

UNCLASSIFIED

AD NUMBER

ADB013480

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

Distribution authorized to U.S. Gov't. agencies only; Test and Evaluation; 01 AUG 1975. Other requests shall be referred to Ballistic Missile Defense Advanced Technology Center, PO Box 1500, Huntsville, AL 35807.

AUTHORITY

usa bmdatc ltr, 6 may 1977

THIS PAGE IS UNCLASSIFIED

THIS REPORT HAS BEEN DELIMITED
AND CLEARED FOR PUBLIC RELEASE
UNDER DOD DIRECTIVE 5200.20 AND
NO RESTRICTIONS ARE IMPOSED UPON
ITS USE AND DISCLOSURE.

DISTRIBUTION STATEMENT A

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

ADBU13480

27332-6921-024

(2)

AD NO. 1
DDC FILE COPY

SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY

CDRL C011

1 SEPTEMBER 1976

Prepared For
BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER
DASG60-75-C-0022

DDC
RECEIVED
SEP 23 1976
C

TRW.
DEFENSE AND SPACE SYSTEMS GROUP
HUNTSVILLE, ALABAMA

TRW

SN27332.000
1780.1.76-5765
1 September 1976

Director, Ballistic Missile Defense
Advanced Technology Center
P. O. Box 1500
Huntsville, Alabama 35807

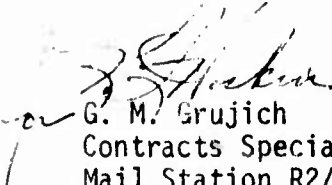
Attention: ATC-P

Subject: Contract DASG60-75-C-0022
CDRL Sequence No. C011
Software Requirements Engineering Methodology

In accordance with the requirements of the subject CDRL, two (2) copies of the Software Requirements Engineering Methodology are herewith submitted.

Distribution of this report is shown below.

TRW SYSTEMS, INC.


G. M. Grujich
Contracts Specialist
Mail Station R2/1054
TRW Defense and Space Systems Group

Enclosures: As stated

cc: BMDPO, Arlington, VA
ATTN: DACS-BMT, Dr. R. Merwin (1)
DACs-BMS (1)
BMDSC, Huntsville, AL
ATTN: BMDSC-C (w/o encl.)
DDC, Alexandria, VA (2)
Aeronutronic Ford, Willow Grove, PA (1)
SDC, Huntsville, AL
ATTN: Mr. Robert Covelli (1)
Library (10)
Texas Instruments, Huntsville, AL
ATTN: Dr. R. Bates (1)

Auburn University, Auburn, AL
ATTN: Dr. H. Troy Nagle (1)
CSC, Huntsville, AL
ATTN: Mr. P. C. Belford (1)
GRC, Santa Barbara, CA
ATTN: Dr. Charles Perkins (1)
Logicon, San Pedro, CA
ATTN: Wm. C. Nielson (1)
SAI, Huntsville, AL
ATTN: Mr. Robert Curry (1)
Univ. of Calif., Berkley, CA
ATTN: Dr. C. V. Ramamoorthy (1)
AFPRO-TRW (w/o encl.)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

20. (Continued)

The methodology systematically develops the specification from source documentation at the system level, documenting omissions and errors of the source materials in the process. The produced requirements are provably consistent, and may be validated against system objectives through the generated simulation. The entire process is subject to systematic management through definable and verifiable milestones supported by REVS.

APPROVAL	
NTES	Write Section ☐
DIS	Box Section ☒
DATE	☐
JUSTIFICATION	
BY DISTRIBUTION/AVAILABILITY NOTES	
Dist.	Avail. Notes
B	

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CDRL C011	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 Software Requirements Engineering Methodology.		5. TYPE OF REPORT & PERIOD COVERED 7 Technical Report.
7. AUTHOR(s) 10 M. D. Richter, J. D. Mason, et al M. W. Alford, I. R. Burns, H. A. Helton		6. PERFORMING ORG. REPORT NUMBER 14 TRW - 27332-6921-024 ✓
9. PERFORMING ORGANIZATION NAME AND ADDRESS TRW Defense and Space Systems Group 7702 Governors Drive, West Huntsville, Alabama 35805		8. CONTRACT OR GRANT NUMBER(s) 15 DASG60-75-C-0022 ✓
11. CONTROLLING OFFICE NAME AND ADDRESS Ballistic Missile Defense Advanced Technology Center, P. O. Box 1500, Huntsville, AL 35807 ATTN: ATC-P		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 6.33.04.A 12/31/76
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE 11 1 Sep 1976
		13. NUMBER OF PAGES 321
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution limited to U. S. Government Agencies only; test and evaluation, 1 August 1975. Other requests for this document must be referred to Ballistic Missile Defense Advanced Technology Center, P. O. Box 1500, Huntsville, AL 35807. ATTN: ATC-P		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Software Specification Software Requirements Software Development		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A methodology is presented for the development and management of software specifications. The technique is built upon a language (RSL) readable both by a computer and by man, and a set of tools termed collectively the Requirements Engineering and Validation System (REVS). The tools provided for retention of all requirements in a relational data base from which documentation, consis- tency analyses, and simulations may be constructed automatically.		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407 674

