provides friendly aircraft
--protection.  A BN-level system can be designated as a
--master BN-level system when a GROUP/BDE-level system
--is inoperable or non-existent due to local conditions.
--The designated master BN-level system will provide the
--interface to other Army tactical data systems and tactical
--command and control systems of other military services.
--  The TSQ-73 operator shall select either the GROUP
--configuration or the BATTALION configuration at runtime
--initialization (site adaptation).  The redesign of the
--TSQ-73 has been limited to the BATTALION (BN)
--configuration.  The TSQ-73 has been repartitioned to
--reveal four concurrent processes:
```

```
package COMMAND AND CONTROL (1.0);
package TARGET CONTROL (2.0);
package REMOTE COMMMUNICATION (3.0);
package RADAR COMMUNIATION (4.0);

--  The numbers in parentheses correspond to the numbers
--on the associated SYSTEM ENTITY DIAGRAM (SED).
--  Each of the above packages will have an associated
--elaboration in the form outlined in CONTROL DATA
--CORPORATION's Ada System Designer's Guide.  Selected
--system requirements such as shelter, cooling fans, power
--equipment, etc., have not been addressed.
--  Reference the "Redesign of the AN/TSQ-73 System
--Engineering Block Diagram", "System Entity Diagram",
--and the "AN/TSQ-73 Hierarchical Chart" for an
--overall view of the AN/TSQ-73 redesign.  The Engineering
--Diagram is the theoretical product of the various
--hardware/software tradeoff decisions.  The SDL provides
--input (for the System being developed) to this decision
--process.

-------------------------------------------------------------

--  The System Design Language (SDL) definition has evolved
--during the redesign of the TSQ-73 System.  As such, the
--first SDL was produced for the REMOTE COMMUNICATION which
--required four iterations but has not been updated due to
--time limitations.  The second and third SDL's were RADAR
--COMMUNICATION and TARGET CONTROL.  RADAR COMMMUNICATION
--can be considered an update for REMOTE COMMUNICATION since
--they are very similar.  TARGET CONTROL's SDL objective was
--to decompose the SDL into a PDL.  Again, time limitations
--have restricted the effort to produce an SDL for COMMAND
--AND CONTROL.
```

```
$ TYPE COMCON.SDL
package COMMAND AND CONTROL (1.0) is
--  The COMMAND AND CONTROL package controls the
--total system resources and allocates them to the
--various tasks in the TSQ-73 system.  The COMMAND
--AND CONTROL package incorporates a number of utilities
--which will reduce application development time, exercise
--the various application packages within the TSQ-73 and
--provide a friendly interface between the operators and
--the TSQ-73 system.  The COMMAND AND CONTROL package
--consists of:
--    package GENERAL OPERATING SYSTEM (1.1);
--    package RUNTIME SUPPORT SOFTWARE (1.2);
--    package RUNTIME INITIALIZATION (1.3);
--    package USER APPLICATION LIBRRY (1.4);
--    package I/O MESSAGE INTERPRETER (1.5);
--    package RUNTIME TERMINATION (1.6);
--    package RUNTIME SYSTEM EXERCISOR (1.7);
--  A subset of the COMMAND AND CONTROL package will be
--utilized by local,remote and radar communication
--processors.
--  It is not the intent of the COMMAND AND CONTROL SDL
--to be complete.  Its intent is to provide an outline
--for COMMAND AND CONTROL and to be utilized as a
--reference as to what is contained within the package.
--This is primarily due to the limited resources allocated
--to it under the existing contract.

-----------------------------------------------------------------

 package GENERAL OPERATING SYSTEM (1.1) is
--  The operating system is a general-purpose,multi-tasking,
--event-driven,secondary-storage-based system that controls
--the system resources for real-time,time-sharing and
--batch applications.  A subset of the general operating
--system will be utilized for the runtime operating system.
--The GENERAL OPERATING SYSTEM consists of:
  package SYSTEM BOOTSTRAP INITIALIZATION (1.1.1);
  package RESOURCE MANAGEMENT (1.1.2);
  package TASK MANAGEMENT (1.1.3);
```

```
   package TASK COMMUNICATION (1.1.4);
   package POWER FAIL/AUTOMATIC RESTART (1.1.5);
   package DEVICE I/O HANDLERS (1.1.6);
   package FILE MANAGEMENT (1.1.7);
   package ERROR RECOVERY AND SYSTEM REPORTING (1.1.8);
   package SYSTEM COMMAND INTERPRETER (1.1.9);
   package MAINTENANCE AND DIAGNOSTICS UTILITIES (1.1.10);
 end GENERAL OPERATING SYSTEM (1.1);

 package body GENERAL OPERATING SYSTEM (1.1) is
  package SYSTEM BOOTSTRAP INITIALIZATION (1.1.1) is
--  SYSTEM BOOTSTRAP INITIALIZATION  performs a system
--boostrap initialization from a cold start (the section
--on POWER FAIL/AUTOMATIC RESTART and INTERRUPT MANAGEMENT).
--This is accomplished by a hardware interrupt processor
--RESET switch.  A preliminary set of hardware diagnostic
--routines are executed on the processor,memory,
--controllers and interfaces, system console and peripheral
--devices, etc.  Hardware faults, if any, will be identified
--along with the reason for their fault.  These faults will
--be displayed by means of:
--       -system console
--       -status display panel
--       -processor front panel indicators
--       -error lamps on the malfunctioning board
--          (processor,controller,interface,memory,peripheral,
--           etc.)
--  NOTE: Each of the following packages, procedures,
--functions or tasks would have an associated description.
   procedure EXECUTES INITIAL START-UP PROCEDURES (1.1.1.1) is
--  Read-Only_Memory (ROM) routines.
    procedure LIGHTS PROCESSOR(S) FRONT PANEL POWER ON
     INDICATOR (1.1.1.1.1);
    procedure TEST POWER FAIL/AUTOMATIC RESTART INDICATORS
     (1.1.1.1.2);
    procedure SYNCHRONIZE DATA CHANNEL BUSES (1.1.1.1.3);
    procedure LIGHT HARDWARE FAULT INDICATORS (1.1.1.1.4);
    procedure PRELIMINARY HARDWARE DIAGNOSTICS (1.1.1.1.5);
    procedure INITIALIZE POWER FAIL/AUTOMATIC RESTART
     INDICATORS (1.1.1.1.6);
```

```
    procedure EXTINGUISH HARDWARE FAULT INDICATORS
     (1.1.1.1.7);
--  Extinguish only hardware fault indicators that have
--successfully passed the preliminary hardware diagnostics.
    procedure START MAINTENANCE AND DIAGNOSTICS PROCESSOR
     (1.1.1.1.8);
   end EXECUTES INITIAL START-UP PROCEDURES (1.1.1.1);

   procedure LOADS AND EXECUTES OPERATING SYSTEM (1.1.1.2) is
--OS is GENERAL OPERATING SYSTEM
    procedure IDENTIFIES OS SOURCE LOCATION (1.1.1.2.1);
    procedure INITIALIZES OS SOURCE LOCATION DEVICE
     (1.1.1.2.2);
    procedure LOADS OS INTO MEMEORY (1.1.1.2.3);
    procedure STARTS OS EXECUTION (1.1.1.2.4);
   end LOADS AND EXECUTES OPERATING SYSTEM (1.1.1.2);

   procedure INITIALIZE SYSTEM CONSOLE DEVICE (1.1.1.3) is
    procedure CONSOLE DEVICE DETERMINATION (1.1.1.3.1);
    procedure OPEN CONSOLE DEVICE FOR I/O (1.1.1.3.2);
    procedure REPORT CURRENT SYSTEM STATUS (1.1.1.3.3);
--  Status includes date, time, CPU identification codes,
--hardware faults (if any), operating system version,
--software exceptions,etc.
   end INITIALIZE SYSTEM CONSOLE DEVICE (1.1.1.3);

   procedure INITIALIZE SYSTEM LOG (1.1.1.4) is
    procedure SYSTEM LOG DEVICE DETERMINATION (1.1.1.4.1);
    procedure OPEN SYSTM LOG DEVICE FOR I/O (1.1.1.4.2);
    procedure REPORT CURRENT SYSTEM STATUS (1.1.1.4.3);
--  Status includes date,time,CPU identification codes,
--hardware faults (if any), operating system version,
--software exceptions, etc.
   end INITIALIZE SYSTEM LOG (1.1.1.4);

   procedure INITIALIZE REMAINING PERIPHERALS (1.1.1.5) is
--  This procedure will initialize the remaining peripheral
--devices: secondary-storage devices (if not initialized for
--system bootstrap), magnetic tape devices (if not
--initialized for system bootstrap), display consoles,
```

```
--auxiliary read-outs, status display panels, etc.
    procedure PERIPHERAL DEVICE DETERMINATION (1.1.1.5.1);
    procedure OPEN PERIPHERAL DEVICE FOR I/O (1.1.1.5.2);
--  Devices are opened based upon characteristics, that is,
--devices only capable of output are only opened for
--output,etc.
    procedure REPORT CURRENT SYSTEM STATUS (1.1.1.5.3);
--  Only for status display panels.
   end INITIALIZE REMAINING PERIPHERALS (1.1.1.5);

   procedure INITIALIZE SYSTEM COPROCESSORS (1.1.1.6) is
--  Similiar to the procedures described in EXECUTES
--INITIAL START-UP PROCEDURES (1.1.1.1) with the addition
-- that the coprocessors may be downline loaded.
   end INITIALIZE SYSTEM COPROCESSORS (1.1.1.6);

   procedure LOAD AND EXECUTE SYSTEM COMMAND INTERPRETER
    (1.1.1.7) is
    procedure IDENTIFIES SYSTEM COMMAND INTERPRETER LOCATION
     (1.1.1.7.1);
    procedure LOADS SYSTEM COMMMAND INTERPRETER INTO MEMORY
     (1.1.1.7.2);
--  Initializes source location device if necessary.
    procedure START SYSTEM COMMMAND INTERPRETER EXECUTION
     (1.1.1.7.3);
   end LOAD AND EXECUTE SYSTEM COMMMAND INTERPRETER
    (1.1.1.7);
  end SYSTEM BOOTSTRAP INITIALIZATION (1.1.1);

  package body SYSTEM BOOTSTRAP INITIALIZATION (1.1.1) is
--package body would be placed here
  end SYSTEM BOOTSTRAP INITIALIZATION (1.1.1);

-------------------------------------------------------------

  package RESOURCE MANAGEMENT (1.1.2) is
--RESOURCE MANAGEMENT performs the management of all the
--system's resources

   procedure PROCESSOR MANAGEMENT (1.1.2.1);
```

```
-- PROCESSOR MANAGEMENT (1.1.2.1) supervises,coordinates
--and manages the processor's resources with the other
--resource managers as well as provide for instruction
--execution,instruction prefetching, etc.

   procedure MEMORY MANAGEMENT (1.1.2.2);
-- MEMORY MANAGEMENT provides resources for:
--      -address relocation translation
--      -storage protection
--      -stacking facilities
--      -dynamic memory management
--      -static memeory management
--      -system common management

   procedure INTERRUPT MANAGEMENT (1.1.2.3);
-- INTERRUPT MANAGEMENT processes I/O interrupts and
--exception interrupts.  I/O interrupts are priority
--interrupts from peripheral I/O devices and sensor
--I/O devices.  Exception interrupt handlers process
--interrupts from errors or exceptions which consist of:
--      -processor front-panel switches
--      -processor checks
--      -power/thermal warnings
--      -software exceptions
--      -system(supervisor) calls

   procedure TIME MANAGEMENT (1.1.2.4);
--  The services of TIME MANAGEMENT are:
--      -date
--      -time of day
--      -event or interval timers
--      -programmable timers
--  The services are provided for time and date dependent
--operations.


   procedure PERIPHERAL MANAGEMENT (1.1.2.5);
--  PERIPHERAL MANAGEMENT supervises,coordinates and
--manages the following peripheral devices.
--      -system consoles
```

```
--      -secondary storage units
--      -magnetic tape units
--      -display consoles
--      -auxiliary read-out
--      -status display panels
--      -processor front panel
--      -error lamp indicators


   procedure COPROCESSOR MANAGEMENT (1.1.2.6);
--  COPROCESSOR MANAGEMENT  supervises, coordinates and
--manages attached floating-point processors and
--attached I/O computers within the system configuration.

  end RESOURCE MANAGEMENT (1.1.2);

  package body RESOURCE MANAGEMENT (1.1.2) is
--package body would be placed here
  end RESOURCE MANAGEMENT (1.1.2);

-----------------------------------------------------------------


  package TASK MANAGEMENT (1.1.3) is
--  TASK MANAGEMENT  supervises, coordinates, schedules
--and manages all tasks for execution.  TASK MANAGEMENT
--supports multi-programming which allows more than one
--task to be executing in the same processor concurrently.
--These concurrent tasks must share CPU time through a
--scheduling policy.  The scheduling policy treats
--differently the scheduling of devices and non-device
--tasks with the same priority.

   package TASK CHARACTERISTICS MANAGER (1.1.3.1) is
    procedure TASK TYPE (1.1.3.1.1);
    procedure TASK STATUS (1.1.3.1.2);
    procedure TASK PRIORITY (1.1.3.1.3);
    procedure SITE EXECUTION LOCATION DETERMINATION
     (1.1.3.1.4);
    procedure CRITICAL TASKS (1.1.3.1.5);
   end TASK CHARACTERISTICS MANAGER (1.1.3.1);
```

```
    package TASK SCHEDULE POLICY MANAGER (1.1.3.2);
    package TASK SYNC MANAGER (1.1.3.3);
    package TASK INVOCATION (1.1.3.4);
    package TASK SUSPENSION (1.1.3.5);
    package TASK RESUMATION (1.1.3.6);
    package TASK TERMINATION (1.1.3.7);
    package TASK CREATION (1.1.3.8);
    package TASK DELETION (1.1.3.9);
  end TASK MANAGEMENT (1.1.3);

  package body TASK MANAGEMENT (1.1.3) is
--package body would be placed here
  end TASK MANAGEMENT (1.1.3);

-----------------------------------------------------------------

  package TASK COMMUNICATION (1.1.4) is
--  TASK COMMUNICATION supervises, coordinates and manages
--various mechanisms available for task communications.
   package LOW LEVEL (1.1.4.1) is
    procedure DEVICE COMM WITH I/O BUS (1.1.4.1.1);
    procedure DEVICE COMM WITH MEMORY I/O BUS (1.1.4.1.2);
    procedure INTERTASK COMM WITH SHARED VARIABLES (1.1.4.1.3);
--  The simplest form of intertask communications is
--accomplished through the sharing of variables or records.
--However, only a single task should be allowed to operate
--on a variable at a time.  A semaphore is used to
--guarantee exclusive access.

    procedure INTERTASK COMM WITH MSG BUFFER (1.1.4.1.4);
--  A message buffer is a shared data structure through
--which messages are transferred and buffered among tasks.
--A message is any data structure (integer,real,character,
--arrays,records or pointers) which can be copied from
--one task to another. A semaphore is used to guarantee
--exlusive access.

   end LOW LEVEL (1.1.4.1);

   package HIGH LEVEL (1.1.4.2) is
```

```
    procedure CHANNELS (1.1.4.2.1);
--  CHANNELS provide message transmission from more than
--one task to one or more tasks.  Messages are any data
--structure (integer, real,character,arrays,records or
--pointers).  A message can be transmitted more than once.
--Messages are transmitted over channels.  A channel
--synchronizes and buffers message flow between two or more
--processes.  Semaphores may be used to guarantee exclusive
--access.

    procedure FILES (1.1.4.2.2);
--  FILES provides a real-time memory interface for logical
--I/O.  (Reference section 1.1.7 for logical I/O.)
--Cooperating tasks may communicate with each other: one
--task may read file components whichare being written
--concurrently by another task.  The input is generated and
--the output consumed in memory by tasks rather than waiting
--for the data to be stored onto and retrieved from some
--secondary storage device.

    procedure INTRASYSTEM TRANSFER (1.1.4.2.3);
   end HIGH LEVEL (1.1.4.2);
  end TASK COMMUNICATION (1.1.4);

-------------------------------------------------------------------

  package POWER FAIL AUTO RESTART (1.1.5) is
   procedure POWER FAIL (1.1.5.1);
   procedure AUTO RESTART (1.1.5.2);
  end POWER FAIL AUTO RESTART (1.1.5);

-------------------------------------------------------------------

  package DEVICE I/O HANDLERS(1.1.6) is
--  The DEVICE I/O HANDLERS package provices a physical
--device I/O interface, a physical to logical interface and
--logical device channels.  Physical devices are:
--    -system console
--    -secondary storage unit
--    -magnetic tape unit
```

```
--    -display console
--    -auxiliary read-out
--    -status display panel
--    -processor front panel

    procedure PHYSICL DEVICE INTERFACE(1.1.6.1);
    procedure PHYSICAL TO LOGICAL INTERFACE (1.1.6.2);
    procedure LOGICAL DEVICE CHANNELS (1.1.6.3);
  end DEVICE I/O HANDLERS (1.1.6);

-------------------------------------------------------------------------

  package FILE MANAGEMENT (1.1.7) is
--   FILE MANAGMENT manages the physical and logical
--organization of information.  Physical organization
--matches data to the requirements of the physical device
--with no regard to the data content or use.  Logical
--organization matches data to the requirements of the user
--by associating and grouping information according to
--content or meaning.
--   FILE MANAGEMENT also manages the recording,retrieving,
--access methods and access privileges to data.

    procedure DATA STRUCTURES (1.1.7.1);
    package FILE MANAGER (1.1.7.2);
    package FILE MANAGER UTILITIES (1.1.7.3);
    procedure FILES TO LOGICAL DEVICE CHANNEL (1.1.7.4);
  end FILE MANAGEMENT (1.1.7);

-------------------------------------------------------------------------

  package ERROR RECOVERY & SYSTEM REPORTING (1.1.8) is
    package ERROR RECOVERY (1.1.8.1);
--   ERROR RECOVERY provides the ability to identify and
--deal with exceptions and the possibility of recovering
--from these exceptions.  ERROR RECOVERY provides for:
--    -task errors
--    -scheduling errors
--    -semaphore errors
--    -interrupt errors
```

```
--    -device I/O errors
--    -memory management errors
--    -file management errors
--    -user errors
--    -exception errors
--    -processor errors
--    -hardware errors
--  For device I/O errors, several attempts will be made
--to successfully complete the I/O operation before the
--error is considered valid.

   procedure SYSTEM REPORTING (1.1.8.2);
--  SYSTEM REPORTING provides for the reporting of
--errors to the system console, status display panel
--(hardware only),display console, and the system log.
--Date and time stamps are appended to the error messages.
--  Application programs and the SYSTEM COMMAND INTERPRETER
--can specify additional message to be recorded in the
--system log.

  end ERROR RECOVERY & SYSTEM REPORTING (1.1.8);

-------------------------------------------------------------------

  package SYSTEM COMMAND INTERPRETER (1.1.9) is
--  The SYSTEM COMMAND INTERPRETER provides an interactive,
--conversational interface between the system and the
--operators terminal.  It also provides a batch processing
--mode.  Services provided through the SYSTEM COMMAND
--INTERPRETER are:
--    -user login/logout
--    -station control (user ID,terminal status)
--    -system task status
--    -system I/O status
--    -logical I/O assignment
--    -program installation,deletion, activation and control
--    -system log
--    -system log activation,listing and termination
--    -file management support
--    -assembler language support
```

```
--    -Ada language support
--    -libraries support: installation and deletion
--    -utilities: ext editors, sort/merge,system generation

    package ASSEMBLER (1.1.9.1);
    package ADA LANGUAGE (1.1.9.2);
    package LIBRARIES (1.1.9.3);
    package EDITORS (1.1.9.4);
    package UTILITIES (1.1.9.5);
   end SYSTEM COMMAND INTERPRETER (1.1.9);

-------------------------------------------------------------------

   package MAINTENANCE & DIAGNOSTIC UTILITIES (1.1.10);
    package DESTRUCTIVE OFFLINE UTILITIES (1.1.10.1);
    package NON-DESTRUCTIVE ON LINE UTILITIES (1.1.10.2);
   end MAINTENANCE & DIAGNOSTIC UTILITIES (1.1.10);
 end GENERAL OPERATING SYSTEM (1.1);

-------------------------------------------------------------------
-------------------------------------------------------------------

 package RUNTIME SUPPORT SOFTWARE (1.2) is
  package RUNTIME OPERATING SYSTEM (1.2.1);
  package RUNTIME SYSTEM LIBRARY (1.2.2);
  package MATHEMATICAL LIBRARY (1.2.3);
  package CHARACTER STRING LIBRARY (1.2.4);
  procedure RUNTIME ON-LINE DEBUGGER (1.2.5);
 end RUNTIME SUPPORT SOFTWARE (1.2);

-------------------------------------------------------------------
-------------------------------------------------------------------

 package RUNTIME INITIALIZATION (1.3) is
  procedure LOAD RUNTIME SUPPORT SOFTWARE (1.3.1);
  procedure OPTION LOAD ONLINE DEBUGGER (1.3.2);
  procedure LOADS USER APP LIB (1.3.3);
  procedure OPTION LOAD SYSTEM EXERCISOR (1.3.4);
  procedure LOAD I/O MSG INTERPRETER (1.3.5);
  procedure OPTION OPEN TRANSACTION MSG LOG (1.3.6);
```

```
  procedure LOAD SITE ADAPTATION (1.3.7);
  procedure LOAD TARGET CONTROL (1.3.8);
  procedure LOAD REMOTE COMMUNICATION (1.3.9);
  procedure LOAD RADAR COMMMUNICATION (1.3.10);
 end RUNTIME INITIALIZATION (1.3);

-------------------------------------------------------------------
-------------------------------------------------------------------

 package USER APP LIBRARY (1.4) is
  function TRANSVERS MERCATOR (1.4.1);
  function STEREOGRAPHIC (1.4.2);
  function SLANT COORDINATE (1.4.3);
  function GEOGRAPHIC (1.4.4);
 end USER APP LIBRARY (1.4);

-------------------------------------------------------------------
-------------------------------------------------------------------

 package I/O MESSAGE INTERPRETER (1.5) is
  procedure MESSAGE ROUTING MANAGER (1.5.1) is
   procedure INPUT MESSAGES (1.5.1.1);
   procedure OUTPUT MESSAGES (1.5.1.2);
  end MESSAGE ROUTING MANAGER (1.5.1);

  procedure DISPLAY UPDATE I/O MANAGER (1.5.2);
 end I/O MESSAGE INTERPRETER (1.5);

-------------------------------------------------------------------
-------------------------------------------------------------------

 package RUNTIME TERMINATION (1.6) is
  procedure GENERATE TERMINATION MSG TO LOCAL/REMOTE USERS
   (1.6.1);
  procedure TERMINATE RADAR COMM (1.6.2);
  procedure TERMINATE REMOTE COMM (1.6.3);
  procedure TERMINATE TARGET CONTROL (1.6.4);
  procedure OPTIONALLY CLOSE TRANSACTION MSG LOG (1.6.5);
  package TERMINATION OPTIONS (1.6.6);
 end RUNTIME TERMINATION (1.6);
```

```
---------------------------------------------------------------------------
---------------------------------------------------------------------------

 package RUNTIME SYSTEM EXERCISOR (1.7) is
  procedure EXERCISOR COMMAND SUPERVISOR (1.7.1);
  procedure USER APPLICATION LIBRARY EXERCISOR (1.7.2);
  procedure I/O MSG INTERPRETER EXERCISOR (1.7.3);
  procedure SITE ADAPTATION EXERCISOR (1.7.4);
  procedure TARGET CONTROL EXERCISOR (1.7.5);
  procedure REMOTE COMM EXERCISOR (1.7.6);
  procedure RADAR COMM EXERCISOR (1.7.7);
 end RUNTIME SYSTEM EXERCISOR (1.7);
end COMMAND AND CONTROL (1.0);

$ }is
%DCL-W-IVVERB, unrecosnized command
  \}IS\
```

```
$ TYPE TARGET.SDL
package TARGET_CONTROL_2_0 is

--   Target control is one of four packages which compose the
--missle minder air defense A/N TSQ-73  system. The target
--control package processes radar report data, updates track
--data based on recent radar reports and if required,
--automatically initiates new tracks. The track data is used
--to evaluate the threat posed by a track to defended points
--and to determine the fire unit(s) best qualified to engage a
--specific hostile. Upon an engagement order, target control
--is responsible for monitoring the engagement as well as
--providing friendly track protection. During system
--initialization target control generates the algorithms
--necessary to perform the air defense computations which
--contain site dependent parameters. Target control is itself
--composed of five packages:
--    o Start_up_initialization_2_1
--    o Tracking_2_2
--    o Track_manager_2_3
--    o Counter_measure_processing_2_4
--    o Target_engagement_2_5

     package START_UP_INITIALIZATION_2_1 is

     --   Upon system initialization, all tracking and
     --target engagement algorithms with site dependent
     --parameters are initialized. In addition sector
     --definition and table initialization is provided.

       procedure INITIALIZE_TRACK_ALGORITHMS_2_1_1;
       --
       -- Not further decomposed
       --

       procedure INITIALIZE_THREAT_EVALUATION_ALGORITHMS_2_1_2;
       --
       -- Not further decomposed
       --
```

```
   procedure INITIALIZE_WEAPON_ASSIGNMENT_ALGORITHMS_2_1_3;
   --
   -- Not further decomposed
   --

   procedure INITIALIZE_TRACK_TABLES_2_1_4;
   --
   -- Not further decomposed
   --

   procedure SECTOR_DEFINITION_2_1_5;
   --
   -- Not further decomposed
   --

end START_UP_INITIALIZATION_2_1 ;

package TRACKING_2_2 is

--   Tracking_2_2 processes radar reports received through
--data_control_2_3_4 in order to perform identification of friend or
--foe (iff/sif), correlate, associate, smooth, and predict new
--positions for existing tracks, automatically initiate
--new tracks, drop tracks, and evaluate track
--quality.

  package IFF_SIF_2_2_1 is

  --   The IFF_SIF package is responsible for all
  --activities which are required to identify air traffic as
  --friendly or unfriendly.

       procedure AUTO_BECON_INTERROGATION_2_2_1_1 (  in
                 MODE; in out POS; out RESULT);

       --   Beacon interrogation shall be performed
       --automatically whenever the category of a track
       --is altered.

       procedure MANUAL_INTERROGATION_2_2_1_2 (in
```

```
                MODE1; in MODE2; in out POS; out RESULT);

        --    Manual interrogation is indirectly requested
        --by the operator when he hooks a specific auto or non-auto
        --track. The operator may specify one or two modes for use;
        --otherwise the current system modes shall be used.

        procedure VERIFY_IFF_CODE_2_2_1_3 (in MODE;
                    in out POS; out OK);

        --    Mode 1,2,3A, and C codes shall be
        --verified as follows:
        --     o If a radar report containing a mode C code associates
        --      with a track, the code shall be accepted for that track.
        --     o The term "certified" applies to mode
        --      1, mode 2, and mode 3A codes. Certification shall be
        --      accomplished if necessary, whenever automatic
        --      interrogation is performed. This certification shall be
        --      attempted by repeated interrogation until one of two
        --      conditions results:
        --        1. Two consecutive matching codes
        --             are received
        --        2. Three consecutive null responses
        --             are received (null condition).

        procedure MODE_4_INTERROGATION_2_2_1_4 (in out
                    POS; out RESULT);

        --    Mode_4_interrogation shall be requested
        --directly by the operator and the response shall be accepted
        --and displayed.

end;


task TARGET_REPORT_PROCESSING_2_2_2 is

--    TARGET_REPORT_PROCESSING is responsible for
--performing the front end processing on all target information
--received by TRACKING_2_0. Sources for this data are local
```

```
--radar, remote radar, remote communication, and the operator.
--Local and remote radar reports are processed in such a
--manner that all slant coordinates are transformed into
--rectangular coordinates, and are subsequently handed to
--the TRACK_WHILE_SCAN_2_2_3 task. The reports received by
--REMOTE_COMMUNICATION_3_0 are received with coordinate data
--referenced with a remote system coordinate center (SCC).
--TARGET_REPORT_PROCESSING_2_2_2 processes these remote
--reports so that they all are referenced with the local SCC
--and have their coordinate data in rectangular form. These
--remote reports are placed in the track storage file and
--are updated with remote reports received on succeeding scans
--The reports generated by the operator are considered as
--rate-aided-manually-initiated-tracks (RAMIT) and are
--updated manually.
--
     entry LOCAL_RADAR(LOCAL_RADAR_BUFFER);
     entry REMOTE_RADAR(REMOTE_RADAR_BUFFER);
     entry REMOTE_COMM(REMOTE_REPORT);
     entry OPERATOR(RAMIT);



end TARGET_REPORT_PROCESSING_2_2_2 ;

task TRACK_WHILE_SCAN_2_2_3   is
--
--    The track while scan task is the central activity
--in the tracking process. It is the responsibility of
--the track while scan task to accept the target
--report information generated by the local and remote
--radars and the remote track information received
--by REMOTE_COMMUNICATION_3.0 and to match the most
--recently received information with the historical
--information stored for each track.
--
     entry NEW_TGT_BUFFER (TGT_REP_BUFF ) ;

end TRACK_WHILE_SCAN_2_2_3 ;
```

```
package TRACK_WHILE_SCAN_UTILITIES_2_2_4   is

--   Track while scan utilities contains the routines
--which handle the automatic initiation and dropping
--of tracks to and from the track storage file as well
--as a routine which estimates the reliability of the
--positional data held for any given track, i.e. track
--quality.


  procedure AUTO_INITIATION_NEW_TRACK_2_2_4_1
    (in NON_ASSOC_TGT_REP ; in NON_CORREL_TGT_REP );

  --  The automatic initiation capability shall apply
  --to the entire radar coverage area when the system
  --initiation mode is AUTOMATIC. When the initiation
  --mode is MANUAL this capability shall apply only within 5DM
  --of non-auto tracks.
  --   After all possible target report/track
  --associations have been made in a given vicinity, those
  --reports that have failed to associate with any existing
  --track shall be used to generate new auto tracks. A new
  --track initiated in this way shall be placed in the
  --tenative track category. For the purpose of the
  --association count, this first report shall count
  --as an association.
  --   Since a single position measurement provides no
  --velocity estimate, all directions and speeds are
  --equiprobable. However, the upper and lower speed limits
  --for automatically initiated tracks limit the area
  --withhn which subsequent target reports must fall
  --in order to associate with these initial tenative
  --tracks.

  task DROP_TRACK_2_2_4_2 is
  --   A RAMIT track shall be dropped if no manual
  --update action is taken before the second manual update
  --warning time period has elapsed. A remote non-auto track
  --shall be dropped if it has received no remote communication
  --message update for 60 seconds.
```

```
--   Provisional and established tracks shall be
--declared to be in poor tracking status if no association
--can be made for three scans. An established track that
--has been declared in poor tracking status shall be changed
--to provisional with a radar association count of 4. An
--operator indication shall be posted. Provisional tracks
--shall be dropped if no association can be made within 45
--seconds after declaration of poor tracking status,
--except as follows:
--   o Remote auto tracks that fail to associate for
--     three scans shall be switched to remote non-auto.
--   o Any track that is locally engaged shall not be dropped.
--     In this case the track shall be dead-reckoned and drop
--    action shall be attempted in each succeeding scan.
--   o For all MBDL designated tracks, if the time since the
--     last update is equal to or greater than 45 seconds
--     a drop indication is warranted and posted.

     entry TRK_DATA ( in NON_ASSOC_TRACK;
                      in NON_CORREL_TRACK);

end DROP_TRACK_2_2_4_2 ;

task DETERMINE_TRACK_QUALITY_2_2_4_3 is

--   Local track quality is an assigned value based
--upon the degree of estimated reliability of the
--positional data. The assigned value for real-time
--tracks shall be limited to range of 1 through 7.
--A track quality of 0 shall indicate a non-real
--time track.
--   The track quality of local auto tracks used for
--ATDL-1 and TADIL-B reporting responsibility shall be:
--   o Initially reported as 4 when a track becomes
--     provisional.
--   o Incremented by one after two successive
--     associations.
--   o Decremented by one after two successive
--     misses.
--   The track quality of local non-auto tracks
```

```
      --(ramit) used for ATDL-1 and TADIL-B reporting
      --responsibility shall be:
      --    o Initially reported as 4 when track is
      --      manually initiated.
      --    o Incremented by one as a result of a manual
      --      update action.
      --    o Decremented by one when two successive
      --      scans have occurred without a manual
      --      update action.
      --    The track quality of pop-up tracks used for
      --ATDL-1 and TADIL-B reporting responsibility
      --shall be:
      --    o Initially reported as 4 when a pop-up
      --      track is declared.
      --    o Decremented by one when two successive
      --      scans have occurred without any  fire unit
      --      update.


           entry TRK_DATA ( in ASSOC_DATA) ;

      end DETERMINE_TRACK_QUALITY_2_2_4_3 ;

   end TRACK_WHILE_SCAN_UTILITIES_2_2_4 ;

end TRACKING_2_2 ;

package TRACK_MANAGER_2_3 is
--
--    The track manager provides the capability necessary
--to receive , send and route all intra-system messages
--pertinent to TARGET_CONTROL_2_0. In addition the
--TRACK_MANAGER is responsible for the maintenance
--of all tables unique to the TARGET_CONTROL_2_0
--activity.

  procedure TRACK_UPDATE_2_3_1;
  --
  -- Not further decomposed
  --
```

```
   procedure MANAGE_TABLES_2_3_2;
   --
   -- Not further decomposed
   --

   procedure DETERMINE_TRACK_STATUS_2_3_3;
   --
   -- Not further decomposewd
   --

   procedure DATA_CONTROL_2_3_4;
   --
   -- Not further decomposed
   --

   procedure OPERATOR_SWITCH_ACTIONS_2_3_5;
   --
   -- Not further decomposed
   --

end TRACK_MANAGER_2_3 ;

package COUNTER_MEASURE_PROCESSING_2_4 is

--   COUNTER_MEASURE_PROCESSING_2_4 provides all
--activities necessary to provide effective
--identification and control for air traffic
--which attempt to avoid detection through use
--of jamming or chaff techniques.

  procedure JAM_STROBE_PROCESSING_2_4_1;
  --
  -- Not further decomposed
  --

  procedure CHAFF_PROCESSING_2_4_2;
  --
  -- Not further decomposed
  --
```

```
end COUNTER_MEASURE_PROCESSING_2_4 ;

package TARGET_ENGAGEMENT_2_5 is
--    The objective of TARGET_ENGAGEMENT_2_5 is to
--evaluate the threat posed to defended points by
--unknown and hostile air traffic. After calculation
--of the threat posed by each track, the fire unit(s)
--best qualified to engage that particular track are
--determined. Upon an engagement order,
--TARGET_ENGAGEMENT_2_5 is responsible for monitoring all
--engagements.

  procedure EVALUATE_THREAT_2_5_1 ;
  --    EVALUATE_THREAT_2_5_1 obtains track information
  --for each track and evaluates the threat posed by that
  --track to each defended point. The threat posed by
  --each track for each defended point shall
  --be evaluated once for each scan.

  procedure EVALUATE_WEAPON_ASSIGNMENT_SCORE_2_5_2;
  --    EVALUATE_WEAPON_ASSIGNMENT_SCORE_2_5_2  computes
  --a value which is used by the WEAPON_ASSIGNMENT_2_5_3
  --activity to assign a target to the appropriate fire
  --unit(s).

  procedure WEAPON_ASSIGNMENT_2_5_3;
  --    WEAPON_ASSIGNMENT_2_5_3 automatically assigns the
  --fire unit(s) best qualified to engage hostile air
  --tracks. It also performs bookeeping operations which are
  --necessary to maintain stockpile balancing as well as
  --distinguish between weapons tight and weapons free.
  --In the weapons tight mode assignment of unknown tracks
  --made be accomplished by manual operation only.

  procedure MONITOR_ENGAGEMENT_2_5_4;
  --    MONITOR_ENGAGEMENT_2_5_4 shll provide friendly
  --protection by monitoring the engagement status of
  --friendly and unknown tracks. In addition, it shall
```

```
      --monitor ensaged tracks relative to safe corridors.
      --Hold fire alerts shall be issued to
      --REMOTE_COMMUNICATION_3_0 if ensaged tarsets are
      --within safe corridors.

   end TARGET_ENGAGEMENT_2_5 ;



end TARGET_CONTROL_2_0 ;

package body TARGET_CONTROL_2_0 is

      package body START_UP_INITIALIZATION_2_1 is

      --    Upon system initialization, all tracking and
      --tarset ensasement alsorithms with site dependent
      --parameters are initialized. In addition sector
      --definition and table initialization is provided.

        procedure INITIALIZE_TRACK_ALGORITHMS_2_1_1 is
        --
        -- Not further decomposed
        --
        end INITIALIZE_TRACK_ALGORITHMS_2_1_1 ;

        procedure INITIALIZE_THREAT_EVALUATION_ALGORITHMS_2_1_2 is
        --
        -- Not further decomposed
        --
        end INITIALIZE_THREAT_EVALUATION_ALGORITHMS_2_1_2 ;

        procedure INITIALIZE_WEAPON_ASSIGNMENT_ALGORITHMS_2_1_3 is
        --
        -- Not further decomposed
        --
        end INITIALIZE_WEAPON_ASSIGNMENT_ALGORITHMS_2_1_3;

        procedure INITIALIZE_TRACK_TABLES_2_1_4 is
```

```
   --
   -- Not further decomposed
   --
   end INITIALIZE_TRACK_TABLES_2_1_4 ;

   procedure SECTOR_DEFINITION_2_1_5 is
   --
   -- Not further decomposed
   --
   end SECTOR_DEFINITION_2_1_5 ;


end START_UP_INITIALIZATION_2_1 ;


package body TRACKING_2_2 is


   package body IFF_SIF_2_2_1 is
   --   The IFF_SIF package is responsible for all
   --activities which are required to identify air traffic as
   --friendly or unfriendly.

      procedure INTERROGATE_BEACON_2_2_1_5
         (in MODE; in out POS; out RESULT) is

      --   Interrogate beacon is the activity which
      --handles the actual callins of the IFF_SIF beacon component
      --of the radar hardware. The interrogation mode currently in
      --effect and the positionof the air traffic in question are
      --accepted as input and a valid or invalid response is
      --received from the beacon and subsequently returned to
      --the callins activity.

      begin

         SEND_INQUIRY_TO_AIRCRAFT(MODE);
         RECEIVE_AIRCRAFT_RESPONSE(AIRCRAFT_RESPONSE);
         if AIRCRAFT_RESPONSE = MODE then
            RESULT = TRUE;
```

```
   end if;
end INTERROGATE_BEACON_2_2_1_5   ;


procedure AUTO_BECON_INTERROGATION_2_2_1_1 (  in
          MODE; in out POS; out RESULT) is

--   Beacon interrogation shall be performed
--automatically whenever the category of a track
--is altered.

begin

   INTERROGATE_BEACON_2_2_1_5 (MODE,POS,RESULT);

end AUTO_BEACON_INTERROGATION_2_2_1_5;


procedure MANUAL_INTERROGATION_2_2_1_2 (in
          MODE1; in MODE2; in out POS; out RESULT) is

--   Manual interrogation is indirectly requested
--by the operator when he hooks a specific auto or non-auto
--track. The operator may specify one or two modes for use;
--otherwise the current system modes shall be used.

begin

   INTERROGATE_BEACON_2_2_1_5 (MODE1,POS,RESULT);
   if MODE2 SPECIFIED then
      INTERROGATE_BEACON(MODE2,POS,RESULT);
   end if;

end MANUAL_INTERROGATION_2_2_1_2 ;


procedure VERIFY_IFF_CODE_2_2_1_3 (in MODE;
           in out POS; out OK) is

--   Mode 1,2,3A, and C codes shall be
```

```
--verified as follows:
--     o If a radar report containing a mode C code associates
--      with a track, the code shall be accepted for that track.
--     o The term "certified" applies to mode
--      1, mode 2, and mode 3A codes. Certification shall be
--      accomplished if necessary, whenever automatic
--      interrogation is performed. This certification shall be
--      attempted by repeated interrogation until one of two
--      conditions results:
--        1. Two consecutive matching codes
--             are received
--        2. Three consecutive null responses
--             are received (null condition)..

begin

   if MODE = C then
      OK := TRUE;
   else
      loop
         INTERROGATE_BEACON_2_2_1_5(MODE,POS,RESULT);
         EVALUATE MATCHING CODES;
         EVALUATE NON_MATCHING CODES;
         exit when CONSEC_MATCH = 2 or
              NON_CONSEC_MATCH = 3;
      end loop;

end VERIFY_IFF_CODE_2_2_1_3;


procedure MODE_4_INTERROGATION_2_2_1_4 (in out
          POS; out RESULT) is

--   Mode_4_interrogation shall be requested
--directly by the operator and the response shall be accepted
--and displayed.

begin

   INTERROGATE_BEACON_2_2_1_5 (MODE,POS,RESULT);
```

```
            DISPLAY_PPI(RESULT);

        end MODE_4_INTERROGATION_2_2_1_4 ;


end IFF_SIF_2_2_1 ;

task body TARGET_REPORT_PROCESSING_2_2_2 is

--not further decomposed

end TARGET_REPORT_PROCESSING_2_2_2 ;



task body TRACK_WHILE_SCAN_2_2_3 is

   package CORRELATION_ACTIONS_2_2_3_1 is
   --  Each target report position shall be compared with
   --all tracks in it's vicinity to determine if the report is
   --close enough to correlate. each report may correlate with
   --more than one track.

      procedure COMPUTE_DEVIATION_VECTOR_2_2_3_1_1
        (in TGT_REP_POS; in TRK_POS; out DEV_VEC );
      --
      --  For all categories of tracks the quantity of
      --interest is the deviation vector between the track predicted
      --position and the target report position.
      --


      procedure INNER_GATE_CORRELATION_2_2_3_1_2
        (in TGT_REP_NO; in TRK_REC_NO; in DEV_VEC; out IG_CORR );
      --
      --  For maneuvering tracks and non-manueverings
      --tracks three types of inner gates shall be used:
      --  o For non-maneuvering tracks
      --       o distance gate check
      --       o range and azimuth error check
      --  o For maneuvering tracks
```

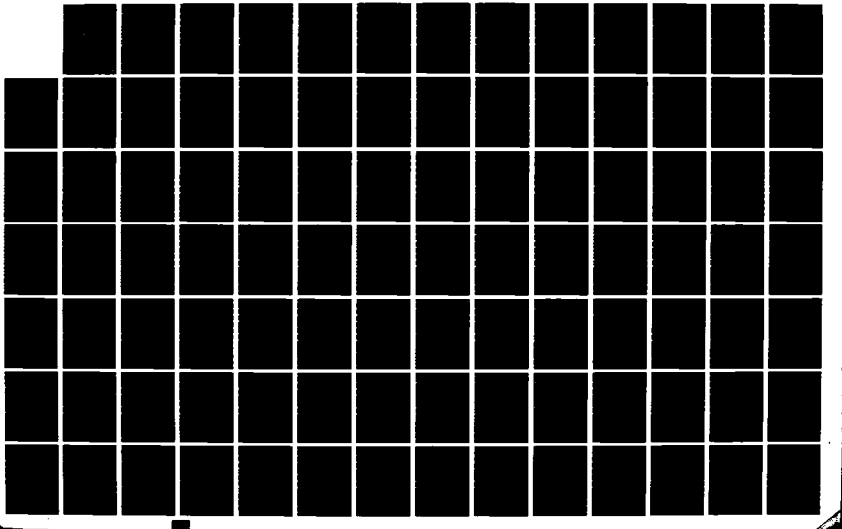
```
--       o maneuver recovery gate check
--


procedure INNER_GATE_CORRELATION_2_2_3_1_2
  (in TGT_REP_NO; in TRK_REC_NO; in DEV_VEC; out IG_CORR );
--
--  The outer gates shall be oriented parallel to
--and normal with each track's velocity vector. The outer gate
--correlation tests shall be performed by decomposing the
--deviation vector into two orthogonal components.
--  o Ud in a direction perpendicular to the
--       track's heading
--  o Wd in the fore and aft direction.
--  Irrespective of the values given by the
--formulas shown in the package body an upper bound of 10 DM
--and a lower bound of 1 DM shall be placed on all outer gate
--values. If all tests are passed, the target report
--correlates with the track in the outer gate. If any test
--fails the target report fails to correlate with the track.
--


 procedure OUTER_GATE_CORRELATION_2_2_3_1_3 (in
           TGT_REP_NO; in TRK_NO; in DEV_VEC; out OG_CORR );


--   he outer gates shall be oriented parallel to
--and normal with each track's velocity vector. The outer gate
--correlation tests shall be performed by decomposing the
--deviation vector into two orthogonal components.
--  o Ud in a direction perpendicular to the
--       track's heading
--  o Wd in the fore and aft direction.
--  Irrespective of the values given by the
--formulas shown in the package body an upper bound of 10 DM
--and a lower bound of 1 DM shall be placed on all outer gate
--values. If all tests are passed, the target report
--correlates with the track in the outer gate. If any test
--fails the target report fails to correlate with the track.
--
```

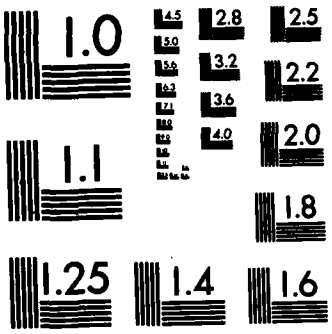
```
procedure NON_AUTO_BEACON_CORRELATION_2_2_3_1_4
    (in TGT_IFF; in TRK_IFF; out
        NON_AUTO_BN_CORR);
--
--  If the absolute miss distance on grid system
--coordinates is greater than 5 DM, no correlation
--may take place. If the report contains no
--IFF/SIF data, no correlation may take place.
--Otherwise, the report is correlated and the
--correlation score is set to 1.
--


procedure ALTITUDE_GATE_CORRELATION_2_2_3_1_5
    (in TGE_REP_ALT; in TRK_ALT; out
        ALT_GATE_CORR);
--
--  If a target report correlates with a track for
--either the inner or the outer gate, the
--possibility of correlation in altitude shall be
--examined. Each track has an altitude estimate
--associated with it which may be derived from 3D
--measurements, from mode C beacon responses or
--from operator inputs. If there are no altitude
--data, the track shall be assigned a default
--altitude of 20,000 feet. If the correlating
--report contains an altitude measurement, either
--from a 3D radar or mode C, this altitude shall
--be compared to the track altitude.
--


procedure CORRELATION_SCORE_2_2_3_1_6
     ( in TGT_REP_NO; in TRK_NO; in POS_CORR; in IFF_CORR;
     in ALT_CORR; out CORR_SCORE);
--
--  During the correlation process for each track,
--a record shall be kept of the target reports
--that correlate with that auto-track, up to a
--maximum of three reports. Each report that has a
--position correlation shall be given a
```

```
   --correlation score with respect to the track.
   --If more than three reports correlate, the three
   --with the highest correlation scores shall be
   --retained for possible use in the association
   --process. In addition, a record shall be
   --maintained of up to three correlating tracks per
   --target report, for use in the association
   --process. There are three separate correlation
   --scores that are used to determine the final
   --correlation score:
   --    o Position correlation
   --    o Altitude correlation
   --    o IFF correlation.
   --

end CORRELATION_ACTIONS_2_2_3_1 ;


package body CORRELATION_ACTIONS_2_2_3_1 is

   procedure COMPUTE_DEVIATION_VECTOR_2_2_3_1_1
     (in TGT_REP_POS; in TRK_POS; out DEV_VEC ) is

   --  For all categories of tracks the quantity of
   --interest is the deviation vector between the track predicted
   --position and the target report position.


   begin

     COMPUTE SCALAR POSITION DIFFERENCES;
     COMPUTE DEVIATION VECTOR LENGTH;

   end COMPUTE_DEVIATION_VECTOR_2_2_3_1_1;

   procedure INNER_GATE_CORRELATION_2_2_3_1
     (in TGT_REP_NO; in TRK_REC_NO;

   --  For maneuvering tracks
   --tracks three types of
```

AD-A123 308 LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE MISSILE MINDER AN/TSQ-73 USING T..(U) CONTROL DATA CORP SHREWSBURY NJ GOVERNMENT SYSTEMS 09 NOV 82 5/6

UNCLASSIFIED DAAK80-81-C-0107 F/G 9/2 NL





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

```
--   o For non-maneuvering tracks
--        o distance gate check
--        o range and azimuth error check
--   o For maneuvering tracks
--        o maneuver recovery gate check

IG_CORR : BOOLEAN := FALSE;

begin

  if NON_MANVR then
     CHECK DISTANCE_GATE_CORR;
     if DISTANCE_GATE_CORR then
        IG_CORR := TRUE;
     else
        CHECK RANGE_AND_AZIMUTH_ERROR_CORR;
        if RANGE_AND_AZIMUTH_ERROR_CORR then
           IG_CORR := TRUE;
        end if;
     end if;
   else
     CHECK R_GATE_CORR;
     if R_GATE_CORR then
        IG_CORR := TRUE;
     end if;
    end if;

end INNER_GATE_CORRELATION_2_2_3_1_2;

 procedure OUTER_GATE_CORRELATION_2_2_3_1_3 (in
          TGT_REP_NO; in TRK_NO; in DEV_VEC; out OG_CORR ) is

--   The outer gates shall be oriented parallel to
--and normal with each track's velocity vector. The outer gate
--correlation tests shall be performed by decomposing the
--deviation vector into two orthogonal components.
--   o Ud in a direction perpendicular to the
--        track's heading
--   o Wd in the fore and aft direction.
--   Irrespective of the values given by the
```

```
--formulas shown in the package body an upper bound of 10 DM
--and a lower bound of 1 DM shall be placed on all outer gate
--values. If all tests are passed, the target report
--correlates with the track in the outer gate. If any test
--fails the target report fails to correlate with the track.

OG_CORR : BOOLEAN := FALSE;

begin

  COMPUTE AFT_GATES;
  COMPUTE LATERAL_GATES;
  CHECK LATERAL_GATE_CORR;
  if LATERAL_GATE_CORR then
     CHECK AFT_GATE_CORR;
     if AFT_GATE_CORR then
        OG_CORR := TRUE;
     end if;
  end if;

end OUTER_GATE_CORRELATION_2_2_3_1_3;

procedure NON_AUTO_BEACON_CORRELATION_2_2_3_1_4
    (in TGT_IFF; in TRK_IFF; out
        NON_AUTO_BN_CORR);
--  If the absolute miss distance on grid system
--coordinates is greater than 5 DM, no correlation
--may take place. If the report contains no
--IFF/SIF data, no correlation may take place.
--Otherwise, the report is correlated and the
--correlation score is set to 1.

-- Documentation provided does not support an
-- accurate description, hence none is offered.

end NON_AUTO_BEACON_CORRELATION_2_2_3_1_4;

procedure ALTITUDE_GATE_CORRELATION_2_2_3_1_5
    (in TGE_REP_ALT; in TRK_ALT; out
        ALT_GATE_CORR) is
```

```
--
--  If a target report correlates with a track for
--either the inner or the outer gate, the
--possibility of correlation in altitude shall be
--examined. Each track has an altitude estimate
--associated with it which may be derived from 3D
--measurements, from mode C beacon responses or
--from operator inputs. If there are no altitude
--data, the track shall be assigned a default
--altitude of 20,000 feet. If the correlating
--report contains an altitude measurement, either
--from a 3D radar or mode C, this altitude shall
--be compared to the track altitude.
--

ALT_GATE_CORR : BOOLEAN := FALSE;

begin

   if IG_CORR or OG_CORR then
      if TRK_ALT not ASSIGNED then
         TRK_ALT := 20_000;
         TRK_ALT_MAX := 25_000;
         TRK_ALT_MIN := 15_000;
      end if;
      if TGT_ALT >= TRK_ALT_MAX and <= TRK_ALT_MIN
         then ALT_GATE_CORR := TRUE;
      end if;
   end if;

end ALTITUDE_GATE_CORRELATION_2_2_3_1_5;


procedure CORRELATION_SCORE_2_2_3_1_6
     ( in TGT_REP_NO; in TRK_NO; in POS_CORR; in IFF_CORR;
     in ALT_CORR; out CORR_SCORE);

--  During the correlation process for each track,
--a record shall be kept of the target reports
```

```
    --that correlate with that auto-track, up to a
    --maximum of three reports. Each report that has a
    --position correlation shall be given a
    --correlation score with respect to the track.
    --If more than three reports correlate, the three
    --with the highest correlation scores shall be
    --retained for possible use in the association
    --process. The final correlation score, between 3
    --and 15, as derived from the following criteria,
    --shall be used as a measure of the quality of
    --correlation. In addition, a record shall be
    --maintained of up to three correlating tracks per
    --target report, for use in the association
    --process. There are three separate correlation
    --scores that are used to determine the final
    --correlation score:
    --    o Position correlation
    --    o Altitude correlation
    --    o IFF correlation.

    begin
      COMPUTE POSITION_SCORE;
      COMPUTE ALTITUDE_SCORE;
      COMPUTE IFF_SCORE;
      COMPUTE FINAL_CORR_SCORE;
      LOAD_PAIRING_BUFFER;
    end CORRELATION_SCORE_2_2_3_1_6;
end CORRELATION_ACTIONS_2_2_3_1 ;


procedure ASSOCIATION_2_2_3_2 ;
--    In the association process, each track shall
--be examined to see which of several correlating target
--reports should be associated with the track.
--After all tracks and target reports in a given area
--have been compared for correlation, the best
--target report for each track shall be selected.
--Auto track categories shall be treated
--separately and in the following priority order:
--    o Established
```

```
--    o Provisional
--    o Tenative.
--    The track-to-report pairings noted under
--correlation shall be used in conJunction with the
--final correlation scores to select the most likely
--report-track associations.
--    If a target report associates with a track in
--one category, it shall become inelisble for
--processing in any lower priority category. The
--association process shall resolve all conflicting
--correlations by examining the correlation scores.
--The report with the hishest score shall
--associate with the track. If the hishest two or
--three scores are the same, the pairing with the minimum
--deviation distance shall be associated.
--    For each track, count shall be kept of the number of
--times the track has associated with reports. This
--association count shall a maximum value of 15.
--    Association will operate on a buffer which
--has been loaded by correlation. There will be one
--buffer per sector. Each buffer contains a table
--that links a track record with all possible
--correlating target reports and correlation scores
--and deviation vector values.

begin

   SORT_PAIRINGS_BY_TRACK_CATEGORY ;
   for I in TRACK_CATEGORY loop
      COMPARE_CORRELATION_SCORES ;
      for K in NO_TRKS_IN_CATEGORY loop
         MERGE_CONDITION_TEST ;
         IFF_EMERGENCY_TEST ;
         UPDATE_ASSOCIATION_COUNT ;
      end loop;
    end loop;
    AUTO_INITIATION_NEW_TRACK_2_2_4_1
        (NON_ASSOC_TGT_REP );

 end ASSOCIATION_2_2_3_2 ;
```

```
procedure SMOOTHING_2_2_3_3 (in TGT_REP_POS;
        in TRK_POS; in TRK_VEL; in
        LAST_SMOOTH_TIME; out SMOOTH_POS; out
        SMOOTH_VEL; out SMOOTH_VEL);

  --    The smoothing process shall take place after
  --a target has been associated with a track
  --and shall involve estimating the current
  --track position / velocity based upon previous
  --track position, velocity, the time of last smoothing,
  --and the new measurement information. Because the
  --predicted position represents the recent history of
  --the track, and the target report position represents
  --the best choice of the latest observations, the
  --smoothing process shall employ a weighted average
  --of the two positions and of the previous and
  --indicated velocities. The weighting factors are
  --called smoothing constants.
  --    Several sets of smoothing constants shall
  --be used to accomodate a range of maneuver
  --performance.

  --The smoothing constants will be governed by the
  --following rules:
  --    o for a non-maneuvering track, the
  --      heaviest smoothing shall be used.
  --    o for a maneuvering track, the smoothing
  --      index shall become one step heavier for
  --      recovery gate association and one step
  --      lighter for each outer gate association.
  --    If no association processing was performed
  --for a track, that track shall not be
  --smoothed in that scan.
  --    The smoothing procedure accepts the track
  --to report pairing table generated by association,
  --as new input pairings.

  begin
```

```
            if ASSOC  then
               OBTAIN SMOOTHING_CONSTANTS;
               SMOOTH POSITION;
               SMOOTH VELOCITY;
               SMOOTH ALTITUDE;
            end if;

      end SMOOTHING_2_2_3_3 ;

      procedure PREDICTION_2_2_3_4 (in SMOOTH_POS; in
                SMOOTH_VEL; out PRED_POS);

      --    Once per radar scan, each track shall be
      --dead_reckoned forward to the point where it
      --should next be seen by the radar. This process
      --shall follow the correlation, association,
      --and smoothing processes.
      --    Prediction accepts a track_record
      --from smoothing or from the file of
      --track records that did not associate with any
      --target report.

      begin

        COMPUTE ASSOC_PREDICTED_POSITION;
        COMPUTE NON_ASSOC_PREDICTED_POSITION;

      end PREDICTION_2_2_3_4;

begin --body of TRACK WHILE SCAN

   accept NEW_TGT_BUFFER(TGT_REP_BUFF) do
      OBTAIN SECTORIZED TRACK DATA;
      CORRELATE;
      ASSOCIATE;
      SMOOTH;
      PREDICT;
      for NON_ASSOC and NON_CORREL_TGT_REP loop
          AUTO_INITIATION_NEW_TRACK_2_2_4_1;
      end loop;
```

```
      for NON_ASSOC and NON_CORREL_TGT_REP loop
          DROP_TRCK_2_2_4_2;
      end loop;
   end accept;

end TRACK_WHILE_SCAN_2_2_3 ;

package body TRACK_WHILE_SCAN_UTILITIES_2_2_4 is
  --   Track while scan utilities contains the routines
  --which handle the automatic initiation and dropping
  --of tracks to and from the track storage file as well
  --as a routine which estimates the reliability of the
  --positional data held for any given track, i.e. track
  --quality.

    procedure AUTO_INITIATION_NEW_TRACK_2_2_4_1
      ( NON_ASSOC_TGT_REP : TGT_REP ) is
    --  The automatic initiation capability shall apply
    --to the entire radar coverage area when the system
    --initiation mode is AUTOMATIC. When the initiation
    --mode is MANUAL this capability shall apply only within 5DM
    --of non-auto tracks.
    --   After all possible target report/track
    --associations have been made in a given vicinity, those
    --reports that have failed to associate with any existing
    --track shall be used to generate new auto tracks. A new
    --track initiated in this way shall be placed in the
    --tenative track category. For the purpose of the
    --association count, this first report shall count
    --as an association.
    --   Since a single position measurement provides no
    --velocity estimate, all directions and speeds are
    --equiprobable. However, the upper and lower speed limits
    --for automatically initiated tracks limit the area
    --withhn which subsequent target reports must fall
    --in order to associate with these initial tenative
    --tracks.

    --not further decomposed
```

```
end AUTO_INITIATION_NEW_TRACK_2_2_4_1 ;

task body DROP_TRACK_2_2_4_2 is
--    A RAMIT track shall be dropped if no manual
--update action is taken before the second manual update
--warning time period has elapsed. A remote non-auto track
--shall be dropped if it has received no remote communication
--message update for 60 seconds.
--    Provisional and established tracks shall be
--declared to be in poor tracking status if no association
--can be made for three scans. An established track that
--has been declared in poor tracking status shall be changed
--to provisional with a radar association count of 4. An
--operator indication shall be posted. Provisional tracks
--shall be dropped if no association can be made within 45
--seconds after declaration of poor tracking status,
--except as follows:
--    o Remote auto tracks that fail to associate for
--      three scans shall be switched to remote non-auto.
--    o Any track that is locally engaged shall not be dropped.
--      In this case the track shall be dead-reckoned and drop
--     action shall be attempterd in each succeeding scan.
--    o For all MBDL designated tracks, if the time since the
--      last update is equal to or greater than 45 seconds
--      a drop indication is warranted and posted.

-- not further decomposed

end DROP_TRACK_2_2_4_2 ;

task body DETERMINE_TRACK_QUALITY_2_2_4_3 is
--    Local track quality is an assigned value based
--upon the degree of estimated reliability of the
--positional data. The assigned value for real-time
--tracks shall be limited to range of 1 through 7.
--A track quality of 0 shall indicate a non-real
--time track.
--    The track quality of local auto tracks used for
--ATDL-1 and TADIL-B reporting responsibility shall be:
--    o Initially reported as 4 when a track becomes
```

```
      --      provisional.
      --    o Incremented by one after two successive
      --      associations.
      --    o Decremented by one after two successive
      --      misses.
      --    The track quality of local non-auto tracks
      --(remit) used for ATDL-1 and TADIL-B reporting
      --responsibility shall be:
      --    o Initially reported as 4 when track is
      --      manually initiated.
      --    o Incremented by one as a result of a manual
      --      update action.
      --    o Decremented by one when two successive
      --      scans have occurred without a manual
      --      update action.
      --    The track quality of pop-up tracks used for
      --ATDL-1 and TADIL-B reporting responsibility
      --shall be:
      --    o Initially reported as 4 when a pop-up
      --      track is declared.
      --    o Decremented by one when two successive
      --      scans have occurred without any  fire unit
      --      update.

      -- not further decomposed

      end DETERMINE_TRACK_QUALITY_2_2_4_3 ;

  end TRACK_WHILE_SCAN_UTILITIES_2_2_4 ;

end TRACKING_2_2 ;

package body TRACK_MANAGER_2_3 is
--
--    The track manager provides the capability necessary
--to receive , send and route all intra-system messages
--pertinent to TARGET_CONTROL_2_0. In addition the
--TRACK_MANAGER is responsible for the maintenance
--of all tables unique to the TARGET_CONTROL_2_0
--activity.
```

```
   procedure TRACK_UPDATE_2_3_1 is
   --
   -- Not further decomposed
   --
   end TRACK_UPDATE_2_3_1 ;

   procedure MANAGE_TABLES_2_3_2 is
   --
   -- Not further decomposed
   --
   end MANAGE_TABLES_2_3_2 ;

   procedure DETERMINE_TRACK_STATUS_2_3_3 is
   --
   -- Not further decomposewd
   --
   end DETERMINE_TRACK_STATUS_2_3_3 ;

   procedure DATA_CONTROL_2_3_4 is
   --
   -- Not further decomposed
   --
   end DATA_CONTROL_2_3_4 ;

   procedure OPERATOR_SWITCH_ACTIONS_2_3_5 is
   --
   -- Not further decomposed
   --
   end OPERATOR_SWITCH_ACTIONS_2_3_5 ;

end TRACK_MANAGER_2_3 ;


package body COUNTER_MEASURE_PROCESSING_2_4 is

--    COUNTER_MEASURE_PROCESSING_2_4 provides all
--activities necessary to provide effective
--identification and control for air traffic
--which attempt to avoid detection through use
```

```
--of jamming or chaff techniques.

  procedure JAM_STROBE_PROCESSING_2_4_1 is
  --
  -- Not further decomposed
  --
  end JAM_STROBE_PROCESSING_2_4_1 ;

  procedure CHAFF_PROCESSING_2_4_2 is
  --
  -- Not further decomposed
  --
  end CHAFF_PROCESSING_2_4_2 ;

end COUNTER_MEASURE_PROCESSING_2_4 ;

package body TARGET_ENGAGEMENT_2_5 is
--    The objective of TARGET_ENGAGEMENT_2_5 is to
--evaluate the threat posed to defended points by
--unknown and hostile air traffic. After calculation
--of the threat posed by each track, the fire unit(s)
--best qualified to engage that particular track are
--determined. Upon an engagement order,
--TARGET_ENGAGEMENT_2_5 is responsible for monitoring all
--engagements.

  procedure EVALUATE_THREAT_2_5_1  is
  --    EVALUATE_THREAT_2_5_1 obtains track information
  --on each track and evaluates the threat posed by that
  --track to each defended point. The threat posed by
  --each track once for each scan.
  --
  -- Not further decomposed
  --
  end EVALUATE_THREAT_2_5_1 ;

  procedure EVALUATE_WEAPON_ASSIGNMENT_SCORE_2_5_2 is
  --    EVALUATE_WEAPON_ASSIGNMENT_SCORE_2_5_2  computes
  --a value which is used by the WEAPON_ASSIGNMENT_2_5_3
  --activity to assign a target to the appropriate fire
```

```
      --unit(s).
      --
      -- Not further decomposed
      --
      end EVALUATE_WEAPON_ASSIGNMENT_SCORE_2_5_2 ;

      procedure WEAPON_ASSIGNMENT_2_5_3 is
      --    WEAPON_ASSIGNMENT_2_5_3 automatically assigns the
      --fire unit(s) best qualified to engage hostile air
      --tracks. It also performs bookeeping operations which are
      --necessary to maintain stockpile balancing as well as
      --distinguish between weapons tight and weapons free.
      --In the weapons tight mode assignment of unknown tracks
      --made be accomplished by manual operation only.
      --
      -- Not fdurther decomposed
      --
      end WEAPON_ASSIGNMENT_2_5_3 ;

      procedure MONITOR_ENGAGEMENT_2_5_4 is
      --    MONITOR_ENGAGEMENT_2_5_4 shll provide friendly
      --protection by monitoring the engagement status of
      --friendly and unknown tracks. In addition, it shall
      --monitor engaged tracks relative to safe corridors.
      --Hold fire alerts shall be issued to
      --REMOTE_COMMUNICATION_3_0 if engaged targets are
      --within safe corridors.
      --
      -- Not further decomposed
      --
      end MONITOR_ENGAGEMENT_2_5_4 ;

   end TARGET_ENGAGEMENT_2_5 ;

end TARGET_CONTROL_2_0;
```

```
$ TYPE REMCOM.SDL
package REMOTE COMMUNICATION (3.0) is
--  The objective of the REMOTE COMMUNICATION SUBSYSTEM (3.0)
--is to enable the TSQ-73 to co unicate digitally with
--various remote sites by means of the previously defined
--military protocols,TADIL-B,ATDL-1, and MODIFIED MBDL.
--The REMOTE COMMMUNICATIONS subsystem shall provide for:
--      -2 group data links (ATDL-1)
--      -2 batallion data links (ATDL-1)
--      -1 tactical operations system data link (ATDL-1)
--      -1 air traffic management system data link (ATDL-1)
--      -1 inter-service communication data link (ATDL-1)
--      -4 fire unit data links (modified MBDL)
-- A maximum 11 eleven (11) active data links will be required plus
--on-line redundant data transmission links for each active link.
--Thus, a total of 22 active links plus two spare links
--will be required. Each link will be capable of supporting
--baud rates 300,600,750,1200,2400,4800 or higher.
--  Each protocol will be logically and/or physically
--independent from the operaton of a different protocol.
--The data will be encrypted at the point of transmission
--and decrypted at the point of reception.  For each eight
--bits of data to be transmitted, a hamming code will be
--added to identify multiple bit errors and correct single-
--bit errors.  This hamming code will be verified
--or reported at the point of reception
--where the hamming code will be removed from each eight-
--bit sequence.
--  Upon initial power-up, all data link paths including
--components will be tested and exercised with a basic set
--of diagnostics. Faulty data paths will be identified as well
--as the reason for their fault.  These faults will be
--displayed by means of:
--      -system console
--      -status display panel
--      -remote communication processor(s) front panel indicator
--      -error lamps on the malfunctioning board (processor,
--       controller, interface,modem,etc.) which may be cor-
--       rected by removal and replacement of a spare board.
--  Diagnostic routines will be executed every 5 minutes or
```

```
--upon request by the system operator on all data links avail-
--able but not currently utilized for transmission or reception.
--  A pool of available data links will be utilized by system
--initialization and/or system operator commands to determine
--how many and what protocol type will be used with which data
--links and at which particular baud rate.  Each protocol
--has a variety of safety and redundancy features built-in,
--but additional system requirements to terminate or auto-
--matically activate a link are:
--      -identification of a data link fault
--      -five successive checksum errors
--  On a power fail condition, a lamp on the REMOTE COM-
--MUNICATION processor(s) front panel will be illuminated and
--the power fail timer started.  The REMOTE COMMMUNICATION
--subsystem will retain its status and message contents for a
--minimum of 24 hours.
--  Upon power restoration, an automatic restart procedure will
--be initiated and will include:
--      -notification of the power fail condition to all users-
--       local and remote
--      -resumption of communication services
--      -extinguishment of the power fail lamp on the front panel
--  A number of operator commands will be available to generate
--free-form text messages, test messages, and pre-defined
--system messages, as well as to initiate and terminate specific
--links.  A number of commands will be available to provide
--information as to data links, message buffers, and message
--queues as well as to provide a single-step execution
--capability following the reception or transmission of a
--message.
--  Each message will have dual link transmission and recep-
--tion but upon reception only one valid message copy is
--required for intra-TSQ-73 requirements.  A priority message
--structure will be used for message transmission.  A message
--broadcast capability to all links with a single message
--will be provided.
--  An analysis of the quantity,size, and type of messages
--flowing in and out of the system is not available since
--detailed analyses of other major packages (COMMAND AND
--CONTROL,TARGET CONTROL, and RADAR COMMUNICATION had
```

```
--not been performed.

-------------------------------------------------------------------

package COMMUNICATION SUPERVISOR (3.1) is
--The objectives of the COMMUNICATION SUPERVISOR are:
-- 1.  To provide a controlling interface between COMMAND
--       AND CONTROL, TARGET CONTROL,RADAR COMMUNICATION, and
--       REMOTE COMMUNICATON.
--2.    To provide for the management of the  REMOTE
--      COMMUNICATION package.
 procedure CONSUME INTERNAL MESSAGE (3.1.1);
 procedure INITIALIZE REMOTE COMMUNICATION (3.1.2);
 procedure MANAGE DATA LINKS (3.1.3);
 package MANAGE MESSAGES (3.1.4);
 package MANAGE FAULT DETECTION,CORRECTION AND REPORTING (3.1.5);
 package TERMINATE REMOTE COMMUNICATION (3.1.6);
 procedure PRODUCE INTERNAL MESSAGE (3.1.7);
end COMMUNICATION SUPERVISOR (3.1);

-------------------------------------------------------------------

package body COMMUNICATON SUPERVISOR (3.1) is
 procedure CONSUME INTERNAL MESSAGE (3.1.1) is
--  Utilize the PUT procedure of INTRASYSTEM TRANSFER
 begin
 loop
--  Validate message origination
  if COMMAND AND CONTROL MESSAGE or COMMUNICATION PROTOCOL
   MESSAGE = TRUE then
--Validate message type and route message
   case MESSAGE TYPE is
--Message type 'I'M ALIVE' handled by GET procedure
    when INITIALIZE REMOTE COMMUNICATION =>
     INITIALIZE REMOTE COMMUNICATION (3.1.2);
    when MANAGE DATA LINKS =>
     INITIATE MANAGE DATA LINKS (3.1.3);
    when MANAGE MESSAGES =>
     INITIATE MANAGE MESSAGES (3.1.4);
    when MANAGE FAULT DETECTION, CORRECTION, AND REPORTING =>
```

```
      INITIATE MANAGE FAULT DETECTION, CORRECTION, AND
      REPORTING (3.1.5);
     when TERMINATE REMOTE COMMUNICATION =>
      TERMINATE REMOTE COMMUNICATION (3.1.6);
     when OTHERS =>
      SET ERROR CODE FOR INVALID MESSAGE TYPE;
      PRODUCE INTERNAL MESSAGE (3.1.7);
    end case;
   else
    SET ERROR CODE FOR INVALID MESSAGE ORIGINATOR;
    PRODUCE INTERNAL MESSAGE (3.1.7);
   end if;
   DELAY INTERNAL TIMING REQUIREMENT;
--  The internal timing requirement is undetermined at
--this time.  A rough estimate places this value in the
--50 msec rnge.  The final value will be determined at the
--time of system integration when actual message flow is
--performed.
   end loop;
 end CONSUME INTERNAL MESSAGE (3.1.1);

-------------------------------------------------------------

 procedure INITIALIZE REMOTE COMMUNICATION (3.1.2) is
  procedure HOUSEKEEPING (3.1.2.1) is
  begin
   if POWER FAIL FLAG = TRUE then
    INITIATE AUTOMATIC RESTART;
   else
    CLEAR MESSAGE BUFFERS (3.1.2.1.1);
    CLEAR MESSAGE QUEUES (3.1.2.1.2);
    CLEAR MESSAGE COUNTERS (3.1.2.1.3);
    CLEAR TRANSACTION LOG INDICATOR (3.1.2.1.4);
    CLEAR POWER FAIL INDICATOR (3.1.2.1.5);
   end if;
  end HOUSEKEEPING (3.1.2.1);
--  Initialize communication susbsystem parameters
--  Primary and redundant line addresses
--  Association of routing I.D., Data link addresses, baud
--rate, protocol assignments and transaction log indicator.
```

```
   procedure INITIALIZE COMM SUBSYSTEM PARAMETERS (3.1.2.2) is
   begin
    if NON-DEFAULT PARAMETERS = TRUE then
     LOAD NON-DEFAULT PARAMETERS (3.1.2.2.1);
    else
     LOAD DEFAULT PARAMETERS (3.1.2.2.2);
    end if;
--Initialize communication tables using parameters
    procedure INITIALIZE COMM TABLES PARAMETERS
     (3.1.2.2.3) is
     INITIALIZE DATA LINK TABLE (3.1.2.2.3.1);
     INITIALIZE MESSAGE TYPE TABLE (3.1.2.2.3.2);
     SET TRANSACTION LOG FLAG (3.1.2.2.3.3);
    end INITIALIZE COMM TABLES PARAMETERS (3.1.2.2.3);
    procedure INITIATE MANAGE MESSAGES (3.1.2.3);
    procedure INITIATEMANAGE DATA LINKS (3.1.2.4);
  end INITIALIZE COMM SUBSYSTEM PARAMETERS (3.1.2.2);
 end INITIALIZE REMOTE COMMUNICATION (3.1.2);

-------------------------------------------------------------

 procedure MANAGE DATA LINKS (3.1.3) is
--  The objective of the MANAGE DATA LINKS procedure is to
--manage the data link table and the initialization and
--termination of each data link.
  procedure MANAGE DATA LINK STATUS TABLE (3.1.3.1) is
   function SEARCH DATA LINK STATUS TABLE (3.1.3.1.1);
   function INSERT ENTRY DATA LINK STATUS TABLE (3.1.3.1.2);
   function DELETE ENTRY DATA LINK STATUS TABLE (3.1.3.1.3);
   procedure INITIALIZE LINK (3.1.3.1.4);
   procedure TERMINATE LINK (3.1.3.1.5);
  end MANAGE DATA LINK STATUS TABLE (3.1.3.1);


  procedure MANAGE PROTOCOLS (3.1.3.2) is
  begin
   case PROTOCOL is
    when TADIL-B =>
     INITIATE TADIL-B PROTOCOL SUPERVISOR;
    when ATDL-1 =>
```

```
      INITIATE ATDL-1 PROTOCOL SUPERVISOR;
     when MBDL =>
      INITIATE MBDL PROTOCOL SUPERVISOR;
     when others =>
      SET ERROR CODE FOR INVALID PROTOCOL;
      PRODUCE INTERNAL MESSAGE (3.1.7);
    end case;
  end MANAGE PROTOCOLS (3.1.3.2);

  procedure MONITOR PROTOCOLS (3.1.3.2.1) is
  begin
   if EOT = TRUE then
    TERMINATE DATA LINK (3.1.3.2.2);
    REPORT PROTOCOL STATUS (3.1.3.2.3);
   end if;
  end MONITOR PROTOCOLS (3.1.3.2.1);
  end MANAGE DATA LINKS (3.1.3);

-------------------------------------------------------

 package MANAGE MESSAGES (3.1.4) is
--  MANAGE MESSAGES has been decomposed to reveal
--three activities.
  procedure MANAGE OUTGOING MESSAGES (3.1.4.1);
  procedure MANAGE INCOMING MESSAGES (3.1.4.2);
  procedure MANAGE MESSAGE BUFFERS AND QUEUES
   (3.1.4.3);
 end MANAGE MESSAGES (3.1.4);

 package body MANAGE MESSAGES (3.1.4) is
  procedure MANAGE OUTGOING MESSAGES (3.1.4.1) is
   procedure DETERMINE MSG ACTION (3.1.4.1.1) is
   begin
    SELECT HIGHEST PRIORITY MESSAGE (return MESSAGE)
     (3.1.4.1.1.1);
    procedure REPLICATE MESSAGE is
    begin
     if MULTIPLE MESSAGES = TRUE then
      for I in 1..NUMBER OF MAILBOXES - 1 loop
       GET MESSAGE BUFFER(I);
```

```
        COPY MESSAGE INTO MAILBOX(I);
        PRODUCE INTERNAL MESSAGE;
       end loop;
      end if;
     end REPLICATE MESSAGE;
    end DETERMINE MSG ACTION (3.1.4.1.1);

    procedure POST TRANSMISSION ACTION (3.1.4.1.2) is
    begin
     SELECT HIGHEST PRIORITY MSG (return MSG)
      (3.1.4.1.2.1);
     POST MESSAGE STATUS;
    if TRANSACTION LOG FLAG and ACK REQUIRED = FALSE then
      RELEASE MESSAGE BUFFER;
     end if;
     if TRANSACTION LOG FLAG and ACK REQUIRED = TRUE then
      REPLICATE MESSAGE;
     end if;
     PRODUCE INTERNAL MESSGE;
    end POST TRANSMISSION ACTION (3.1.4.1.2);

    procedure PRE-TRANSMISSION ACTION is
    begin
     function MESSAGE TABLE LOOKUP;
--   Searches message type table and returns message type and
--duplication codes/i.d.'s.
     function DATA LINK TABLE LOOKUP;
--   Searches data link table by destination i.d. and returns
--protocol status, data link availability, and protocol
--mailbox.
     MESSAGE TYPE = MESSAGE TABLE LOOKUP;
     if MESSAGE TYPE /= OUTGOING MESSAGE TYPE then
      SET ERROR CODE FOR INVALID OUTGOING MESSAGE;
      PRODUCE INTERNAL MESSAGE;
     end if;
     LINK = DATA LINK TABLE LOOKUP;
     if PROTOCOL STATUS = INACTIVE then
      SET ERROR CODE FOR INACTIVE PROTOCOL;
      PRODUCE INTERNAL MESSAGE;
     end if;
```

```
    if DATALINK AVAILABILITY = FALSE then
     SET ERROR CODE FOR INACTIVE DATA LINK;
     PRODUCE INTERNAL MESSAGE;
    end if;
    REPLICATE MESSAGE;
    PRODUCE INTERNAL MESSAGE;
   end PRE TRANSMISSION ACTION;

    loop
    if MESSAGE QUEUE COUNT > 0 THEN
     SELECT HIGHEST PRIORITY MESSAGE;
     if POST TRANSMISSION FLAG = TRUE THEN
      POST TRANSMISSION ACTION
     else
      PRE TRANSMISSION ACTION;
     end if;
    end if;
    DELAY INTERNAL TIMING REQUIREMENT;
    end loop;
   end DETERMINE MSG ACTION (3.1.4.1.1);
  end MANAGE OUTGOING MESSAGES (3.1.4.1);

  procedure MANAGE INCOMING MESSAGE (3.1.4.2);
--Not decomposed for validation effort

  procedure MANAGE MESSAGE BUFFERS AND QUEUES (3.1.4.3);
--Not decomposed for validation effort

 end MANAGE MESSAGES (3.1.4);

-------------------------------------------------------

 package MANAGE FAULT DETECTION,CORRECTION AND REPORTING
  (3.1.5) is
  procedure IDENTIFY FAULTS (3.1.5.1);
  procedure UPDATE ERROR COUNTERS (3.1.5.2);
  procedure ATTEMPT CORRECTIVE ACTION (3.1.5.3);
  procedure REPORT FAULTS (3.1.5.4);
 end MANAGE FAULT DETECTION,CORRECTION AND REPORTING (3.1.5);
```

```
package body MANAGE FAULT DETECTION,CORRECTION AND REPORTING
 (3.1.5) is

 procedure IDENTIFY FAULTS (3.1.5.1) is
  procedure IDENTIFY HARDWARE FAULTS (3.1.5.1.1);
  procedure IDENTIFY SOFTWARE FAULTS (3.1.5.1.2);
 end IDENTIFY FAULTS (3.1.5.1);
end MANAGE FAULT DETECTION,CORRECTION AND REPORTING (3.1.5);

-------------------------------------------------------------

 package TERMINATE REMOTE COMMUNICATON (3.1.6) is
  procedure BUILD TERMINATION MESSAGE (3.1.6.1);
  procedure START TERMINATE DATA LINK (3.1.6.2);
  procedure UPDATE DATA LINK STATUS TABLE (3.1.6.3);
  procedure STOP DATA LINK MANAGER (3.1.6.4);
  procedure STOP MESSAGE MANAGER (3.1.6.5);
  procedure UPDATE TRANSACTION LOG (3.1.6.6);
 end TERMINATE REMOTE COMMUNICAION (3.1.6);

-------------------------------------------------------------

 procedure PRODUCE INTERNAL MESSAGE (3.1.7) is
  begin
   if INTERNAL MESSAGE QUEUE NOT EMPTY then
    GET MESSAGE FROM QUEUE (3.1.7.1);
    case INTENRAL ROUTING ID is
     when COMMAND AND CONTROL or TARGET CONTROL  or
      RADAR COMMUNICATION =>
      PUT MESSAGE TO COMMAND AND CONTROL;
     when COMMUNICATION PROTOCOL =>
      PUT MESSAGE TO COMMUNICATION PROTOCOL (3.1.7.2);

     when others =>
      SET ERROR CODE FOR INVALID INTERNAL ROUTING I.D.;
      PUT ERROR CODE TO COMMAND AND CONTROL;
    end case;
   end if;
 end PRODUCE INTERNAL MESSAGE (3.1.7);
```

```
end COMMUNICATION SUPERVISOR (3.1.);

------------------------------------------------------------
------------------------------------------------------------

package INTRASYSTEM TRANSFER (3.2) is
--  The objective of the intrasystem transfer subentity is to
--provide facilities which allow message flow between the
--REMOTE COMMUNICATON and COMMAND AND CONTROL (RADAR COM-
--MUNICATION and TARGET CONTROL) as well as support message
--flow within the REMOTE COMMUNICATION, i.e., between
--the REMOTE COMMUNICATON supervisor and the communicaton
--protocols.  To insure message flow integrity,intra-system
--transfers will employ mailbox concepts with semaphore
--counters and message time stamps to identify potential
--bottlenecks and provide periodic status reports as needed.
--  The INTRASYSTEM TRANSFER package is initiated and
--terminated by COMMAND AND CONTROL.
--  The INTRASYSTEM TRANSFER package is intended to operate
--concurrently with various other operations and is itself
--comprised of a MESSAGE POSTING TIMER procedure and two other
--procedures, GET INTRASYSTEM TRANSFER and PUT INTRASYSTEM
--TRANSFER.
--  The followins descriptions assume that the COMMAND
--AND CONTROL package has seneric mailbox features which
--utilize the semaphore concept.  The initialization
--function of COMMAND AND CONTROL will initialize the
--proper number and type of mailboxes necessary to
--implement this activity.
--  Mailboxes are producer-consumer points, the number
--of which is determined at initialization time by the
--number of protocols, the communication supervisor,
--and the COMMAND AND CONTROL package.
 procedure GET INTRASYSTEM TRANSFER (3.2.1);
 procedure PUT INTRASYSTEM TRANSFER (3.2.2);
 procedure MESSAGE POSTING TIMER (3.2.3);
end INTRASYSTEM TRANSFERS (3.2);

------------------------------------------------------------
```

```
package body INTRASYSTEM TRANSFER (3.2) is
 procedure GET INTRASYSTEM TRANSFER (3.2.1) is
--  The set and put features of the INTRASYSTEM TRANSFER
--package may be implemented in two ways.  The first method
--is to have each mailbox employ its own set and put
--procedures.  The second method is separate procedures
--for set and put with multiple mailboxes - a logical
--multiplexer.
  begin
   for I in I..NUMBER OF MAILBOXES loop
    TIMEOUT FLAG(I) = FALSE;
   end loop;
   loop
    for I in I..NUMBER OF MAILBOXES loop
     if GET MESSAGE BUFFER = EMPTY then
      if TIMEOUT FLAG(I) = TRUE then
       ISSUE PRODUCER ERROR;
       TIMEOUT FLAG(I) = FALSE;
      else
       TIMEOUT FLAG(I) = TRUE;
      end if;
     else
      TIMEOUT FLAG(I) = FALSE;
      GET MESSAGE FROM BUFFER(I);
      UPDATE TIME STAMP QUEUE(I);
--Determine message type
      if MESSAGE = 'I'M ALIVE' then
       DISCARD MESSAGE;
      else
       MESSAGE AVAILABLE FOR PROCESSING;
      end if;
     end if;
     DELAY INTERNAL TIMING REQUIREMENT;
    end loop;
 end GET INTRASYSTEM TRANSFER (3.2.1);

-------------------------------------------------------------

 procedure PUT INTRASYSTEM TRANSFER (3.2.2) is
--  The set and put features of the INTRASYSTEM TRANSFER
```

```
--procedure may be implemented in two ways.  The first method is
--to have each mailbox employ its own set and put procedures.
--The second method is separate procedure for set and put with
--multiple mailboxes - a logical multiplexer.
  procedure TEST BUFFER (DESTINATION) is
  begin
   if DESTINATION BUFFER ='NOT EMPTY then
    if DESTINATION TIMEOUT FLAG = TRUE then
     ISSUE CONSUMER ERROR;
     DESTINATION TIMEOUT FLAG = FALSE;
    else
     DESTINATION TIMEOUT FLAG = TRUE;
    end if;
   else
    PUT MESSAGE IN DESTINATION MAILBOX;
    DESTINATION TIMEOUT FLAG = FALSE;
   end if;
  end TEST BUFFER;
  begin
   loop
    ACCEPT MESSAGE (MESSAGE);
    DETERMINE DESTINATION; --FROM MESSAGE CONTENT
    case DESTINATION is
     when COMMAND AND CONTROL or TARGET CONTROL or
      RADAR COMMUNICATION =>
      TEST BUFFER (COMMAND AND CONTROL);
     when COMMUNICATION SUPERVISOR =>
      TEST BUFFER (COMMUNICATION SUPERVISOR);
     when TADIL-B PROTOCOL=>
      TEST BUFFER (TADIL-B PROTOCOL);
     when ATDL-1 PROTOCOL =>
      TEST BUFFER (ATDL-1 PROTOCOL);
     when MBDL PROTOCOL =>
      TEST BUFFER (MBDL PROTOCOL);
    end case;
    DELAY INTERNAL TIMING REQUIREMENT;
   end loop
 end PUT INTRASYSTEM TRANSFER (3.2.2);
```

```
procedure MESSAGE POSTING TIMER (3.2.3) is
 procedure CONSUMER TIMER (3.2.3.1);
 begin
  loop
   for I in I..NUMBER OF MAILBOXES loop
    if GET MESSAGE BUFFER(I) = NOT EMPTY then
     if CURRENT TIME - TIME STAMP(I) >
      INTERNAL TIMING REQUIREMENT then
      ISSUE TIMEOUT ERROR(I);
     end if;
     if SEMAPHORE COUNTER(I) > MAX MESSAGES OR SEC then
      ISSUE SEMAPHORE ERROR(I);
     end if;
    end if;
   end loop;
   DELAY INTERNAL TIMING REQUIREMENT;
  end loop;
 end CONSUMER TIMER (3.2.3.1);

 procedure PRODUCER TIMER (3.2.3.2) is
 begin
  loop
   for I in I..NUMBER OF MAILBOXES loop
    if PUT MESSAGE BUFFER(I) = EMPTY then
     if CURRENT TIME - TIME STAMP >
      INTERNAL TIME REQUIREMENT then
      BUILD 'I'M ALIVE' MESSAGE;
      PUT 'I'M ALIVE' MESSAGE IN MAILBOX(I);
     end if;
    elseif
     SEMAPHORE COUNTER(I) > MAX MESSAGES OR SEC then
      ISSUE SEMAPHORE ERROR;
     end if;
   end loop;
   DELAY INTERNAL TIMING REQUIREMENT;
  end loop;
 end PRODUCER TIMER (3.2.3.2);
--  Additional details (queue management, buffer management, inter-
--ernal routing table, etc.) are not included due to project
```

```
--schedule restrictions.
 end MESSAGE POSTING TIMER (3.2.3);
end INTRASYSTEM TRANSFERS (3.2);

------------------------------------------------------------------
------------------------------------------------------------------

package COMMUNICATION PROTOCOL (3.3) is
--  The objective of the COMMUNICATION PROTOCOL package is to
--handle the physical level of the transmission and reception
--of digital messages according to the predefined military
--protocols TADIL-B,ATDL-1, and modified MBDL.
--  The COMMUNICAION PROTOCOL package is initiated and
--terminated by COMMUNICATION SUPERVISOR (3.1).
 procedure TADIL-B SUPERVISOR (3.3.1);
--NOT INVESTIGATED
 procedure ATDL-1 SUPERVISOR (3.3.2);
--NOT INVESTIGATED
 procedure MBDL PROTOCOL SUPERVISOR (3.3.3);
--  MBDL PROTOCOL SUPERVISOR is a bisynchronous protocol
--utilizing full duplex data lines with point-to-point
--transmission.
  procedure COMSUME COMMUNICATION SUPERVISOR MESSAGE (3.3.3.1);
  procedure INITIALIZE DATA LINE (3.3.3.2);
  procedure TRANSMIT MESSAGE (3.3.3.3);
  procedure RECEIVE MESSAGE (3.3.3.4);
  package PROTOCOL UTILITIES (3.3.3.5);
  procedure TERMINATE DATA LINK (3.3.3.6);
  procedure PRODUCE COMMUNICATION SUPERVISOR MESSAGE (3.3.3.7);
 end MBDL PROTOCOL SUPERVISOR (3.3.3);
end COMMUNICATON PROTOCOL (3.3);

------------------------------------------------------------------

package body COMMUNICATON PROTOCOL (3.3) is
 procedure MBDL PROTOCOL SUPERVISOR (3.3.3) is
   procedure CONSUME COMMUNICATION SUPERVISOR MESSAGE (3.3.3.1)
    is
   begin
    loop
```

```
--Utilize the GET procedure of INTRASYSTEM TRANSFER
--validate message originator
     if COMM SUPERVISOR MESSAGE = TRUE then
--validate message type and routes message
      case MESSAGE TYPE is--message type 'I'M ALIVE' handled by
--     GET procedure
       when INITIALIZE MESSAGE TYPE => QUEUE TO
        INITIALIZE DATA LINK (3.3.3.2);
       when TRANSMIT MESSAGE => QUEUE TO TRANSMIT
        MESSAGE (3.3.3.3);
       when UTILITY MESSAGE TYPE => QUEUE TO
        PROTOCOL UTILITIES (3.3.3.5);
       when TERMINATE MESSAGE TYPE => QUEUE TO
        TERMINATE DATA LINK (3.3.3.6);
       when others =>
        SET ERROR CODE FOR INVALID MESSAGE TYPE;
        QUEUE TO PRODUCE COMM SUPERVISOR MSG (3.3.3.7);
      end case;
     else
      SET ERROR CODE FOR INVALID MESSAGE ORIGINATOR;
      QUEUE TO PRODUCE COMM SUPERVISOR MSG (3.3.3.7);
     end if;
     DELAY INTERNAL TIMING REQUIREMENT;
    end loop;
   end CONSUME COMMUNICATION SUPERVISOR MESSGE (3.3.3.1);

-----------------------------------------------------------------

  procedure INITIALIZE DATA LINK (3.3.3.2) is
   procedure CONNECT DATA LINK (3.3.3.2.1) is
   begin
    START DATA LINK SELF TEST (3.3.3.2.1.1);
    START INITIALIZE LINE (3.3.3.2.1.2);
    START LINE SUPERVISOR (3.3.3.2.1.3);
    START DATA LINK LOOPBACK TEST (3.3.3.2.1.4);
   end CONNECT DATA LINK (3.3.3.2.1);
   procedure START PROTOCOL FOR DATA LINK (3.3.3.2.2);
   procedure TRANSMIT INITIALIZATION MESSAGE (3.3.3.2.3);
   procedure VALIDATE DATA LINK PATH (3.3.3.2.4);
   procedure QUEUE DATA LINK AND PROT STAT TO PROD COMM SUPER MSG
```

```
    (3.3.3.2.5);
  end INITIALIZE DATA LINK (3.3.3.2);

-------------------------------------------------------------

  procedure TRANSMIT MESSAGE (3.3.3.3) is
    procedure ENCODE MESSAGE TO PROTOCOL (3.3.3.3.1) is
    begin
     loop
      procedure SELECT MESSAGE FOR ENCODING (3.3.3.3.1.1) is
      begin
       if MESSAGE QUEUE COUNT > ENCODED MESSAGE QUEUE COUNT
        then
        SELECT HIGHEST PRIORITY MESSGE (3.3.3.3.1.2);
        ENCODE MESSAGE HEADER (3.3.3.3.1.3);
        ENCODE MESSAGE TEXT (3.3.3.3.1.4);
        if TEXT FORM = PREDEFINED FORM then
         ENCODE PREDEFINED TEXT;
        else
         ENCODE FREE FORM TEXT;
        end if;
        ENCODE MESSAGE TRAILER (3.3.3.3.1.5);
--Checksum calculations if not calculated in transmit/receive
--character.
        ASSEMBLE MESSAGE FRAME (3.3.3.3.1.6);
        UPDATE MESSAGE TRANS AVAIL (3.3.3.3.1.7);
        UPDATE ENCODED MESSAGE QUEUE COUNT (3.3.3.3.1.8);
       end if;
       DELAY INTERNAL TIMING REQUIREMENT;
     end SELECT MESSAGE FOR ENCODING (3.3.3.3.1.1);
    end loop;
   end ENCODE MESSAGE TO PROTOCOL (3.3.3.3.1);

   procedure SEND MESSAGE (3.3.3.3.2);
    begin
     procedure SELECT MESSAGE FOR TRANSMISSION (3.3.3.3.2.1) is
     begin
      loop
       if ENCODED MESSAGE QUEUE > 0 then
        SELECT HIGHEST PRIORITY MESSAGE (3.3.3.3.2.1.1);
```