mt

27332-6921-024

SOFTWARE REQUIREMENTS
ENGINEERING METHODOLOGY

CDRL C011

1 SEPTEMBER 1976

DISTRIBUTION LIMITED TO U. S. GOVERNMENT AGENCIES ONLY;
TEST AND EVALUATION, 1 AUG 75. OTHER REQUESTS FOR THIS
DOCUMENT MUST BE REFERRED TO BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER, ATTN: ATC-P, P.O. BOX 1500,
HUNTSVILLE, ALABAMA 35807.

THE FINDINGS OF THIS REPORT ARE
NOT TO BE CONSTRUED AS AN OFFICIAL
DEPARTMENT OF THE ARMY POSITION.

Prepared For

BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER

DASG60-75-C-0022

TRW.

DEFENSE AND SPACE SYSTEMS GROUP
Huntsville, Alabama

27332-6921-024

SOFTWARE REQUIREMENTS
ENGINEERING METHODOLOGY

CDRL C011

1 SEPTEMBER 1976

Principal Authors:

M. D. Richter - Technical
J. D. Mason - Management

Approved By:



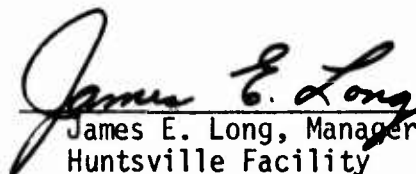
L. R. Marker, Manager
Software Requirements
Engineering Program

Principal Contributors:

M. W. Alford
I. F. Burns
H. A. Helton
J. T. Lawson



M. D. Richter, Head
Methodology Research
and Development



James E. Long, Manager
Huntsville Facility

Prepared For

BALLISTIC MISSILE DEFENSE
ADVANCED TECHNOLOGY CENTER

DASG60-75-C-0022

TRW

DEFENSE AND SPACE SYSTEMS GROUP
Huntsville, Alabama

TABLE OF CONTENTS

<u>Section</u>	<u>Title</u>	<u>Page</u>
1.0	INTRODUCTION.	1-1
1.1	BACKGROUND	1-1
1.2	SCOPE AND CONTENT OF THIS MANUAL	1-3
1.3	OVERVIEW OF SREM	1-4
	1.3.1 The Requirements Statement Language (RSL) . .	1-5
	1.3.2 The Requirements Engineering and Validation System (REVS)	1-5
	1.3.3 The Engineering Methodology	1-7
	1.3.4 Specification Management.	1-8
1.4	APPLICABILITY.	1-8
1.5	TERMINOLOGY.	1-8

PART I - TECHNICAL APPROACH

2.0	SREM OVERVIEW	2-1
2.1	THE TRACK LOOP SYSTEM EXAMPLE.	2-3
	2.1.1 Preliminary Ballistic Missile Defense System.	2-5
	2.1.2 TLS Requirements.	2-5
2.2	SUMMARY OF APPENDICES.	2-5
3.0	FUNCTIONAL REQUIREMENTS	3-1
3.1	PHASE 1 - DEFINITION OF SUBSYSTEM ELEMENTS	3-3
	3.1.1 Initial Inputs.	3-4
	3.1.2 Interface Definition.	3-5
	3.1.3 Message Definition.	3-6
	3.1.4 The Interface Data Hierarchy.	3-14
	3.1.5 Problems of Definition.	3-16
	3.1.6 R NET Definition.	3-17
	3.1.7 Entity Definition	3-25
	3.1.8 The Entity Data Hierarchy	3-27
	3.1.9 Independent FILES	3-30
	3.1.10 Summary of Phase I.	3-33
3.2	PHASE 2 - EVALUATION OF THE KERNEL	3-33
	3.2.1 Data Naming Conventions	3-34
	3.2.2 Structural Data Definitions	3-35
	3.2.3 Entering R NETs in the ASSM	3-36
	3.2.4 The STRUCTURE OF an R NET	3-38
	3.2.5 Checking the Kernel with the Aid of RADX. . .	3-39
	3.2.6 Summary of Phase 2.	3-41