```
--Select highest priority message is the same as 3.3.3.3.1.1;
--but is repeated here because this procedure may be executing
--on a different processor.
        DATA LINK TABLE LOOKUP (3.3.3.3.2.1.2);
--Searches the data link table by destination i.d. and returns
--physical line numbers for both primary and redundant lines.
        if PRIMARY LINE NUMBER and REDUNDANT LINE NUMBER
         NOT VALID then
         SET ERROR CODE (INVALID OR INACTIVE) FOR BOTH LINES;
         QUEUE TO PRODUCE COMM SUPERVISOR MSG (3.3.3.7);
        else if
         PRIMARY LINE NUMBER and REDUNDANT LINE NUMBER VALID
          then
          INSERT PRIMARY LINE NUMBER INTO MESSAGE;
          INSERT REDUNDANT LINE INTO MESSAGE;
          QUEUE MSG TO POST MSG TO LINE (3.3.3.3.2.1.3);
        else if
          PRIMARY LINE VALID then
           INSERT PRIMARY LINE NUMBER INTO MESSAGE;
           SET ERROR CODE (INVALID OR INACTIVE) FOR PRIMARY
            LINE;
           QUEUE MSG TO POST MSG TO LINE (3.3.3.3.2.1.3);
        .else
          INSERT REDUNDANT LINE NUMBER INTO MESSAGE;
          SET ERROR CODE (INVALID OR INACTIVE) FOR PRIMARY LINE;
          QUEUE MSG TO POST MSG TO LINE (3.3.3.3.2.1.3);
         end if;
        end if;
       DELAY INTERNAL TIMING REQUIREMENT;
      end loop;

    procedure POST MESSAGE TO LINE (3.3.3.3.2.1.3) is
    begin
     if INITIALIZATION MESSAGE then
      POST MESSAGE TO LINE(I);
--   Where I is the physical line number
      POSTING MESSAGE TIMER (3.3.3.3.2.1.3.3);
     end if;
     procedure POST MESSAGE TO PRIMARY LINE (3.3.3.3.2.1.3.1) is
     begin
```

```
      loop
       if PRIMARY LINE(I) AVAILABLE and ENCODED MESSAGE(I)
        AVAILABLE then
        POST MESSAGE TO LINE(I);
       else
        DELAY UPON PRIMARY LINE(I) AVAILABILITY;
        ENCODED MESSAGE(I) AVAILABILITY;
       end if;
      end loop;
     end POST MESSAGE TO PRIMARY LINE (3.3.3.3.2.1.3.1);

     procedure POST MESSAGE TO REDUNDANT LINE (3.3.3.3.2.1.3.2);
     begin
      loop
       IF REDUNDANT LINE(I) AVAILABLE and ENCODED MESSAGE(I)
        AVAILABLE then
        POST MESSAGE TO LINE(I);
       else
        DELAY UPON REDUNDANT LINE(I) AVAILABILITY or
         ENCODED MESSAGE(I) AVAILABILITY;
       end if;
      end loop;
     end POST MESSAGE TO REDUNDANT LINE (3.3.3.3.2.1.3.2);

     procedure POSTING MESSAGE TIMER (3.3.3.3.2.1.3.3);
--   The objective of the POSTING MESSAGE TIMER  is to issue
--(set error code and queue to comm supervisor task) a posting
--time-out error.  This is when a message cannot be posted to
--a specific active line within a specific time interval which
--is in relation (<=>) to the interval timing requirement.  This
--is not investigated further.
    end SELECT MESSAGE FOR TRANSMISSION (3.3.3.3.2.1);


-------------------------------------------------------------

    procedure LINE PROTOCOL MANAGER (3.3.3.3.2.2) is
     procedure PROTOCOL REQUIREMENTS (3.3.3.3.2.2.1) is
     begin
      procedure DISASSEMBLE MESSAGE FRAME (3.3.3.3.2.2.1.1);
```

```
      procedure SUPERVISOR LINE (3.3.3.3.2.2.1.2);
      procedure LINE I/O DRIVER (3.3.3.3.2.2.1.3) is
       INITIALIZE LINE (3.3.3.3.2.2.1.3.1);
       procedure TRANSMIT CHARACTER (3.3.3.3.2.2.1.3.2) is
        begin
         CHECKSUM CALCULATION (3.3.3.3.2.2.1.3.2.1);
         ENCRYPT CHARACTER (3.3.3.3.2.2.1.3.2.2);
         CALC CHAR HAM CODE (3.3.3.3.2.2.1.3.2.4);
         CHARACTER TIME OUT (3.3.3.3.2.2.1.3.2.4);
        end TRANSMIT CHARACTER (3.3.3.3.2.2.1.3.2);
       procedure RECEIVE CHARACTER (3.3.3.3.2.2.1.3.3) is
       begin
        VERIFY CHAR HAMM CODE (3.3.3.3.2.2.1.3.3.1);
        DECRYPT CHARACTER (3.3.3.3.2.2.1.3.3.2);
        CHECKSUM CALCULATION (3.3.3.3.2.2.1.3.3.3);
        CHARACTER TIME OUT (3.3.3.3.2.2.1.3.3.4);
       end RECEIVE CHARACTER (3.3.3.3.2.2.1.3.3);
       TERMINATE LINE (3.3.3.3.2.2.1.3.4);
       REAL LINE STATUS (3.3.3.3.2.2.1.3.5);
       LOOP BACK TEST (3.3.3.3.2.2.1.3.6);
       SELF TEST (3.3.3.3.2.2.1.3.7);
       AUTOMATIC DIAL (3.3.3.3.2.2.1.3.8);
       AUTOMATIC ANSWER (3.3.3.3.2.2.1.3.9);
       EXCEPTION HANDLER (3.3.3.3.2.2.1.3.10);
      end LINE I/O DIRVER (3.3.3.3.2.2.1.3);
      procedure MONITOR DISPLAY MODE (3.3.3.3.2.2.1.4);
--Use to display character transmission to a terminal
      begin
       TERMINAL COMMANDS (3.3.3.3.2.2.1.4.1);
       TERMINAL I/O DRIVER (3.3.3.3.2.2.1.4.2);
      end MONITOR DISPLAY MODE (3.3.3.3.2.2.1.4);
      REPORT MESSAGE TRANSMISSION STATUS (3.3.3.3.2.2.1.5);
     end PROTOCOL REQUIREMENTS (3.3.3.3.2.2.1);
    end LINE PROTOCOL MANAGER (3.3.3.3.2.2);
   end SEND MESSAGE (3.3.3.3.2);

-------------------------------------------------------------

  procedure RECEIVE MESSAGE (3.3.3.4) is
--not decomposed for validation effort
```

```
  end RECEIVE MESSAGE (3.3.3.4);

-------------------------------------------------------------

  package PROTOCOL UTILITIES (3.3.3.5) is
--  The objective is to provide utilities (display,list,modify)
--capabilities into the communication protocol.  These utilities
--will not be decomposed further but are listed for completeness
   package BUFFER UTILITIES (3.3.3.5.1);
   package DATA LINK TABLE UTILITIES (3.3.3.5.2);
   package MAILBOX UTILTIES (3.3.3.5.3);
   SEMAPHORE UTILITES (3.3.3.5.4);
   TASK PROCEDURE UTILITIES (3.3.3.5.5);
  end PROTOCOL UTILITIES (3.3.3.5);

-------------------------------------------------------------

  procedure TERMINATE DATA LINK (3.3.3.6) is
--Not decomposed for validation effort
   procedure TRANSMIT TERMINATION MESSAGE (3.3.3.6.1);
   procedure HALT PROTOCOL FOR DATA LINK (3.3.3.6.2);
   procedure DISCONNECT DATA LINK (3.3.3.6.3);
   procedure QUEUE DATA LINK AND PROT STAT TO PRODUCE COMM SUPER
    MSG (3.3.3.3.6.4);
  end TERMINATE DATA LINK (3.3.3.6);

-------------------------------------------------------------

  procedure PRODUCE COMM SUPERVISOR MESSAGE (3.3.3.7);
  begin
   loop
    DEQUEUES MESSAGE FROM COMM SUPERVISOR QUEUES (3.3.3.7.1);
    PUT MESSAGE (3.3.3.7.2);
--Utilizes the put task of INTRASYSTEM TRANSFER
   end loop;
   DELAY INTERNAL TIMING REQUIREMENT;
  end PRODUCE COMM SUPERVISOR MESSAGE (3.3.3.7);
 end MBDL PROTOCOL SUPERVISOR (3.3.3);
end COMMUNICATION PROTOCOL (3.3);
```

```
end REMOTE COMMUNICATION (3.0);
```

```
$ TYPE RADAR.SDL
package RADAR COMMUNICATION (4.0) is
--   RADAR COMMUNICATION will execute concurrently with
--REMOTE COMMUNICATIONS, COMMAND & CONTROL and TARGET
--CONTROL packages.
--   The objective of the radar communication subsystem is to
--enable the TSQ_73 to communicate digitally (analog
--conversion will be performed by the radar sites)
--with various local and remote sites by means
--of the previously defined military radar interface
--(ATDL-1).
--  The radar communication subsystem shall provide
--ATDL-1 data links to four (4) radar sites.
--  A maximum of four (4) active full duplex data links will be
--required plus on-line redundant data transmission links
--for each active link.  Thus, a total of eight (8) active links
--plus a minimum of two spare links will be required.  Each
--link will be capable of supporting baud rates of 300, 600,
--750, 1200, 1500, 2400, 4800, 9600 or higher.
--  The data will be encrypted at the point of transmission
--and decrypted at the point of reception.  For each eight
--bits of data to be transmitted, a hamming code will be
--added to identify multiple-bit errors and correct single-
--bit errors. This hamming code will be verified or reported
--at the point of reception and the hamming code will be
--removed from each eight bit sequence.
--  Upon initial power-up, all data link paths including
--components will be tested and exercised with a basic set
--of diagnostics.  Faulty data paths will be identified as
--well as the reason for their fault.  These faults will be
--displayed by means of:
--  -system console
--  -status display panel
--  -radar communications processor(s) front panel indicator
--  -error lamps on the malfunctioning board (processor,
--   controller, interface, modem, etc.) which may be
--   corrected by removal and replacement of a spare board.
--  Diagnostic routines will be executed every five minutes
--or upon request by the system operator on all data links
--available but not currently utilized for transmission or
```

```
--reception.
--  A pool of available data links will be utilized by
--system initialization parameters and/or system operator
--commands to determine how many will be used with which
--data links and at which particular baud rate.  Radar
--interface protocols have a variety of safety and
--redundancy features built-in, but additional system
--requirements to terminate or automatically activate a
--data link are:
--  -identification of a data link fault.
--  -five successive checksum errors.
--  Upon a power fail condition, a lamp/indicator on the
--radar communications processor(s) front panel and the
--status display panel will be illuminated and the power
--fail timer initialized and started. The radar communication
--subsystem will retain its status and message contents for
--a minimum of 24 hours.
--  Upon power restoration, an automatic restart pro-
--cedure will be initiated which will include:
--  -Power-fail duration of less than one second
--   o  Notification of the power fail condition and
--      duration to the system console device.
--   o  Resumption of communication services.
--   o  Extinguish the power fail lamp/indicator on
--      the processor front panel and the status
--      display panel.
--  -Power-fail duration of greater than one second
--   and less than the radar scan time of 360 degrees
--   (3, 6 or 10 seconds).
--   o  Notification of the power fail condition and
--      duration to the system console device.
--   o  Notification of the power fail condition and
--      duration to target control.
--   o  Resumption of communication services.
--   o  Extinguish the power fail lamp/indicator on
--      the processor front panel and the status
--      display panel.
--  -Power fail duration greater than the radar scan
--   of 360 degrees  (3, 6, or 10 seconds).
--   o  Notification of the power fail condition and
```

```
--       duration to the system console device.
--    o  Notification of the power fail condition and
--       duration to target control.
--    o  Notification of the power fail condition and
--       duration to all users - local and remote.
--    o  A deletion of all existing (in-bound) XY
--       coordinate messages.
--    o  Resumption of communication services.
--    o  Extinguish the power fail lamp/indicator on
--       the processor front panel and the status
--       display panel.
--  A number of operator commands will be available to
--generate free-form text messages, test messages and
--predefined system messages as well as initiate and
--terminate specific data links.  A number of commands
--will be available to provide information as to data
--links, message buffers and message queues as well as
--provide a single-step execution capability of following
--the reception or transmission of a message.
--  Each message will have dual link transmission and
--reception but upon reception only one valid message
--copy is required for intra-TSQ-73 requirements.  A
--priority message structure will be used for message
--transmissions.  A message broadcast capability to
--all data links of a single message will be provided.
--  The various radar units digitally communicating
--to the TSQ-73 are capable of providing a 360 degree
--scan every three, six or ten seconds.  A 360 degree
--scan is currently divided into twenty 18 degree  sectors.
--Variable sector sizes are desired to support efficient
--target control processing.  For each sector, the radar
--sites will transmit to the TSQ-73:
--  -one XY coordinate message whether a target is
--   reported or not.
--  -one XY coordinate message for every target
--   reported.
--  -a maximum of 96 XY coordinate messages per
--   360 degree  scan.
--Approximate XY coordinate message size is 20 bytes in-
--cluding header and checksum.
```

```
--  The TSQ-73 free-form text messages, radar command
--(requests) messages are a variable length message but
--will have a minimum of 20 bytes and a maximum of 64
--bytes including header and checksum.  A maximum of
--one free-form message per display console operator
--will be generated per three seconds.  The radar
--units are capable of processing free-form text messages
--and request messages.
--  An analysis of the quanity, size and type of message
--transmitted between the TSQ-73 and radar sites is as
--follows:
--   maximum messages per scan and second.
--   max. mss. = max XY coor mss per scan x max active data
--    links + (free form mss per scan x max display
--    console operators x 2 (primary and redundant
--    links).
--   ------------------------------------------------
--       min. scan time (seconds)
--   max. mss = 96 x 8 + (1 x 8 x 2) = 784
--              -------------------   ---
--                        3            3
--   max. mss. per scan = 784 mss.
--   max mss per second = 262 mss.
-- NOTE: The volume of text/cmd. messages from radar sites
--       to the TSQ-73 is insignificant - less than 1%.
--   maximum characters per scan and second per data link.
--   max. char. = max. XY mss. per scan x char. per mss
--    + (text/cmd mss. per scan x max. display console
--    operators x char. per mss.
--    ------------------------------------------------
--         min. scan time (seconds)
--    max. char. = 96 x 20 + (1 x 8 x 64) = 2432
--                 ----------------------   ----
--                           3                3
--    max. char. per scan per link = 2432 char.
--    max. char. per second per link = 811 char.
--  The radar communication subsystem shall be blocked and
--deblocked XY coordinate messages to and from the TSQ-73
--internal message size for intrasystems transfer with
--target control.  XY coordinate messages shall be deblocked
```

```
--for transmission and reception to and from the radar
--sites.


type RADAR_COMMUNICATION_CAPACITIES_PERFORMANCE is
   record
--this indicates record maximum required capacities for
--hardware/software tradeoff decisions.
     NO_OF_DEGRESS_PER_SCAN:constant:=360;
     NO_OF_DEGREES_PER_SECTOR:constant:=18;
     NO_OF_SECTORS_PER_SCAN:=NO_OF_DEGREES_PER_SCAN/NO_OF_
       DEGREES_PER_SECTOR;
     MAX_NO_OF_DISPLAY_CONSOLE_OPERATORS:constant:=8
     MAX_FREE_FORM_MSG_SIZE:constant:=64;
     AVG_FREE_FORM_MSG_SIZE:constant:=42;
     MIN_FREE_FORM_MSG_SIZE:constant:=20;
     XY_COOR_MSG_SIZE:constant:=20;
     MAX_XY_COOR_MSG_PER_SCAN:constant:=96;
     MAX_FREE_FORM_MSG_PER_SEC:constant:=5.334
     MAX_MSG_PER_SCAN:constant:=784;
     MAX_MSG_PER_SEC:constant:=262;
     MAX_CHAR_PER_SEC_PER_LINK:constant:=811;
     MAX_NO_OF_TASKS:constant:=20;
--estimated number of tasks capacity required at this time.
--(Does not include I/O handler tasks). A more
--accurate number will be provided later in the development
--process.
     ONE_SECOND:constant:=1.0;
     INTERNAL_TIMING_REQUIREMENT:constant:=0.010;
--internal timing requirement is measured in milliseconds
     MIN_NO_OF_TASK_EXEC_PER_SEC:=ONE_SECOND/
       INTERNAL_TIMING_REQUIREMENT;
     MIN_NO_OF_EXEC_PER_SEC_PER_TASK:=MIN_NO_OF_TASK_EXEC_
       PER_SEC/MAX_NO_OF_TASKS;
     MAX_DATA_LINK:constant:=10;
     MAX_ACTIVE_DATA_LINK:constant:=8;
     MAX_RADAR_SITES:constant:=4;
--there is a ratio of two data links per radar site
--plus two data link spares
   end record;
```

```
type SCAN_RATE is (SEC3,SEC6,SEC10);
type BAUD_RATE is (B300,B600,B750,B1200,B1500,B2400,B4800,B9600);
type PROTOCOL is (TADIL_B,ATDL_1,MBDL);
type ORIENTATION is (BIT_ORIENTED,CHAR_ORIENTED);
type TRANSMISSION is (PT_TO_PT,MULTI_DROP);
type LINE is (FULL_DUPLEX,HALF_DUPLEX);
type MSG_TYPE is (FREE_FORM_MSG,XY_COOR_MSG);
type SYNC is (SYNCHRONOUS,ASYNCHRONOUS);
type MODE is (PRIMARY,REDUNDANT);

type RADAR_DATA_LINK_SPEC is
   record
     DATA_LINK_BAUD_RATE:BAUD_RATE;
     RADAR_PROTOCOL:PROTOCOL:=ATDL_1;
     RADAR_ORIENTATION:ORIENTATION:=BIT_ORIENTED;
     RADAR_TRANSMISSION:TRANSMISSION:=PT_TO_PT;
     RADAR_LINE:LINE:=FULL_DUPLEX;
     RADAR_SYNC:SYNC:=ASYNCHRONOUS;
     RADAR_MODE:MODE;
     case MSG_TYPE is
      when FREE_FORM_MSG=>
       RADAR_REQ_MSG_BUFFER:string (1..
        MAX_FREE_FORM_MSG_SIZE);
      when XY_COOR_MSG =>
       RADAR_REQ_MSG_BUFFER:string (1..
        XY_COOR_MSG_SIZE);
     end case;
   end record;

PRIMARY_DATA_LINK:DATA_LINK_SPEC range 1..MAX_RADAR_SITES;
REDUNDANT_DATA_LINK:DATA_LINK_SPEC range 1..MAX_RADAR_SITES;

type RADAR_CONFIGURATION is
    record
-- This is where the hardware configuration (if any) would
--be described along with minimum execution characteristics
--for radar communications.  The record would include the
--following:
-- CPU: characteristics (instruction execution speed, type
```

```
--      instruction mix)
-- DATA CHANNELS: number & type
-- MEMORY: size & type
-- INTERRUPTS: number & type
-- CLOCKS,TIMERS,DATE DEVICES
-- INPUT/OUTPUT INTERFACES: number & types
-- REDUNDANCY CHARACTERISTICS
-- PERIPHERALS: number,types & characteristics
    end record;


-------------------------------------------------------------------


package COMMUNICATION SUPERVISOR (4.1) is
--The objectives of the radar communication supervisor are:
--1. To provide a controlling interface between COMMAND &
--   CONTROL,TARGET CONTROL and REMOTE COMMUNICATION.
--2. To provide for the management of the radar
--   communication itself.

  procedure CONSUME INTERNAL MESSAGE (4.1.1);
  procedure INITIALIZE RADAR COMMUNICATION (4.1.2);
  package MANAGE DATA LINKS (4.1.3);
  package MANAGE MESSAGES (4.1.4);
  package MANAGE FAULT DETECTION,CORRECTION AND REPORTING
          (4.1.5);
  procedure TERMINATE REMOTE COMMUNICATION (4.1.6);
  procedure PRODUCE INTERNAL MESSAGE (4.1.7);
end COMMUNICATION SUPERVISOR;


-------------------------------------------------------------------


package body COMMUNICATION SUPERVISOR (4.1) is

--Prior to initialization of RADAR COMMUNICATION it is
--assumed that:
--1.  Preliminary diagnostic routines have been performed to
--    identify and report various hardware faults.
```

```
--2.  Initiation of the power-fail monitor,intra-system
--    (message) transfers and the radar communication
--    supervisor have been performed by COMMAND AND CONTROL.
  procedure CONSUME INTERNAL MESSAGE (4.1.1) is

  begin
    loop
      while INTERNAL TIMING REQUIREMENT>0 loop
        if MAILBOX COUNT <=0 then
          exit loop;
        else get INTERNAL MESSAGE;
--utilizes the GET task of INTRASYSTEM TRANSFERS
--validate internal message orisinator
          if COMMAND AND CONTROL or TARGET CONTROL INTERNAL
            MESSAGE = TRUE then
--validate internal message type and queue message
              case INTERNAL MESSAGE TYPE is
--message type 'I'M ALIVE' is handled by GET TASK
                when INITIALIZE RADAR COMM >= queue to
                  INITIALIZE RADAR COMMUNICATION (4.1.2);
                when MANAGE DATA LINKS=>queue to MANAGE DATA
                  LINKS (4.1.3);
                when MANAGE MESSAGE=>queue to MANAGE
                  MESSAGES (4.1.4);
                when MANAGE FAULT DETECTION,CORRECTION
                  AND REPORTING=>queue to MANAGE FAULT
                  DETECTION,CORRECTION AND REPORTING (4.1.5);
                when TERMINATE RADAR COMMUNICATION=>queue to
                  TERMINATE RADAR COMMUNICATION (4.1.6);
                when others=>
                  SET ERROR CODE FOR INVALID MESSAGE TYPE;
                  queue to PRODUCE INTERNAL MESSAGE (4.1.7);
              end case;
            else
              SET ERROR CODE FOR INVALID INTERNAL MESSAGE
                ORIGINATOR;
              queue to PRODUCE INTERNAL MESSAGE (4.1.7);
          end if;
        end if;
      end loop;
```

```
      DELAY INTERNAL TIMING REQUIREMENTS * NUMBER OF TASKS;
     end loop;
    end CONSUME INTERNAL MESSAGE (4.1.1);

-------------------------------------------------------------------------------

    procedure INITIALIZE RADAR COMMUNICATION (4.1.2) is

     begin
       DEQUEUE INTERNAL MESSAGE FROM CONSUME INTERNAL
        MESSAGE (4.1.1);
       use package HOUSEKEEPING (4.1.2.1);
        if POWER FAIL FLAG and AUTOMATIC RESTART FLAG = TRUE then
          goto MANAGE FAULT DETECTION,CORRECTION
          AND REPORTING (4.1.5).AUTOMATIC RESTART
        else
          CLEAR MESSAGE BUFFERS (4.1.2.1.1);
          CLEAR MESSAGE QUEUES (4.1.2.1.2);
          CLEAR MESSAGE COUNTERS (4.1.2.1.3);
          CLEAR TRANSACTION LOG INDICATOR
           (4.1.2.1.4);
          CLEAR POWER FAIL INDICATOR (4.1.2.1.5);
        end if;
        procedure INITIALIZE COMMUNICATION SUBSYSTEM PARAMETERS
         (4.1.2.2) is
        begin
--   primary and redundant line adresses,
--   association of routing id, data link adresses,
--   baud rate, protocol assignments, and transaction
--   log indicator.
        if NON-DEFAULT PARAMETER FLAG = TRUE then
          LOAD NON-DEFAULT PARAMETERS (4.1.2.2.1);
        else
          LOAD DEFAULT PARAMETERS (4.1.2.2.2);
        end if;
        procedure INITIALIZE COMMUNICATION TABLES USING
          PARAMETERS (4.1.2.2.3);
        procedure INITIALIZE DATA LINK TABLE (4.1.2.2.3.1);
        procedure INITIALIZE MESSAGE TYPE TABLE (4.1.2.2.3.2);
        procedure SET TRANSACTION LOG FLAG (4.1.2.2.3.3);
```

```
        procedure INITIATE MANAGE MESSAGES (4.1.2.3);
        procedure INITIATE MANAGE DATA LINKS (4.1.2.4);
    end INITIALIZE RADAR COMMUNICATION (4.1.2);

-------------------------------------------------------------------

  package MANAGE DATA LINKS (4.1.3) is
   procedure MANAGE DATA LINK STATUS TABLE (4.1.3.1);
   procedure MANAGE PROTOCOLS (4.1.3.2);
  end MANAGE DATA LINKS (4.12.3);

  package body MANAGE DATA LINKS (4.1.3) IS
--  THE OBJECTIVE OF THE MANAGE DATA LINKS PACKAGE IS TO
--MANAGE THE DATA LINK TABLE, PROTOCOL AND THE
--INITIALIZATION AND TERMINATION OF A SINGLE DATA LINK.
  begin
    procedure MANAGE DATA LINK STATUS TABLE (4.1.3.1) is
    begin
      function SEARCH DATA LINK STATUS TABLE (4.1.3.1.1);
      function INSERT ENTRY (4.1.3.1.2);
      function DELETE ENTRY (4.1.3.1.3);
      procedure INITIATE INITIALIZE DATA LINK (4.1.3.1.4);
      procedure INITIATE TERMINATE DATA LINK (4.1.3.1.5);
    end MANAGE DATA LINK STATUS TABLE (4.1.3.1);
    procedure MANAGE PROTOCOLS (4.1.3.2) is
--START PROTOCOLS
    begin
      case PROTOCOL is
        when ATDL-1 =>
          initiate ATDL-1 PROTOCOL SUPERVISOR;
        when others =>
          SET ERROR CODE TO INVALID PROTOCOL;
          queue to PRODUCE INTERNAL MESSAGE (4.1.7);
      end case;
      procedure MONITOR PROTOCOL (4.1.3.2.1);
      procedure INITIATE TERMINATE DATA LINK (4.1.3.2.2);
      procedure REPORT PROTOCOL STATUS (4.1.3.2.3);
    end MANAGE PROTOCOLS (4.1.3.2);
  end MANAGE DATA LINKS (4.1.3);
```

```
------------------------------------------------------------------

package MANAGE MESSAGES (4.1.4) is
  procedure MESSAGE SUPERVISOR (4.1.4.1);
  procedure OUTGOING MESSAGES (4.1.4.2);
  procedure INCOMING MESSAGES (4.1.4.3);
  procedure INTERNAL MESSAGES (4.1.4.4);
  procedure MANAGE MESSAGE BUFFERS AND QUEUES (4.1.4.5);
end;

package body MANAGE MESSAGES (4.1.4) is
  procedure MESSAGE SUPERVISOR (4.1.4.1) is

   begin
    loop
     while INTERNAL TIMING REQUIREMENT <=0 loop
       if MESSAGE QUEUE COUNT >0 then
       DETERMINE MESSAGE ACTION (4.1.4.1.1);
       SELECT HIGHEST PRIORITY MESSAGE (4.1.4.1.2);
--      SELECT HIGHEST PRIORITY MESSAGE
--      WILL BE REPEATED A NUMBER OF TIMES IN REMOTE AND
--      RADAR COMMUNICATIONS AND WOULD PROBABLY BE
--      IMPLEMENTED AS PART OF THE USERS' RUNTIME LIBRARY.
--      TO VALIDATE MESSAGE SUBTYPE
       case MESSAGE SUBTYPE
         when POST TRANSMISSION MESSAGE ACTION =>
           queue to POST TRANSMISSION MESSAGE ACTION
             (4.1.4.1.3)
         when OUTGOING MESSAGE =>
           queue to OUTGOING MESSAGE (4.1.4.2)
         when INCOMING MESSAGE =>
           queue to INCOMING MESSAGE (4.1.4.3)
         when INTERNAL MESSAGE =>
           queue to INTERNAL MESSAGES (4.1.4.4)
         when others =>
           SET ERROR CODE FOR INVALID MESSAGE SUBTYPE;
           QUEUE TO PRODUCE INTERNAL MESSAGE (4.1.7);
       end case;
       else
         exit loop;
```

```
        end if;
      end loop;
      delay INTERNAL TIMING REQUIREMENT * NUMBER OF TASKS
    end loop;
procedure POST TRANSMISSION MESSAGE ACTION (4.1.4.1.3) is
begin
  loop
    while INTERNAL TIMING REQUIREMENT <= loop
      if POST MESSAGE QUEUE >0 then
       function SELECT HIGHEST PRIORITY MESSAGE
       (4.1.4.1.3.1);
--      SELECT HIGHEST PRIORITY MESSAGE
--      WILL BE REPEATED A NUMBER OF TIMES IN REMOTE AND
--      RADAR COMMUNICATIONS AND WOULD PROBABLY BE
--      IMPLEMENTED AS PART OF THE USERS' RUNTIME LIBRARY.
       procedure POST MESSAGE STATUS (4.1.4.1.3.2);
       if TRANSACTION LOG FLAG and MESSAGE ACKNOWLEDGEMENT
         REQUIRED = FALSE then
         RELEASE MESSAGE BUFFER (4.1.4.1.3.2.1);
       else if TRANSACTION LOG FLAG AND MESSAGE
         ACKNOWLEDGEMENT REQUIRED = TRUE then
        GET MESSAGE BUFFER (4.1.4.1.3.2.2);
        COPY MESSAGE TO NEW BUFFER (4.1.4.1.3.2.3);
       end if;
       procedure QUEUE MESSAGE(S) TO PRODUCE INTERNAL
        MESSAGE (4.1.7);
      end if;
     end loop;
     delay INTERNAL TIMING REQUIREMENTS * NUMBER OF TASKS
   end loop;
  end POST TRANSMISSION MESSAGE ACTION (4.1.4.1.3);
  end MESSAGE SUPERVISOR (4.1.4.1);


procedure OUTGOING MESSAGES (4.1.4.2) is

begin
  loop
    while INTERNAL TIMING REQUIREMENT <=0 loop
      if OUTGOING MESSAGE QUEUE COUNT >0 then
```

```
      SELECT HIGHEST PRIORITY MESSAGE (4.1.4.2.1)
--       SELECT HIGHEST PRIORITY MESSAGE
--       WILL BE REPEATED A NUMBER OF TIMES IN REMOTE
--       AND RADAR COMMUNICATIONS AND WOULD PROBABLY BE
--       IMPLEMENTED AS PART OF THE USERS' RUNTIME LIBRARY.
        procedure VALIDATE OUTGOING MESSAGE (4.1.4.2.2) is
        begin
         function MESSAGE TYPE TABLE LOOKUP (4.1.4.2.2.1);
--        SEARCHES MESSAGE TYPE TABLE AND RETURNS MESSAGE
--        TYPE AND DUPLICATION CODES/IDS.
         if MESSAGE TYPE NOT OUTGOING MESSAGE TYPE then
          SET ERROR CODE FOR INVALID OUTGOING MESSAGE;
          queue to PRODUCE INTERNAL MESSAGE (4.1.7);
         end if;
         function DATA TABLE LOOKUPS (4.1.4.2.2.2);
--       SEARCHES DATA LINK TABLE BY DESTINATION
--       ID AND RETURNS PROTOCOL STATUS, DATA LINK
--       AVAILABILITY AND PROTOCOL MAILBOX.
         if PROTOCOL STATUS INACTIVE then
          SET ERROR CODE FOR INACTIVE PROTOCOL STATUS;
          queue to PRODUCE INTERNAL MESSAGE (4.1.7);
         else if DATA LINK NOT AVAILABLE then
          SET ERROR CODE FOR INACTIVE DATA LINK;
          queue to PRODUCE INTERNAL MESSAGE (4.1.7);
         end if;
        procedure REPLICATE MESSAGE FOR MULTIPLE MESSAGES
         (4.1.4.2.2.3) is
        begin
         if MULTIPLE MESSAGES then
          for I IN I....NUMBER OF MESSAGES - 1 loop
           GET MESSAGE BUFFER (4.1.4.2.2.3.1);
           COPY MESSAGE TO NEW BUFFER
            (4.1.4.2.2.3.2);
          end loop;
         end if;
        end REPLICATE MESSAGE FOR MULTIPLE MESSAGES
          (4.1.4.2.2.3);
        procedure QUEUE MESSAGE(S) TO PRODUCE INTERNAL
         MESSAGE (4.1.7)
       end VALIDATE OUTGOING MESSAGE (4.1.4.2.2);
```

```
      end if;
    end loop;
    delay INTERNAL TIMING REQUIREMENT * NUMBER OF TASKS
  end loop;
end OUTGOING MESSAGES (4.1.4.2);

procedure INCOMING MESSAGES (4.1.4.3) is
begin
end INCOMING MESSAGES (4.1.4.3);

procedure INTERNAL MESSAGES (4.1.4.4) is
begin
end INTERNAL MESSAGES (4.1.4.4);

procedure MANAGE MESSAGE BUFFERS AND QUEUES (4.1.4.5) is
begin
end MANAGE MESSAGE BUFFERS AND QUEUES (4.1.4.5)
end MANAGE MESSAGES (4.1.4);
```

```
package MANAGE FAULT DETECTION, CORRECTION AND REPORTING
 (4.1.5) is

--AUTOMATIC RESTART WOULD APPEAR IN THIS AREA
  procedure IDENTIFY FAULTS (4.1.5.1) is
  begin
   procedure HARDWARE FAULTS (4.1.5.1.1);
   procedure SOFTWARE FAULTS (4.1.5.1.2);
  end IDENTIFY FAULTS (4.1.5.1);
  procedure UPDATE ERROR COUNTERS (4.1.5.2);
  procedure ATTEMPT CORRECTIVE ACTION (4.1.5.3);
  procedure REPORT FAULTS (4.1.5.4);
end MANAGE FAULT DETECTION, CORRECTION AND REPORTING
 (4.1.5);
package body MANAGE FAULT DETECTION, CORRECTION AND
  REPORTING (4.1.5) is
end MANAGE FAULT DETECTION, CORRECTION AND REPORTING
```

```
  (4.1.5);
-------------------------------------------------------------

procedure TERMINATE REMOTE COMMUNICATION (4.1.6) is
begin
  procedure BUILD TERMINATION MESSAGE (4.1.6.1);
  procedure START TERMINATE DATA LINK (4.1.6.2);
  procedure UPDATE DATA LINK STATUS TABLE (4.1.6.3);
  procedure ABORT DATA LINK MANAGER (4.1.6.4);
  procedure ABORT MESSAGE MANAGER (4.1.6.5);
  procedure UPDATE TRANSACTION LOG (4.1.6.6);
end TERMINATE REMOTE COMMUNICATIONS (4.1.6);
-------------------------------------------------------------

 procedure PRODUCE INTERNAL MESSAGE (4.1.7) IS
  begin
    loop
      while INTERNAL TIMING REQUIREMENT >0 loop
        if PRODUCE INTERNAL MESSAGE COUNT <=0 then
          exit loop;
        else SELECT HIGHEST PRIORITY MESSAGE (4.1.7.1);
--       SELECT HIGHEST PRIORITY MESSAGE
--       WILL BE REPEATED A NUMBER OF TIMES IN REMOTE
--       AND RADAR COMMUNICATIONS AND WOULD PROBABLY BE
--       IMPLEMENTED AS PART OF THE USERS' RUNTIME LIBRARY.

        procedure DEQUEUE MESSAGE FROM PRODUCE INTERNAL MESSAGE
         QUEUE (4.1.7.2);
--  VALIDATE INTERNAL MAILBOX ID AND PUT MESSAGE
        case INTERNAL MAILBOX ID is
         when COMMAND AND CONTROL ID => PUT
          MESSAGE TO COMMAND AND CONTROL MAILBOX;
--  UTILIZE THE PUT TASK OF INTRASYSTEM TRANSFERS
         when TARGET CONTROL ID => PUT
          MESSAGE TO TARGET CONTROL MAILBOX;
         when REMOTE COMMUNICATIONS ID => PUT
          MESSAGE TO REMOTE COMMUNICATIONS MAILBOX;
         when others =>
          SET ERROR CODE FOR INVALID INTERNAL MAILBOX ID;
          PUT MESSAGE TO COMMAND AND CONTROL MAILBOX;
```

```
          end case;
         end if;
        end loop;
        delay INTERNAL TIMING DELAY * NUMBER OF TASKS;
      end loop;
 end PRODUCE INTERNAL MESSAGE (4.1.7);
end COMMUNICATION SUPERVISOR (4.1);
-------------------------------------------------------------------
-------------------------------------------------------------------

package INTRASYSTEM TRANSFER (4.2) is
--The objective of the INTRASYSTEM TRANSFER subentity is to
--provide facilities which allow message flow between the
--RADAR COMMUNICATIONS and the COMMAND & CONTROL and other
--processors as well as support message flow within the RADAR
--COMMUNICATIONS.  To insure message flow integrity,
--INTRASYSTEM TRANSFERS will employ mailbox concepts with
--semaphore counters and message time stamps to identify
--potential bottlenecks and provide periodic status reports as
--needed.  INTRASYSTEM TRANSFER is started and terminated by
--COMMAND & CONTROL.
--The INTRASYSTEM TRANSFER package is intended to operate
--concurrently with various other operations and is itself
--comprised of MESSAGE POSTING TIMER, GET INTRASYSTEM
--TRANSFER and PUT INTRASYSTEM TRANSFER.
--  The following descriptions assume that COMMAND &
--CONTROL has generic mailbox box features which utilize
--the semaphore concept. The initialization function of
--COMMAND & CONTROL will initialize the proper number and
--type of mailboxes necessary to implement this activity.
--  Mailboxes are producer-consumer points, the number of
--which is determined at initialization time by the number of
--protocols.
  procedure GET INTRASYSTEM TRANSFERS (4.2.1);
  procedure PUT INTRASYSTEM TRANSFERS (4.2.2);
  procedure MESSAGE POSTING TIMER (4.2.3);
end INTRASYSTEM TRANSFERS (4.2);

package body INTRASYSTEM COMMUNICATIONS (4.2) is
 procedure GET INTRASYSTEM TRANSFER (4.2.1) is
```

```
--The set and put features of INTRASYSTEM TRANSFER
-- may be implemented in two ways. The first method
--is to have each mailbox employ its own set and put
--tasks.  The second method is separate procedures
--for set and put with multiple mailboxes - a logical
--multiplexer. This is a concurrent task.
  begin
    loop
      while INTERNAL TIMING REQUIREMENT >0 loop
       if MAILBOX COUNT <0 then
         exit loop;
       else for I IN I...NUMBER OF MAILBOXES loop
        TIMEOUT FLAG(I) = FALSE;
       end loop;
       end if;
    loop
      for I IN I...NUMBER OF MAILBOXES loop
        if GET MESSAGE BUFFER = EMPTY then
          if TIMEOUT FLAG (I) TRUE then
            ISSUE PRODUCER ERROR;
            TIMEOUT FLAG (I) = FALSE;
          else
            TIMEOUT FLAG (I) = TRUE;
          end if;
        else
          TIMEOUT FLAG (I) = FALSE;
          set MESSAGE FROM BUFFER (I);
          UPDATE TIME STAMP QUEUE (I);
--    determine message type
          if MESSAGE = 'I'M ALIVE' then
           DISCARD MESSAGE;
          else
           MESSAGE AVAILABLE FOR PROCESSING;
          end if;
        end if;
      end loop;
      delay INTERNAL TIMING REQUIREMENT * NUMBER OF TASKS
    end loop;
 end GET INTRASYSTEM TRANSFER (4.2.1);
```

```
 procedure PUT INTRASYSTEM TRANSFER (4.2.2) is
--The set and put features of the INTRASYSTEM TRANSFER
--may be implemented in two ways. The first method
--is to have each mailbox employ its own set and put
--procedures.
--The second method is separate procedures for set and put with
--multiple mailboxes - a logical multiplexer.
  procedure TEST BUFFER (DESTINATION) is
  begin
   loop
    while INTERNAL TIMING REQUIREMENT > 0 loop
     if MAILBOX COUNT < 0 then
      exit loop;
     elsif DESTINATION BUFFER = NOT EMPTY then
      if DESTINATION TIMEOUT FLAG = TRUE then
        ISSUE CONSUMER ERROR;
        DESTINATION TIMEOUT FLAG = FALSE;
      else
        DESTINATION TIMEOUT FLAG = TRUE;
      end if;
     else
      PUT MESSAGE IN DESTINATION MAILBOX;
      DESTINATION TIMEOUT FLAG = FALSE;
     end if;
    end loop;
  end TEST BUFFER;
  loop
    accept MESSAGE (MESSAGE);
    DETERMINE DESTINATION; --FROM MESSAGE CONTENT
    case DESTINATION is
      when COMMAND CONTROL or TARGET CONTROL or
       RADAR COMMUNICATION = >
        TEST BUFFER (COMMAND AND CONTROL);
      when COMMUNICATION SUPERVISOR = >
        TEST BUFFER (COMMUNICATION SUPERVISOR);
      when TADIL-B PROTOCOL =>
        TEST BUFFER (TADIL-B PROTOCOL);
```

```
        when ATDL-1 PROTOCOL =>
          TEST BUFFER (ATDL-1 PROTOCOL);
        when MBDL PROTOCOL =>
          TEST BUFFER (MBDL PROTOCOL);
      end case;
      delay INTERNAL TIMING REQUIREMENT;
    end loop;
end put INTRASYSTEM TRANSFER (4.2.2);

procedure MESSAGE POSTING TIMER (4.2.3) is
--The objectives of the MESSAGE POSTING TIMER
--are to periodically generate 'I'M ALIVE' messages
--if no other message activity has occurred in
--any particular area for a period equal to or
--greater than the internal timing requirement
--and to insure that messages generated by the
--producing tasks are actually being consumed.
--In the case that generated messages are not
--being consumed the message posting timer alerts
--the message producing task and the operator.
  procedure CONSUMER TIMER (4.2.3.1) is
  begin
    loop
      for I IN I...NUMBER OF MAILBOXES loop
        if GET MESSAGE BUFFER(I) = NOT EMPTY then
          if CURRENT TIME-TIME STAMP(I) >
          INTERNAL TIMING REQUIREMENT then
            ISSUE TIMEOUT ERROR(I);
          elsif SEMAPHORE COUNTER(I)>MAX MESSAGES/SEC
          then
            ISSUE SEMAPHORE ERROR(I);
          end if;
        end if;
      end loop;
      delay INTERNAL TIMING REQUIREMENT;
    end loop;
  end CONSUMER TIMER (4.2.3.1);

  procedure PRODUCER TIMER (4.2.3.2) is
  begin
```

```
    loop
      for I IN I...NUMBER OF MAILBOXES loop
        if PUT MESSAGE BUFFER(I) = EMPTY then
          if CURRENT TIME - TIME STAMP>
            INTERNAL TIMING REQUIREMENT then
            BUILD 'I'M ALIVE' MESSAGE;
            put 'I'M ALIVE' MESSAGE IN MAILBOX(I);
          end if;
        elsif
          SEMAPHORE COUNTER(I)>MAX MESSAGES/SEC
          then
            ISSUE SEMAPHORE ERROR;
        end if;
      end loop;
      delay INTERNAL TIMING REQUIREMENT;
    end loop;
  end PRODUCER TIMER (4.2.3.2);
--Additional details (queue management, buffer management,
--internal routing table, etc.), are not included due to
--project schedule restrictions.
 end MESSAGE POSTING TIMER (4.2.3);
end INTRASYSTEM TRANSFERS (4.2);

------------------------------------------------------------------------
------------------------------------------------------------------------


package COMMUNICATION PROTOCOL (4.3) is
-- The objective of the COMMUNICATION PROTOCOL package is to
--handle the physical level of the transmission and
--reception of digital messages according to the predefined
--military protocol ATDL-1.
-- The COMMUNICATION PROTOCOL package is initiated and
-- The protocol used by the COMMUNICATION PROTOCOL package
--is ATDL-1, which cannot be discussed further due to
--it classification.  However, it will be handled by the
--following procedures.
  procedure CONSUME COMM SUPERVISOR MESSAGE (4.3.1);
  procedure INITIALIZE DATA LINK (4.3.2);
  procedure TRANSMIT MESSAGE (4.3.3);
```