TABLE OF CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.3	PHASE 3 - COMPLETION OF THE FUNCTIONAL DEFINITION. .	3-42
3.3.1	Data Transactions	3-43
3.3.2	RADX Evaluation of Data Transactions.	3-45
3.3.3	Hierarchy Transitions	3-47
3.3.4	Further Data Definition	3-49
3.3.5	Evaluation of the ASSM Using RADX	3-53
3.3.6	Summary of Phase 3.	3-59
3.4	PHASE 4 - DEVELOPMENT OF FUNCTIONAL MODELS	3-60
3.4.1	Betas	3-60
3.4.2	Local Data.	3-61
3.5	TRACEABILITY	3-63
3.5.1	Originating Requirements.	3-63
3.5.2	References.	3-63
3.5.3	Decisions	3-64
3.5.4	Relating to Sources	3-64
3.6	INFORMATIVE MATERIAL	3-65
3.6.1	Description	3-65
3.6.2	Synonym	3-66
3.6.3	Authorship.	3-66
3.6.4	Complementary Relationships	3-66
3.6.5	Structural References	3-66
3.7	ANALYTIC MODELS.	3-67
4.0	PERFORMANCE REQUIREMENTS.	4-1
4.1	LOCATE TEST POINTS	4-5
4.2	DEFINE DATA AND TESTS.	4-7
4.3	DEFINE SUPPLEMENTAL VALIDATION POINTS AND DATA . . .	4-13
<u>PART II - MANAGEMENT APPROACH</u>		
5.0	INTRODUCTION.	5-1
6.0	DEFINING MEASURABLE MILESTONES.	6-1
6.1	SOFTWARE REQUIREMENTS DEVELOPMENT.	6-3
6.2	SOFTWARE REQUIREMENTS VALIDATION	6-7
6.3	SUMMARY.	6-8

TABLE OF CONTENTS (Continued)

<u>Section</u>	<u>Title</u>	<u>Page</u>
7.0	PLANNING.	7-1
	7.1 PRELIMINARY GUIDELINES	7-1
	7.2 COST MODEL	7-2
	7.3 SCHEDULING	7-3
8.0	MANAGEMENT CONTROL.	8-1
	8.1 CONTROL MECHANISMS	8-1
	8.2 CHANGE CONTROL	8-4
	8.3 SELLING THE SOFTWARE REQUIREMENTS.	8-7
9.0	CONCLUSIONS	9-1
APPENDIX A	- RSL TERMINOLOGY	A-1
APPENDIX B	- DRAFT REPRESENTATION OF THE KERNEL OF TLS	B-1
APPENDIX C	- TLS KERNEL.	C-1
APPENDIX D	- TLS REQUIREMENTS NETWORKS	D-1
APPENDIX E	- COMPLETE TLS DATA BASE.	E-1
APPENDIX F	- TLS SOURCE SPECIFICATIONS	F-1
REFERENCES		R-1

LIST OF FIGURES

<u>Figure</u>	<u>Title</u>	<u>Page</u>
2-1	Track Loop System	2-4
3-1	RSL Subsystem Entries	3-7
3-2	An Elementary Interface Hierarchy	3-9
3-3	C ² Input Hierarchy.	3-12
3-4	RSL Message Entries	3-13
3-5	Interface Data Hierarchy.	3-15
3-6	RSL Decision Entry.	3-18
3-7	Three Asynchronous R_NETs	3-23
3-8	Entity Data Hierarchy	3-28
3-9	Entity Hierarchy.	3-31
3-10	RSL Entity Entry.	3-32
3-11	RSL Data Entry.	3-37
3-12	RSL Initial ALPHA Entry	3-46
3-13	RSL Additional ALPHA Entry.	3-48
3-14	RSL BETA Entry.	3-62
4-1	Performance Requirements Statements Representation at the Completion of SREM Step - Locate Test Points.	4-8
4-2	Performance Requirements Statements Representation at the Completion of SREM Step - Define Data and Tests	4-14
6-1	Overview of SREM Activities (Development and Validation of Functional Requirements)	6-2
6-2	Sample Activity Network for Software Requirements Development	6-4
6-3	Sample Decisions from Track Loop.	6-6
6-4	Sample Activity Network for Software Requirements Validation.	6-9
7-1	Rough Spread of Activities by Level	7-7
7-2	Rough Spread of Manpower for Example.	7-9
7-3	Sample Milestone Schedule for Small Project	7-11
7-4	Sample Milestone Schedule of Figure 7-3 With Slack Time Added	7-12
8-1	Interests Shared by SRE	8-2
8-2	Change Flow Into Update of Baseline	8-6