```
    procedure RECEIVE MESSAGE (4.3.4);
    procedure PROTOCOL UTILITIES (4.3.5);
    procedure TERMINATE DATA LINK (4.3.6);
    procedure COMM SUPERVISOR MESSAGE PRODUCT (4.3.7);
end COMMUNICATION PROTOCOLS (4.3)
end RADAR COMMUNICATION (4.0);
```

APPENDIX E

DATA FLOW DIAGRAMS

APPENDIX E
DATA FLOW DIAGRAM

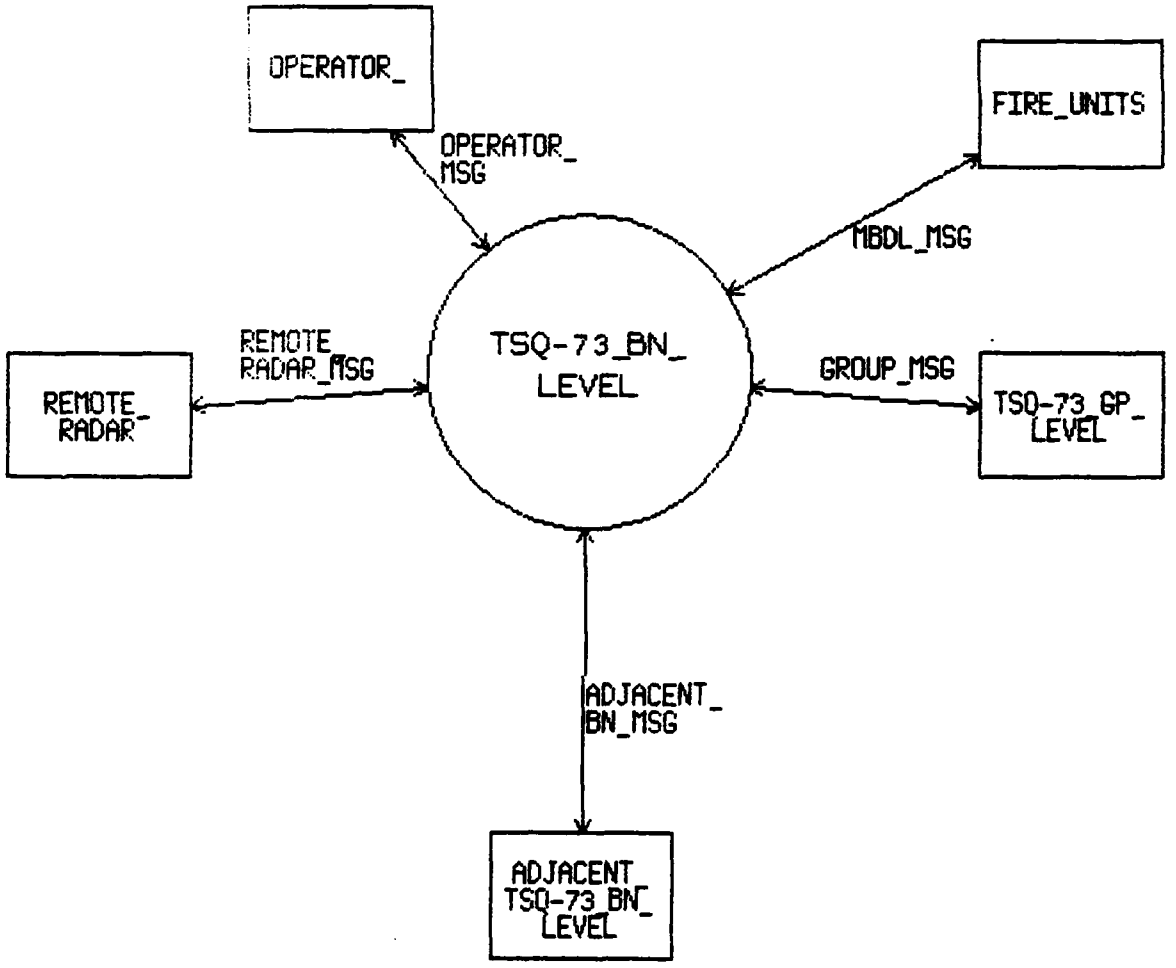
FINAL REPORT

LARGE SCALE SOFTWARE SYSTEM DESIGN
FOR THE
MISSILE MINDER AN/TSQ-73
USING
THE ADA PROGRAMMING LANGUAGE

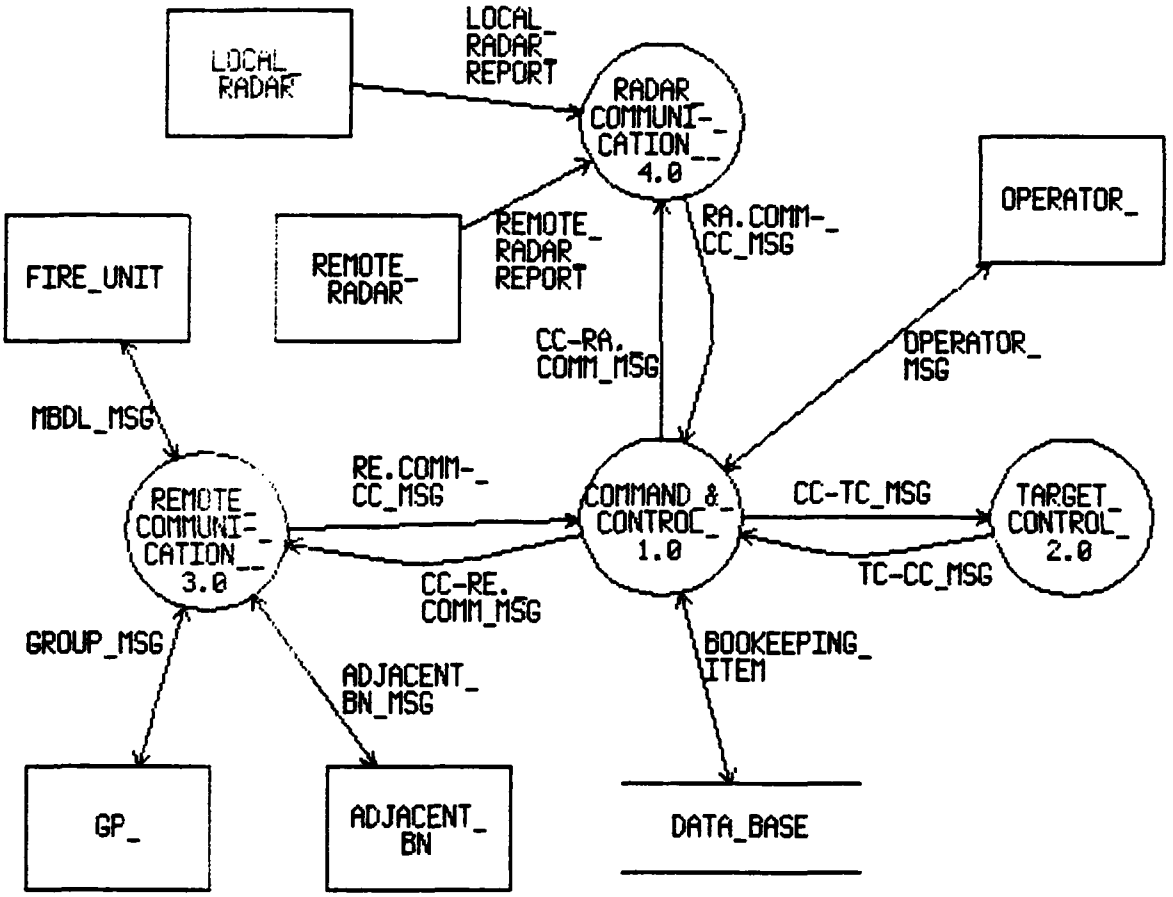
PREPARED FOR

U.S. ARMY COMMUNICATIONS ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703

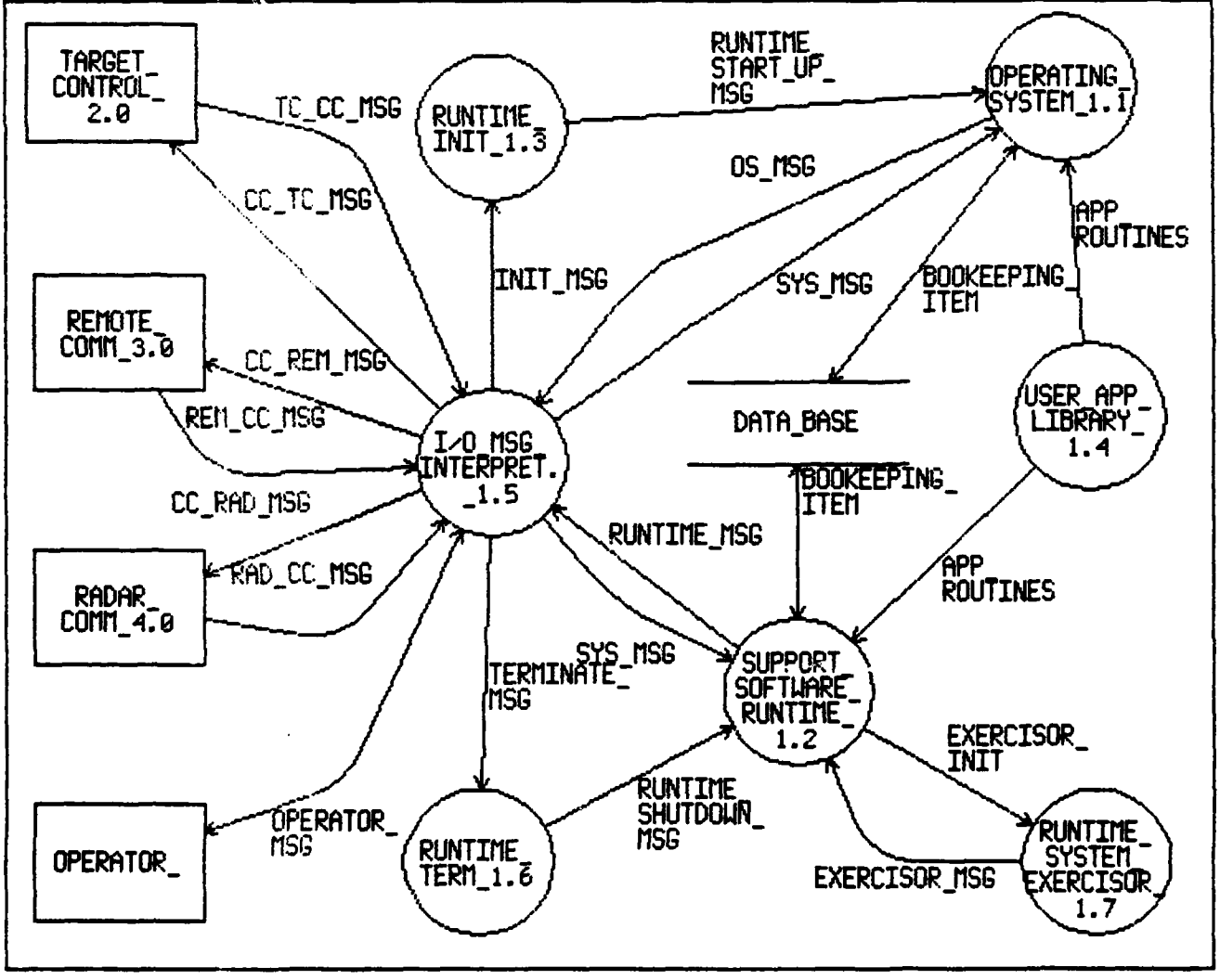
TITLE: AN/TSQ-73 CDFD		PAGE: 900
REVISION 1	AUTHOR: LWD	DATE: 5/5/82
NOTES:		



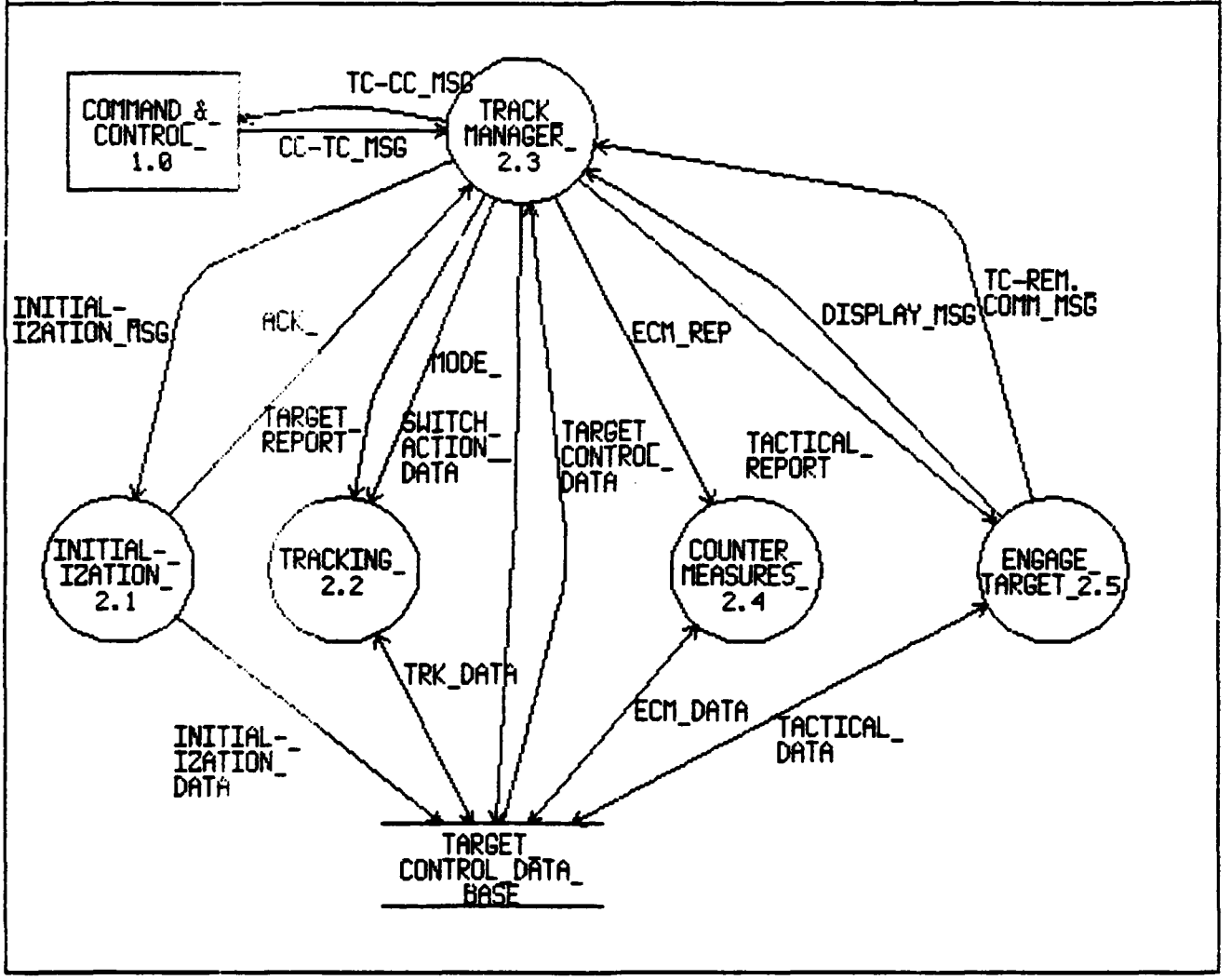
TITLE: AN/TSQ-73 BN-LEVEL SYSTEM DFD		PAGE: 901
REVISION 1	AUTHOR: LWD	DATE: 5/5/82
NOTES:		ROOT:



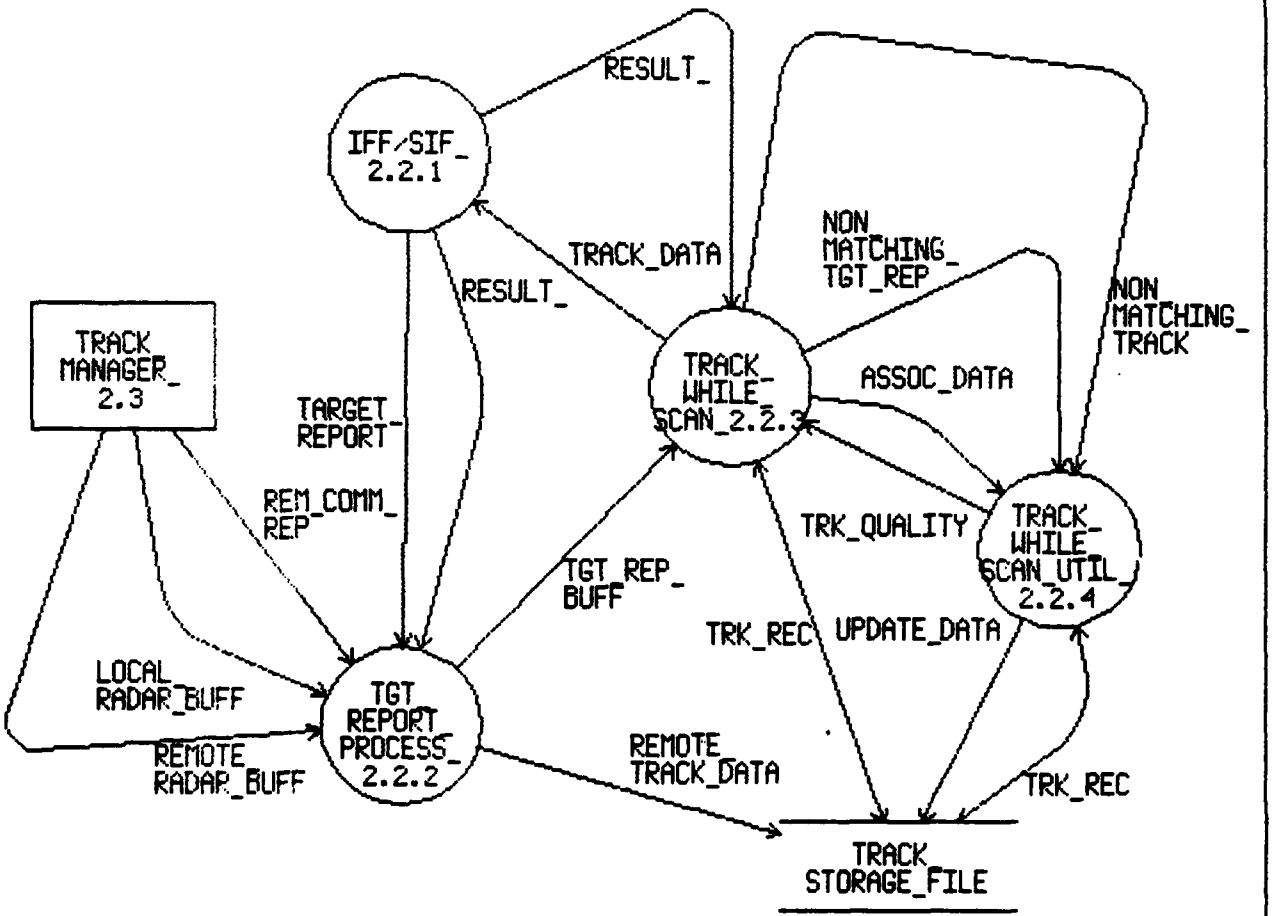
TITLE: 1.0 COMMAND & CONTROL SYSTEM DFD		PAGE: 902
REVISION 1	AUTHOR: LWD	DATE: 5/17/82
NOTES:		ROOT: 1.0



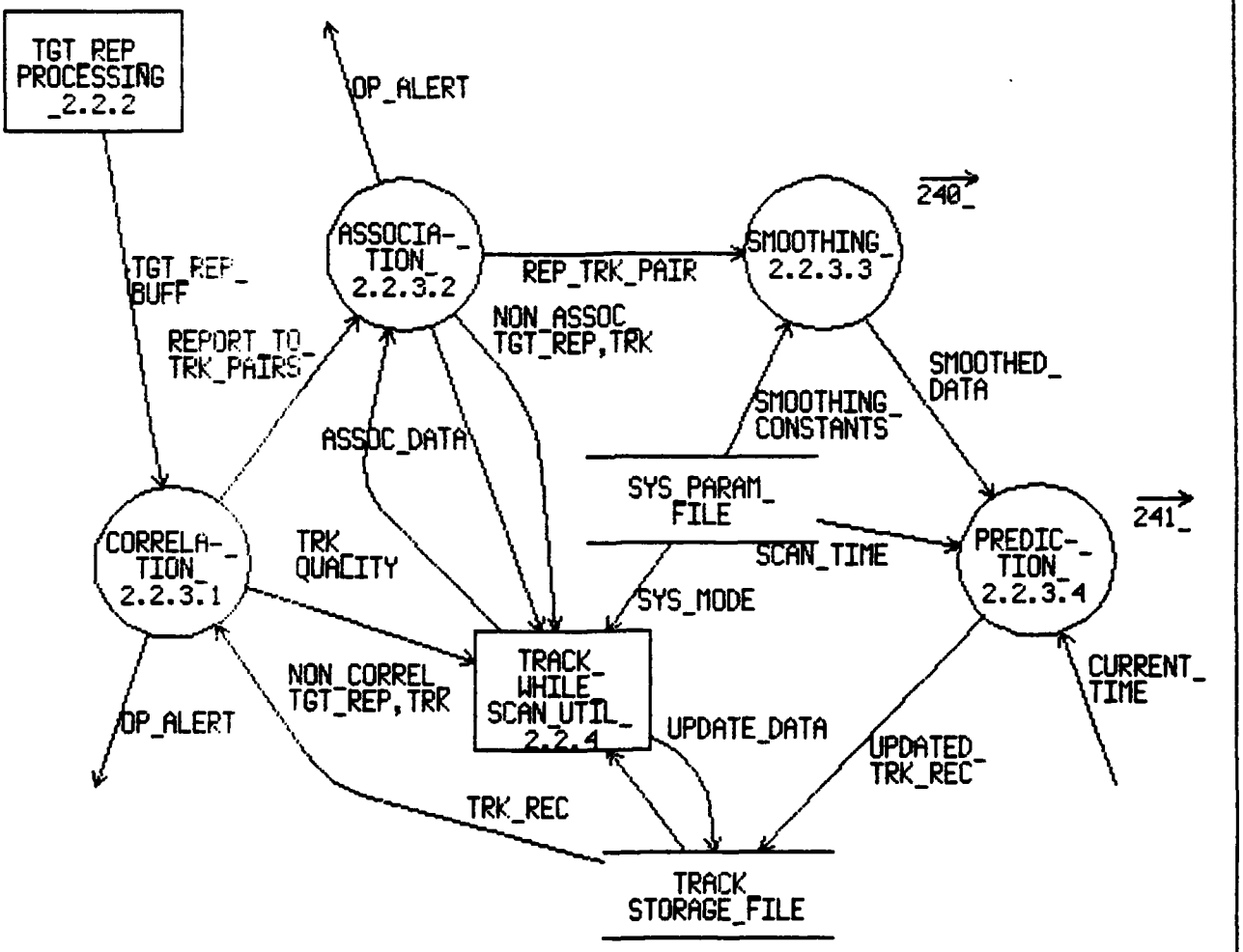
TITLE: 2.0 TARGET CONTROL SYSTEM DFD		PAGE: 904
REVISION 1	AUTHOR: MJD/WAS	DATE: 4/19/82
NOTES:		ROOT:



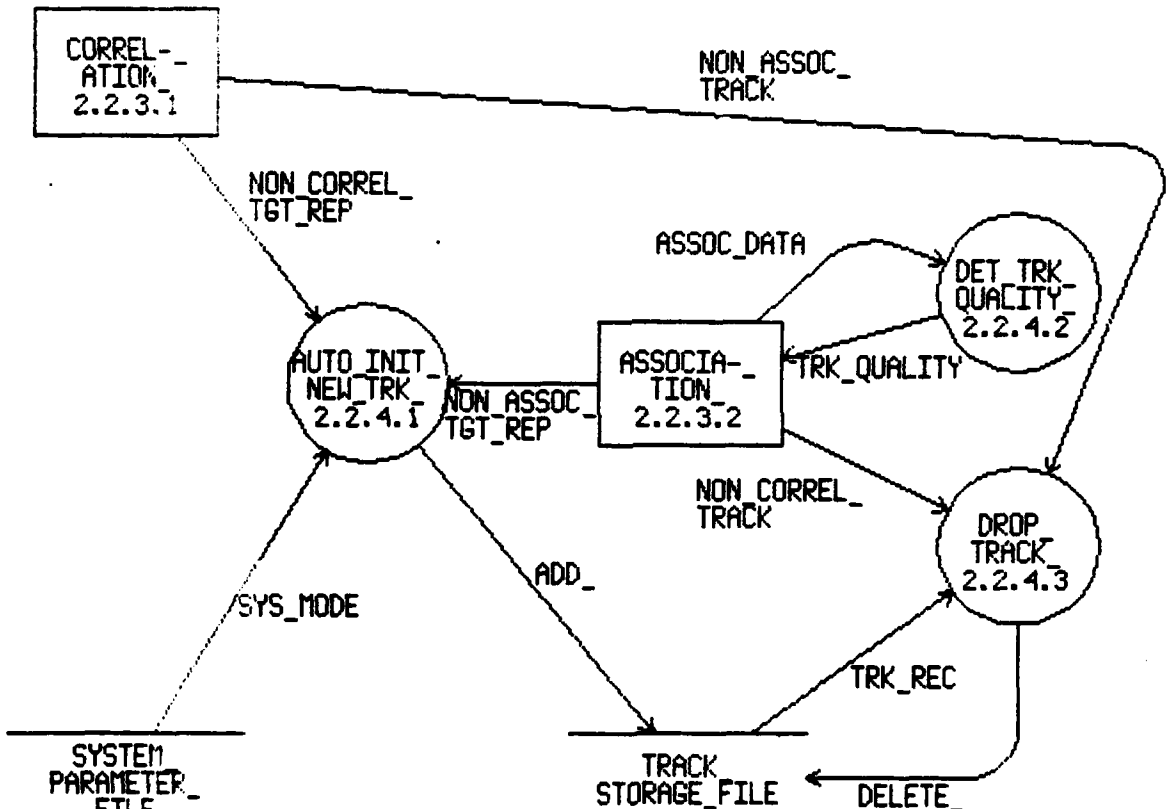
TITLE: 2.2 TRACKING SYSTEM DFD		PAGE: 905
REVISION 1	AUTHOR: LWD	DATE: 4/12/82
NOTES:		ROOT: 2.0



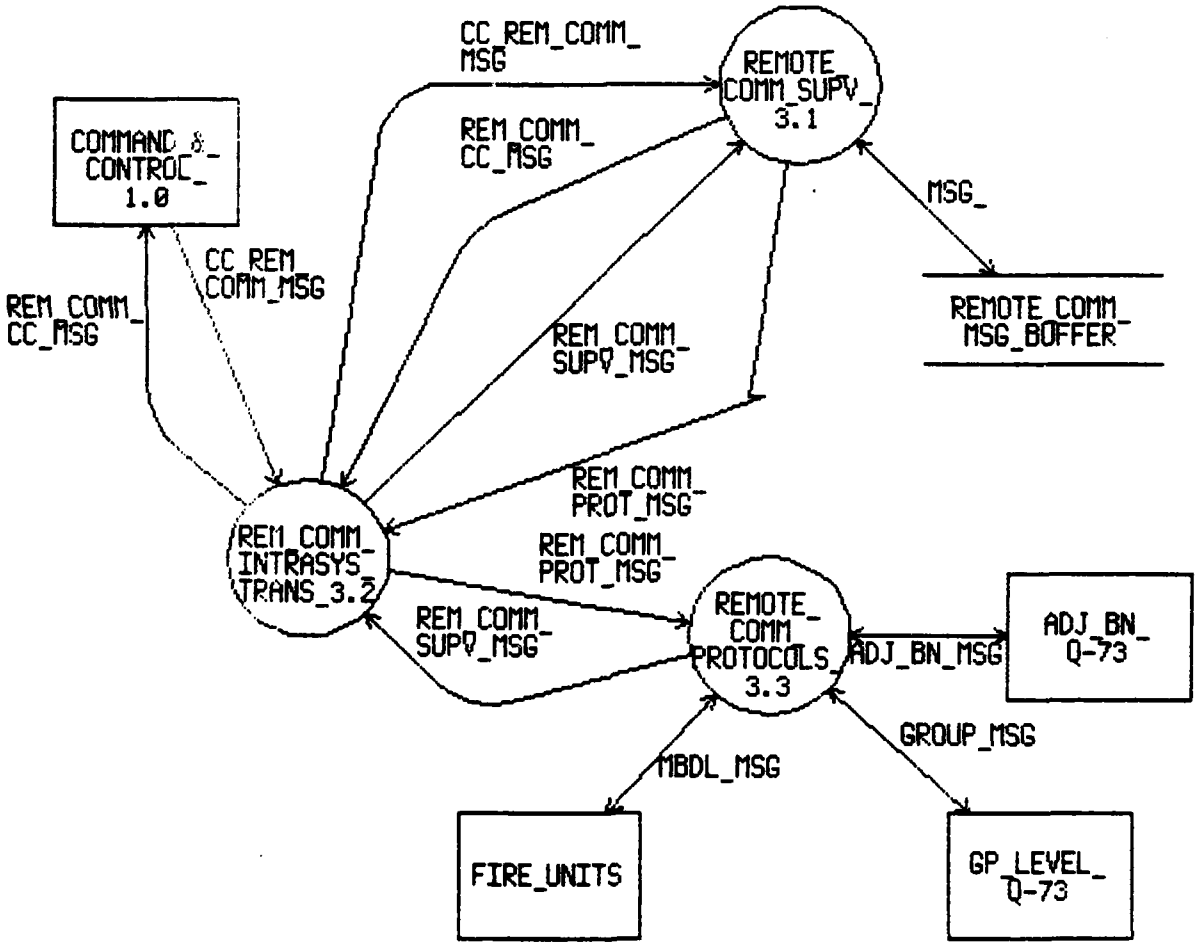
TITLE: 2.2.3 TRACK WHILE SCAN SYSTEM DFD		PAGE: 906
REVISION 1	AUTHOR: LWD	DATE: 5/12/82
NOTES:		ROOT: 2.0



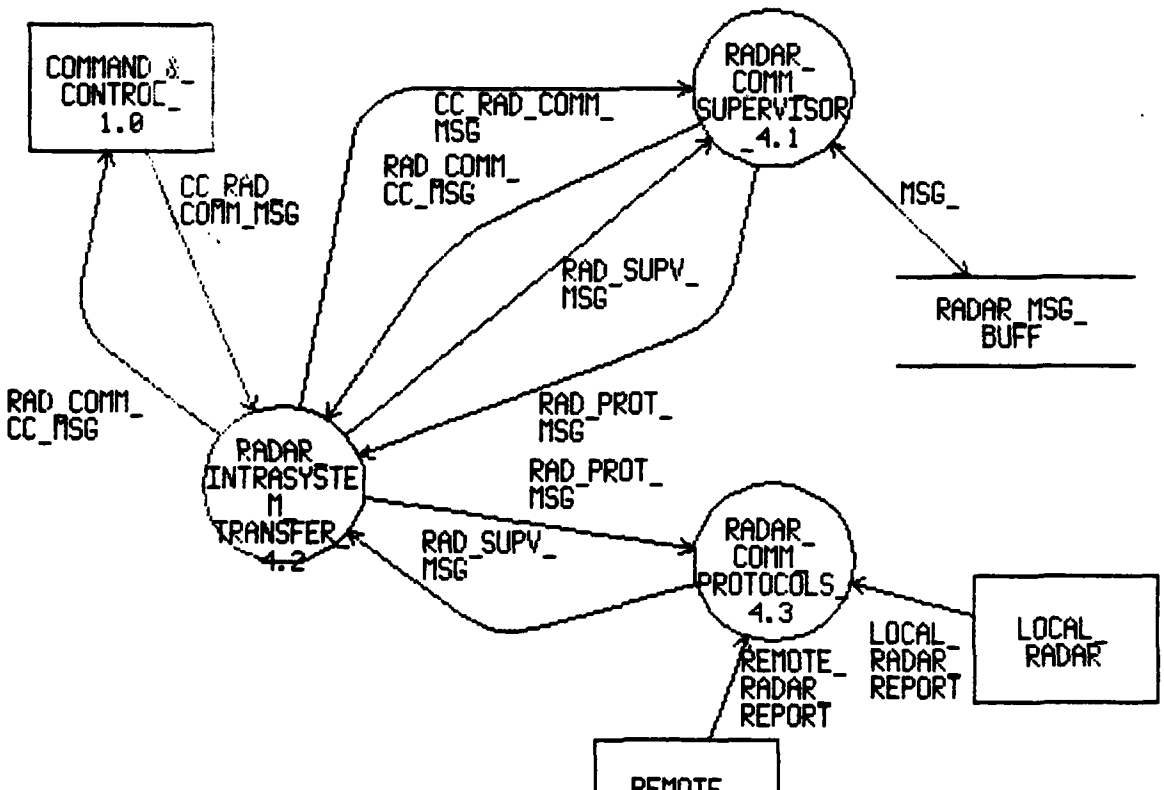
TITLE: 2.2.4 TRACK WHILE SCAN UTILITIES SYSTEM DFD		PAGE: 907
REVISION 1	AUTHOR: LLWD	DATE: 4/12/82
NOTES:		ROOT: 2.0



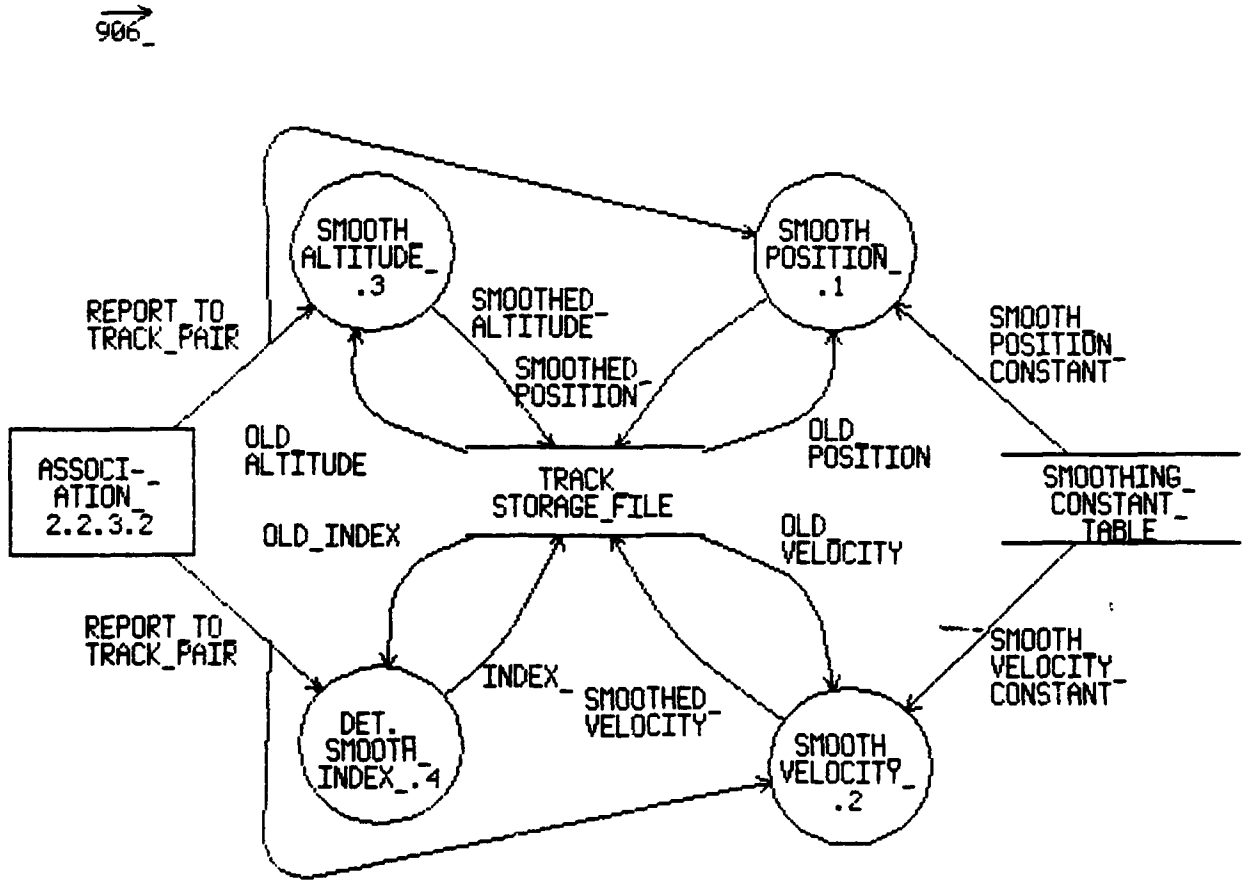
TITLE: 3.0 REMOTE COMMUNICATION SYSTEM DFD		PAGE: 908
REVISION 1	AUTHOR: LWD	DATE: 4/12/82
NOTES:		ROOT: 3.0



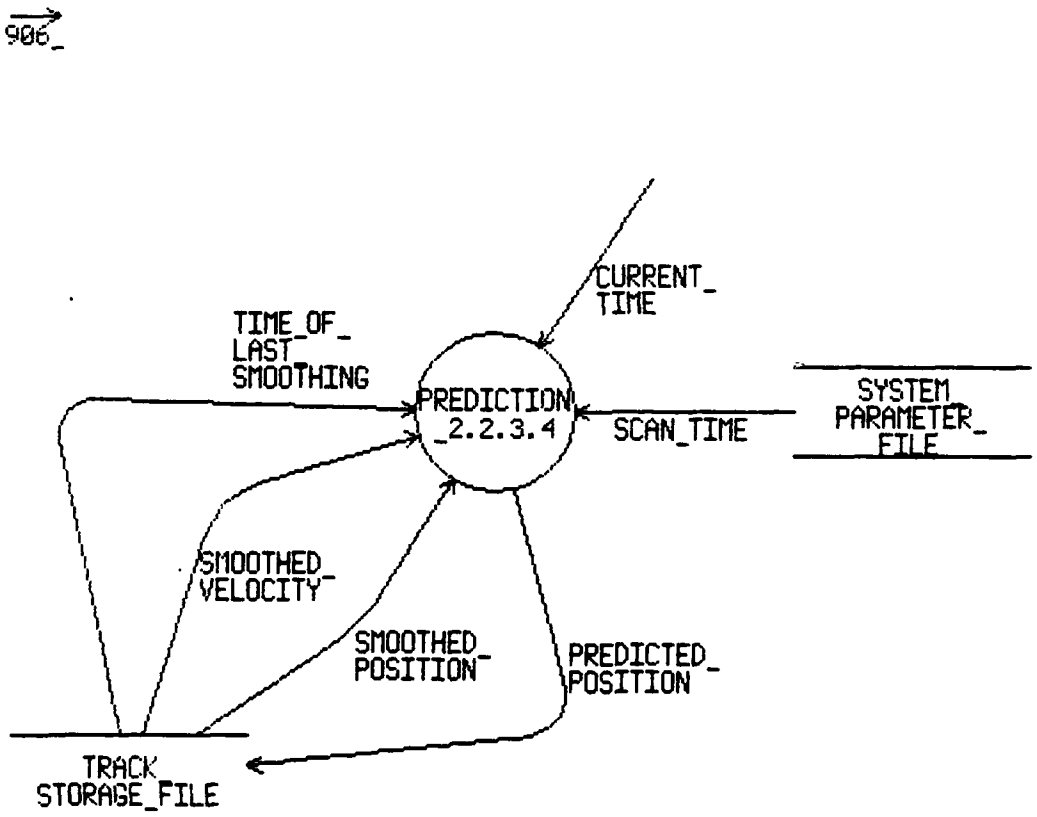
TITLE: 4.0 RADAR COMMUNICATION SYSTEM DFD		PAGE: 909
REVISION 1	AUTHOR: LWD	DATE: 4/12/82
NOTES:		ROOT: 4.0



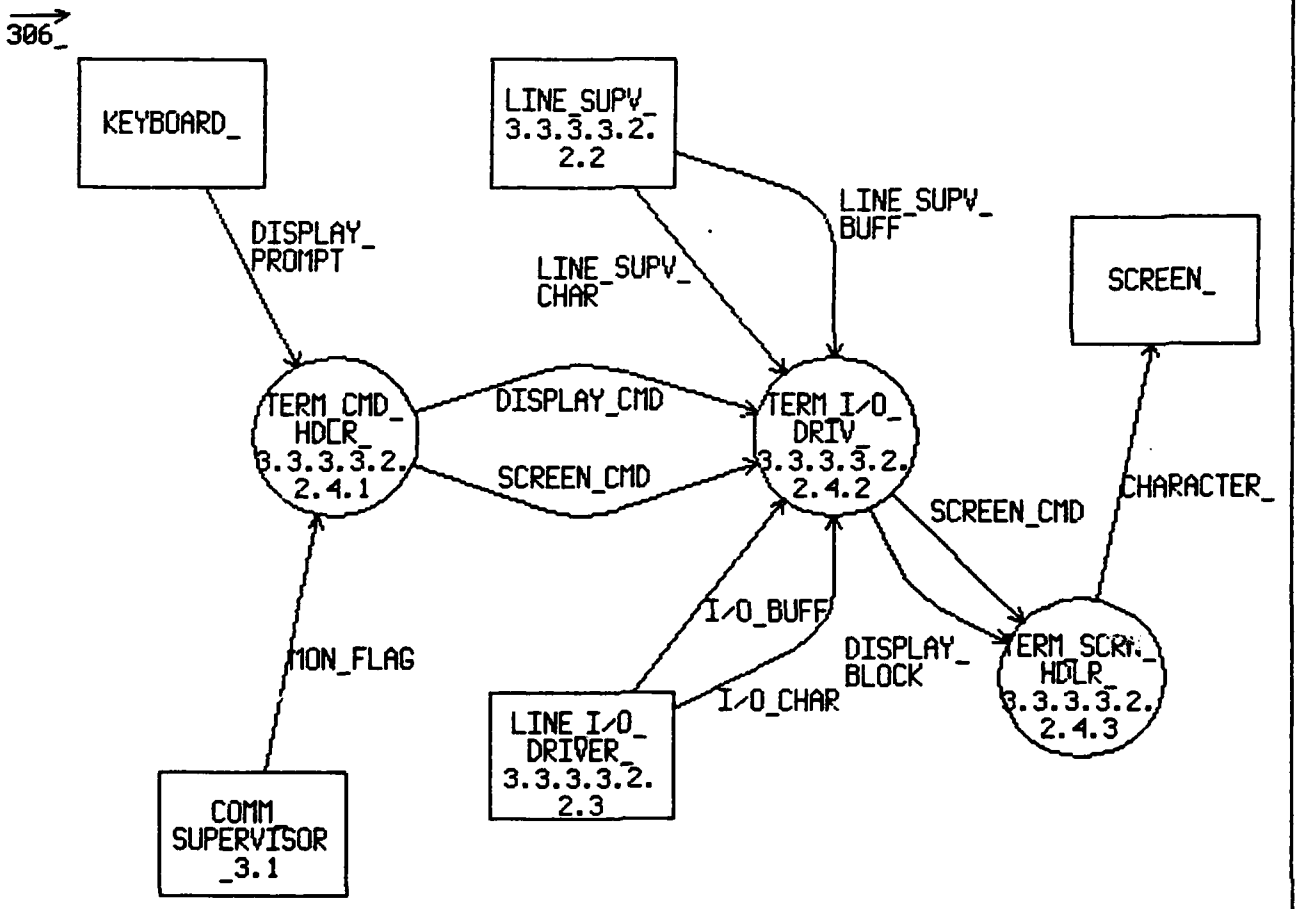
TITLE: 2.2.3.3 SMOOTHING DFD		PAGE: 240
REVISION 4	AUTHOR: MJD/CWM/PBM/LWD	DATE: 5/27/82
NOTES:		ROOT: 2.2.3.3



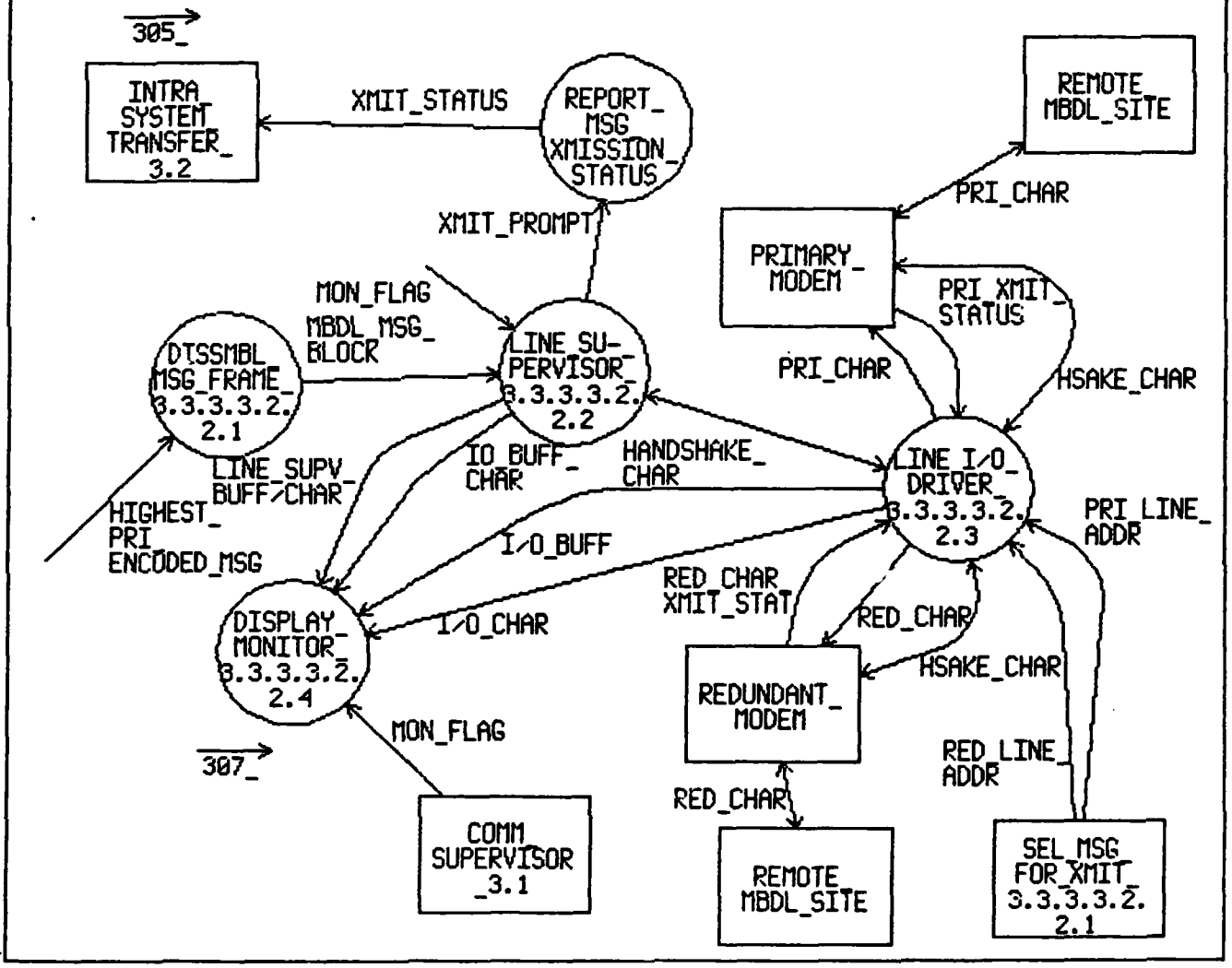
TITLE: 2.2.3.4 PREDICTION DFD		PAGE: 241
REVISION 3	AUTHOR: MJD/LWD	DATE: 5/27/82
NOTES:		ROOT: 2.2.3.4



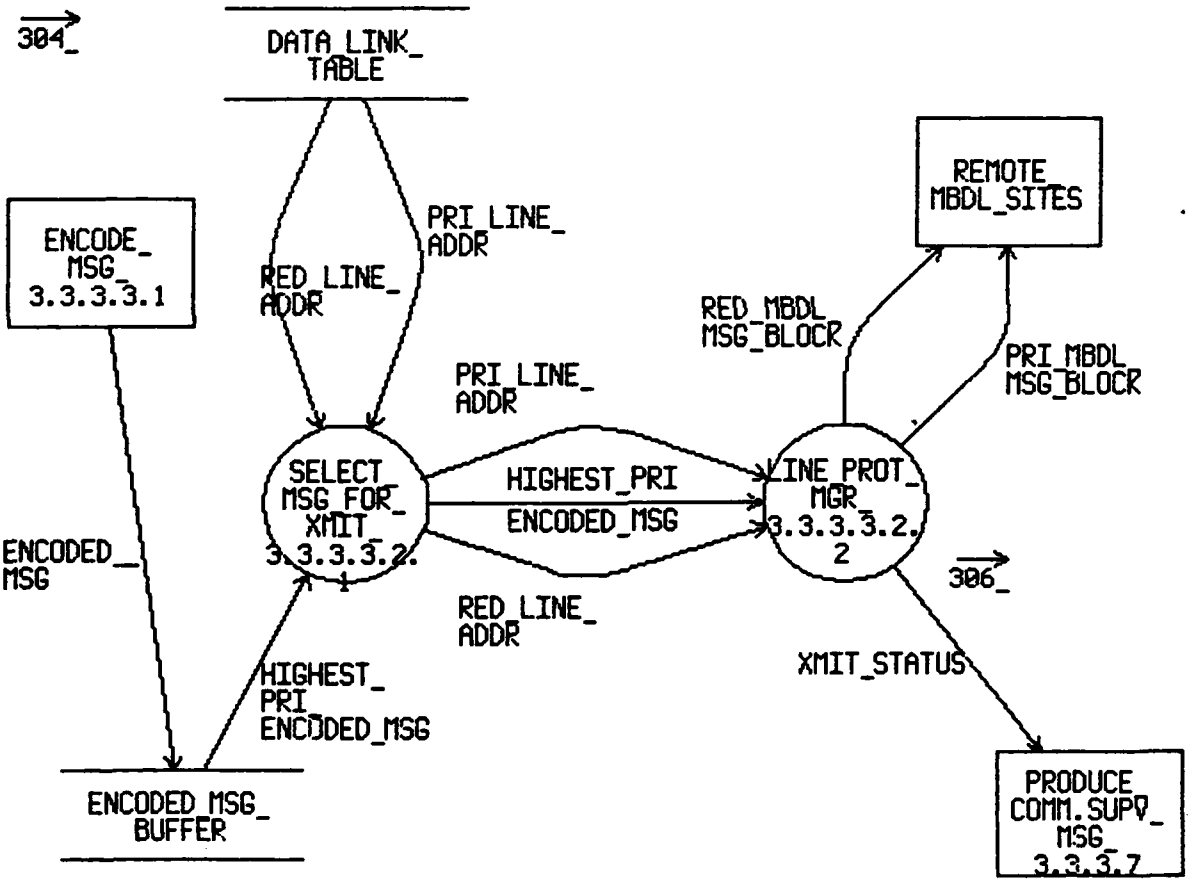
TITLE: 3.3.3.3.2.2.4 DISPLAY MONITOR DFD		PAGE: 307
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 2/5/82
NOTES:		ROOT: 3.0



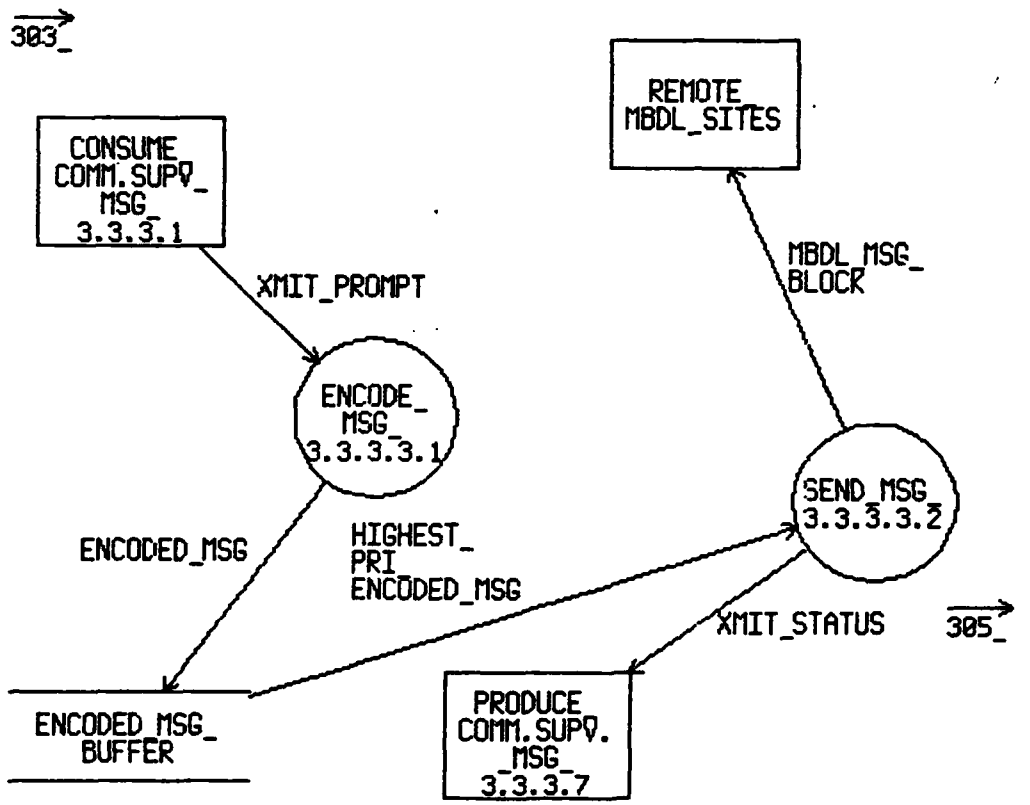
TITLE: 3.3.3.3.2.2 PROTOCOL MANAGER DFD		PAGE: 306
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 2/5/82
NOTES:		ROOT: 3.0



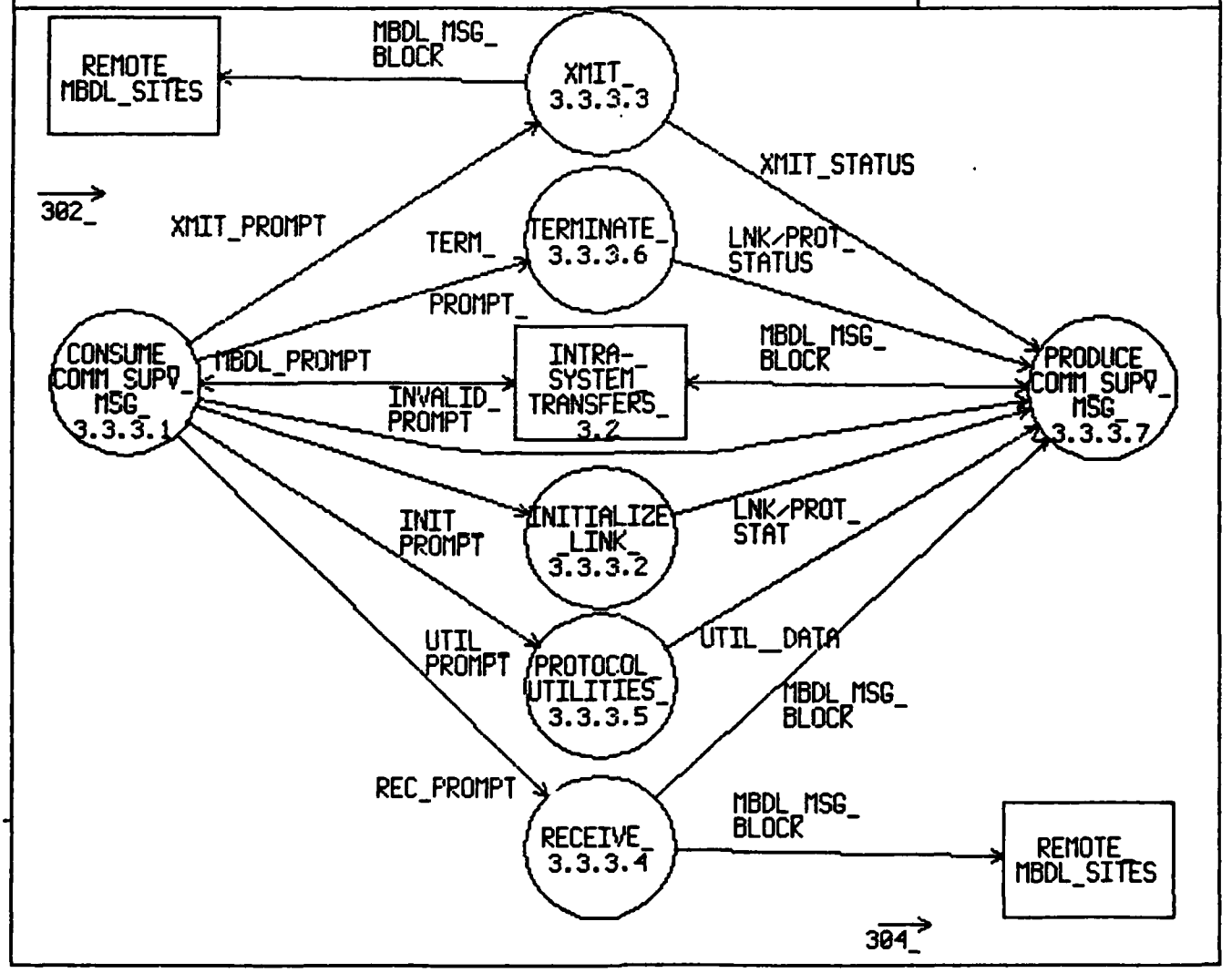
TITLE: 3.3.3.3.2 SEND MESSAGE DFD		PAGE: 305
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 2/5/82
NOTES:		ROOT: 3.0



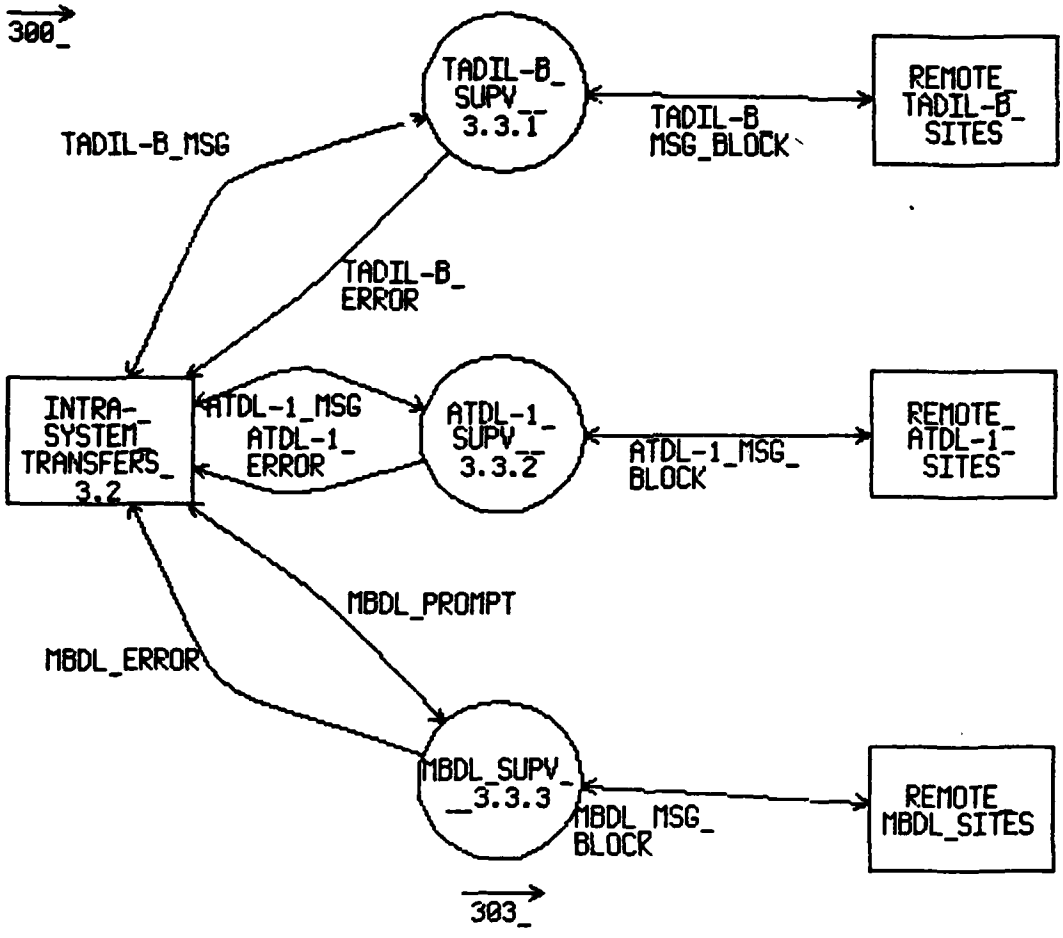
TITLE: 3.3.3.3 XMIT MESSAGE DFD		PAGE: 304
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 2/5/82
NOTES:		ROOT: 3.0



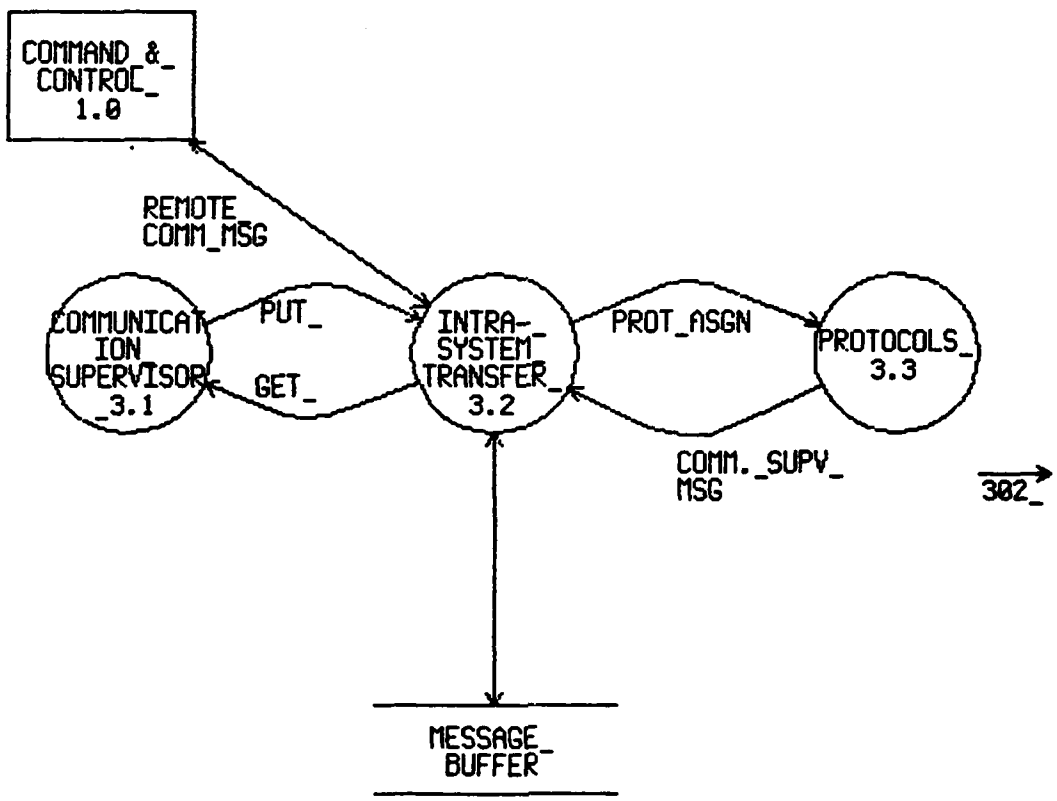
TITLE: MBDL SUPERVISOR 3.3.3		PAGE: 303
REVISION 1	AUTHOR: LWD/RAW/GIC	DATE: 2/5/82
NOTES:		ROOT: 3.0



TITLE: 3.3 PROTOCOLS DFD		PAGE: 302
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/28/82
NOTES:		ROOT: 3.0



TITLE: 3.0 REMOTE COMMUNICATIONS DFD		PAGE: 300
REVISION 0	AUTHOR: LWD/RAW/GIC	DATE: 1/14/82
NOTES:		ROOT: 3.0



APPENDIX F

DATA DICTIONARY

APPENDIX F

DATA DICTIONARY

FINAL REPORT

LARGE SCALE SOFTWARE SYSTEM DESIGN FOR THE MISSILE MINDER AN/TSQ-73 USING THE ADA PROGRAMMING LANGUAGE

PREPARED FOR

U.S. ARMY COMMUNICATIONS ELECTRONICS COMMAND

FORT MONMOUTH, NEW JERSEY 07703