LIST OF TABLES

<u>Table</u>	<u>Title</u>	<u>Page</u>
7.1	Definition of Symbols Used in Cost Model.	7-4
8.1	SREM Focus of Management Control on Substantive Issues. .	8-3
8.2	Control Mechanisms for Major Managerial Control Issues. .	8-5
8.3	Selling Software Requirements	8-8

1.0 INTRODUCTION

1.1 BACKGROUND

The history of digital computing can be meaningfully traced for some three decades, to the ENIAC and UNIVAC I systems of the 1940's. For the first half of that period, limitations of computer hardware were the primary constraint on the application of digital systems. In the past ten to fifteen years, however, hardware technology has improved to the point where software technology has become the limiting factor. The tremendous speed and computational power of modern computers has made possible very complex and sophisticated systems. The software imbedded in such systems provides not only the mathematical data transformations required, but also provides the control functions of many of the system components (such as radars) and of the system as a whole. Therefore, the software is uniquely critical to the successful operation and performance of the system.

The need for improving the techniques of designing, building, testing, and managing software has been well understood for several years and has resulted in vigorous research and development within the Government, industry, and the universities. This has led to the development of improved programming languages, development support tools, and management approaches, as well as a strong theoretical basis in such areas as queueing theory and dynamic programming and many pragmatic development approaches such as structured programming and top down design. While significant improvements have been obtained in the state-of-the-art in program design, implementation, and testing, much additional research is needed and is being actively pursued -- especially in such areas as proof-of-correctness, data base design, etc.

There is an additional phase of software development which is especially critical: the definition and specification of the functional and performance requirements which the software must satisfy. This phase is especially critical due to the very high cost and schedule leverage which exists. Simple errors in the requirements, if not detected until after the software has been built, are extremely expensive in terms of time and manpower to correct. While it has been apparent for quite some time that the state-of-the-art in developing requirements needed improvement, it was not possible to accurately

identify and quantify the improvements needed until after the design implementation, and test phases became more predictable and controlled. Thus, in the fall of 1973, BMDATC initiated the Software Requirements Engineering Program with the objective of developing a set of tools and techniques for defining and specifying the software requirements for ballistic missile defense (BMD) software. The result of that research is the Software Requirements Engineering Methodology (SREM) which is described in this report.

The first step in developing the Software Requirements Engineering Methodology was to determine the properties required of a specification and of the individual requirements of which it is composed. Returning to first principles, we note that:

- A specification is the set of all requirements which must be satisfied, and the identification of the subsets which must be met concurrently; and
- A specification is neither legally binding nor realizable unless it is consistent with both the laws of logic and the laws of nature.

In addition, we observe that

- A specification defines the properties required of a product such that any delivery satisfying the specification satisfies the objectives of the specifier.

Taken together, the above truisms lead to a set of properties which a specification must have from a technical point of view:

- Internal Consistency
- Consistency with the physical universe
- Freedom from ambiguity.

Economic and management considerations lead to an additional set of properties which a good specification must exhibit:

- Clarity
- Minimality
- Predictability of specification development
- Controllability of software development.

Since freedom from ambiguity is mandatory, we naturally looked to a machine-readable statement of the requirements. It is a known principle of computer operation that input ambiguities can be tolerated only insofar as they are designed into the software. Thus, by employing an unambiguous language, and by translating and analyzing it with a program intolerant of ambiguity, we can ensure an unambiguous statement of requirements. However, the need for clarity of communication strongly suggests a language resembling common speech, so that the specification can be read by managers, systems engineers, and others who are not specially trained in the language.

To provide an internally consistent specification, analyses of the requirements statements are incorporated into the system supporting the language. These analyses include semantic and syntactic decomposition of the individual statements, and analysis of the composite flow of data and processing. Support of consistency with the physical universe is accomplished by converting the specification unambiguously into a model (simulation) which can be executed against a model of the real world.

Finally, to support control of both the specification process and software development, a means of selective documentation and analysis of the specification is provided. The integration of these tools with a sound and methodical engineering and management approach provides predictability in the specification process and aids in avoiding overspecification.

1.2 SCOPE AND CONTENT OF THIS MANUAL

This manual is essentially a SREM User's Guide for development of software requirements specifications. It is not a cookbook, in that it does not attempt the inherently impossible task of converting the genuinely creative aspects of specification development into rote, deductive operations. However, it does define guidelines through which these creative operations are recognized, applied and restricted to their natural roles in the specification development process. In this manner, the scale and range of creativity can be defined and contained, thus allowing the specification development process to be scheduled with some degree of confidence. It should be noted that creative features remain in the methodology and as a consequence the major development effort must be conducted by experienced, knowledgeable engineers. However, SREM has been structured in a manner such

that many elements of specification development can be identified and isolated to permit junior engineering personnel to perform the details of specification statement definition and documentation preparation.

This manual is organized into two parts: Part I deals with the technical aspects of software requirements engineering. The methodology is described in detail, step-by-step, in the context of an example beginning in Section 2. Part II discusses the management of the specification development process with emphasis on how the specific features of SREM and its tools can be used to advantage in the management of the activities.

This document is intended as a User's Guide for the requirements engineer. It describes the steps of the Software Requirements Engineering Methodology and defines the techniques, procedures and tools to be used during application of the methodology steps to the development of a Process Performance Requirement Specification. The language (RSL) and tools which form the Requirements Engineering and Validation System (REVS) are described in Reference [1] and should be familiar to the reader prior to attempting to apply the methodology.

1.3 OVERVIEW OF SREM

The desired properties of a requirements specification discussed above are rather general in nature. These can be precisely defined in terms of nine characteristics of a good specification:

- Communicability
- Testability
- Consistency
- Completeness
- Traceability
- Correctness
- Design Freedom
- Flexibility (Changeability)
- Feasibility

These characteristics, which are self-explanatory, formed the specific objectives which influenced every aspect of the development of SREM. They are repeated here to establish the context of our objectives. Justification of the methodology presented here against these goals is contained in [2] and will not be repeated here.

The reader who wishes to learn how to write software requirements using the SREM techniques should first study the language and support software capabilities described in the REVS Users Manual [1]. However, a general understanding can be obtained from this manual alone. To facilitate this, a brief overview of RSL and REVS is provided here.

1.3.1 The Requirements Statement Language (RSL)

RSL is an extensible language which means that certain primitive concepts are built in and the user can use these to define more complex language concepts. The primitives are elements, attributes, relationships, and structures. From these, we have defined a nucleus of concepts which to date have proven sufficient. Future users of the language can add to these by means of the extension features as required. These concepts are introduced as they are used in this manual, and are presented in full in Appendix A.

The Requirements Statement Language is a user-oriented mechanism for specifying requirements. It is oriented heavily toward colloquial English, and uses nouns for elements and attributes and transitive verbs for relationships; a complementary relationship uses the passive form of the verb. Both syntax and semantics echo English usage, so that many simple RSL sentences may be read as English with the same meaning. However, the precision of RSL, enforced through machine translation, is not typical of colloquial speech; as a result, most complex RSL sentences are a somewhat stylized form of English.

1.3.2 The Requirements Engineering and Validation System (REVS)

REVS is an integrated set of tools used to support the definition, analysis, simulation, and documentation of software requirements. A key concept of REVS is that all requirements are translated into a central data base called the Abstract System Semantic Model (ASSM). The RSL statements themselves are not stored in the ASSM. Instead, they are translated into representations of the information content of the requirements statements. This provides an efficient and flexible means of maintaining a large software specification in a relatively small computer data base.

The ASSM is a relational data base providing a common source for all requirements analysis and modelling and for documentation. The commonality of all data ensures that any combination of extractions from the ASSM at any time (e.g., a document and a simulation) will be mutually consistent. That consistency is essential to asserting that the requirements modelled in validation of the specification are equivalent in every sense to those written in the specification.

REVS provides two mechanisms for entry of data into the ASSM, translation and interactive graphics, and a powerful set of tools for analysis termed collectively Requirements Analysis and Data Extraction (RADX). Translation is the process of converting RSL statements into the ASSM information, where the source of the statements may be cards, card images on tape, or keyboard entry from a terminal. Interactive graphics (RNETGEN) is a software package executing in conjunction with the the Anagraph color graphics console to provide ASSM entry and illustrative documentation. It permits entry of structures and referenced elements in a manner parallel with the translator, and in fact may be used in conjunction with translation in an operational environment. Significantly, RNETGEN allows the user to attribute graphical information to his structure, both for multicolor display on the Anagraph and for documentation via CALCOMP.

Information held in the ASSM may be selected and output using RADX. That tool is responsive to user direction in selecting either a recreation of the information translated into the ASSM, or the formatted abstraction of that information in a user-defined HIERARCHY. The combination of these features allows complex selections to be effected, so that all information needed for documentation and much that is essential to configuration management can be abstracted from the system without the encumbrance of irrelevant data. Since all data abstractions are drawn from a common ASSM (and since that data base is confirmably consistent within itself), even redundant assertions in data extractions are absolutely consistent with one another.