```
ACK_ (FLOW):

        - ACKNOWLEDGEMENT;

        AFFIRM_ACK !
        NEG_ACK!


ADD_ (FLOW):

        "TRACK RECORD TO BE ADDED TO TRACK STORAGE FILE"

        TRACK_REC!


AFFIRM_ACK (ELEMENT):

        - ACKNOWLEDGEMENT INDICATOR; !


ALTITUDE_ (ELEMENT):

        - RANGE 1000-2000 FEET;

        {DIGIT}!


ALTITUDE_GATE (FLOW):

        - INTEGER RANGE LOWER ALT GATE..UPPER ALT GATE;
        - CLASSIFIED;

        {DIGIT}!


ASSOCIATION_COUNT (ELEMENT):

        - INTEGER RANGE 1..15;
        "HOW MANY OTHER TARGETS THAT HAVE BEEN ASSOCIATED
            WITH A CERTAIN TARGET"

        (DIGIT)!
        (DIGIT&
        DIGIT)!


ASSOC_DATA (FLOW):

        TRACK_NO &
        CONSECUTIVE_HIT &
        CONSECUTIVE_MISS!


AZIMUTH_ (ELEMENT):

        - MEASURED IN DEGREES;

        [DIGIT]!
```

```
CC_TC_MSG (FLOW):

        (TACTICAL_REPORT &
        ECM_REPORT &
        SWITCH_ACTION_DATA &
        MODE_ &
        TARGET_REPORT &
        INITIALIZATION_MSG)!


CONSECUTIVE_HIT (ELEMENT):

        - CONSEC ASSOCIATION HITS;
        - INTEGER FROM 1..15;

        DIGIT!
        2{DIGIT}2!


CONSECUTIVE_MISS (ELEMENT):

        - CONSEC ASSOCIATION MISSES;

        (DIGIT)!
        (DIGIT &
        DIGIT)!


CONSECUTIVE_OUTERGATE_COUNT (ELEMENT):

        "NUMBER OF CONSECUTIVE OUTERGATE COUNTS"

        (DIGIT)!


CORRELATION_COUNT (FLOW):

        (R_GATE_CNT!
        OUTERGATE_COUNT!
        INNERGATE_COUNT!
        CONSECUTIVE_OUTERGATE_COUNT)!


CURRENT_ASSOCIATED_REPORT_FILE (FLOW):

        REPORT_TO_TRACK_PAIR!


CURRENT_TIME (FLOW):

        TIME_!


DELETE_ (FLOW):

        "TRACK RECORD TO BE DELETED FROM TRACK STORAGE FILE"

        TRACK_REC!
```

```
DEVIATION_VECTOR (FLOW):

        VECTOR_!


DISPLAY_MSG (FLOW):

        THREAT_PRIORITY!
        OPERATOR_ALERT!
        HOLD_FIRE_MSG!


DROP_DATA (FLOW):

        DELETE_!


ECM_DATA (FLOW):

        JAM_STROBE_DATA!
        CHAFF_DATA!


ECM_REPORT (FLOW):

        TRACK_NO &
        ECM_DATA!


IFF_ALERT (ELEMENT):

        "IF MODE 3/A CODE OF ASSOCIATION REPORT CONTAINS AN
            AIRCRAFT, COMMUNICATION, ; HIJACK EMERGENCY CODE"

        DIGIT &
        DIGIT!


IFF_DATA (FLOW):

        VALID_CODE!
        INVALID_CODE!


IFF_RECORD (FLOW):

        IFF_DATA!


INDEX_ (ELEMENT):

        - USED TO INDICATE DEGREE OF MANEUVERABILITY OF
          TARGET;

        1!
        2!
        3!
        4!
        5!
```

```
INITIALIZATION_DATA (FLOW):

        TARGET_CONTROL_PARAMETER!
        TARGET_CONTROL_ALGORITHM!


INITIALIZATION_MSG (FLOW):

        ALGORITHM_INDICATOR &
        INITIALIZATION_DATA!


INITIAL_TRACK_RECORD (FLOW): !


INNERGATE_COUNT (ELEMENT):

        - INTEGER RANGE 0..15;

        DIGIT&
        2{DIGIT}2!


INVALID_CODE (ELEMENT):

        - UNFRIENDLY; !


LOCAL_RADAR_BUFF (FLOW):

        REPORT_NO &
        SECTOR_NO &
        RANGE_ &
        AZIMUTH_ &
        [IFF_DATA] &
        [ALTITUDE_] &
        TRACK_REC!


MODE_ (FLOW):

        TRACKING_MODE &
        BEACON_INTERROGATION &
        TRACK_INITIATION!


NEGATIVE_ACK (ELEMENT):

        - LACK OF ACKNOWLEDGEMENT INDICATOR; !


NON_ASSOC_TGT_REP (FLOW):

        "TARGET REPORT NOT ASSOCIATED WITH ANY CORRELATED
            REPORT"

        TARGET_REPORT!
```

```
NON_ASSOC_TRACK (FLOW):

        TRACK_NO!


NON_CORREL_TGT_REP (FLOW):

        "TARGET REPORT NOT CORRELATED WITH ANY OTHER TARGET
            REPORT"

        TARGET_REPORT!


NON_CORREL_TRACK (FLOW):

        TRACK_NO!


NON_MATCHING_TGT_REP (FLOW):

        NON_ASSOC_TGT_REP ;
        NON_CORREL_TGT_REP!


NON_MATCHING_TRACK (FLOW):

        NON_ASSOC_TRACK &
        NON_CORREL_TRACK!


OLD_ALTITUDE (FLOW):

        - ALTITUDE PRIOR TO SMOOTHING;

        ALTITUDE_!


OLD_INDEX (FLOW):

        - INDEX PRIOR TO PROCESSING BY 2.2.3.4;

        INDEX_!


OLD_POSITION (FLOW):

        - POSITION PRIOR TO SMOOTHING;

        POSITION_!


OLD_VELOCITY (FLOW):

        - VELOCITY PRIOR TO SMOOTHING;

        VELOCITY_!


OPERATOR_ALERT (FLOW):
```

```
        MERGE_ALERT!
        ENGAGEMENT_ALERT!
        IFF_ALERT!


OUTERGATE_COUNT (ELEMENT):

        - INTEGER RANGE 0..15;

        DIGIT &
        2{DIGIT}2!


POOR_TRACKING_STATUS_INDICATOR (FLOW):

        - STANDARD MESSAGE FORMAT; !


POSITION_ (FLOW):

        VECTOR_!


PREDICTED_POSITION (FLOW):

        - POSITION AFTER PREDICTION 2.2.3.4;

        POSITION_!


PREDICTED_VELOCITY (FLOW):

        VELOCITY_!


RADAR_REPORT (FLOW):

        TARGET_REPORT!


RADAR_REPORT_BUFFER (FLOW):

        {RADAR_REPORT}!


REMOTE_COMM_MSG (FLOW):

        PRIMARY_ASSIGN &
        SECONDARY_ASSIGN!
        COMMAND_MSG!


REMOTE_COMM_REPORT (FLOW):

        TRACK_NO &
        POSITION WITH REMOTE CENTER &
        IFF_DATA!
```

```
REMOTE_RADAR_BUFF (FLOW):

        REPORT_NO &
        SECTOR_NO &
        RANGE_ &
        AZIMUTH_ &
        [IFF_DATA] &
        [ALTITUDE_] &
        TRACK_REC!


REMOTE_TRACK_DATA (FLOW):

        TRACK_NO &
        POSITION WITH LOCAL SYSTEM COORDINATE CENTER &
        IFF_DATA!


REPORT_ALTITUDE (FLOW):

        ALTITUDE_!


REPORT_NO (ELEMENT):

        - INTEGER RANGE 1..NO OF TARGETS DETECTED;

        {DIGIT} CORRELATION_SCORE (ELEMENT): -INTEGER RANGE
            0..15; {DIGIT}!


REPORT_TO_TRACK_PAIR (FLOW):

        REPORT_NO &
        TRACK_NO!


RESULT_ (FLOW):

        INVALID_CODE !
        VALID_CODE!


R_GATE (ELEMENT):

        - BOOLEAN;

        'TRUE'!
        'FALSE'!


R_GATE_CNT (ELEMENT):

        - INTEGER RANGE 1..15;

        DIGIT &
        2{DIGIT}2!


SCAN_TIME (FLOW):
```

```
        - DEPENDS ON RADAR TYPE;
        - DETERMINED AT INITIALIZATION;
        - GIVEN IN SECONDS;

        '3' |
        '6' |
        '10'!


SECTOR_NO (ELEMENT):

        - INTEGER RANGE 1..20;
        - SECTOR IS 18 DEGREES IN SIZE;
        - 20 SECTORS;

        ('0' |
        '1' |
        '2') &
        (DIGIT)!


SMOOTHED_ALTITUDE (FLOW):

        - ALTITUDE AFTER SMOOTHING 2.2.3.3;

        ALTITUDE_!


SMOOTHED_DATA (FLOW):

        SMOOTHED_POSITION &
        SMOOTHED_VELOCITY &
        [SMOOTHED_ALTITUDE] &
        [SMOOTHING_INDEX] &
        TIME_OF_LAST_SMOOTHING!


SMOOTHED_POSITION (FLOW):

        POSITION_!


SMOOTHED_VELOCITY (FLOW):

        VELOCITY_!


SMOOTHING_CONSTANTS (FLOW):

        SMOOTH_POSITION_CONSTANT |
        SMOOTH_VELOCITY_CONSTANT!


SMOOTHING_CONSTANT_TABLE (FILE):

        "TABLE OF SMOOTHING CONSTANTS"

        {SMOOTHING_CONSTANTS}!
```

```
SMOOTHING_INDEX (ELEMENT):

        - INTEGER RANGE 1..5;

        '1'|
        '2'|
        '3'|
        '4'|
        '5'!


SMOOTH_POSITION_CONSTANT (ELEMENT):

        ('7/16'|
        '9/16'|
        '11/16'|
        '13/16'|
        '15/16')!


SMOOTH_VELOCITY_CONSTANT (ELEMENT):

        ('1/16'|
        '3/16'|
        '5/16'|
        '7/16'|
        '9/16')!


STATUS_ (FLOW):

        - TRACK STATUS; !


SWITCH_ACTION_DATA (FLOW):

        TRACK_DATA|
        ECM_DATA|
        TACTICAL_DATA|
        OPERATOR_REQUESTS!


SYSTEM_PARAMETER_FILE (FILE):

        SMOOTHING_CONSTANTS &
        SYS_MODE &
        SCAN_TIME!


SYS_MODE (FLOW):

        MODE_!


TACTICAL_DATA (FLOW):

        (TRACK_NO &
        THREAT_PRIORITY &
        SLACK_TIME)!
```

```
TACTICAL_REPORT (FLOW):

        TAC_RPT_HEADER &
        TAC_RPT_BODY!


TARGET_CONTROL_DATA (FLOW):

        SWITCH_ACTION_DATA !
        TRACK_DATA !
        ECM_DATA !
        TACTICAL_DATA!


TARGET_CONTROL_DATA_BASE (FILE):

        (TRACK_STORAGE_FILE &
        DEFENDED_POINT_FILE &
        FU_STATUS_FILE &
        SYSTEM_PARAMETER_FILE &
        JAM_STROBE_FILE &
        SMOOTHING_CONSTANT_TABLE &
        TRACK_PARAMETER_FILE &
        TRACK_ALGORITHM_FILE &
        THREAT_PARAMETER_FILE &
        WEAPON_ASGN &
        THREAT_ALGORITHM_FILE &
        WEAPON_ASSIGN_ALGORITHM_FILE &
        ALGORITHM_LIBRARY &
        SAFE_CORRIDOR_FILE &
        CLUTTER_MAP_FILE)!


TARGET_MSG (FLOW):

        (INITIALIZATION_MSG !
        TARGET_REPORT !
        ECM_REPORT !
        TACTICAL_REPORT!
        MODE_!
        SECTOR_PULSE!
        NORTH_SECTOR_PULSE)!


TARGET_REPORT (FLOW):

        LOCAL_RADAR_BUFF &
        REMOTE_RADAR_BUFF &
        REMOTE_COMM_REPORT!


TC_CC_MSG (FLOW):

        (ACK_ &
        TARGET_CONTROL_DATA &
        DISPLAY_MSG &
        TC_REM_COMM_MSG)!
```

```
TGT_REP_BUFF (FLOW):

        REPORT_TO_TRACK_PAIR &
        TRACK_QUALITY &
        NON_MATCHING_TGT_REP &
        ASSOC_DATA!


THREAT_PRIORITY (ELEMENT):

        - HIGHEST TO LOWEST PRIORITY OF THREAT;

        (DIGIT)!


TIME_ (ELEMENT):

        - HH MM SS SS;
        - HOURS, MINUTES, SECONDS AND HUNDREDTHS OF SECONDS;

        (0|
        1|
        2) &
        DIGIT &
        (0|
        1|
        2|
        3|
        4|
        5) &
        DIGIT &
        (0|
        1|
        2|
        3|
        4|
        5) &
        DIGIT &
        '.' &
        DIGIT &
        DIGIT!


TIME_FROM_LAST_SMOOTHING (FLOW):

        TIME_!


TIME_OF_LAST_SMOOTHING (FLOW):

        TIME_!


TRACK_ALTITUDE (FLOW):

        ALTITUDE_!
```



```
TRACK_DATA (FLOW):
```

```
        (STATUS_ &
        TRACK_NO &
        PREDICTED_POSITION &
        PREDICTED_VELOCITY &
        ALTITUDE_ &
        OUTERGATE_COUNT &
        INNERGATE_COUNT &
        ASSOCIATION_COUNT &
        SMOOTHING_INDEX &
        SMOOTHED_ALTITUDE &
        SMOOTHED_VELOCITY &
        SMOOTHED_POSITION &
        TIME_OF_LAST_SMOOTHING &
        DEVIATION_VECTOR)!


TRACK_NO (ELEMENT):

        - INTEGER RANGE 1..NO OF TRACKS IN SYSTEM CAPACITY;
        - CLASSIFIED;

        {DIGIT}!


TRACK_QUALITY (FLOW):

        - INTEGER RANGE 0..7;

        (0!
        1!
        2!
        3!
        4!
        5!
        6!
        7)!


TRACK_REC (FLOW):

        TRACK_DATA!


TRACK_STORAGE_FILE (FILE):

        {TRACK_DATA}!


UPDATED_TRACK_REC (FLOW):

        ADD_ !
        DELETE_!


UPDATE_DATA (FLOW):

        UPDATED_TRACK_REC!
```

```
VALID_CODE (ELEMENT):

        - FRIENDLY; !


VECTOR_ (FLOW):

        X_COORD &
        Y_COORD!


VELOCITY_ (ELEMENT): !


X_COORD (ELEMENT):

        {DIGIT}&
        ". "&
        {DIGIT}!


Y_COORD (ELEMENT):

        {DIGIT}&
        ". "&
        {DIGIT}!


SMOOTHING_ (PROCESS 2.2.3.3):

  FOR EACH REPORT_TO_TRACK_PAIR LOOP:
    DET_SMOOTHING_INDEX;
    SMOOTH_POSITION;
    SMOOTH_VELOCITY;
    IF REPORT_ALTITUDE IS PRESENT:
      SMOOTH_ALTITUDE.
    EN IF;
    UPDATE TRACK_STORAGE_FILE.
  OOP!



SMOOTH_POSTIION (PROCESS 2.2.3.3.1):

  FROM TRACK STORAGE FILE GET:
    OLD_POSITION;
    DEVIATION_VECTOR.
  FROM SMOOTHING_CONSTANT_TABLE GET:
    SMOOTH_POSITION_CONSTANT.
  SMOOTH_POSITION = OLD_POSITION + SMOOTH_POSITION_CONSTANT
      * DEVIATION_VECTOR!



SMOOTH_VELOCITY (PROCESS 2.2.3.3.2):
```

```
      OLD_VELOCITY;
      DEVIATION_VECTOR.
    FROM SMOOTHING CONSTANT GET:
      SMOOTH_VELOCITY_CONSTANT.
    SMOOTHED_VELOCITY = OLD_VELOCITY + (
        SMOOTH_VELOCITY_CONSTANT / SCAN_TIME) *
        DEVIATION_VECTOR.
    - THIS FORMULA WILL BE COMPLETED BY ADDING THE RADAR SCAN;
    -TIME DURING SYSTEM INITIALIZATION; !
```

```
SMOOTH_ALTITUDE (PROCESS 2.2.3.3.3):

  IF RADAR_REPORT CONTAINS REPORT_ALTITUDE THEN:
    CASE 1:
      ALTITUDE_GATE IS AT MAXIMUM;
      SMOOTHED_ALTITUDE = REPORT_ALTITUDE.
    CASE 2:
      ALTITUDE_GATE IS UNDER MAXIMUM;
      SMOOTHED_ALTITUDE = (REPORT_ALTITUDE + OLD_ALTITUDE) /
          2.
    END CASE.
  ELSE:
    SMOOTHED_ALTITUDE = OLD_ALTITUDE.
  END IF!
```

```
DET_SMOOTHING_INDEX (PROCESS 2.2.3.3.4):

  - INDICES TO ACCESS THE SMOOTHING CONSTANTS RANGE FROM;
  - 1 TO 5- AN INDEX OF 1 REPRESENTS A NON MANEUVERING
    TRACK;
  - AND SHALL HAVE THE HEAVIEST SMOOTHING PERFORMED;

  CASE 1:
    TRACK IS NONMANEUVERING;
    - INDEX IS 1;
    IF OUTERGATE HAS TWO CONSECUTIVE HITS THEN:
      INDEX = 5;
      RESET TO 0.
    ELSE:
      INDEX = 1.
  CASE 2:
    TRACK IS MANEUVERING;
    - INDEX IS 2,3,4,5;
    IF TRACK CORRELATED IN R_GATE:
      INDEX = INDEX -1.
    END IF;
    IF TRACK CORRLATED IS OUTERGATE:
      INDEX = INDEX + 1.
    END IF!
```

```
PREDICTION_ (PROCESS 2.2.3.4):

  FOR EACH TRACK LOOP:
    FROM TRACK_STORAGE_FILE GET:
      SMOOTHED_POSITION;
      SMOOTHED_VELOCITY;
      CALCULATE TIME_FROM_LAST_SMOOTHING;
      PREDICTED_POSITION = SMOOTHED_POSITION + (
          TIME_FROM_LAST_SMOOTHING + SCAN_TIME) *
          SMOOTHED_VELOCITY;
      UPDATE TRACK_STORAGE_FILE!



EOI ENCOUNTERED.
```

```
    ACK_ (FLOW):

                  ~~ MAKES REFERENCES TO ~~
          AFFIRM_ACK . . . . . . . . . . . (ELEMENT)
          NEG_ACK . . . . . . . . . . . . . (UNDEF)

                   ~~ IS REFERENCED BY ~~
          TC_CC_MSG . . . . . . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    ADD_ (FLOW):

                  ~~ MAKES REFERENCES TO ~~
          TRACK_REC . . . . . . . . . . . . (FLOW)

                   ~~ IS REFERENCED BY ~~
          UPDATED_TRACK_REC . . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    AFFIRM_ACK (ELEMENT):

                  ~~ MAKES REFERENCES TO ~~


                   ~~ IS REFERENCED BY ~~
          ACK_ . . . . . . . . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    ALGORITHM_INDICATOR (UNDEF):

                  ~~ MAKES REFERENCES TO ~~


                   ~~ IS REFERENCED BY ~~
          INITIALIZATION_MSG . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    ALGORITHM_LIBRARY (UNDEF):

                  ~~ MAKES REFERENCES TO ~~


                   ~~ IS REFERENCED BY ~~
          TARGET_CONTROL_DATA_BASE . . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -
```

ALTITUDE_ (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

```
LOCAL_RADAR_BUFF . . . . . . . . (FLOW)
```



```
OLD_ALTITUDE . . . . . . . . . . (FLOW)
REMOTE_RADAR_BUFF . . . . . . . . (FLOW)
REPORT_ALTITUDE . . . . . . . . . (FLOW)
SMOOTHED_ALTITUDE . . . . . . . . (FLOW)
TRACK_ALTITUDE . . . . . . . . . (FLOW)
TRACK_DATA . . . . . . . . . . . (FLOW)
```

- -

ALTITUDE_GATE (FLOW):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

```
SMOOTH_ALTITUDE . . . . . . . . . (PROCESS 2.2.3.3.3)
```

- -

ASSOCIATION_COUNT (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

```
TRACK_DATA . . . . . . . . . . . (FLOW)
```

- -

ASSOC_DATA (FLOW):

~~ MAKES REFERENCES TO ~~

```
CONSECUTIVE_HIT . . . . . . . . . (ELEMENT)
CONSECUTIVE_MISS . . . . . . . . (ELEMENT)
TRACK_NO . . . . . . . . . . . . (ELEMENT)
```

~~ IS REFERENCED BY ~~

```
TGT_REP_BUFF . . . . . . . . . . (FLOW)
```

- -

AZIMUTH_ (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

```
LOCAL_RADAR_BUFF . . . . . . . . (FLOW)
REMOTE_RADAR_BUFF . . . . . . . . (FLOW)
```

- -

BEACON_INTERROGATION (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~


```
MODE_ . . . . . . . . . . . . . . (FLOW)
```

- -

CC_TC_MSG (FLOW):

~~ MAKES REFERENCES TO ~~

```
ECM_REPORT . . . . . . . . . . . (FLOW)
INITIALIZATION_MSG . . . . . . . (FLOW)
MODE_ . . . . . . . . . . . . . . (FLOW)
SWITCH_ACTION_DATA . . . . . . . (FLOW)
TACTICAL_REPORT . . . . . . . . . (FLOW)
TARGET_REPORT . . . . . . . . . . (FLOW)
```

~~ IS REFERENCED BY ~~

- -

CHAFF_DATA (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

```
ECM_DATA . . . . . . . . . . . . (FLOW)
```

- -

CLUTTER_MAP_FILE (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

```
TARGET_CONTROL_DATA_BASE . . . . (FILE)
```

- -

COMMAND_MSG (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
REMOTE_COMM_MSG . . . . (FLOW)

- -
- - - - -

CONSECUTIVE_HIT (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
ASSOC_DATA . . . . (FLOW)

- -
- - - -

CONSECUTIVE_MISS (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
ASSOC_DATA . . . . (FLOW)

- -
- - - -

CONSECUTIVE_OUTERGATE_COUNT (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
CORRELATION_COUNT . . . . (FLOW)

- -
- - - -

CORRELATION_COUNT (FLOW):

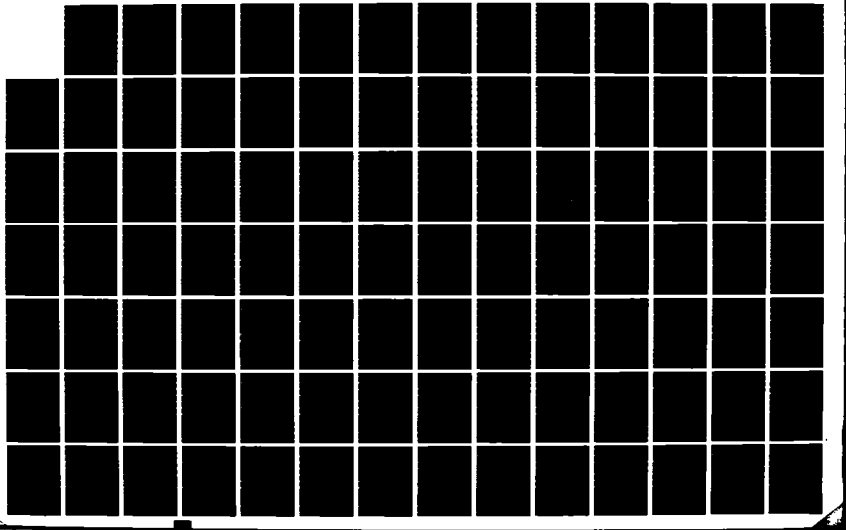
~~ MAKES REFERENCES TO ~~
CONSECUTIVE_OUTERGATE_COUNT . . (ELEMENT)
INNERGATE_COUNT . . . . (ELEMENT)
OUTERGATE_COUNT . . . . (ELEMENT)
R_GATE_CNT . . . . (ELEMENT)

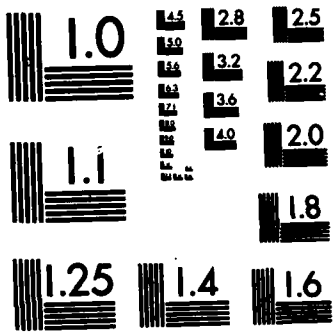
~~ IS REFERENCED BY ~~

- -
- - - -

AD-A123 308 LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE MISSILE MINDER AN/TSQ-73 USING T..(U) CONTROL DATA CORP SHREWSBURY NJ GOVERNMENT SYSTEMS 09 NOV 82 6/6

UNCLASSIFIED DAAK80-81-C-0107 F/G 9/2 NL





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

CORRELATION_SCORE (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

- -
- - - -

CURRENT_ASSOCIATED_REPORT_FILE (FLOW):

~~ MAKES REFERENCES TO ~~

REPORT_TO_TRACK_PAIR . . . . . (FLOW)

~~ IS REFERENCED BY ~~

- -
- - - -

CURRENT_TIME (FLOW):

~~ MAKES REFERENCES TO ~~

TIME_ . . . . (ELEMENT)

~~ IS REFERENCED BY ~~

- -
- - - -

DEFENDED_POINT_FILE (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

TARGET_CONTROL_DATA_BASE . . . (FILE)

- -
- - - -

DELETE_ (FLOW):

~~ MAKES REFERENCES TO ~~

TRACK_REC . . . . (FLOW)

~~ IS REFERENCED BY ~~

DROP_DATA . . . . (FLOW)
UPDATED_TRACK_REC . . . . (FLOW)

- -
- - - -

DET_SMOOTHING_INDEX (PROCESS 2.2.3.3.4):

~~ MAKES REFERENCES TO ~~

R_GATE . . . . (ELEMENT)

~~ IS REFERENCED BY ~~

SMOOTHING_ . . . . (PROCESS 2.2.3.3)

- -
- - - - -

DEVIATION_VECTOR (FLOW):

~~ MAKES REFERENCES TO ~~

VECTOR_ . . . . (FLOW)

~~ IS REFERENCED BY ~~

SMOOTH_POSTIION . . . . (PROCESS 2.2.3.3.1)
SMOOTH_VELOCITY . . . . (PROCESS 2.2.3.3.2)
TRACK_DATA . . . . (FLOW)

- -
- - - -

DISPLAY_MSG (FLOW):

~~ MAKES REFERENCES TO ~~

HOLD_FIRE_MSG . . . . (UNDEF)
OPERATOR_ALERT . . . . (FLOW)
THREAT_PRIORITY . . . . (ELEMENT)

~~ IS REFERENCED BY ~~

TC_CC_MSG . . . . (FLOW)

- -
- - - -

DROP_DATA (FLOW):

~~ MAKES REFERENCES TO ~~

DELETE_ . . . . (FLOW)

~~ IS REFERENCED BY ~~

- -
- - - -

ECM_DATA (FLOW):

~~ MAKES REFERENCES TO ~~

CHAFF_DATA . . . . (UNDEF)
JAM_STROBE_DATA . . . . (UNDEF)

~~ IS REFERENCED BY ~~

ECM_REPORT . . . . (FLOW)
SWITCH_ACTION_DATA . . . . (FLOW)
TARGET_CONTROL_DATA . . . . (FLOW)

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    ECM_REPORT (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            ECM_DATA . . . . . . . . . . . (FLOW)
            TRACK_NO . . . . . . . . . . . (ELEMENT)

                     ~~ IS REFERENCED BY ~~
            CC_TC_MSG . . . . . . . . . . . (FLOW)
            TARGET_MSG . . . . . . . . . . (FLOW)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    ENGAGEMENT_ALERT (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                     ~~ IS REFERENCED BY ~~
            OPERATOR_ALERT . . . . . . . . (FLOW)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    FU_STATUS_FILE (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                     ~~ IS REFERENCED BY ~~
            TARGET_CONTROL_DATA_BASE . . . (FILE)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -
```



```
    HOLD_FIRE_MSG (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                     ~~ IS REFERENCED BY ~~
            DISPLAY_MSG . . . . . . . . . . (FLOW)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    IFF_ALERT (ELEMENT):

                    ~~ MAKES REFERENCES, TO ~~


                     ~~ IS REFERENCED BY ~~
            OPERATOR_ALERT . . . . . . . . (FLOW)
```

- -
- - - -

IFF_DATA (FLOW):

~~ MAKES REFERENCES TO ~~

INVALID_CODE . . . . (ELEMENT)
VALID_CODE . . . . (ELEMENT)

~~ IS REFERENCED BY ~~

IFF_RECORD . . . . (FLOW)
LOCAL_RADAR_BUFF . . . . (FLOW)
REMOTE_COMM_REPORT . . . . (FLOW)
REMOTE_RADAR_BUFF . . . . (FLOW)
REMOTE_TRACK_DATA . . . . (FLOW)

- -
- - - -

IFF_RECORD (FLOW):

~~ MAKES REFERENCES TO ~~

IFF_DATA . . . . (FLOW)

~~ IS REFERENCED BY ~~

- -
- - - -

INDEX_ (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

OLD_INDEX . . . . (FLOW)

- -
- - - -

INITIALIZATION_DATA (FLOW):

~~ MAKES REFERENCES TO ~~

TARGET_CONTROL_AL(?RITHM . . . (UNDEF)
TARGET_CONTROL_PARAMETER . . . (UNDEF)

~~ IS REFERENCED BY ~~

INITIALIZATION_MSG . . . . (FLOW)

- -
- - - -

INITIALIZATION_MSG (FLOW):

~~ MAKES REFERENCES TO ~~

ALGORITHM_INDICATOR . . . . (UNDEF)
INITIALIZATION_DATA . . . . (FLOW)

~~ IS REFERENCED BY ~~
CC_TC_MSG . . . . (FLOW)
TARGET_MSG . . . . (FLOW)

- -

INITIAL_TRACK_RECORD (FLOW):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

- -

INNERGATE_COUNT (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
CORRELATION_COUNT . . . . (FLOW)
TRACK_DATA . . . . (FLOW)

- -

INVALID_CODE (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
IFF_DATA . . . . (FLOW)
RESULT_ . . . . (FLOW)

- -

JAM_STROBE_DATA (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

ECM_DATA . . . . (FLOW)

- -

JAM_STROBE_FILE (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

```
TARGET_CONTROL_DATA_BASE . . . (FILE)
```

- -

LOCAL_RADAR_BUFF (FLOW):

~~ MAKES REFERENCES TO ~~

```
ALTITUDE_ . . . . . . . . . . . . . (ELEMENT)
AZIMUTH_  . . . . . . . . . . . . . (ELEMENT)
IFF_DATA  . . . . . . . . . . . . . (FLOW)
RANGE_  . . . . . . . . . . . . . . (UNDEF)
REPORT_NO . . . . . . . . . . . . . (ELEMENT)
SECTOR_NO . . . . . . . . . . . . . (ELEMENT)
TRACK_REC . . . . . . . . . . . . . (FLOW)
```

~~ IS REFERENCED BY ~~

```
TARGET_REPORT . . . . . . . . . . . (FLOW)
```

- -

MERGE_ALERT (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

```
OPERATOR_ALERT  . . . . . . . . . . (FLOW)
```