Both static and dynamic analysis are provided by REVS in order to determine the internal consistency of the ASSM and its validity with respect to the laws of nature. Static analysis is performed in RADX which examines the data connectivity through the requirements to determine

that the laws of logic and the conventions of the language are fully satisfied throughout. Some forms of completeness testing are also accomplished, determining, for example, that constants are provided as required; the scope of completeness testing is largely at the discretion of the user, since he may define extensive static analyses through RADX commands to supplement those inherent in the system.

No amount of static testing can fully validate a set of requirements. To do so, the system they represent must be exercised against a model of the environment in which the system is to execute. Such simulations are provided by an automated simulation builder (SIMGEN), and a software package supporting its execution (SIMXQT). Two different levels of simulation are supported: analytic, in which high-fidelity models of the environment and explicit performance measures are provided, and functional, in which the connectivity of the system is validated with non-analytic models.

1.3.3 The Engineering Methodology

Historically, the methods for developing a software specification have been as numerous as the developers of such documents. In fact, few cases can be cited in which any formal methodology could be quoted. Until the specification appeared (often after tens or hundreds of man-years of effort), nothing was in hand to show that it would be generated. In addition, it has frequently been true that the quality of the specification even with respect to elementary consistency from one requirement to another, could be verified only very late in software development. Since the problems were discovered only when the cost of correction was prohibitive, the requirements were frequently changed, degrading system performance in order to have some workable product.

The methodology developed within SREP is not only formal, in that it provides an explicit sequence of steps leading to the specification, but also manageable, in that it illuminates multiple phases for management review and analysis. Along the way, it supports early detection of high-level anomalies, since it works from the highest levels of software definition (processing and data flows) to the most detailed (analytic models and data content) in a systematic manner. A key feature of SREM is that the processing functions and data communications are considered in parallel,

rather than have either follow the other. As a result, the connectivity of the system is always complete, and it becomes possible to partition the requirements effort among several groups early in the process without risking divergence, omissions, or inconsistencies.

1.3.4 Specification Management

The management of a specification developed under SREM benefits most from the common source in the ASSM for all representations of the requirements. Thus, the simulation of the specification and the documentation of its requirements must be consistent at any time, since both have a single source of data which generate each without human intervention. In addition to a common data base, the methodology itself supports an orderly development which can be annotated with milestones, recorded on PERT charts, and otherwise controlled with the tools of the last several decades to provide predictability and control. This is not to suggest that the creativity of the specification process can either be scheduled or bypassed; it is still needed, but the methodology isolates it into segments with high visibility, supporting management cognizance of its progress and impact.

1.4 APPLICABILITY

These tools and techniques have been developed to address the needs of BMD software development. With perhaps minor exceptions, however, SREM is directly applicable to the specification of the requirements for any central software process for a large real-time system. In fact, since the methodology is inherently and deliberately computer independent, the techniques are not limited strictly to software in the form of computer programs. The requirements for any process composed of logical decisions and computations performed on data can be expressed via SREM -- regardless of whether the end product will be software, hardware, firmware, or some combination of these.

1.5 TERMINOLOGY

At the risk of introducing confusion, we have introduced some non-standard terminology. This has been done for two purposes: (1) to emphasize the different interpretations given to some concepts, and (2) to emphasize

the generality of the methodology application. An example of the first is the use of the term ALPHA for a processing step. The more common term "function" would be misleading to some because there is, in fact, a wide variety of common interpretations of "function". To avoid misunderstanding, we use the new, unfamiliar term in order to emphasize its specific meaning. An example of the second is the name applied to the resulting requirements specification which we call the Process Performance Requirements (PPR). No documentation system currently in use recognizes a document called a PPR. Here, our point is that any software requirements specification, whether called a B-5 (in MIL-STD 490) or something else in some other system, must contain a certain set of information. That set of information is what we call a PPR.

If this use of new terminology causes confusion, we apologize. However, once the techniques are understood, they can be applied to any program and the terminology adapted to the needs of the user.

PART II - TECHNICAL APPROACH

2.0 SREM OVERVIEW

The Software Requirements Engineering Methodology has been developed during the past several years in conjunction with development of the Requirements Engineering and Validation System and as a consequence has resulted in a clean, clear and comprehensible compatibility between the methodology and the instruments it uses to formulate and test a requirements specification. While REVS embodies the language and tools required for orderly development of process requirements specifications, SREM defines the techniques and procedures within which the tools and sound engineering and management practices are combined to generate a specification containing the desired properties under a controlled environment.