- -

MODE_ (FLOW):

~~ MAKES REFERENCES TO ~~

```
BEACON_INTERROGATION  . . . . . . . (UNDEF)
TRACKING_MODE . . . . . . . . . . . (UNDEF)
TRACK_INITIATION  . . . . . . . . . (UNDEF)
```

~~ IS REFERENCED BY ~~

```
CC_TC_MSG . . . . . . . . . . . . . (FLOW)
SYS_MODE  . . . . . . . . . . . . . (FLOW)
TARGET_MSG  . . . . . . . . . . . . (FLOW)
```

- -

NEGATIVE_ACK (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

NEG_ACK (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

ACK_ . . . . . (FLOW)

NON_ASSOC_TGT_REP (FLOW):

~~ MAKES REFERENCES TO ~~

TARGET_REPORT . . . . . (FLOW)

~~ IS REFERENCED BY ~~

NON_MATCHING_TGT_REP . . . . . (FLOW)

NON_ASSOC_TRACK (FLOW):

~~ MAKES REFERENCES TO ~~

TRACK_NO . . . . . (ELEMENT)

~~ IS REFERENCED BY ~~

NON_MATCHING_TRACK . . . . . (FLOW)

NON_CORREL_TGT_REP (FLOW):

~~ MAKES REFERENCES TO ~~

TARGET_REPORT . . . . . (FLOW)

~~ IS REFERENCED BY ~~

NON_MATCHING_TGT_REP . . . . . (FLOW)

NON_CORREL_TRACK (FLOW):

~~ MAKES REFERENCES TO ~~

TRACK_NO . . . . . (ELEMENT)

~~ IS REFERENCED BY ~~

NON_MATCHING_TRACK . . . . . (FLOW)

```
- - - -

     NON_MATCHING_TGT_REP (FLOW):

1       CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD                PA
GE 11

                    ~~ MAKES REFERENCES TO ~~
               NON_ASSOC_TGT_REP . . . . . (FLOW)
               NON_CORREL_TGT_REP . . . . . (FLOW)

                     ~~ IS REFERENCED BY ~~
               TGT_REP_BUFF . . . . . (FLOW)

     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

     NON_MATCHING_TRACK (FLOW):

                    ~~ MAKES REFERENCES TO ~~
               NON_ASSOC_TRACK . . . . . (FLOW)
               NON_CORREL_TRACK . . . . . (FLOW)

                     ~~ IS REFERENCED BY ~~


     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

     NORTH_SECTOR_PULSE (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                     ~~ IS REFERENCED BY ~~
               TARGET_MSG . . . . . (FLOW)

     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

     OLD_ALTITUDE (FLOW):

                    ~~ MAKES REFERENCES TO ~~
               ALTITUDE_ . . . . . (ELEMENT)

                     ~~ IS REFERENCED BY ~~
               SMOOTH_ALTITUDE . . . . . (PROCESS 2.2.3.3.3)

     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

     OLD_INDEX (FLOW):

                    ~~ MAKES REFERENCES TO ~~
               INDEX_ . . . . . (ELEMENT)

                     ~~ IS REFERENCED BY ~~

     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
- - - -

    OLD_POSITION (FLOW):

                          ~~ MAKES REFERENCES TO ~~
            POSITION_ . . . . . . . . . . . . . (FLOW)

                           ~~ IS REFERENCED BY ~~
            SMOOTH_POSTIION . . . . . . . . .   (PROCESS 2.2.3.3.1)

1      CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD                PA
GE 12

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    OLD_VELOCITY (FLOW):

                          ~~ MAKES REFERENCES TO ~~
            VELOCITY_ . . . . . . . . . . . . . (ELEMENT)

                           ~~ IS REFERENCED BY ~~
            SMOOTH_VELOCITY . . . . . . . . .   (PROCESS 2.2.3.3.2)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    OPERATOR_ALERT (FLOW):

                          ~~ MAKES REFERENCES TO ~~
            ENGAGEMENT_ALERT  . . . . . . . .   (UNDEF)
            IFF_ALERT . . . . . . . . . . . .   (ELEMENT)
            MERGE_ALERT . . . . . . . . . . .   (UNDEF)

                           ~~ IS REFERENCED BY ~~
            DISPLAY_MSG . . . . . . . . . . .   (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    OPERATOR_REQUESTS (UNDEF):

                          ~~ MAKES REFERENCES TO ~~


                           ~~ IS REFERENCED BY ~~
            SWITCH_ACTION_DATA  . . . . . . .   (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

=   OUTERGATE_COUNT (ELEMENT):

                          ~~ MAKES REFERENCES TO ~~


                           ~~ IS REFERENCED BY ~~
            CORRELATION_COUNT . . . . . . . .   (FLOW)
            TRACK_DATA  . . . . . . . . . . .   (FLOW)
```

- -

POOR_TRACKING_STATUS_INDICATOR (FLOW):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

- -

POSITION_ (FLOW):

~~ MAKES REFERENCES TO ~~


```
          VECTOR_ . . . . . . . . . . . . . . (FLOW)

                   ~~ IS REFERENCED BY ~~
          OLD_POSITION  . . . . . . . . . . . (FLOW)
          PREDICTED_POSITION  . . . . . . . . (FLOW)
          SMOOTHED_POSITION . . . . . . . . . (FLOW)
```

- -

PREDICTED_POSITION (FLOW):

```
                   ~~ MAKES REFERENCES TO ~~
          POSITION_ . . . . . . . . . . . . . (FLOW)

                   ~~ IS REFERENCED BY ~~
          PREDICTION_ . . . . . . . . . . . . (PROCESS 2.2.3.4)
          TRACK_DATA  . . . . . . . . . . . . (FLOW)
```

- -

PREDICTED_VELOCITY (FLOW):

```
                   ~~ MAKES REFERENCES TO ~~
          VELOCITY_ . . . . . . . . . . . . . (ELEMENT)

                   ~~ IS REFERENCED BY ~~
          TRACK_DATA  . . . . . . . . . . . . (FLOW)
```

- -

PREDICTION_ (PROCESS 2.2.3.4):

```
                   ~~ MAKES REFERENCES TO ~~
          PREDICTED_POSITION  . . . . . . . . (FLOW)
          SCAN_TIME . . . . . . . . . . . . . (FLOW)
          SMOOTHED_POSITION . . . . . . . . . (FLOW)
          SMOOTHED_VELOCITY . . . . . . . . . (FLOW)
```

```
        TIME_FROM_LAST_SMOOTHING . . . (FLOW)
        TRACK_STORAGE_FILE . . . . . . (FILE)

                  ~~ IS REFERENCED BY ~~


    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    PRIMARY_ASSIGN (UNDEF):

                 ~~ MAKES REFERENCES TO ~~


                  ~~ IS REFERENCED BY ~~
        REMOTE_COMM_MSG . . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    RADAR_REPORT (FLOW):

                 ~~ MAKES REFERENCES TO ~~
1      CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD              PA
GE 14

        TARGET_REPORT . . . . . . . . . (FLOW)

                  ~~ IS REFERENCED BY ~~
        RADAR_REPORT_BUFFER . . . . . . (FLOW)
        SMOOTH_ALTITUDE . . . . . . . . (PROCESS 2.2.3.3.3)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    RADAR_REPORT_BUFFER (FLOW):

                 ~~ MAKES REFERENCES TO ~~
        RADAR_REPORT . . . . . . . . . (FLOW)

                  ~~ IS REFERENCED BY ~~


    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    RANGE_ (UNDEF):

                 ~~ MAKES REFERENCES TO ~~


                  ~~ IS REFERENCED BY ~~
        LOCAL_RADAR_BUFF . . . . . . . (FLOW)
        REMOTE_RADAR_BUFF . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    REMOTE_COMM_MSG (FLOW):
```

```
                    ~~ MAKES REFERENCES TO ~~
          COMMAND_MSG . . . . . (UNDEF)
          PRIMARY_ASSIGN . . . . . (UNDEF)
          SECONDARY_ASSIGN . . . . . (UNDEF)

                    ~~ IS REFERENCED BY ~~

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

     REMOTE_COMM_REPORT (FLOW):

                    ~~ MAKES REFERENCES TO ~~
          IFF_DATA . . . . . (FLOW)
          TRACK_NO . . . . . (ELEMENT)

                    ~~ IS REFERENCED BY ~~
          TARGET_REPORT . . . . . (FLOW)

     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

     REMOTE_RADAR_BUFF (FLOW):

                    ~~ MAKES REFERENCES TO ~~
          ALTITUDE_ . . . . . (ELEMENT)
          AZIMUTH_ . . . . . (ELEMENT)
          IFF_DATA . . . . . (FLOW)
1       CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD                PA
GE 15

          RANGE_ . . . . . (UNDEF)
          REPORT_NO . . . . . (ELEMENT)
          SECTOR_NO . . . . . (ELEMENT)
          TRACK_REC . . . . . (FLOW)

                    ~~ IS REFERENCED BY ~~
          TARGET_REPORT . . . . . (FLOW)

     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

     REMOTE_TRACK_DATA (FLOW):

                    ~~ MAKES REFERENCES TO ~~
          IFF_DATA . . . . . (FLOW)
          TRACK_NO . . . . . (ELEMENT)

                    ~~ IS REFERENCED BY ~~


     - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

     REPORT_ALTITUDE (FLOW):

                    ~~ MAKES REFERENCES TO ~~
          ALTITUDE_ . . . . . (ELEMENT)
```

```
                    ~~ IS REFERENCED BY ~~
            SMOOTHING_ . . . . . (PROCESS 2.2.3.3)
            SMOOTH_ALTITUDE . . . . . (PROCESS 2.2.3.3.3)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    REPORT_NO (ELEMENT):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            LOCAL_RADAR_BUFF . . . . . (FLOW)
            REMOTE_RADAR_BUFF . . . . . (FLOW)
            REPORT_TO_TRACK_PAIR . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    REPORT_TO_TRACK_PAIR (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            REPORT_NO . . . . . (ELEMENT)
            TRACK_NO . . . . . (ELEMENT)

                    ~~ IS REFERENCED BY ~~
            CURRENT_ASSOCIATED_REPORT_FILE (FLOW)
            SMOOTHING_ . . . . . (PROCESS 2.2.3.3)
            TGT_REP_BUFF . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    RESULT_ (FLOW):
1       CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD                PA
GE 16


                    ~~ MAKES REFERENCES TO ~~
            INVALID_CODE . . . . . (ELEMENT)
            VALID_CODE . . . . . (ELEMENT)

                    ~~ IS REFERENCED BY ~~


    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    R_GATE (ELEMENT):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            DET_SMOOTHING_INDEX . . . . . (PROCESS 2.2.3.3.4)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -
```

R_GATE_CNT (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
CORRELATION_COUNT . . . . . (FLOW)

SAFE_CORRIDOR_FILE (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
TARGET_CONTROL_DATA_BASE . . . (FILE)

SCAN_TIME (FLOW):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
PREDICTION_ . . . . . (PROCESS 2.2.3.4)
SMOOTH_VELOCITY . . . . . (PROCESS 2.2.3.3.2)
SYSTEM_PARAMETER_FILE . . . . . (FILE)

SECONDARY_ASSIGN (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

REMOTE_COMM_MSG . . . . . (FLOW)

SECTOR_NO (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
LOCAL_RADAR_BUFF . . . . . (FLOW)
REMOTE_RADAR_BUFF . . . . . (FLOW)

```
SECTOR_PULSE (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                     ~~ IS REFERENCED BY ~~
        TARGET_MSG  . . . . . . . . . . .  (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

SLACK_TIME (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                     ~~ IS REFERENCED BY ~~
        TACTICAL_DATA . . . . . . . . . .  (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

SMOOTHED_ALTITUDE (FLOW):

                    ~~ MAKES REFERENCES TO ~~
        ALTITUDE_ . . . . . . . . . . . .  (ELEMENT)

                     ~~ IS REFERENCED BY ~~
        SMOOTHED_DATA . . . . . . . . . .  (FLOW)
        SMOOTH_ALTITUDE . . . . . . . . .  (PROCESS 2.2.3.3.3)
        TRACK_DATA  . . . . . . . . . . .  (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

SMOOTHED_DATA (FLOW):

                    ~~ MAKES REFERENCES TO ~~
        SMOOTHED_ALTITUDE . . . . . . . .  (FLOW)
        SMOOTHED_POSITION . . . . . . . .  (FLOW)
        SMOOTHED_VELOCITY . . . . . . . .  (FLOW)
        SMOOTHING_INDEX . . . . . . . . .  (ELEMENT)
        TIME_OF_LAST_SMOOTHING  . . . . .  (FLOW)

                     ~~ IS REFERENCED BY ~~
```



```
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

SMOOTHED_POSITION (FLOW):

                    ~~ MAKES REFERENCES TO ~~
        POSITION_ . . . . . . . . . . . .  (FLOW)

                     ~~ IS REFERENCED BY ~~
```

```
            PREDICTION_ . . . . . (PROCESS 2.2.3.4)
            SMOOTHED_DATA . . . . . (FLOW)
            TRACK_DATA  . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTHED_VELOCITY (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            VELOCITY_ . . . . . (ELEMENT)

                    ~~ IS REFERENCED BY ~~
            PREDICTION_ . . . . . (PROCESS 2.2.3.4)
            SMOOTHED_DATA . . . . . (FLOW)
            SMOOTH_VELOCITY . . . . . (PROCESS 2.2.3.3.2)
            TRACK_DATA  . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTHING_ (PROCESS 2.2.3.3):

                    ~~ MAKES REFERENCES TO ~~
            DET_SMOOTHING_INDEX . . . . . (PROCESS 2.2.3.3.4)
            REPORT_ALTITUDE . . . . . (FLOW)
            REPORT_TO_TRACK_PAIR  . . . . . (FLOW)
            SMOOTH_ALTITUDE . . . . . (PROCESS 2.2.3.3.3)
            SMOOTH_POSITION . . . . . (UNDEF)
            SMOOTH_VELOCITY . . . . . (PROCESS 2.2.3.3.2)
            TRACK_STORAGE_FILE  . . . . . (FILE)

                    ~~ IS REFERENCED BY ~~


    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTHING_CONSTANTS (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            SMOOTH_POSITION_CONSTANT  . . . (ELEMENT)
            SMOOTH_VELOCITY_CONSTANT  . . . (ELEMENT)

                    ~~ IS REFERENCED BY ~~
            SMOOTHING_CONSTANT_TABLE  . . . (FILE)
            SYSTEM_PARAMETER_FILE . . . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTHING_CONSTANT_TABLE (FILE):

1       CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD              PA
GE 19

                    ~~ MAKES REFERENCES TO ~~
            SMOOTHING_CONSTANTS . . . . . (FLOW)

                    ~~ IS REFERENCED BY ~~
```

```
            SMOOTH_POSTIION . . . . . (PROCESS 2.2.3.3.1)
            TARGET_CONTROL_DATA_BASE . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTHING_INDEX (ELEMENT):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            SMOOTHED_DATA . . . . . (FLOW)
            TRACK_DATA . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTH_ALTITUDE (PROCESS 2.2.3.3.3):

                    ~~ MAKES REFERENCES TO ~~
            ALTITUDE_GATE . . . . . (FLOW)
            OLD_ALTITUDE . . . . . (FLOW)
            RADAR_REPORT . . . . . (FLOW)
            REPORT_ALTITUDE . . . . . (FLOW)
            SMOOTHED_ALTITUDE . . . . . (FLOW)

                    ~~ IS REFERENCED BY ~~
            SMOOTHING_ . . . . . (PROCESS 2.2.3.3)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTH_POSITION (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            SMOOTHING_ . . . . . (PROCESS 2.2.3.3)
            SMOOTH_POSTIION . . . . . (PROCESS 2.2.3.3.1)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTH_POSITION_CONSTANT (ELEMENT):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            SMOOTHING_CONSTANTS . . . . . (FLOW)
            SMOOTH_POSTIION . . . . . (PROCESS 2.2.3.3.1)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SMOOTH_POSTIION (PROCESS 2.2.3.3.1):
```

~~ MAKES REFERENCES TO ~~

DEVIATION_VECTOR . . . . (FLOW)
OLD_POSITION . . . . (FLOW)
SMOOTHING_CONSTANT_TABLE . . . (FILE)
SMOOTH_POSITION . . . . (UNDEF)
SMOOTH_POSITION_CONSTANT . . . (ELEMENT)

~~ IS REFERENCED BY ~~

- -

SMOOTH_VELOCITY (PROCESS 2.2.3.3.2):

~~ MAKES REFERENCES TO ~~

DEVIATION_VECTOR . . . . (FLOW)
OLD_VELOCITY . . . . (FLOW)
SCAN_TIME . . . . (FLOW)
SMOOTHED_VELOCITY . . . . (FLOW)
SMOOTH_VELOCITY_CONSTANT . . . (ELEMENT)

~~ IS REFERENCED BY ~~

SMOOTHING_ . . . . (PROCESS 2.2.3.3)

- -

SMOOTH_VELOCITY_CONSTANT (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

SMOOTHING_CONSTANTS . . . . (FLOW)
SMOOTH_VELOCITY . . . . (PROCESS 2.2.3.3.2)

- -

STATUS_ (FLOW):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

TRACK_DATA . . . . (FLOW)

- -

SWITCH_ACTION_DATA (FLOW):

~~ MAKES REFERENCES TO ~~

ECM_DATA . . . . (FLOW)
OPERATOR_REQUESTS . . . . (UNDEF)
TACTICAL_DATA . . . . (FLOW)
TRACK_DATA . . . . (FLOW)

```
                    ~~ IS REFERENCED BY ~~
            CC_TC_MSG . . . . . (FLOW)
            TARGET_CONTROL_DATA . . . . . (FLOW)
1       CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD                PA
GE 21


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SYSTEM_PARAMETER_FILE (FILE):

                    ~~ MAKES REFERENCES TO ~~
            SCAN_TIME . . . . . (FLOW)
            SMOOTHING_CONSTANTS . . . . . (FLOW)
            SYS_MODE . . . . . (FLOW)

                    ~~ IS REFERENCED BY ~~
            TARGET_CONTROL_DATA_BASE . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    SYS_MODE (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            MODE_ . . . . . (FLOW)

                    ~~ IS REFERENCED BY ~~
            SYSTEM_PARAMETER_FILE . . . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TACTICAL_DATA (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            SLACK_TIME . . . . . (UNDEF)
            THREAT_PRIORITY . . . . . (ELEMENT)
            TRACK_NO . . . . . (ELEMENT)

                    ~~ IS REFERENCED BY ~~
            SWITCH_ACTION_DATA . . . . . (FLOW)
            TARGET_CONTROL_DATA . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TACTICAL_REPORT (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            TAC_RPT_BODY . . . . . (UNDEF)
            TAC_RPT_HEADER . . . . . (UNDEF)

                    ~~ IS REFERENCED BY ~~
            CC_TC_MSG . . . . . (FLOW)
            TARGET_MSG . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

- - - -

TAC_RPT_BODY (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
TACTICAL_REPORT . . . . (FLOW)

- -
- - - -

TAC_RPT_HEADER (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
TACTICAL_REPORT . . . . (FLOW)

- -
- - - -

TARGET_CONTROL_ALGORITHM (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
INITIALIZATION_DATA . . . . (FLOW)

- -
- - - -

TARGET_CONTROL_DATA (FLOW):

~~ MAKES REFERENCES TO ~~
ECM_DATA . . . . (FLOW)
SWITCH_ACTION_DATA . . . . (FLOW)
TACTICAL_DATA . . . . (FLOW)
TRACK_DATA . . . . (FLOW)

~~ IS REFERENCED BY ~~
TC_CC_MSG . . . . (FLOW)

- -
- - - -

TARGET_CONTROL_DATA_BASE (FILE):

~~ MAKES REFERENCES TO ~~
ALGORITHM_LIBRARY . . . . (UNDEF)
CLUTTER_MAP_FILE . . . . (UNDEF)
DEFENDED_POINT_FILE . . . . (UNDEF)
FU_STATUS_FILE . . . . (UNDEF)
JAM_STROBE_FILE . . . . (UNDEF)

```
            SAFE_CORRIDOR_FILE . . . . . (UNDEF)
            SMOOTHING_CONSTANT_TABLE . . . (FILE)
            SYSTEM_PARAMETER_FILE . . . . . (FILE)
            THREAT_ALGORITHM_FILE . . . . . (UNDEF)
            THREAT_PARAMETER_FILE . . . . . (UNDEF)
            TRACK_ALGORITHM_FILE . . . . . (UNDEF)
            TRACK_PARAMETER_FILE . . . . . (UNDEF)
            TRACK_STORAGE_FILE . . . . . (FILE)
            WEAPON_ASGN . . . . . (UNDEF)
            WEAPON_ASSIGN_ALGORITHM_FILE . (UNDEF)

                    ~~ IS REFERENCED BY ~~

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -
1      CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD              PA
GE 23


    TARGET_CONTROL_PARAMETER (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            INITIALIZATION_DATA . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TARGET_MSG (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            ECM_REPORT . . . . . (FLOW)
            INITIALIZATION_MSG . . . . . (FLOW)
            MODE_ . . . . . (FLOW)
            NORTH_SECTOR_PULSE . . . . . (UNDEF)
            SECTOR_PULSE . . . . . (UNDEF)
            TACTICAL_REPORT . . . . . (FLOW)
            TARGET_REPORT . . . . . (FLOW)

                    ~~ IS REFERENCED BY ~~


    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TARGET_REPORT (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            LOCAL_RADAR_BUFF . . . . . (FLOW)
            REMOTE_COMM_REPORT . . . . . (FLOW)
            REMOTE_RADAR_BUFF . . . . . (FLOW)

                    ~~ IS REFERENCED BY ~~
            CC_TC_MSG . . . . . (FLOW)
            NON_ASSOC_TGT_REP . . . . . (FLOW)
            NON_CORREL_TGT_REP . . . . . (FLOW)
            RADAR_REPORT . . . . . (FLOW)
```

```
          TARGET_MSG . . . . . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TC_CC_MSG (FLOW):

                    ~~ MAKES REFERENCES TO ~~
          ACK_ . . . . . . . . . . . . . . . (FLOW)
          DISPLAY_MSG . . . . . . . . . . . (FLOW)
          TARGET_CONTROL_DATA . . . . . . . (FLOW)
          TC_REM_COMM_MSG . . . . . . . . . (UNDEF)

                     ~~ IS REFERENCED BY ~~

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TC_REM_COMM_MSG (UNDEF):
```



```
                    ~~ MAKES REFERENCES TO ~~

                     ~~ IS REFERENCED BY ~~
          TC_CC_MSG . . . . . . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TGT_REP_BUFF (FLOW):

                    ~~ MAKES REFERENCES TO ~~
          ASSOC_DATA . . . . . . . . . . . (FLOW)
          NON_MATCHING_TGT_REP . . . . . . (FLOW)
          REPORT_TO_TRACK_PAIR . . . . . . (FLOW)
          TRACK_QUALITY . . . . . . . . . . (FLOW)

                     ~~ IS REFERENCED BY ~~

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    THREAT_ALGORITHM_FILE (UNDEF):

                    ~~ MAKES REFERENCES TO ~~

                     ~~ IS REFERENCED BY ~~
          TARGET_CONTROL_DATA_BASE . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    THREAT_PARAMETER_FILE (UNDEF):
```

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
TARGET_CONTROL_DATA_BASE . . . (FILE)

- -

THREAT_PRIORITY (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
DISPLAY_MSG . . . . (FLOW)
TACTICAL_DATA . . . . (FLOW)

- -

TIME_ (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

CURRENT_TIME . . . . (FLOW)
TIME_FROM_LAST_SMOOTHING . . . (FLOW)
TIME_OF_LAST_SMOOTHING . . . . (FLOW)

- -

TIME_FROM_LAST_SMOOTHING (FLOW):

~~ MAKES REFERENCES TO ~~
TIME_ . . . . (ELEMENT)

~~ IS REFERENCED BY ~~
PREDICTION_ . . . . (PROCESS 2.2.3.4)

- -

TIME_OF_LAST_SMOOTHING (FLOW):

~~ MAKES REFERENCES TO ~~
TIME_ . . . . (ELEMENT)

~~ IS REFERENCED BY ~~
SMOOTHED_DATA . . . . (FLOW)
TRACK_DATA . . . . (FLOW)

- -

```
TRACKING_MODE (UNDEF):

                    ~~ MAKES REFERENCES TO ~~

                     ~~ IS REFERENCED BY ~~
        MODE_ . . . . . . . . . . . . . . (FLOW)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TRACK_ALGORITHM_FILE (UNDEF):

                    ~~ MAKES REFERENCES TO ~~

                     ~~ IS REFERENCED BY ~~
        TARGET_CONTROL_DATA_BASE  . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TRACK_ALTITUDE (FLOW):

                    ~~ MAKES REFERENCES TO ~~
        ALTITUDE_ . . . . . . . . . . . . (ELEMENT)

                     ~~ IS REFERENCED BY ~~

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    TRACK_DATA (FLOW):
1      CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD          PA
GE 26

                    ~~ MAKES REFERENCES TO ~~
        ALTITUDE_ . . . . . . . . . . . . (ELEMENT)
        ASSOCIATION_COUNT . . . > . . . (ELEMENT)
        DEVIATION_VECTOR  . . . . . . . . (FLOW)
        INNERGATE_COUNT . . . . . . . . . (ELEMENT)
        OUTERGATE_COUNT . . . . . . . . . (ELEMENT)
        PREDICTED_POSITION  . . . . . . (FLOW)
        PREDICTED_VELOCITY  . . . . . . (FLOW)
        SMOOTHED_ALTITUDE . . . . . . . (FLOW)
        SMOOTHED_POSITION . . . . . . . (FLOW)
        SMOOTHED_VELOCITY . . . . . . . (FLOW)
        SMOOTHING_INDEX . . . . . . . . (ELEMENT)
        STATUS_ . . . . . . . . . . . . . (FLOW)
        TIME_OF_LAST_SMOOTHING  . . . . (FLOW)
        TRACK_NO  . . . . . . . . . . . . (ELEMENT)

                     ~~ IS REFERENCED BY ~~
        SWITCH_ACTION_DATA  . . . . . . (FLOW)
        TARGET_CONTROL_DATA . . . . . . (FLOW)
        TRACK_REC . . . . . . . . . . . . (FLOW)
        TRACK_STORAGE_FILE  . . . . . . (FILE)
```

- -
- - - -

TRACK_INITIATION (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

MODE_ . . . . (FLOW)

- -
- - - -

TRACK_NO (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

ASSOC_DATA . . . . (FLOW)
ECM_REPORT . . . . (FLOW)
NON_ASSOC_TRACK . . . . (FLOW)
NON_CORREL_TRACK . . . . (FLOW)
REMOTE_COMM_REPORT . . . . (FLOW)
REMOTE_TRACK_DATA . . . . (FLOW)
REPORT_TO_TRACK_PAIR . . . . . . (FLOW)
TACTICAL_DATA . . . . (FLOW)
TRACK_DATA . . . . (FLOW)

- -
- - - -

TRACK_PARAMETER_FILE (UNDEF):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

TARGET_CONTROL_DATA_BASE . . . (FILE)

- -
- - - -

TRACK_QUALITY (FLOW):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~

TGT_REP_BUFF . . . . (FLOW)

- -
- - - -

TRACK_REC (FLOW):

~~ MAKES REFERENCES TO ~~
TRACK_DATA . . . . (FLOW)

~~ IS REFERENCED BY ~~
ADD_ . . . . (FLOW)
DELETE_ . . . . (FLOW)
LOCAL_RADAR_BUFF . . . . (FLOW)
REMOTE_RADAR_BUFF . . . . (FLOW)

- -

TRACK_STORAGE_FILE (FILE):

~~ MAKES REFERENCES TO ~~
TRACK_DATA . . . . (FLOW)

~~ IS REFERENCED BY ~~
PREDICTION_ . . . . (PROCESS 2.2.3.4)
SMOOTHING_ . . . . (PROCESS 2.2.3.3)
TARGET_CONTROL_DATA_BASE . . . (FILE)

- -

UPDATED_TRACK_REC (FLOW):

~~ MAKES REFERENCES TO ~~
ADD_ . . . . (FLOW)
DELETE_ . . . . (FLOW)

~~ IS REFERENCED BY ~~
UPDATE_DATA . . . . (FLOW)

- -

UPDATE_DATA (FLOW):

~~ MAKES REFERENCES TO ~~
UPDATED_TRACK_REC . . . . (FLOW)

~~ IS REFERENCED BY ~~

- -

VALID_CODE (ELEMENT):

~~ MAKES REFERENCES TO ~~

~~ IS REFERENCED BY ~~
IFF_DATA . . . . (FLOW)
RESULT_ . . . . (FLOW)

```
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    VECTOR_ (FLOW):

                    ~~ MAKES REFERENCES TO ~~
            X_COORD . . . . . . . . . . . . . (ELEMENT)
            Y_COORD . . . . . . . . . . . . . (ELEMENT)

                    ~~ IS REFERENCED BY ~~
            DEVIATION_VECTOR . . . . . . . . (FLOW)
            POSITION_ . . . . . . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    VELOCITY_ (ELEMENT):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            OLD_VELOCITY . . . . . . . . . . (FLOW)
            PREDICTED_VELOCITY . . . . . . . (FLOW)
            SMOOTHED_VELOCITY . . . . . . . . (FLOW)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    WEAPON_ASGN (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            TARGET_CONTROL_DATA_BASE . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    WEAPON_ASSIGN_ALGORITHM_FILE (UNDEF):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
            TARGET_CONTROL_DATA_BASE . . . (FILE)

    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

1       CROSS REFERENCE REPORT FOR DD "TCDD" for REF = DFD                PA
GE 29

    X_COORD (ELEMENT):

                    ~~ MAKES REFERENCES TO ~~


                    ~~ IS REFERENCED BY ~~
```

```
            VECTOR_ . . . . . (FLOW)
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

    Y_COORD (ELEMENT):

                    ~~ MAKES REFERENCES TO ~~

                    ~~ IS REFERENCED BY ~~
            VECTOR_ . . . . . (FLOW)
    - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - -

 EOI ENCOUNTERED.
```

APPENDIX G

STRUCTURE CHARTS

APPENDIX G
STRUCTURE CHARTS

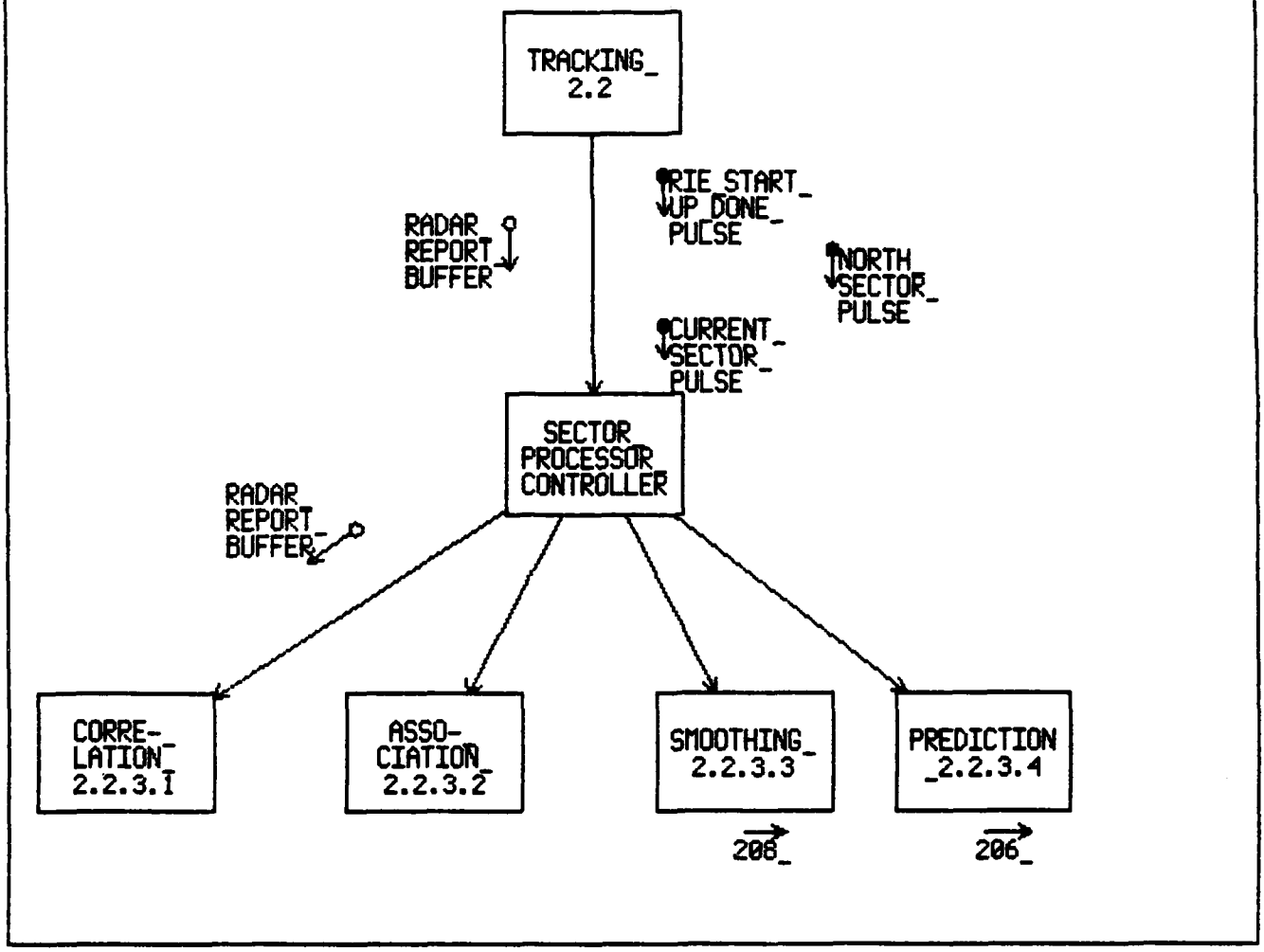
FINAL REPORT

LARGE SCALE SOFTWARE SYSTEM DESIGN
FOR THE
MISSILE MINDER AN/TSQ-73
USING
THE ADA PROGRAMMING LANGUAGE

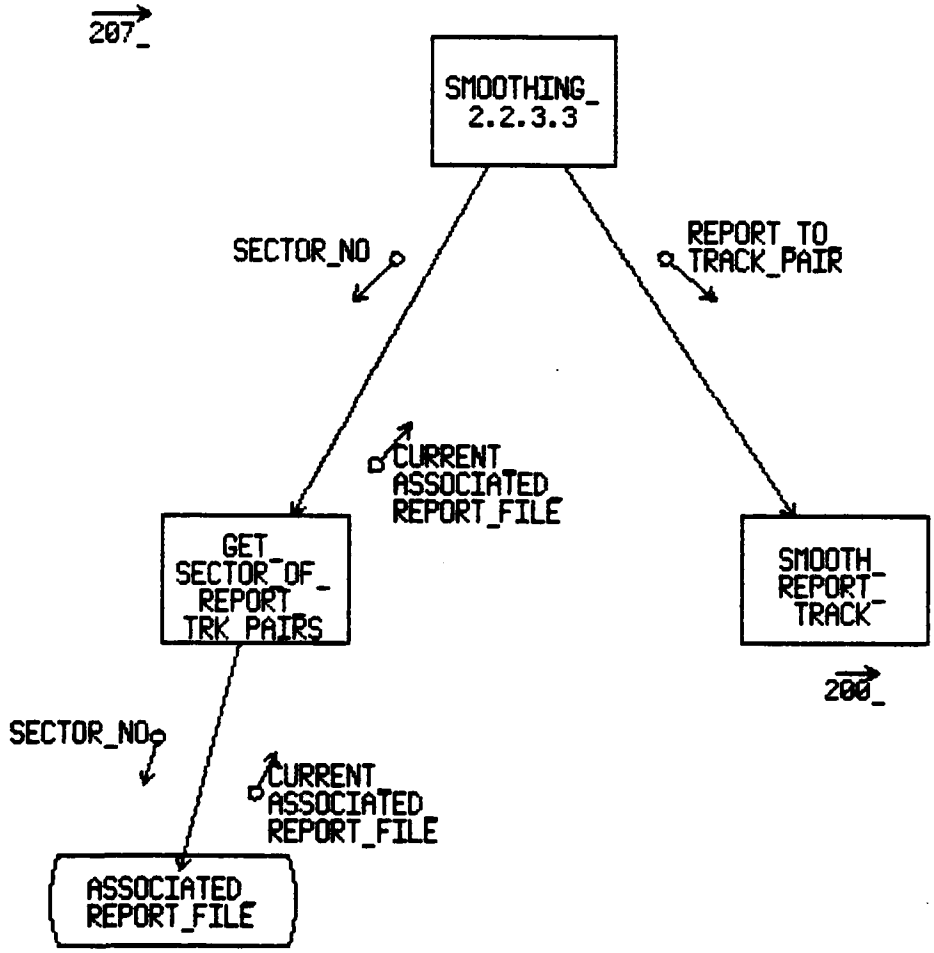
PREPARED FOR

U.S. ARMY COMMUNICATIONS ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703

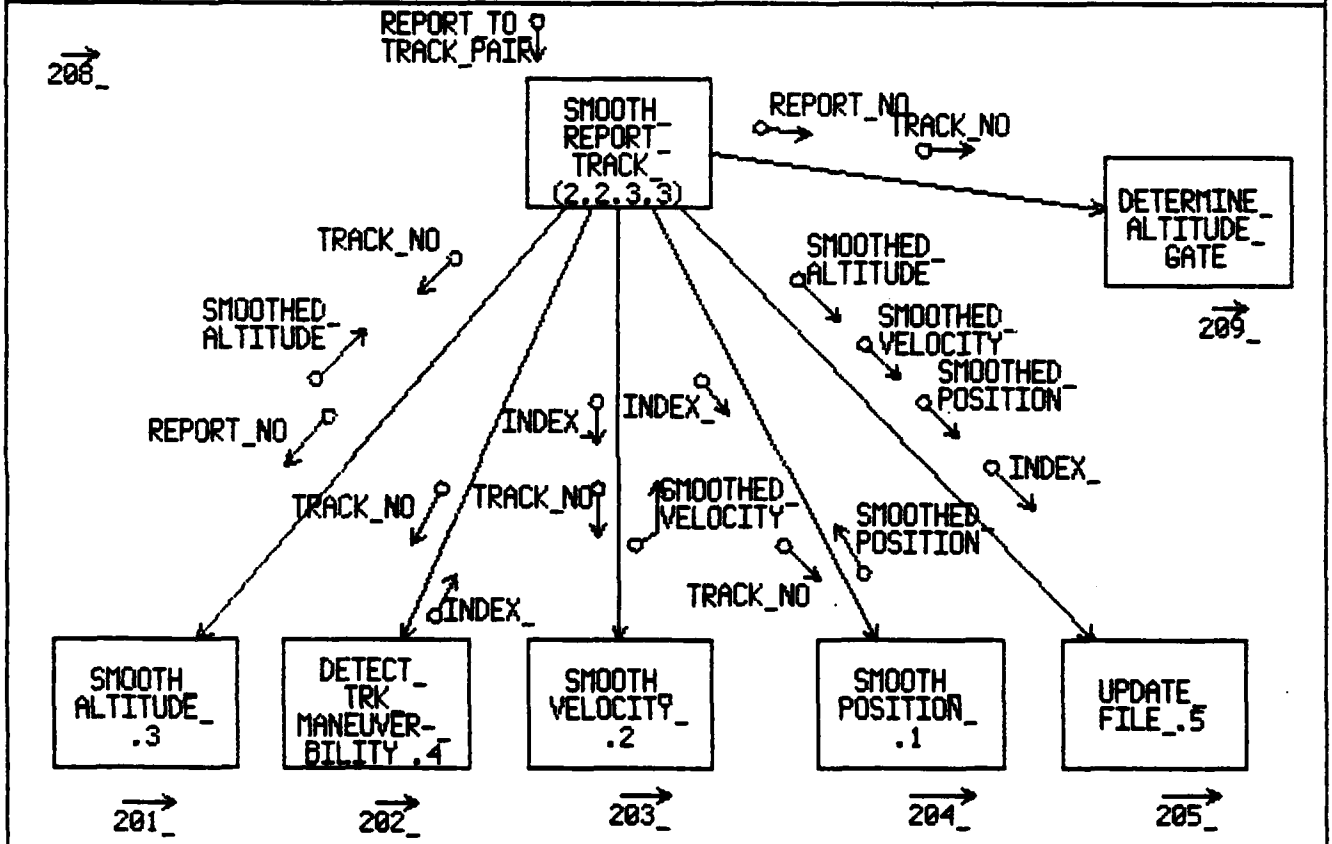
TITLE: 2.2 TRACKING SCT		PAGE: 207
REVISION 2	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



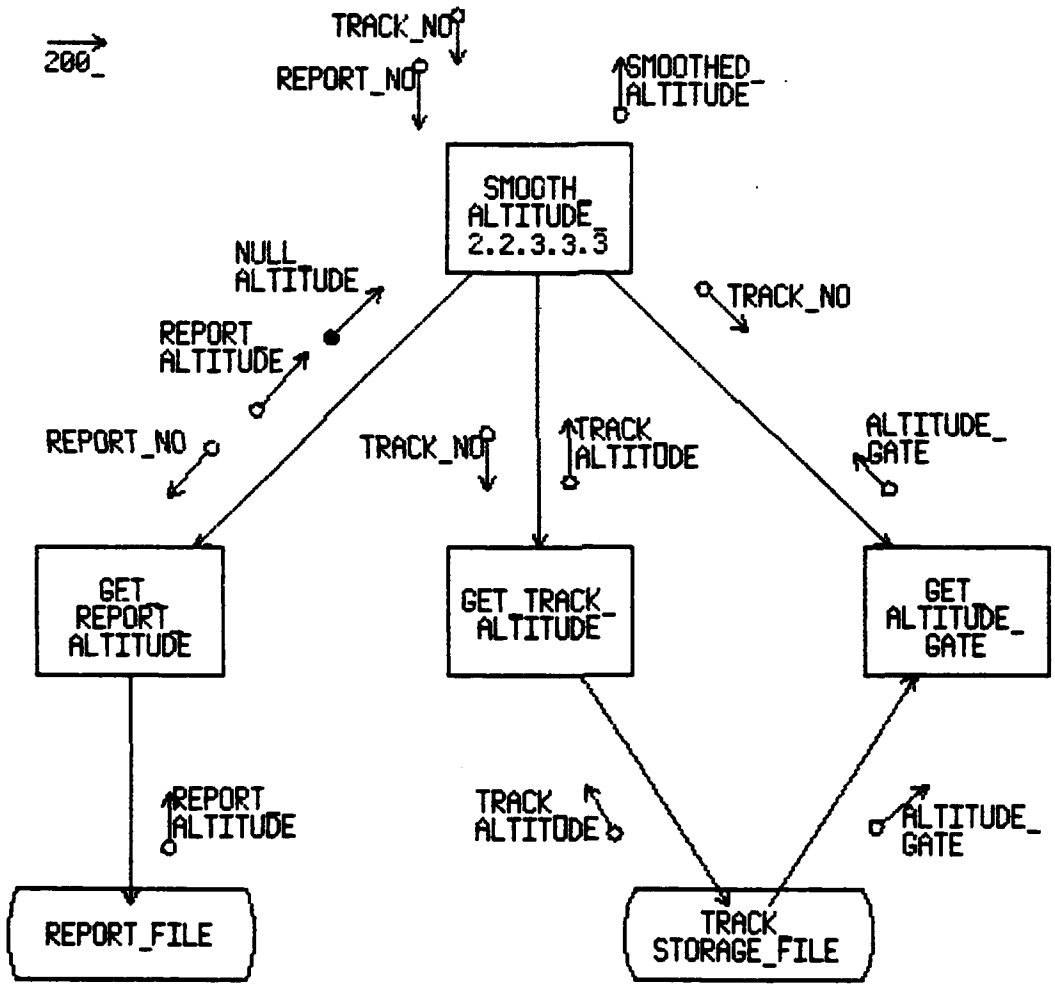
TITLE: 2.2.3.3 SMOOTHING SCT		PAGE: 208
REVISION 2	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



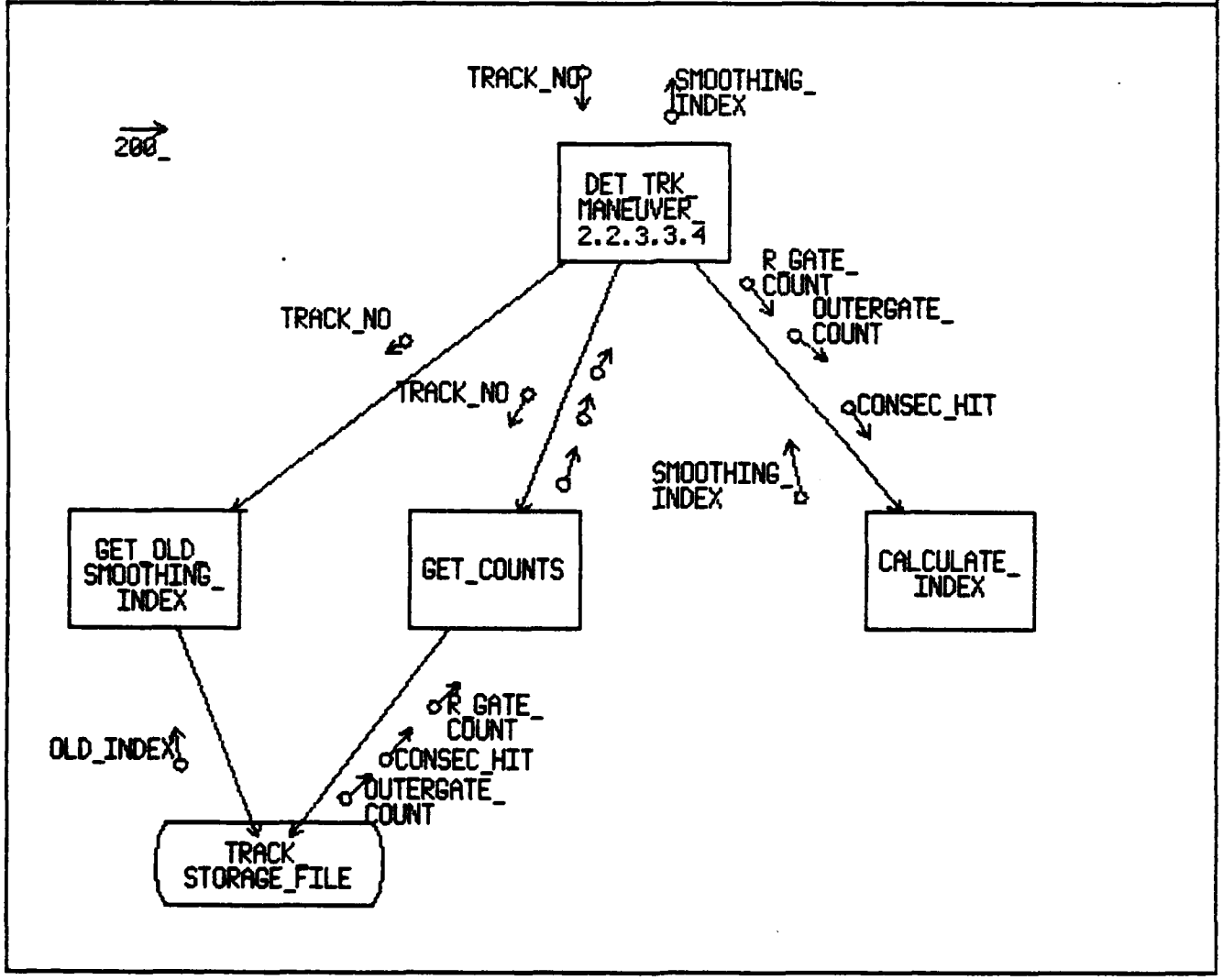
TITLE: (2.2.3.3) SMOOTH REPORT TRACK SCT		PAGE: 200
REVISION 3	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



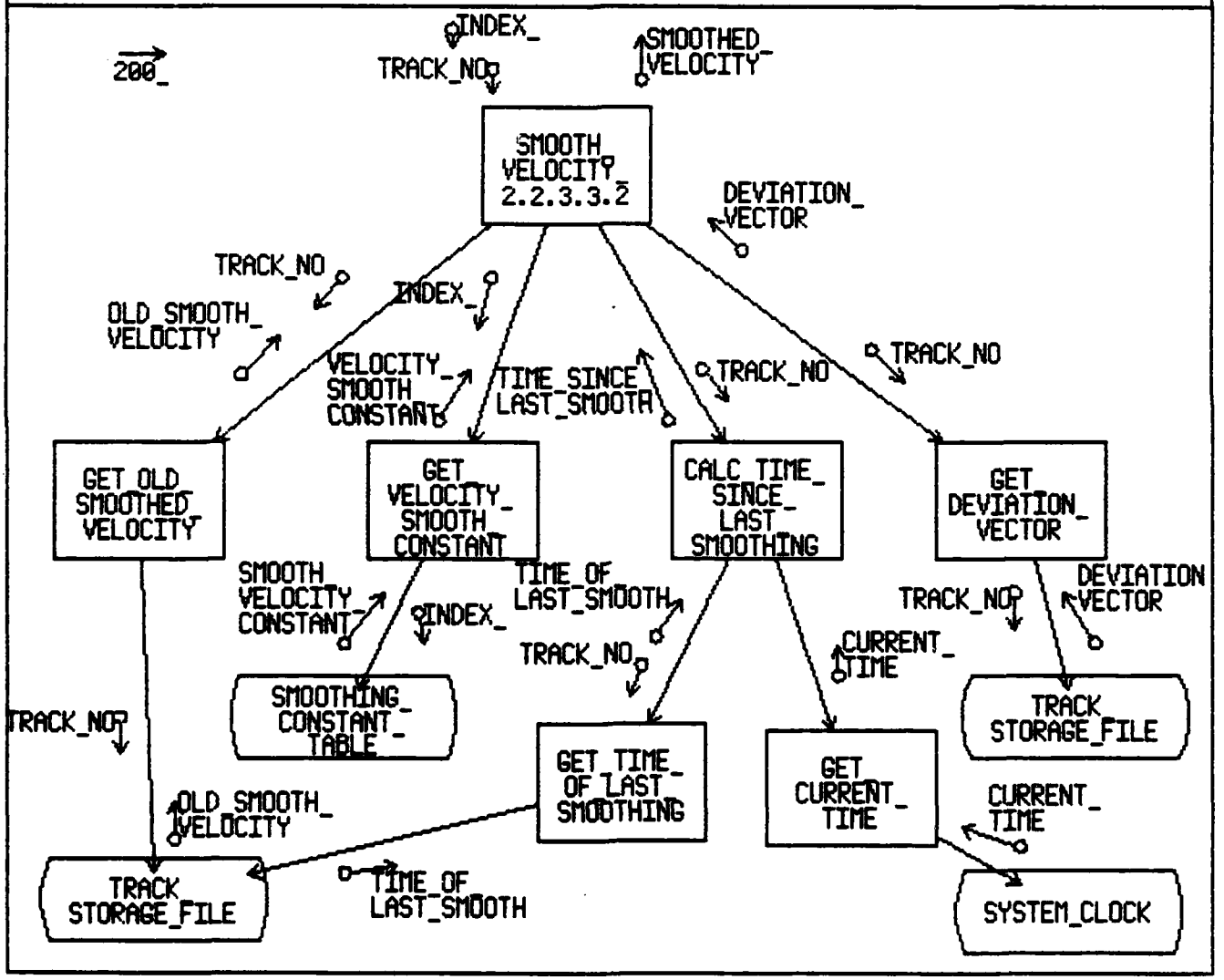
TITLE: 2.2.3.3.3 SMOOTH ALTITUDE SCT		PAGE: 201
REVISION 2	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



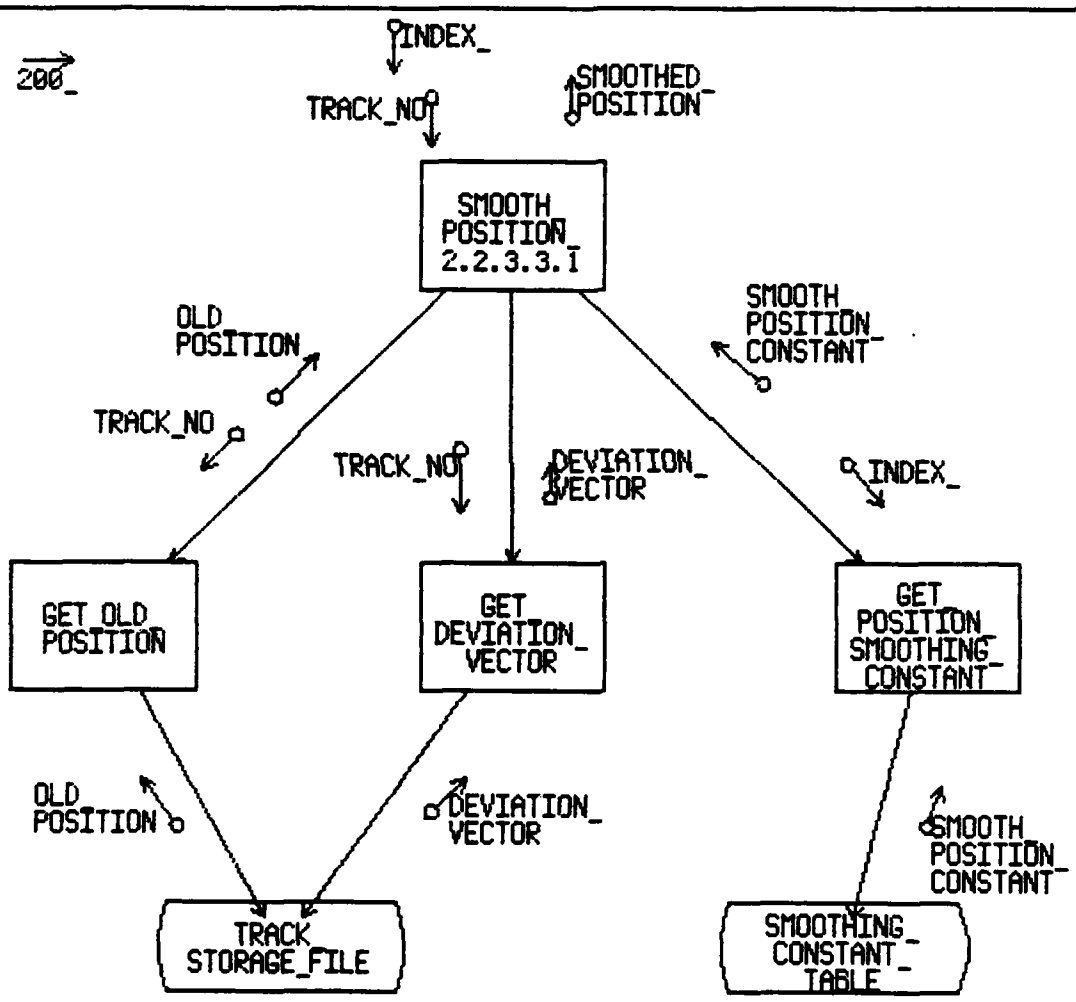
TITLE: 2.2.3.3.4 DETECT TRACK MANEUVERABILITY SCT		PAGE: 202
REVISION 2	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



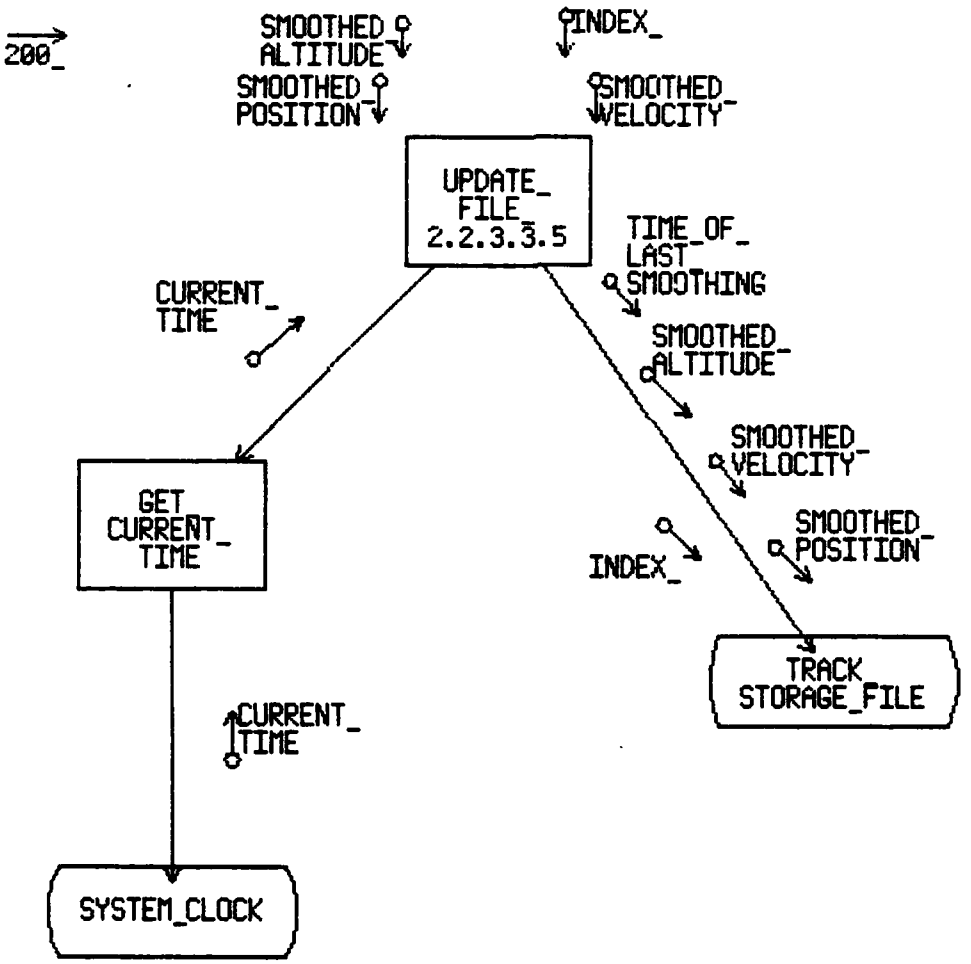
TITLE: 2.2.3.3.2 SMOOTH VELOCITY SCT		PAGE: 203
REVISION 2	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



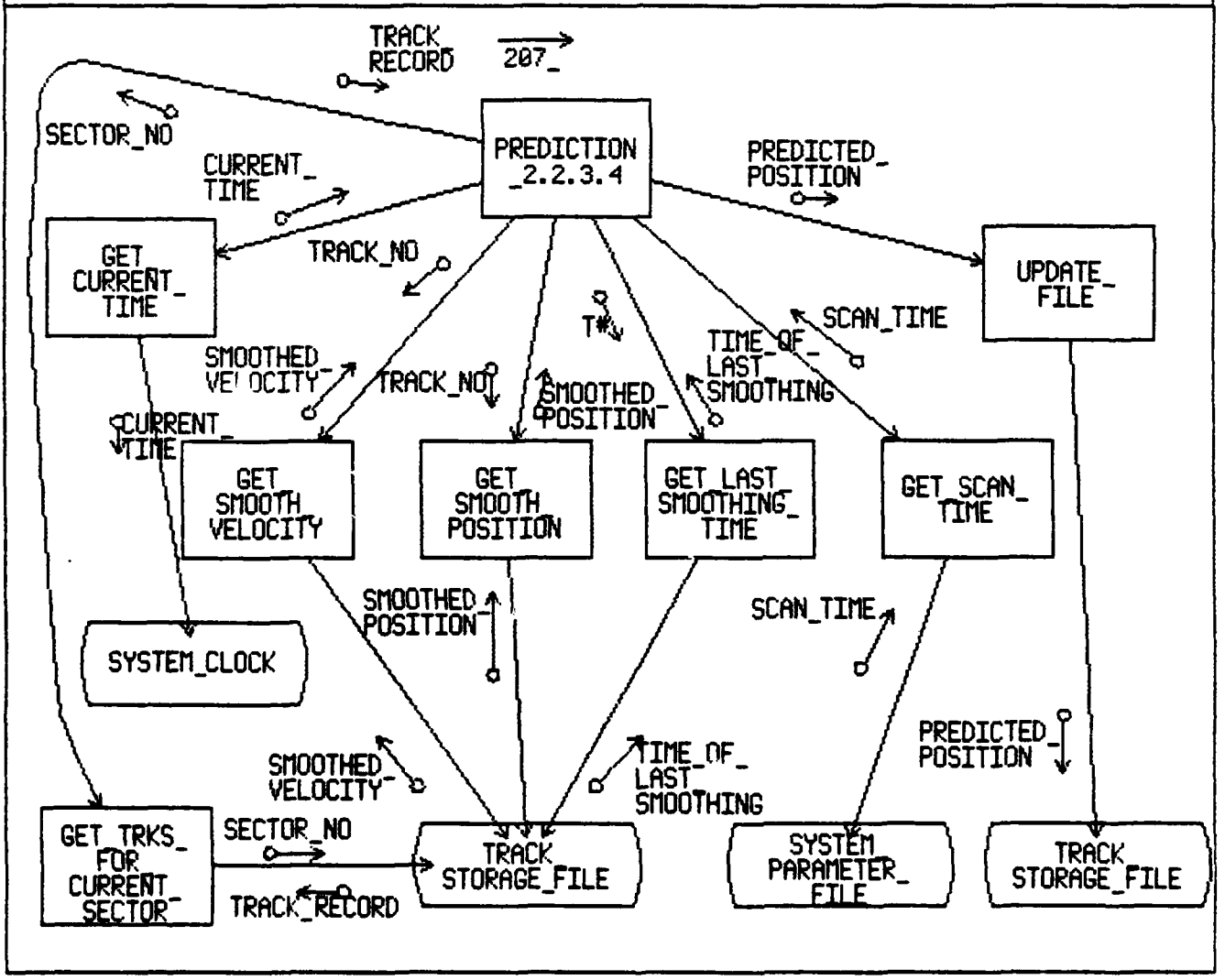
TITLE: 2.2.3.3.2 SMOOTH POSITION SCT		PAGE: 204
REVISION 2	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



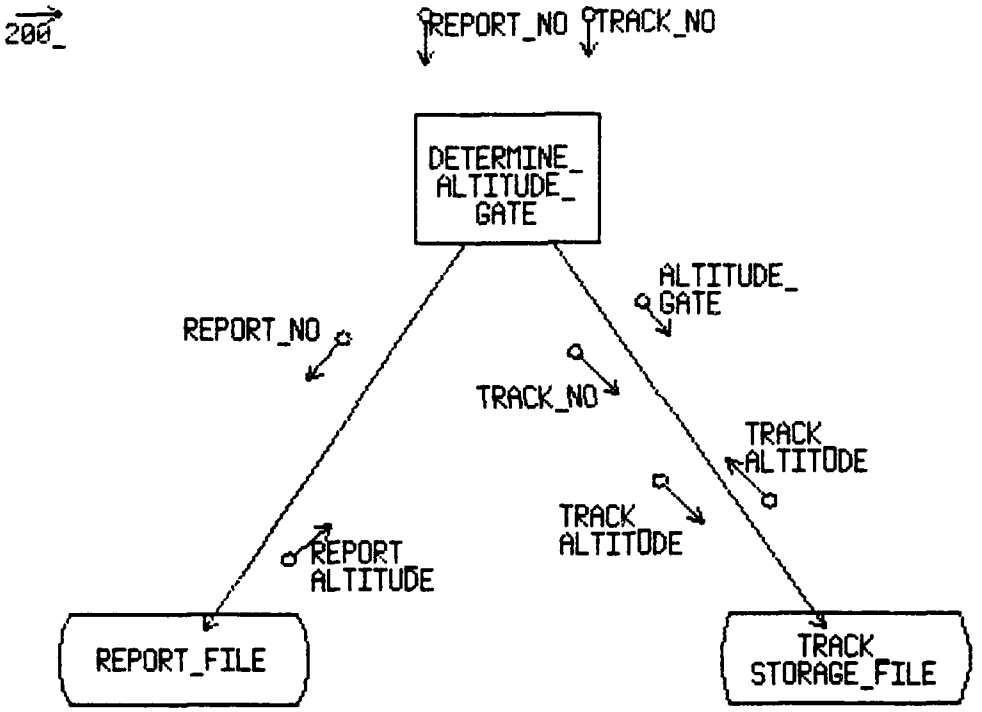
TITLE: 2.2.3.3.5 UPDATE FILE SCT		PAGE: 205
REVISION 2	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



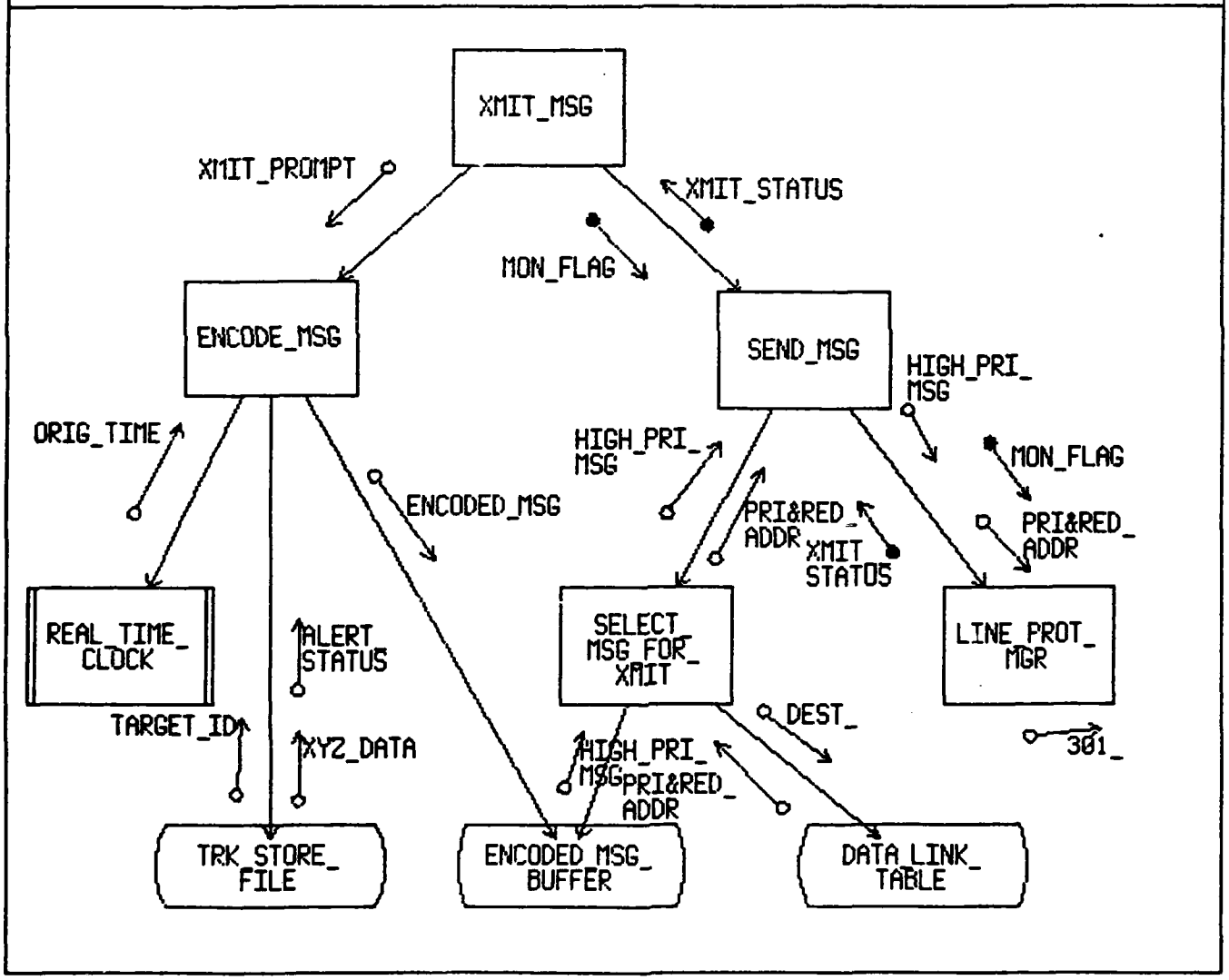
TITLE: 2.2.3.4 PREDICTION SCT		PAGE: 206
REVISION 2	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



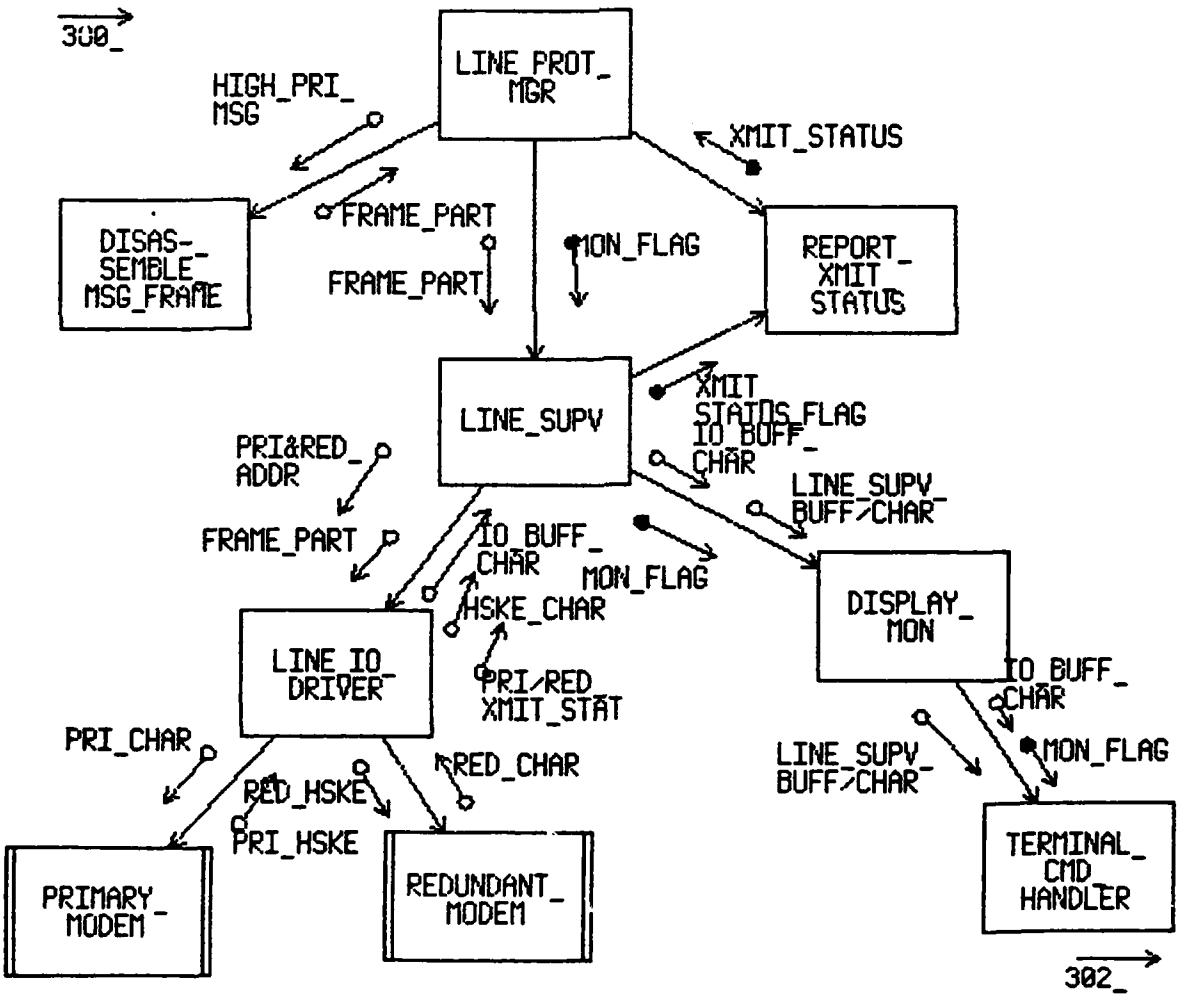
TITLE: DETERMINE ALTITUDE GATE SCT		PAGE: 209
REVISION: 1	AUTHOR: MJD/WAS	DATE: 6/4/82
NOTES:		



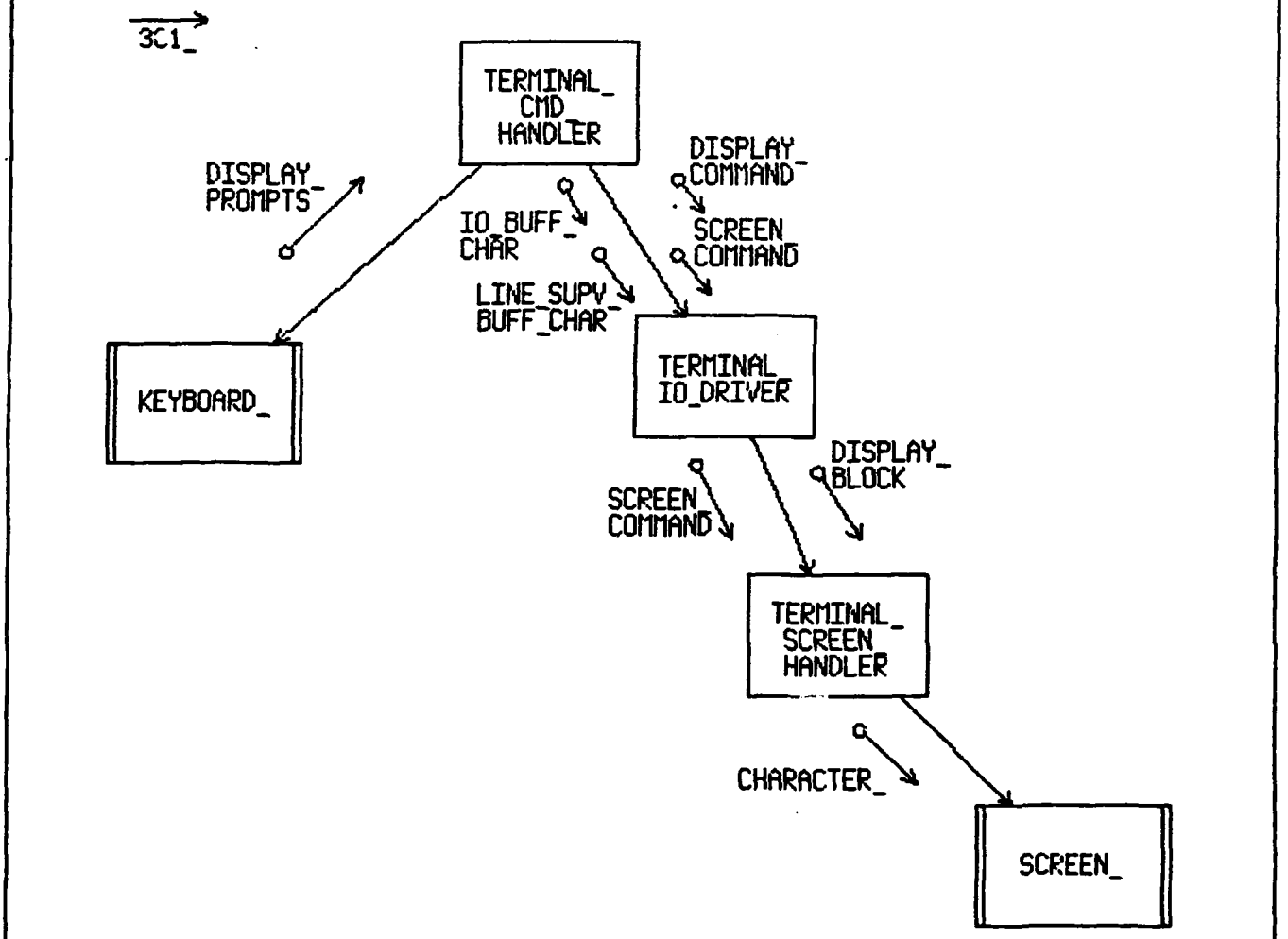
TITLE: XMIT MSG SCT (REMOTE COMM)		PAGE: 300
REVISION 1	AUTHOR: LWD/GIC	DATE: 2/17/82
NOTES:		



TITLE: "LINE PROTOCOL MANAGER SCT (REMOTE COMM)		PAGE: 301
REVISION 1	AUTHOR: LWD/GIC	DATE: 2/19/82
NOTES:		



TITLE: TERMINAL COMMAND HANDLER SCT (REMOTE COMM)		PAGE: 302
REVISION 1	AUTHOR: LWD/GIC	DATE: 2/18/82
NOTES:		



APPENDIX H

ADA-BASED PROGRAM DESIGN LANGUAGE

APPENDIX H

ADA-BASED PROGRAM DESIGN LANGUAGE

(ALSO DELIVERED UNDER CDRL A001)

FINAL REPORT

LARGE SCALE SOFTWARE SYSTEM DESIGN FOR THE MISSILE MINDER AN/TSQ-73 USING THE ADA PROGRAMMING LANGUAGE

PREPARED FOR

U.S. ARMY COMMUNICATIONS ELECTRONICS COMMAND

FORT MONMOUTH, NEW JERSEY 07703

```
--   The SECTOR_PROCESSING_CONTROLLER is the highest level program unit in the
-- implementation of the structure chart entity TRACKING.  It contains the
-- control logic necessary to implement the scheme of having one task to
-- process each sectors worth of radar reports.

-- <><><><><><><><><><> INITIALIZATION <><><><><><><><><><><><><><><><><><><><><><>

--   The SECTOR_PROCESSING_CONTROLLER is started up by the reception of
-- a signal from the radar interface equiptment (RIE) indicating that
-- this device is ready for normal service.  The SECTOR_PROCESSING_
-- controller then waits for the first north sector pulse to be trans-
-- mitted by the RIE.  The reception of this pulse accomplishes the
-- initial synchronization of the SECTOR_PROCESSING_CONTROLLER with the
-- radar scan interval.

-- <><><><><><><><><><> NORMAL OPERATION <><><><><><><><><><><><><><><><><><><><><>

--   Upon reception of the pulse immediately following the north sector pulse
```

```
-- the SECTOR_PROCESSING_CONTROLLER begins normal operation.  Normal operation
-- consists of a sequence of three activities which take place in synchronizati
n
-- with radar scan time.  The three activities are: 1. rendezvous with the RIE
-- to obtain the radar reports for the sector just scanned, 2. notification of
-- the task processing three sectors "ahead" that it must begin prediction, and
-- 3. initiation of processing the current sector by the appropriate task.

-- <><><><><><><><><><> ERROR SITUATIONS <><><><><><><><><><><><><><><><><><><><>

--   Synchronization with the RIE must be maintained.  This synchronization is
-- checked after the reception of each sectors worth of radar reports.  In the
-- event of loss of synchronization the SECTOR_PROCESSING_CONTROLLER will re-
-- synch itself to the sector that the RIE passed.  Tasks processing the
-- sectors passed over by this re-synch will return to synch on the next scan


task SECTOR_PROCESSING_CONTROLLER is
   entry RIE_START_UP_DONE;
   entry NORTH_SECTOR_PULSE;
   entry NEW_SECTOR_PULSE(R : in RADAR_REPORT_BUFFER);
end SECTOR_PROCESSING_CONTROLLER;

task body SECTOR_PROCESSING_CONTROLLER is

   PROCESSOR_LIST : array (SECTOR'FIRST..SECTOR'LAST) of TRACK_WHILE_SCAN;
   R              : RADAR_REPORT_BUFFER;
   CURRENT_SECTOR : SECTOR := 1; -- local sector counter

begin
   accept RIE_START_UP_DONE;
   accept NORTH_SECTOR_PULSE;

   loop
      accept NEW_SECTOR_PULSE(R : in RADAR_REPORT_BUFFER);

      if SECTOR_OF(R) /= CURRENT_SECTOR then--reestablish lost synchronization
         do
  --        send a message to the C&C
            CURRENT_SECTOR := SECTOR_OF(R);
```

```
         end;
      end if;

      -- terminate task processing three sectors "ahead" of current sector
      PROCESSOR_LIST(CURRENT_SECTOR + 2 mod SECTOR'LAST + 1).STOP;

      -- initiate task to process the sector data just received
      PROCESSOR_LIST(CURRENT_SECTOR).NEXT_SECTOR(R);

      CURRENT_SECTOR := CURRENT_SECTOR mod SECTOR'LAST + 1;
   end loop;

end SECTOR_PROCESSING_CONTROLLER;



   -- The task type TRACK_WHILE_SCAN (2.2.3) is the heart of the
   -- tracking process.
   -- One task of this type will process reports for each sector of the radar
   -- coverage area.

   -- <><><><><><><><><><><> NORMAL OPERATION <><><><><><><><><><><><><><><><><><><>

   --   After a rendezvous with the SECTOR_PROCESSING_CONTROLLER to obtain a
   -- sectors worth of radar reports, TRACK_WHILE_SCAN makes calls to perform
   -- the basic radar report processing sequence: correlation, association,
   -- SMOOTHING and prediction.  TRACK_WHILE_SCAN can also be told, via
   -- rendezvous with SECTOR_PROCESSING_CONTROLLER, to stop current
   -- processing and proceed immediately to prediction

   task type TRACK_WHILE_SCAN is    --(2.2.3)
      entry NEXT_SECTOR(R : in RADAR_REPORT_BUFFER);
      entry STOP;
   end TRACK_WHILE_SCAN;

   task body TRACK_WHILE_SCAN is
      R : RADAR_REPORT_BUFFER;

   begin
      loop
```

```
         select
            when STOP'COUNT = 0  =>
               accept NEXT_SECTOR(R : in RADAR_REPORT_BUFFER) do
                  PROCESS_SECTOR(R); -- initiate sequence
               end;
            or
               accept STOP do
                  BEGIN_PREDICTION;  -- begin prediction immediately
               end;
         end select;
      end loop;

   end TRACK_WHILE_SCAN;


-- <><><><><><><><><><><> TASK RIE <><><><><><><><><><><><><><><><><><><><><>

--   The task RIE simulates the radar interface equiptment hardware.  It
-- sends "pulses" to the SECTOR_PROCESSING_CONTROLLER to signal the end
-- of start up procedures and to signal the change of sector.  Since the
-- RIE acts as a master clock for sector processing it contains a start
-- entry point to allow it to be started by the test driver.

   task RIE is
      entry START;
   end RIE;

   task body RIE is

   begin

      -- simulate radar hardware activation
      accept START;

      -- simulate completion of radar hardware warm-up
      SECTOR_PROCESSING_CONTROLLER.RIE_START_UP_DONE;

      -- simulate first north sector pulse transmission
      SECTOR_PROCESSING_CONTROLLER.NORTH_SECTOR_PULSE;
```

```
      loop
         -- simulate sector change pulse and pass radar report buffer
         SECTOR_PROCESSING_CONTROLLER.NEW_SECTOR_PULSE(R);
         -- delay one sector scan time
      end loop;
   end RIE;

-- <><><><><><><><><><><><><> PACKAGE TRACK_STORES <><><><><><><><><><><><><>

--   The track stores package contains the track storage file and the
-- access entry points to the file.  Based on the PDL for the track
-- while scan functions entries to read and write one track record and
-- entries to read and write one sector have been provided


package TRACK_STORES is --(2.3.2)

   task FILE_MGR is -- (2.3.)
      entry READ_SECTOR (S : in SECTOR_NO; D : out SECTOR_DATA);
      entry WRITE_SECTOR(S : in SECTOR_NO; D : in  SECTOR_DATA);
      entry READ (S : in SECTOR_NO; T : in TRACK_NO; R : out TRACK_RECORD);
      entry WRITE(S : in SECTOR_NO; T : in TRACK_NO; R : in  TRACK_RECORD);
   end FILE_MGR;


   type TRACK_RECORD is
      RECORD
         STA      :STATUS;
         SEC_NO   :SECTOR_NO;
         TK_NO    :TRACK_NO;
         EXISTS   :BOOLEAN;
         PRED_POS:PREDICTED_POS;
         S_POS    :SMOOTHED_POS;
         VEL      :SMOOTH_VEL;  --THE PREDICTED VELOCITY
         ALT      :ALTITUDE;    --THE PREDICTED ALTITUDE
         INDEX    :SMOOTHING_INDEX;
         RGATE    :BOOLEAN;
         OUTGATE :BOOLEAN;
         ALTGATE :ALTITUDE_GATE;
         A_CNT    :ASSOCIATION_COUNT;
```

```
        T        :TIME_OF_LAST_SMOOTHING;
        D_VEC    :DEVIATION_VECTOR;
        CONSEC   :CONSEC_HIT;

      END RECORD;


   type SECTOR_DATA is array(TRACK_NO'FIRST..TRACK_NO'LAST) of TRACK_RECORD;

end TRACK_STORES;

PACKAGE VECTOR_OPERATIONS_PDL IS

   -- THIS PACKAGE MUST OVERLOAD THE
   --  BINARY OPERATORS OF '+' AND  '-' TO ALLOW THE CAPABILITY
   --  OF OPERATING ON VECTORS OF TWO DIMENTIONS AND RETURN
   --  THE SAME TYPE.
   --  ADDITIONALLY IT MUST PROVIDE THE CAPABILITY TO OVERLOAD
   --  THE BINARY OPERATOR '*' TO ALLOW FOR MULTIPLICATION OF
   --  A FLOATING POINT SCALAR TYPE AND A VECTOR OF TWO DIMENSIONS.


  TYPE VECT IS
   RECORD
    X:FLOAT;
    Y:FLOAT;
   END RECORD;

  FUNCTION "+" (A,B:VECT) RETURN VECT;
  FUNCTION "*" (A:FLOAT;B:VECT) RETURN VECT;
  FUNCTION "-" (A,B:VECT) RETURN VECT;

END VECTOR_OPERATIONS;


PACKAGE TRACKING_OPERATIONS_PDL IS
   --  THIS PACKAGE CONTAINS THE FUNCTIONS, PROCEDURES AND TYPES REQUIRED
   --  BY THE SMOOTHING AND PREDICTION MODULES OF THE TRACK_WHILE_SCAN
   --  TASKS.  THESE PROCESSES REQUIRE KNOWLEDGE OF THE RECORD COMPONENTS OF
   --  OF THE TRACK_STORAGE_FILE.  THE PACKAGE TRAKE_STORES CONTAINS A
```

```
--  DEFINITION OF THE TRACK_RECORD AND SHOULD BE REFERRED TO WHEN
--  GENERATING THE CODE FOR THIS PACKAGE.

--  THE OBJECTIVE OF THIS PACKAGE IS TO DEFINE THE FUNCTIONS REQUIRED
--  FOR THE TRACKING OPERATIONS AND THEIR INTERFACES.  TO AID THIS
--  DEVELOPMENT THE ADA TYPES REQUIRED BY THESE OPERATIONS ARE ALSO
--  DEFINED.  THIS PACKAGE MAY BE COMPILED BUT IT IS NOT INTENDED TO BE
--  EXECUTED.

TYPE VECT IS
RECORD
   X:FLOAT;
   Y:FLOAT;
END RECORD;

SUBTYPE DEVIATION_VECT IS VECT;
SUBTYPE PREDICTED_POS IS VECT;
SUBTYPE SMOOTH_POS IS VECT;
SUBTYPE SMOOTH_VEL IS VECT;
SUBTYPE REPORT_POS IS VECT;

SUBTYPE ALTITUDE_GATE IS INTEGER RANGE ALTITUDE_GATE'FIRST..
ALTITUDE_GATE'LAST;


TYPE TRACK_NO IS NEW INTEGER RANGE 1..512;
TYPE SECTOR_NO IS NEW INTEGER RANGE 1..20;
TYPE REPORT_NO IS NEW INTEGER RANGE 1..512;
TYPE SMOOTHING_INDEX IS NEW INTEGER RANGE 1..5;

TYPE SMOOTH_CONSTANTS IS
   RECORD
      POS : FLOAT;
      VEL : FLOAT;
   END RECORD;

SMOOTH_TABLE : CONSTANT ARRAY (SMOOTHING_INDEX) OF SMOOTH_CONSTANTS :=
   ((0.4375,0.625),
    (0.5625,0.1875),
    (0.6875,0.3125),
```

```
    (0.8125,0.4375),
    (0.9375,0.5625));

--  A TYPE OF 'TIME'  MUST BE DEFINED AT THIS POINT
--  THE FORMAT FOR 'TIME' SHOULD PROVIDE THE CAPABILITY TO
--  DETERMINE THE CURRENT SYSTEM TIME NECESSARY FOR CALCULATIONS
--  BELOW

--<<<<<  FUNCTIONS REQUIRED BY TRACKING COMPONENTS  >>>>>

   FUNCTION DET_DEVIATION_VECT (A:REPORT_POS;B:PREDICTED_POS)
   RETURN DEVIATION_VECT;

   FUNCTION SMOCON(I: SMOOTHING_INDEX) RETURN SMOOTH_CONSTANTS;

--.....................................................


--<<<<<  PROCEDURES REQUIRED BY TRACKING COMPONENTS  >>>>>


  PROCEDURE PREDICTION (X: IN SECTOR_NO); -- (2.2.3.4)

     -- ONCE PER RADAR SCAN EACH TRACK IN EACH SECTOR SHALL BE DEAD
     -- RECONED FORWARD TO A POSITION WHERE THE RADAR EXPECTS TO
     -- FIND THE TRACK ON THE SUBSEQUENT SWEEEP
     -- THIS SHALL BE ACCOMPLISHED BY READING IN ALL THE TRACKS
     -- FROM THE CURRENT SECTOR OF THE TRACK STORAGE FILE
     -- AND PROCESSING THE TRACKS FROM THAT
     -- SECTOR SEQUENTIALLY
     -- THE FORMULA THAT SHALL BE USED TO PREDICT THE TRACK'S
     -- POSITION  IS AS FOLLOWS:
     --
     --        PREDICTED POSITION = SMOOTHED_POSITION + (RADAR_SCAN_TIME
     --           + CURRENT_TIME) * SMOOTHED_VELOCITY
     --
     -- AFTER ALL TRACKS IN THE CURRENT SECTOR HAVE BEEN PREDICTED
     -- THE CURRENT SECTOR OF THE TRACK STORAGE FILE SHALL BE UPDATED
```

```
PROCEDURE SMOOTHING (RN: IN REPORT_NO; T: IN TRACK_NO; S: IN SECTOR_NO);
   --  (2.2.3.3)SMOOTHING SHALL TAKE PLACE AFTER ASSOCIATION HAS
   --  DETERMINED WHICH REPORT IS PAIRED WITH A PARTICULAR TRACK
   --  SMOOTHING SHALL CONSIST OF SEVERAL SEQUENTIAL STEPS
   --  THEY ARE:
   --           DETERMINE ALTITUDE GATE
   --           SMOOTH ALTITUDE (2.2.3.3.3)
   --           DETECT TRACK MANEUVERABILITY(2.2.3.3.4)
   --           SMOOTH VELOCITY(2.2.3.3.2)
   --           SMOOTH POSITION(2.2.3.3.1)
   --
   --  DETERMINE ALTITUDE GATE
   --     THE REPORT RECORD SHALL BE RETRIVED FROM THE REPORT FILE
   --     AND THE FOLLOWING SHALL TAKE PLACE
   --     AN ALTITUDE GATE SHALL BE FORMED BY ADDING AND SUBTRACTING
   --     AN INTEGER VALUE TO THE TRACK'S LAST RECORDED
   --     ALTITUDE.  THIS INTEGER VALUE IS STORED IN THE TRACK_RECORD
   --     IN THE OBJECT ALTGATE.
   --     IF THE REPORT ALTITUDE LIES INSIDE THIS GATE
   --     THEN ALTGATE  SHALL BE REDUCED BY 500 FEET
   --     IF THE REPORT ALTITUDE LIES OUTSIDE OF THE GATE THEN
   --     THE ALTGATE SHALL BE INCREASED BY 500 FEET.
   --     THE ALTGATE HAS MAXIMUM AND MINIMUM VALUES
   --     DEFINED IN THE CLASSIFIED PACKAGE.
   --     THEY ARE STORED IN ALTITUDE_GATE'FIRST AND
   --     ALTITUDE_GATE'LAST

   --   SMOOTH ALTITUDE
   --      IF THE TRACK'S ALTGATE IS LESS THAN
   --      ALTITUDE_GATE'LAST THEN
   --      THE SMOOTHED ALTITUDE SHALL BE COMPUTED BY
   --      THE ARITHMETIC MEAN BETWEEN THE REPORT ALTITUDE
   --      AND THE TRACK'S ALTITUDE
   --      OTHERWISE THE REPORT ALTITUDE SHALL REPLACE
   --      THE TRACK'S ALTITUDE.

   --   DETECT TRACK MANEUVERABILITY
   --      THIS SHALL BE THE EQUIVALENT OF DETERMINING

   --      THE SMOOTHING INDEX.
```

```
        --      IF THE TRACKS'S INDEX IS 1 AND IF THERE HAVE
        --      BEEN TWO CONSECUTIVE OUTERGATE HITS THEN
        --      THE INDEX SHALL BE SWITCHED TO 5.
        --      IF THE TRACK'S INDEX IS 2,3,4 OR 5 THEN
        --      THE INDEX SHALL BE DECREASED BY ONE FOR
        --      A RECOVERY GATE HIT AND INCREASED BY
        --      ONE FOR AN OUTERGATE HIT.
        --      ALL SMOOTHING INDICES SHALL BE RESTRICTED TO
        --      A RANGE OF 1..5.
        --
        --   SMOOTH VELOCITY
        --      THE SMOOTHED VELOCITY SHALL BE CALCULATED ACCORDING
        --      TO THE FOLLOWING FORMULA:
        --         SMOOTHED_VELOCITY = SMOOTHED_VELOCITY + (VELOCITY_
        --         SMOOTHING_CONSTANT/ TIME_SINCE_LAST_SMOOTHING) * DEV-
        --         IATION_VECTOR
        --         THE VELOCITY_SMOOTHING_CONSTANT SHALL BE OBTAINED FROM
        --         THE FUNCTION SMOCON DESCRIBED ABOVE.