SREM encompasses four major areas of engineering activity that begin with receipt of the set of information which defines the system level requirements on the Data Processing Subsystem. We call this set the Data Processing System Performance Requirements Specification (DPSPR). The DPSPR as used here includes the Data Processing System Interface Requirements Specifications and any external subsystem Performance Requirements Specifications which influence the definition of the Process Performance Requirements. Using these source documents as a stimulus, the requirements engineer becomes involved in the four major engineering activities defined by SREM to develop the Process Performance Requirements Specification. These engineering activities are:

- Identification, definition and development of the functional requirements,
- Identification, definition and development of the performance requirements,
- Development of the Process Performance Requirements Specification and
- Development of the process design feasibility demonstrations which are generally conducted sequentially and separately.

The inherently sequential nature of the steps of the methodology appeared at first to make incremental specification of software awkward. Experience on many programs, notably Systems Technology Program, has made it clear that the new technology should assume that knowledge of requirements will increase continuously throughout the development of the software specification, rather than be complete when software requirements are first initiated. Thus, the tools and methodology of SREP were developed to allow for incremental development of the specification. Specific features, such as VERSION and the qualified inclusion of R_METs in a simulation provide the capability either for defining segments of the software requirements at a time, or for augmenting a full subsystem with additional functions. The consistency and integrity enforced by the system are fundamental to success in incremental specification. They ensure that:

- Portions of the system specified later than some segments will be consistent since their connectivity with the early segments was defined at the highest levels; and
- Any extension of the system will be compatible with prior specification, since any inconsistency would preclude entering the extension into the ASSM.

During each activity of SREM the features of REVS are utilized to control, monitor, test and maintain the evolving collection of requirements statements. The functional requirements are defined in RSL statements and catalogued by REVS in the ASSM through the TRANSLATOR segment. The accuracy and correctness of these RSL statements is verified by the Static Analyzer portion of the RADX segment of REVS. Continuity and completeness of these RSL statements is analyzed through the SIMGEN and simulation execution segments of REVS using algorithms for each functional requirement represented as executable PASCAL procedures implemented as BETA models. Next, the performance requirements are defined in RSL statements and catalogued by REVS in the ASSM through the TRANSLATOR and attached to the functional requirements each CONSTRAINS. The accuracy and correctness of these RSL statements is again verified by the Static Analyzer portion of the RADX segment of REVS. Continuity and completeness of these RSL statements is analyzed through the SIMGEN and simulation execution segments of REVS using algorithms for each functional requirements, represented as executable PASCAL procedures implemented as GAMMA models. Validation of the func-

tional and performance requirements testability is confirmed by REVS through the Post-Processing Analyzer using executable PASCAL procedures implemented as EXTRACTOR and TEST models. In this way, the existence of a feasible design solution for the collection of functional and performance requirements statements is confirmed by REVS through use of candidate algorithms used as the GAMMA models, and a model of the system environment and threat (SETS). The models are executed against one another with a variety of scenarios to demonstrate the existence of a solution to the requirements statement in the ASSM. Finally, data collected through RADX are formatted and published as a Process Performance Requirements Specification.

The preceding information has been provided to introduce and orient the reader to the global view of SREM and the REVS instruments used in the methodology to create a PPR, and to validate it through automated simulation.

The detailed description of the SREM technique of specifying software functional and performance requirements is presented in Sections 3 and 4. The methodology is described in the context of an example which is worked out to the degree necessary to illustrate the method. The example is a hypothetical system called Track Loop System (TLS). TLS is representative of the kind and complexity of real BMD systems, and yet is simple enough to serve as a comprehensible example. A complete DPSPR (including the interface specifications) for TLS is provided as Appendix F. The system is summarized below.

2.1 THE TRACK LOOP SYSTEM EXAMPLE

The Track Loop System (TLS) is a subset of a Preliminary Ballistic Missile Defense System that is capable of nearly autonomous execution in response to external stimuli. It is the simplest known subsystem with properties of interest for software definition, and it is one which has been studied extensively, both in the academic literature and in such practical programs as Site Defense. Therefore, it has been selected as the testbed for supporting experimentation in development of the methodology for software requirements. A pictorial representation of the TLS is provided in Figure 2-1.