        --   SMOOTH POSITION
        --      THE SMOOTHED POSITION SHALL BE A FUNCTION OF THE LAST
        --      SMOOTHED POSITION AND THE DEVIATION_VEC RELATED IN THE
        --      FOLLOWING FORMULA:
        --         SMOOTH_POSITION := SMOOTHED_POSITION + POSTION_SMOOTHING_
        --         CONSTANT * DEVIATION_VECTOR


        --

END TRACKING_OPERATIONS;



PACKAGE VECTOR_OPERATIONS_PDL IS

   -- THIS PACKAGE MUST OVERLOAD THE
   --  BINARY OPERATORS OF '+' AND  '-' TO ALLOW THE CAPABILITY.
   --  OF OPERATING ON VECTORS OF TWO DIMENTIONS AND RETURN
   --  THE SAME TYPE.
   --  ADDITIONALLY IT MUST PROVIDE THE CAPABILITY TO OVERLOAD
   --  THE BINARY OPERATOR '*' TO ALLOW FOR MULTIPLICATION OF
```

```
    --  A FLOATING POINT SCALAR TYPE AND A VECTOR OF TWO DIMENSIONS.


   TYPE VECT IS
    RECORD
     X:FLOAT;
     Y:FLOAT;
    END RECORD;

   FUNCTION "+" (A,B:VECT) RETURN VECT;
   FUNCTION "*" (A:FLOAT;B:VECT) RETURN VECT;
   FUNCTION "-" (A,B:VECT) RETURN VECT;

END VECTOR_OPERATIONS;
```

APPENDIX I

ADA CODE LISTING (DELIVERED UNDER CDRL A001)

APPENDIX I

ADA CODE LISTING
(ALSO DELIVERED UNDER CDRL A001)

FINAL REPORT

LARGE SCALE SOFTWARE SYSTEM DESIGN

FOR THE

MISSILE MINDER AN/TSQ-73

USING

THE ADA PROGRAMMING LANGUAGE

PREPARED FOR

U.S. ARMY COMMUNICATIONS ELECTRONICS COMMAND

FORT MONMOUTH, NEW JERSEY 07703

Summary of Complexity Measure

Procedure	Complexity Meausre
Prediction	3
Smoothing	2
Smooth_Report_Track	8

The code reproduced in Appendix I is organized into the following four Ada packages:

1) VECTOR_OPERATIONS

2) TRACK_TYPES

3) MANAGE_TABLES

4) TRACKING_OPERATIONS

Collectively, they represent the code necessary to implement the following functional areas of the AN/TSQ-73:

- Sector Processing - process radar reports by predefined sectors implemented in Ada by: task type TRACK_WHILE_SCAN (2.2.3).
- Sector Processing Control - provide for control of sector processing, its timing requirements, and synchronization of that processing with the Radar Interface Equipment

  implemented in Ada code by: task SECTOR_PROCESSING_CONTROLLER in package TRACKING_OPERATIONS
- Altitude Gate - provides the measure of altitude correlation history

  implemented in Ada code as: a subset of procedure Smoothing (2.2.3.3) and enclosed in comments -- DETERMINE ALTITUDE GATE
- Maneuver Detection and Recovery - detects the maneuverability of a track

  implemented in Ada as a subset of procedure Smoothing (2.2.3.3) and offset with comment -- Maneuver Detection
- Smoothing - the application of mathematical formulas to associated Radar reports in order to determine a smoothed position, smoothed altitude and smoothed velocity while accounting for the maneuverability of the track

  implemented in Ada code as procedure Smoothing (2.2.3.3)
- Prediction - predict the position of each track by determining the X and Y coordinates of where the radar should expect to find the track on the subsequent sweep

  implemented in Ada code as procedure Prediciton (2.2.3.4)

In addition to these explicit requirements these implicit functional areas were also coded:

- Table Manager provides authorized access to the Track Storage File while guarding against conflicting access requests.

  implemented in Ada code by: task FILE_MGR in package MANAGER_TABLES (2.3.2)

- Support Routines - routines to support the mathematical calculations of Smoothing and Prediction

  implemented in Ada code as: package VECTOR_OPERATIONS (would reside in the USER APPLICATION LIBRARY (1.4)).

The following is a brief description of the contents of each of the four packages:

VECTOR_OPERATIONS

Defines an Ada record type named VECT consisting of two components X and Y. Overloads the operations '+', '-', and '*' to allow operations on objects of type VECT.

TRACK_TYPES

Defines the subtypes, enumeration types, records and arrays required by package TRACKING_OPERATIONS; the TRACK_RECORD, REPORT, REPORT_FILE, et al.

MANAGE_TABLES

Defines the TRACK_STORAGE_FILE and a task FILE_MGR which provides the access mechanism described earlier.

TRACKING_OPERATIONS

Defines the task SECTOR_PROCESSING_CONTROLLER the task type TRACK_WHILE_SCAN procedure SMOOTHING, procedure PREDICTION, and other subprograms required by these two procedures.

For testing purposes, each package was considered to be a unit. Since packages are passive, it was necessary to create an Ada procedure for testing each unit utilizing the subprograms and tasks of that package and check the results manually. TRACK_TYPES

was not considered for its own unit test since it contained only definitions. It was, however, included in other unit tests since they required the types defined there.

The task SECTOR_PROCESSING_CONTROLLER and the twenty objects of task type TRACK_WHILE_SCAN from the package TRACKING_OPERATIONS were considered as one unit for testing. They were grouped together as a single unit since they perform numerous rendevous with each other, for this reason individual testing would be meaningless if not impossible. This test required the most time and effort since a third task simulating the pulses from radar interface equipment names task RIE had to be generated in order to activate the SECTOR_PROCESSING_CONTROLLER.

A test procedure with embedded PUT statements shows when pulses are sent, received and when processing for sector commences and terminates.

The package VECTOR_OPERATIONS was also tested as a unit. An Ada procedure was written declaring objects of type VECT with initialized values. The objects were then operated on using the overloaded operations and the results were checked manually.

Since Smoothing and Prediction both required the package VECTOR_OPERATIONS, TRACK_TYPES, and MANAGE_TABLES a unit test for each was discarded in favor of testing the four packages together. At the time of this writing, the test code was being generated according to the following plan.

The task RIE (mentioned earlier) will activate the SECTOR_PROCESSING_CONTROLLER by simulating pulses from the radar. The SECTOR_PROCESSING_CONTROLLER will activate and terminate according to the timing requirements the twenty TRACK_WHILE_SCAN tasks. Smoothing and Prediction will operate within each TRACK_WHILE_SCAN task on simulated data. The data will reside in three files, the TRACK_STORAGE_FILE, which contains track records, the ASSOCIATED_REPORT_FILE which contains report to track pairs, and the REPORT_FILE which contains radar reports. Since Smoothing and Prediction affect the TRACK_

STORAGE_FILE, this file will be printed before the simulation action and afterwards by means of a procedure SNAPSHOT. The calculations can then be checked manually.

The following tables illustrate the test output produced for one track during the processing of one radar scan. The table items are the components of the track storage file that are manipulated by the radar processing algorithms.

TRACK_STORAGE_FILE BEFORE SNAPSHOT

SELECTOR NUMBER	1
TRACK NUMBER	1
STATUS	Local
PREDICTED POSITION	(16,99)
PREDICTED VELOCITY	(0,.25)
ALTITUDE	10,000
ALTITUDE GATE	5,000
SMOOTHED POSITION	(16,98.75)
SMOOTH INDEX	5
TIME	
R-GATE	T
OUTERGATE	F
CONSECUTIVE HITS	0
ASSOCIATION COUNT	6
DEVIATION VECTOR	(0.0,25)

TRACK_STORAGE_FILE AFTER SNAPSHOT

SELECTOR NUMBER	1
TRACK NUMBER	1
STATUS	Local
PREDICTED POSITION	(16,100.2551)
PREDICTED VELOCITY	(00.2573)
ALTITUDE	9,500
ALTITUDE GATE	4,500
SMOOTHED POSITION	(16,97.8313)
SMOOTH INDEX	4
TIME	
R-GATE	T
OUTERGATE	F
CONSECUTIVE HITS	0
ASSOCIATION COUNT	6
DEVIATION VECTOR	(00,0.1)

The listing for the procedure required to test the package VECTOR_OPERATIONS follows. The procedure declared objects of type VECT and assigned an initial value to each.

The overloaded operators were then tested by performing the given operations on these objects and verifying the results.

```
$ ada tmath.ada
```

```
NYU Ada/ED 16.3(5/29/82)                    THU  10 JUN 82  12:03:36   PAGE     1

ADAfile:   TMATH.ADA

    1   with TEXT_IO;use TEXT_IO;
    2
    3
    4   procedure TEST_VECTOR_OPERATIONS is
    5
    6   package F is NEW FLOAT_IO(FLOAT);use F;
    7
    8   package VECTOR_OPERATIONS is
    9
   10      type VECT is
   11         record
   12            X:FLOAT;
   13            Y:FLOAT;
   14         end record;
   15
   16       function "+" (A,B:VECT)        return VECT;
   17       function "*" (A:FLOAT;B:VECT) return VECT;
   18       function "-" (A,B:VECT)        return VECT;
   19
   20   end VECTOR_OPERATIONS;
   21
   22   package BODY VECTOR_OPERATIONS is
   23
   24       function "+" (A,B:VECT) return VECT is
   25       begin
   26          return ( (X => A.X + B.X,
```

```
27                      Y => A.Y +B.Y));
28
29          end;
30
31
32          function "*" (A:FLOAT;B:VECT) return VECT is
33          begin
34             return ( (X => A * B.X,
35                       Y => A * B.Y));
36
37          end;
38
39          function "-" (A,B:VECT) return VECT is
40
41          begin
42             return ((X => A.X - B.X,
43                      Y => A.Y - B.Y));
44
45          end;
46
47    end VECTOR_OPERATIONS;
48      use VECTOR_OPERATIONS;
49      M:VECT:=(0.5,0.25);
50      N:VECT:=(-3.5,4.1);
51      R,S,T,U,V:VECT;
52      X:FLOAT:=5.2;
53
54
55      begin
56         R:=M+N;
57         S:=M-N;
58         T:=N-M;
59         U:=X*M;
60         V:=X*N;
61       PUT("M=");    F.PUT(M.X);    F.PUT(M.Y);   new_line;
62       PUT("N=");    F.PUT(N.X);    F.PUT(N.Y);   new_line;
63       PUT("X=");     F.PUT(X);      new_line;
64       PUT("M+N="); F.PUT(R.X);    F.PUT(R.Y);   new_line;
65       PUT("M-N="); F.PUT(S.X);    F.PUT(S.Y);   new_line;
66       PUT("N-M="); F.PUT(T.X);    F.PUT(T.Y);   new_line;
```

```
 67       PUT('X*M='); F.PUT(U.X);    F.PUT(U.Y);  new_line;
 68       PUT('X*N='); F.PUT(V.X);    F.PUT(V.Y);  new_line;
 69     end;
 70

no parse errors detected
Parsing time: 99 seconds

no semantic errors detected
Translation time: 145 seconds

Binding time: 0.9 seconds

Begin Ada execution

M=5.000000E-012.500000E-01
N=-3.500000E+004.100000E+00
X=5.200000E+00
M+N=-3.000000E+004.350000E+00
M-N=4.000000E+00-3.850000E+00
N-M=-4.000000E+003.850000E+00
X*M=2.600000E+001.300000E+00
X*N=-1.820000E+012.132000E+01

Execution complete
Execution time: 137 seconds
I-code statements executed: 146
```

On the succeeding pages is the code and the results of the execution of the code that tests the task SECTOR_PROCESSING_CONTROLLER and the twenty tasks TRACK_WHILE_SCAN. Pulses from the Radar Interface Equipment are simulated to activate the task SECTOR_PROCESSING_CONTROLLER. The SECTOR_PROCESSING_CONTROLLER then synchronizes the TRACK_WHILE_SCAN tasks. The results indicate when processing for a current sector has begun and for which sector stop orders have been sent. Also indicated are the sending and receiving of the new sector pulses. The file name used for the test procedure was CODE2.ADA. The results are found in CODE2.AIS.

Test code for SECTOR_PROCESSING_CONTROLLER and TRACK_WHILE_SCAN, CODE2.ADA

```
$ TYPE CODE2.ADA
with TEXT_IO; use TEXT_IO;
package TRACKING_DATA is
   subtype SECTOR is INTEGER range 1..20;
   subtype RADAR_REPORT_BUFFER is INTEGER range 1..20;

   package BUFF_IO   is new INTEGER_IO(RADAR_REPORT_BUFFER); use BUFF_IO;
   package SECTOR_IO is new INTEGER_IO(SECTOR); use SECTOR_IO;

   task type TRACK_WHILE_SCAN is
      entry NEXT_SECTOR(R : in RADAR_REPORT_BUFFER);
      entry STOP(R : in RADAR_REPORT_BUFFER);
   end TRACK_WHILE_SCAN;


end TRACKING_DATA;-- SPEC
package body TRACKING_DATA is

   task body TRACK_WHILE_SCAN is

   begin
      loop
         select
            when STOP'COUNT = 0  =>
               accept NEXT_SECTOR(R : in RADAR_REPORT_BUFFER) do
                  PUT('                                   PROCESSOR a ');
                  BUFF_IO.PUT(R);
                  PUT_LINE( ' ACCEPTING NEW DATA');
               end;
            or
               accept STOP(R : in RADAR_REPORT_BUFFER) do
                  PUT('                                   PROCESSOR b ');
                  BUFF_IO.PUT(R);
                  PUT_LINE( ' ACCEPTING STOP COMMAND');
               end;
         end select;
     end loop;

  end TRACK_WHILE_SCAN; -- BODY
```

```
end TRACKING_DATA;

with TRACKING_DATA;use TRACKING_DATA;
with TEXT_IO; USE TEXT_IO;
package TRACKING is


   task RIE is
      entry START;
   end RIE;

   task SECTOR_PROCESSING_CONTROLLER is
      entry RIE_START_UP_DONE;
      entry NORTH_SECTOR_PULSE;
      entry NEW_SECTOR_PULSE(R : in RADAR_REPORT_BUFFER);
   end SECTOR_PROCESSING_CONTROLLER;

end TRACKING; -- SPEC

package body TRACKING is

task body SECTOR_PROCESSING_CONTROLLER is

   PROCESSOR_LIST : array (SECTOR'FIRST..SECTOR'LAST) of TRACK_WHILE_SCAN;
   R              : RADAR_REPORT_BUFFER;
   CURRENT_SECTOR : SECTOR := 1;

begin
   accept RIE_START_UP_DONE do
      PUT_LINE('                RIE_START_UP_DONE RECEIVED');
   end;

   accept NORTH_SECTOR_PULSE do
      PUT_LINE('                NORTH_SECTOR_PULSE RECEIVED');
   end;

   loop
      accept NEW_SECTOR_PULSE(R : in RADAR_REPORT_BUFFER) do
         PUT('                NEW SECTOR PULSE RECEIVED FOR SECTOR ');
         BUFF_IO.PUT(R); NEW_LINE;
```

```
      -- terminate task processing three sectors 'ahead' of current sector
      PUT('                STOP ORDER SENT FOR PROCESSOR ');
      SECTOR_IO.PUT((CURRENT_SECTOR + 2) mod SECTOR'LAST + 1); NEW_LINE;
      PROCESSOR_LIST((CURRENT_SECTOR + 2) mod SECTOR'LAST + 1).STOP(
      (CURRENT_SECTOR + 2) mod SECTOR'LAST +1);

      -- initiate task to process the sector data just received
      PUT('                PROCESSOR STARTED FOR ');
      SECTOR_IO.PUT(CURRENT_SECTOR); NEW_LINE;
      PROCESSOR_LIST(CURRENT_SECTOR).NEXT_SECTOR(R);
      end;
      CURRENT_SECTOR := CURRENT_SECTOR mod SECTOR'LAST + 1;
   end loop;

end SECTOR_PROCESSING_CONTROLLER;


   task body RIE is
      TEST_BUFFER : RADAR_REPORT_BUFFER;
      COUNT       : SECTOR;

   begin

      accept START do
         PUT_LINE('RIE TURNED ON d ');
      end;

      COUNT       := SECTOR'FIRST;
      TEST_BUFFER := RADAR_REPORT_BUFFER(COUNT);

      PUT_LINE('RIE_START_UP_DONE SENT');
      SECTOR_PROCESSING_CONTROLLER.RIE_START_UP_DONE;

      PUT_LINE('NORTH SECTOR PULSE SENT');
      SECTOR_PROCESSING_CONTROLLER.NORTH_SECTOR_PULSE;

      loop
         PUT('NEW_SECTOR_PULSE SENT FOR ');
```

```
            SECTOR_IO.PUT(COUNT); NEW_LINE;
            SECTOR_PROCESSING_CONTROLLER.NEW_SECTOR_PULSE(TEST_BUFFER);

            COUNT       := COUNT mod SECTOR'LAST + 1;
            TEST_BUFFER := RADAR_REPORT_BUFFER(COUNT);
         end loop;
      end RIE;
   end TRACKING;

   with TRACKING; use TRACKING;
   with TRACKING_DATA;use TRACKING_DATA;
   with TEXT_IO; USE TEXT_IO;
   procedure TEST is
      task TEST_DRIVER;

      task body TEST_DRIVER is
         begin
            put_line('task TEST_DRIVER ENTERED');
            RIE.START;
            put_line('RIE TURNED ON e ');
            DELAY 25000.0;-- 8 HR
            PUT_LINE(' task TEST_DRIVER ENDS ');
      end TEST_DRIVER;
      begin
         PUT_LINE('PROCEDURE TEST ENTERED');

   end TEST;
```

Results produced by test code for SECTOR_PROCESSING_CONTROLLER and TRACK_WHILE_SCAN, CODE2.AIS.

```
NYU Ada/ED 16.3(5/29/82)                THU  10 JUN 82  09:49:32   PAGE      1

AISfile:   CODE2.AIS

  Binding time: 2.0 seconds

  Begin Ada execution

 task TEST_DRIVER ENTERED
 RIE TURNED ON d
 PROCEDURE TEST ENTERED
 RIE TURNED ON e
 RIE_START_UP_DONE SENT
                 RIE_START_UP_DONE RECEIVED
 NORTH SECTOR PULSE SENT
                 NORTH_SECTOR_PULSE RECEIVED
 NEW_SECTOR_PULSE SENT FOR 1
                 NEW SECTOR PULSE RECEIVED FOR SECTOR 1
                 STOP ORDER SENT FOR PROCESSOR 4
                                 PROCESSOR b 4 ACCEPTING STOP COMMAND
                 PROCESSOR STARTED FOR 1
                                 PROCESSOR a 1 ACCEPTING NEW DATA
    NEW_SECTOR_PULSE SENT FOR 2
                 NEW SECTOR PULSE RECEIVED FOR SECTOR 2
                 STOP ORDER SENT FOR PROCESSOR 5
                                 PROCESSOR b 5 ACCEPTING STOP COMMAND
                 PROCESSOR STARTED FOR 2
                                 PROCESSOR a 2 ACCEPTING NEW DATA
 NEW_SECTOR_PULSE SENT FOR 3
                 NEW SECTOR PULSE RECEIVED FOR SECTOR 3
                 STOP ORDER SENT FOR PROCESSOR 6
                                 PROCESSOR b 6 ACCEPTING STOP COMMAND
                 PROCESSOR STARTED FOR 3
                                 PROCESSOR a 3 ACCEPTING NEW DATA
 NEW_SECTOR_PULSE SENT FOR 4
```

```
                NEW SECTOR PULSE RECEIVED FOR SECTOR 4
                STOP ORDER SENT FOR PROCESSOR 7
                                PROCESSOR b 7 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 4
                                PROCESSOR a 4 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 5
                NEW SECTOR PULSE RECEIVED FOR SECTOR 5
                STOP ORDER SENT FOR PROCESSOR 8
                                PROCESSOR b 8 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 5
                                PROCESSOR a 5 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 6
                NEW SECTOR PULSE RECEIVED FOR SECTOR 6
                STOP ORDER SENT FOR PROCESSOR 9
                                PROCESSOR b 9 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 6
                                PROCESSOR a 6 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 7
                NEW SECTOR PULSE RECEIVED FOR SECTOR 7
                STOP ORDER SENT FOR PROCESSOR 10
                                PROCESSOR b 10 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 7
                                PROCESSOR a 7 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 8
                NEW SECTOR PULSE RECEIVED FOR SECTOR 8
                STOP ORDER SENT FOR PROCESSOR 11
                                PROCESSOR b 11 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 8
                                PROCESSOR a 8 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 9
                NEW SECTOR PULSE RECEIVED FOR SECTOR 9
                STOP ORDER SENT FOR PROCESSOR 12
                                PROCESSOR b 12 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 9
                                PROCESSOR a 9 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 10
                NEW SECTOR PULSE RECEIVED FOR SECTOR 10
                STOP ORDER SENT FOR PROCESSOR 13
                                PROCESSOR b 13 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 10
```

```
                                PROCESSOR a 10 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 11
                NEW SECTOR PULSE RECEIVED FOR SECTOR 11
                STOP ORDER SENT FOR PROCESSOR 14
                                PROCESSOR b 14 ACCEPTING STOP COMMAND*
                PROCESSOR STARTED FOR 11
                                PROCESSOR a 11 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 12
                NEW SECTOR PULSE RECEIVED FOR SECTOR 12
                STOP ORDER SENT FOR PROCESSOR 15
                                PROCESSOR b 15 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 12
                                PROCESSOR a 12 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 13
                NEW SECTOR PULSE RECEIVED FOR SECTOR 13
                STOP ORDER SENT FOR PROCESSOR 16
                                PROCESSOR b 16 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 13
                                PROCESSOR a 13 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 14
                NEW SECTOR PULSE RECEIVED FOR SECTOR 14
                STOP ORDER SENT FOR PROCESSOR 17
                                PROCESSOR b 17 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR *4
                                PROCESSOR a 14 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 15
                NEW SECTOR PULSE RECEIVED FOR SECTOR 15
                STOP ORDER SENT FOR PROCESSOR 18
                                PROCESSOR b 18 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 15
                                PROCESSOR a 15 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 16
                NEW SECTOR PULSE RECEIVED FOR SECTOR 16
                STOP ORDER SENT FOR PROCESSOR 19
                                PROCESSOR b 19 ACCEPTING STOP COMMAND
                PROCESSOR STARTED FOR 16
                                PROCESSOR a 16 ACCEPTING NEW DATA
NEW_SECTOR_PULSE SENT FOR 17
                NEW SECTOR PULSE RECEIVED FOR SECTOR 17
              * STOP ORDER SENT FOR PROCESSOR 0
```

```
                                PROCESSOR b 20 ACCEPTING STOP COMMAND
                    PROCESSOR STARTED FOR 17
                                PROCESSOR a 17 ACCEPTING NEW DATA
 NEW_SECTOR_PULSE SENT FOR 18
                    NEW SECTOR PULSE RECEIVED FOR SECTOR 18
                    STOP ORDER SENT FOR PROCESSOR 1
                                PROCESSOR b 1 ACCEPTING STOP COMMAND
                    PROCESSOR STARTED FOR 18
                                PROCESSOR a 18 ACCEPTING NEW DATA
 NEW_SECTOR_PULSE SENT FOR 19
                    NEW SECTOR PULSE RECEIVED FOR SECTOR 19
                    STOP ORDER SENT FOR PROCESSOR 2
                                PROCESSOR b 2 ACCEPTING STOP COMMAND
                    PROCESSOR STARTED FOR 19
                                PROCESSOR a 19 ACCEPTING NEW DATA
 NEW_SECTOR_PULSE SENT FOR 20
                    NEW SECTOR PULSE RECEIVED FOR SECTOR 20
                    STOP ORDER SENT FOR PROCESSOR 3
                                PROCESSOR b 3 ACCEPTING STOP COMMAND
                   PROCESSOR STARTED FOR 20
                                PROCESSOR a 20 ACCEPTING NEW DATA
 NEW_SECTOR_PULSE SENT FOR 1
                   NEW SECTOR PULSE RECEIVED FOR SECTOR 1
                   STOP ORDER SENT FOR PROCESSOR 4
                                PROCESSOR b 4 ACCEPTING STOP COMMAND
                   PROCESSOR STARTED FOR 1
                                PROCESSOR a 1 ACCEPTING NEW DATA
 NEW_SECTOR_PULSE SENT FOR 2
^Y
```

The file QCODE.ADA contains the four packages described in the beginning of this appendix.

Ada/ED 16.3(5/29/82) TUE 22 JUN 82 08:54:44

QCODE.ADA
QCODE.AIS

```
1
2   package VECTOR_OPERATIONS is
3
4      --This package defines a type VECT which can be viewed logically
5      --as an ordered pair (X,Y).
6      --The functions '+' and '-' have been overloaded to allow the
7      --operations of addition and subtraction on object of this type.
8      --The function '*' has been overloaded to allow multiplication
9      --of objects of type FLOAT and objects of type VECT.  NOTE: This
10     --is left hand multiplication only.
11     --Users of this package may declare objects of type VECT and FLOAT
12     --and perform these operations without concerning themselves with
13     --the details of implementation.
14     --This package would reside in the USER APPLICATION LIBRARY (1.4)
```

```
15      --of COMMAND and CONTROL (1.0).
16
17
18      type VECT is
19         record
20            X : FLOAT;
21            Y : FLOAT;
22         end record;
23
24      function "+" (A,B:VECT)        return VECT;
25      function "*" (A:FLOAT;B:VECT) return VECT;
26      function "-" (A,B:VECT)        return VECT;
27
28   end VECTOR_OPERATIONS;
29
30
31   package body VECTOR_OPERATIONS is
32
33      function "+" (A,B:VECT) return VECT IS
34
35      begin
36         return ( (X => A.X + B.X,
37                   Y => A.Y + B.Y));
38
39      end;
40
41      function "*" (A : FLOAT; B : VECT) return VECT is
42
43      begin
44         return ((X => A * B.X,
45                  Y => A * B.Y));
46      end;
47
48       function "-" (A,B : VECT) return VECT IS
49
50       begin
51          return ((X => A.X - B.X,
52                   Y => A.Y - B.Y));
53       end;
54
```

```
55   end VECTOR_OPERATIONS;
56
57
58   with VECTOR_OPERATIONS; use VECTOR_OPERATIONS;
59   with CALENDAR; use CALENDAR;
60
61   package TRACK_TYPES IS
62
63      --This package contains the type definitions required by subprograms
64      --and tasks of TRACKING_OPERATIONS.  It requires the use of the
65      --package VECTOR_OPERATIONS to define subtypes of VECT.
66
67
68      subtype      DEVIATION_VECT      is VECT;    --deviation vector
69      subtype      PREDICTED_POS       is VECT;    --predicted position
70      subtype      SMOOTH_POS          is VECT;    --smoothed position
71      subtype      SMOOTH_VEL          is VECT;    --smoothed velocity
72      subtype      REPORT_POS          is VECT;    --report position
73
74      subtype      ALTITUDE_GATE       is INTEGER range 2500..5000;
75      subtype      ASSOCIATION_COUNT   is INTEGER range 0..15;  --number
76         --of times a track has associated
77      type         STATUS              is (LOCAL, REMOTE);  --origin of track
78      subtype      CONSEC_HIT          is INTEGER range 0..10;  --consecutive
79         --hits in the outergate during correlation
80      subtype      ALTITUDE            is INTEGER range 1000..20_000;
81      subtype      TRACK_NO            is integer range 1..20;--explicit for test
82      subtype      SECTOR_NO           is INTEGER range 1..20;
83      subtype      SMOOTHING_INDEX     is INTEGER range 1..5;
84      type         REPORT_NO           is range 1..20;--explicit for test
85      subtype      RADAR_REPORT_BUFFER is INTEGER range 1..20;
86
87
88      type SMOOTH_CONSTANTS IS
89         RECORD
90            POS : FLOAT;
91            VEL : FLOAT;
```

```
92          end RECORD;
93
94
95       SMOOTH_TABLE :constant  array (SMOOTHING_INDEX) of SMOOTH_CONSTANTS:=
96          ((0.4375,0.625),
97           (0.5625,0.1875),
98           (0.6875,0.3125),
99           (0.8125,0.4375),
100          (0.9375,0.5625));
101         --constants used by smoothing.  Components are 'POS' for position

102         --'VEL' for velocity.
103
104      type TRACK_RECORD is
105         RECORD
106            STA       : STATUS;
107            SEC_NO    : SECTOR_NO;
108            TK_NO     : TRACK_NO;
109            EXISTS    : BOOLEAN;
110            PRED_POS  : PREDICTED_POS;
111            S_POS     : SMOOTH_POS;
112            VEL       : SMOOTH_VEL;  --THE PREDICTED VELOCITY
113            ALT       : ALTITUDE;    --THE PREDICTED ALTITUDE
114            INDEX     : SMOOTHING_INDEX;
115            RGATE     : BOOLEAN;     --'True' indicates report correlated during

116                                      --recovery gate correlation.
117            OUTGATE   : BOOLEAN;     --'True' indicates report correlated during

118                                      --outergate correlation.
119            ALTGATE   : ALTITUDE_GATE;
120            A_CNT     : ASSOCIATION_COUNT;
121            T         : TIME;        --Time since last smoothing
122            D_VEC     : DEVIATION_VECT;  --The deviation vector.
123            CONSEC    : CONSEC_HIT;  --Consecutive hits in the outergate.
124
125         end RECORD;
126
127      type REPORT_TRACK_PAIR is
128         record
```

```
129            TK_NO : TRACK_NO;
130            RP_NO : REPORT_NO;
131         end record;
132
133      type REPORT is
134         record
135            RN     : REPORT_NO;
136            SEC_NO : SECTOR_NO;
137            POS    : REPORT_POS;
138            ALT    : ALTITUDE;
139         end record;
140
141      type REPORT_FILE is array (REPORT_NO) of REPORT;
142      R_F : REPORT_FILE;
143
144      type ASSOCIATED_SECTOR is
145         record
146            NUM_ASSOC: INTEGER RANGE 1..INTEGER(TRACK_NO'LAST) ;
147            PAIRS     : array(0..10) of REPORT_TRACK_PAIR;-- explicit for test
148         end record;
149
150      type ASSOCIATED_REPORT_FILE is array(SECTOR_NO) of ASSOCIATED_SECTOR;
151      ARF : ASSOCIATED_REPORT_FILE;
152
153      type SECTOR_DATA is array (TRACK_NO'FIRST..TRACK_NO'LAST) of TRACK_RECORD
154
155   end TRACK_TYPES; --SPEC
156
157   with TRACK_TYPES; use TRACK_TYPES;
158   package MANAGE_TABLES is -- (2.3.2)
159
160   -- <><><><><><><><><><><><> PACKAGE MANAGE_TABLES<><><><><><><><><><><><><
161
162   --    The manage tables package contains the track storage file and the
163   -- access entry points to the file.  Based on the PDL for the track
164   -- while scan functions entries to read and write one track record and
165   -- entries to read and write one sector have been provided
```

```
166
167
168       task FILE_MGR is --(2.3)
169          entry READ_SECTOR (S : in SECTOR_NO; D : out SECTOR_DATA);
170          entry WRITE_SECTOR(S : in SECTOR_NO; D : in  SECTOR_DATA);
171          entry READ (S : in SECTOR_NO; T : in TRACK_NO; R : out TRACK_RECORD)

172          entry WRITE(S : in SECTOR_NO; T : in TRACK_NO; R : in  TRACK_RECORD)

173       end FILE_MGR;
174
175       type T_S_F is array (SECTOR_NO'FIRST..SECTOR_NO'LAST,
176            TRACK_NO'FIRST..TRACK_NO'LAST) of TRACK_RECORD;
177       TRACK_STORAGE_FILE : T_S_F;
178       --  THIS TYPE DEFINITION AND OBJECT DECLARATION WOULD NORMALLY RESIDE
179       --  IN THE BODY OF MANAGE_TABLES BECAUSE THE PHYSICAL IMPLEMENTATION
180       --  OF THE FILE SHOULD REMAIN HIDDEN.  IN THIS CASE DUE TO TESTING
181       --  REQUIREMENTS IT MUST BE VISIBLE.
182
183
184    end MANAGE_TABLES;
185
186    package body MANAGE_TABLES is
187
188       task body FILE_MGR is
189
190          begin
191             loop
192                select
193                   accept READ_SECTOR(S : in SECTOR_NO; D : out SECTOR_DATA)

194                      for I in TRACK_NO'FIRST..TRACK_NO'LAST loop
195                         D(I) := TRACK_STORAGE_FILE(S,I);
196                      end loop;
197                   end;
198                or
199                   accept WRITE_SECTOR(S : in SECTOR_NO; D : in SECTOR_DATA)

200                      for I in TRACK_NO'FIRST..TRACK_NO'LAST loop
201                         TRACK_STORAGE_FILE(S,I) := D(I);
```

```
202                     end loop;
203                  end;
204               or
205                  accept READ(S : in SECTOR_NO; T : in TRACK_NO;
206                              R : out TRACK_RECORD) do
207                              R := TRACK_STORAGE_FILE(S,T);
208                  end;
209               or
210                  accept WRITE(S : in SECTOR_NO; T : in TRACK_NO;
211                              R : in TRACK_RECORD) do
212                              TRACK_STORAGE_FILE(S,T) := R;
213                  end;
214
215            end select;
216         end loop;
217      end FILE_MGR; -- body
218
219   end MANAGE_TABLES;
220
221
222   with VECTOR_OPERATIONS, CALENDAR, MANAGE_TABLES,TRACK_TYPES;
223   use  VECTOR_OPERATIONS, CALENDAR, MANAGE_TABLES,TRACK_TYPES;
224
225   package TRACKING_OPERATIONS IS
226
227
228      --<<<<<  FUNCTIONS REQUIRED BY TRACKING COMPONENTS  >>>>>
229
230      function DET_DEVIATION_VECT (A : REPORT_POS;B : PREDICTED_POS)
231         return DEVIATION_VECT;
232
233      function SMOCON(I : SMOOTHING_INDEX) return SMOOTH_CONSTANTS;
234
235
236      --.....
237
238
239
240      --<<<<<  PROCEDURES REQUIRED BY TRACKING COMPONENTS  >>>>>
241
```

```
242
243      procedure PREDICTION (X : in SECTOR_NO); --(2.2.3.4)
244
245      procedure SMOOTHING (X : in SECTOR_NO);  --(2.2.3.3)
246
247
248      procedure SMOOTH_REPORT_TRACK(RN : in REPORT_NO; T : in TRACK_NO;
249         S : in SECTOR_NO);
250         -- (2.2.3.3)
251      --.....................................................
252
253
254      task type TRACK_WHILE_SCAN is  --(2.2.3)
255         entry NEXT_SECTOR(R : in RADAR_REPORT_BUFFER);
256         entry STOP(S : in SECTOR_NO);
257      end TRACK_WHILE_SCAN;
258
259      task SECTOR_PROCESSING_CONTROLLER is
260         entry RIE_START_UP_DONE;
261         entry NORTH_SECTOR_PULSE;
262         entry NEW_SECTOR_PULSE(R : in RADAR_REPORT_BUFFER);
263      end SECTOR_PROCESSING_CONTROLLER;
264
265
266
267   end TRACKING_OPERATIONS;
268
269   package body TRACKING_OPERATIONS IS
270
271
272
273
274      function DET_DEVIATION_VECT (A : REPORT_POS;B : PREDICTED_POS)
275         return DEVIATION_VECT IS
276
277            --This function determines the deviation vector between a report
278            --and a track.
279
280            LOCAL : DEVIATION_VECT;
```

```
281
282      begin
283
284         LOCAL := A - B;
285         return LOCAL;
286      end DET_DEVIATION_VECT;
287
288
289     function SMOCON(I : SMOOTHING_INDEX) return SMOOTH_CONSTANTS IS
290
291        --This function returns both velocity and position smoothing
292        --constants.  See definition of SMOOTH_CONSTANTS from
293        --package TRACK_TYPES.
294
295        begin
296
297          return(SMOOTH_TABLE(I));
298        end;
299
300
301      procedure PREDICTION (X : in SECTOR_NO) IS --(2.2.3.4)
302
303         --Prediction reads a sector of TRACK_RECORD(s) utilizing the
304         --task FILE_MGR.
305         --The position of each track is predicted using the following
306         --equation:   Predicted_position = smoothed_position +
307         -- (radar_scan_time + time_since_previous_smoothing) *
308         -- smoothed_velocity.
309         --After each track has been predicted the TRACK_RECORDS(s) are
310         --updated using the task FILE_MGR.
311
312         Y          : SECTOR_DATA;
313         TS         : DURATION;
314         SCAN_TIME : DURATION := 3.0; -- EXPLICIT FOR TEST
315
316      begin
317
318         FILE_MGR.READ_SECTOR(X,Y);
319         FOR I in TRACK_NO'FIRST..TRACK_NO'LAST LOOP
320            if Y(I).EXISTS then
```

```
321              TS := CLOCK() - Y(I).T;
322              --Y(I).PRED_POS := Y(I).S_POS + (SCAN_TIME + TS) * Y(I).VEL;
323              --PRECEEDING LINE REPLACED FOR TEST PURPOSES BY NEXT LINE
324              -- IN ORDER TO SIMULATE ONE RADAR SWEEP
325              Y(I).PRED_POS := Y(I).S_POS + 6.0 * Y(I).VEL;
326           end IF;
327        end LOOP;
328        FILE_MGR.WRITE_SECTOR(X,Y);
329     end PREDICTION;
330
331  procedure SMOOTHING (X : in SECTOR_NO) is
332
333     --Smoothing assigns the report_to_track pairings determined
334     --during ASSOCIATION and stored in ARF (the associated_report_file)
335     --to the object 'C'.
336     --The procedure SMOOTH_REPORT_TRACK is then called to perform the
337     --actual mathematical calculations.  The update of TRACK_STORAGE_FILE
338     --is donedurins the SMOOTH_REPORT_TRACK procedure.
339
340     C  : ASSOCIATED_SECTOR;
341     TN : TRACK_NO;
342     RN : REPORT_NO;
343
344  begin
345     C := ARF(X);
346     for I in 1..ARF(X).NUM_ASSOC loop
347        TN := C.PAIRS(I).TK_NO;
348        RN := C.PAIRS(I).RP_NO;
349        SMOOTH_REPORT_TRACK (RN, TN, X);
350     end loop;
351  end SMOOTHING;
352
353
354  procedure SMOOTH_REPORT_TRACK (RN : in REPORT_NO;
355     T : in TRACK_NO;    S : in SECTOR_NO) is  --(2,2,3,3)
356
357       R  : REPORT;
358       TR : TRACK_RECORD;
359       C  : TIME;
360       SC : SMOOTH_CONSTANTS;
```

```
361
362   begin
363
364       FILE_MGR.READ(S,T,TR);
365       --<<<<<  DETERMINE ALTITUDE GATE  >>>>>
366       R := R_F(RN);
367       if R.ALT in TR.ALT - TR.ALTGATE..TR.ALT + TR.ALTGATE then
368          if TR.ALTGATE > ALTITUDE_GATE'FIRST then
369             TR.ALTGATE := TR.ALTGATE - 500;
370          end if;
371       else
372          if TR.ALTGATE < ALTITUDE_GATE'LAST then
373             TR.ALTGATE := TR.ALTGATE + 500;
374          end if;
375       end if;
376
377       --<<<<<  END  DETERMINE ALTITUDE GATE  >>>>>
378
379       --<<<<<  BEGIN SMOOTHING of ALTITUDE  (2.2.3.3.3) >>>>>
380       if TR.ALTGATE < ALTITUDE_GATE'LAST then
381          TR.ALT := (TR.ALT + R.ALT) / 2;
382       else
383          TR.ALT := R.ALT;
384       end IF;
385
386       --<<<<<  end SMOOTHING of ALTITUDE  >>>>>
387
388       --<<<<<  DETECT TRACK MANEUVERABILITY  (2.2.3.3.4) >>>>>
389       -- This block also determines the smoothing index.
390
391       if TR.INDEX = 1 then
392          if TR.CONSEC >= 2 then
393             TR.INDEX   := 5;
394             TR.CONSEC  := 0;
395          end if;
396       else
397          if TR.RGATE then
398             TR.INDEX := TR.INDEX - 1;
399          elsif  TR.OUTGATE and TR.INDEX < 5 then
400             TR.INDEX := TR.INDEX + 1;
```

```
401          end if;
402       end if;
403
404       --<<<<<  END DETECT TRACK MANEUVERABILITY  >>>>>
405
406       --<<<<<  SMOOTH VELOCITY (2.2.3.3.2) >>>>>
407       TR.D_VEC := DET_DEVIATION_VECT(R.POS , TR.PRED_POS);
408       --deviation vector required for smoothing of velocity
409
410       C  := CLOCK();
411       SC := SMOCON(TR.INDEX);  -- A FUNCTION CALL RETURNING THE
412                                -- SMOOTHING CONSTANTS
413       --  TR.VEL := TR.VEL + (SC.VEL / (C - TR.T)) * TR.D_VEC;
414       --  THE PRECEDING LINE OF CODE IS REPLACED BY THE
415       --  FOLLOWING LINE TO SIMULATE ONE RADAR SWEEP FOR TEST PURPOSE
416       TR.VEL := TR.VEL + ((SC.VEL/6.0) * TR.D_VEC);
417       --<<<<<  end SMOOTH VELOCITY  >>>>>
418
419       --<<<<<  SMOOTH POSITION  (2.2.3.3.1) >>>>>
420       TR.S_POS := TR.S_POS + SC.POS * TR.D_VEC;
421
422       --<<<<<  end SMOOTH POSITION  >>>>>
423
424       TR.T := CLOCK();
425       FILE_MGR.WRITE(S,T,TR);
426
427    end SMOOTH_REPORT_TRACK;
428
429       -- The task type TRACK_WHILE_SCAN (2.2.3) is the heart of the
430       -- tracking process.
431       -- One task of this type will process reports for each sector of the
adar
432       -- coverage area.
433
434       -- <><><><><><><><><><><><> NORMAL OPERATION <><><><><><><><><><><><><>
><><>
435
436       --   After a rendezvous with the SECTOR_PROCESSING_CONTROLLER to obta
n a
437       -- sectors worth of radar reports, TRACK_WHILE_SCAN makes calls to pe
```

```
438      -- the basic radar report processing sequence: correlation, association
439      -- SMOOTHING and prediction.  TRACK_WHILE_SCAN can also be told, via
440      -- rendezvous with SECTOR_PROCESSING_CONTROLLER, to stop current
441      -- processing and proceed immediately to prediction
442
443     task body TRACK_WHILE_SCAN is
444      X : SECTOR_NO;
445
446      begin
447         loop
448            select
449               when STOP'COUNT = 0  =>
450                  accept NEXT_SECTOR(R : in RADAR_REPORT_BUFFER);
451                  X := SECTOR_NO(R);
452                  SMOOTHING (X);
453               or
454                  accept STOP(S : in SECTOR_NO);
455                  PREDICTION(S);
456
457            end select;
458        end loop;
459
460     end TRACK_WHILE_SCAN;
461
462
463   --   The SECTOR_PROCESSING_CONTROLLER is the highest level program unit IN
 the
464   -- implementation of the structure chart entity TRACKING.  It contains THE
465   -- control logic necessary to implement the scheme of having one task to
466   -- process each sectors worth of radar reports.
467
468   -- <><><><><><><><><><> INITIALIZATION <><><><><><><><><><><><><><><><><><><><
<><><><>
469
470   --   The SECTOR_PROCESSING_CONTROLLER is started up by the reception of
471   -- a signal from the radar interface equipment (RIE) indicating that
472   -- this device is ready for normal service.  The SECTOR_PROCESSING_
```

```
473    -- controller then waits for the first north sector pulse to be trans-
474    -- mitted by the RIE.  The reception of this pulse accomplishes the
475    -- initial synchronization of the SECTOR_PROCESSING_CONTROLLER with the
476    -- radar scan interval.
477
478    -- <><><><><><><><><><> NORMAL OPERATION <><><><><><><><><><><><><><><><><><><><
   <><><>
479
480    --   Upon reception of the pulse immediately following the north sector pulse
481    -- the SECTOR_PROCESSING_CONTROLLER begins normal operation.  Normal operation
482    -- consists of a sequence of three activities which take place in synchron
   ization
483    -- with radar scan time.  The three activities are: 1. rendezvous with the
    RIE
484    -- to obtain the radar reports for the sector just scanned; 2. notification
      of
485    -- the task processing three sectors 'ahead' that it must begin prediction,
      and
486    -- 3. initiation of processing the current sector by the appropriate task
487
488    -- <><><><><><><><><><><> ERROR SITUATIONS <><><><><><><><><><><><><><><><><><><
   <><><>
489
490    --   Synchronization with the RIE must be maintained.  This synchronization
      is
491    -- checked after the reception of each sectors worth of radar reports.  In the
492    -- event of loss of synchronization the SECTOR_PROCESSING_CONTROLLER will
   re-
493    -- synch itself to the sector that the RIE passed.  Tasks processing the
494    -- sectors passed over by this re-synch will return to synch on the next
   scan
495    task body SECTOR_PROCESSING_CONTROLLER is
496
497       PROCESSOR_LIST : array (SECTOR_NO'FIRST..SECTOR_NO'LAST)
498                                of TRACK_WHILE_SCAN;
499       R              : RADAR_REPORT_BUFFER;
```

```
500       CURRENT_SECTOR : SECTOR_NO := 1;
501
502    begin
503       accept RIE_START_UP_DONE;
504       accept NORTH_SECTOR_PULSE;
505
506       loop
507          accept NEW_SECTOR_PULSE(R : in RADAR_REPORT_BUFFER);
508
509    --    if SECTOR_OF(R) /= CURRENT_SECTOR then--reestablish lost synchronization
510    --       do
511    --          send a message to the C&C
512    --          CURRENT_SECTOR := SECTOR_OF(R);
513    --       end;
514    --    end if;
515
516          -- terminate task processing three sectors 'ahead' of current sector
517          PROCESSOR_LIST((CURRENT_SECTOR + 2) mod SECTOR_NO'LAST + 1).STOP(
518             (CURRENT_SECTOR + 2) mod SECTOR_NO'LAST + 1);
519
520
521          -- initiate task to process the sector data just received
522          PROCESSOR_LIST(CURRENT_SECTOR).NEXT_SECTOR(R);
523
524          CURRENT_SECTOR := CURRENT_SECTOR mod SECTOR_NO'LAST + 1;
525       end loop;
526
527    end SECTOR_PROCESSING_CONTROLLER;
528
529
530    end TRACKING_OPERATIONS;
531
532
533
534

no parse errors detected
Parsing time: 426 seconds
```