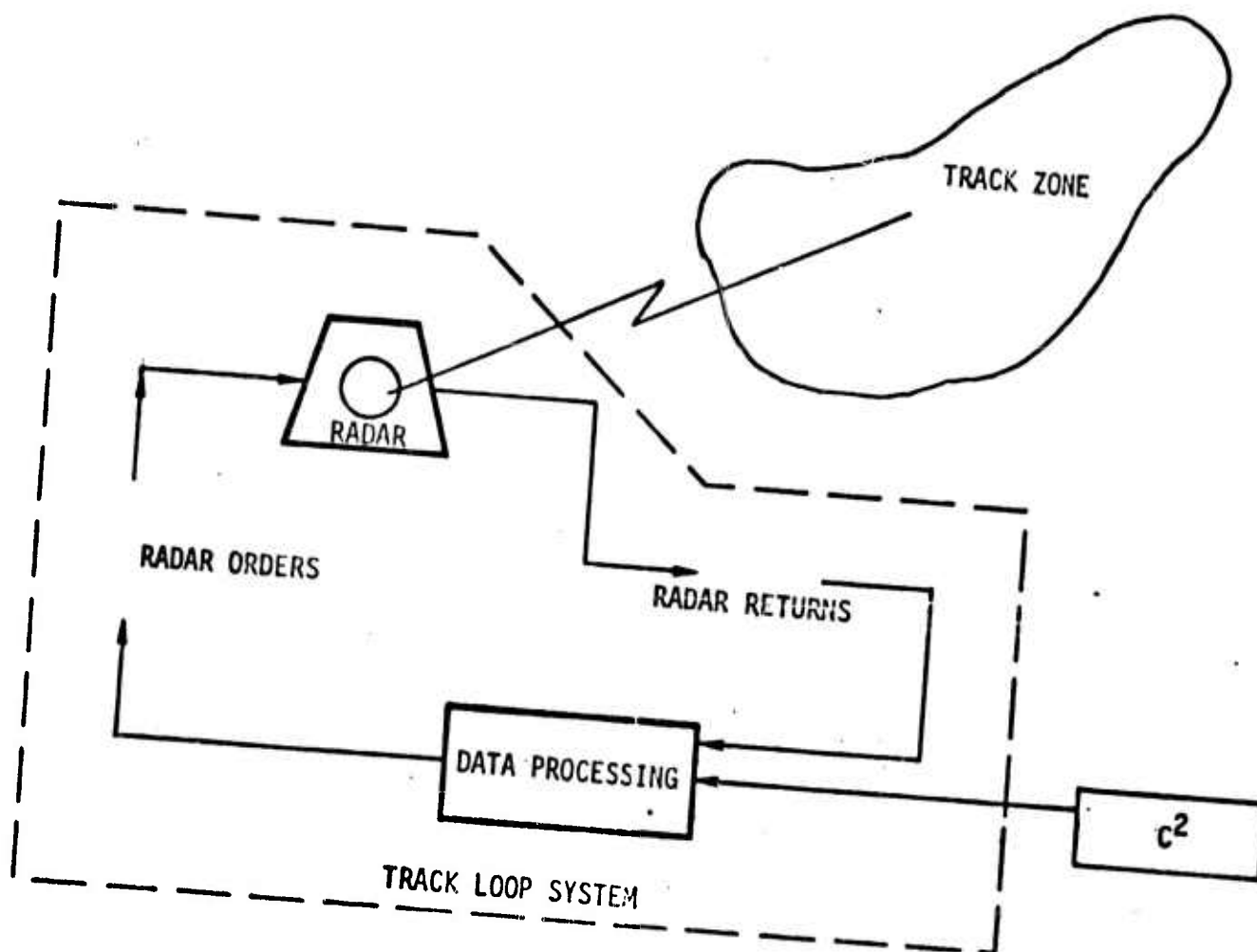


Figure 2-1 Track Loop System

2.1.1 Preliminary Ballistic Missile Defense System

A Preliminary Ballistic Missile Defense System (PBMDs) has been postulated as an environment in which the TLS would execute. It is a generalized representative of the class of systems currently in development, and is particularized for the TLS through representative but non-real specifications where required. In the Conduct Engagement mode, an object entering the search region will be detected and designated, tracked, discriminated, and engaged (as required) in defense of the ground facilities. Those functions are implemented through the Data Processing System (DPS), a radar or other sensor, and a means of neutralizing hostile objects. For the purpose of the TLS, only the radar need be defined in detail; other system elements are identified only to the extent that they impact DPS requirements.

2.1.2 TLS Requirements

The TLS is required to perform five system level functions: 1) system initialization and engagement initiation, 2) engagement termination, 3) target tracking, 4) control of system resources and 5) recording of data during the engagement. The system includes: the DPS, the Radar and the recording media and directly interfaces with the external environment through communications and control (C^2).

In general, the functions of TLS are initiated by messages from C^2 ; however, track maintenance and certain control functions are autonomous. The engagement is initiated and terminated by C^2 messages; during engagement, radar data are reported periodically autonomously. When an image is handed over to TLS through C^2 , it is tracked without further direction, until it is dropped either by command or by determination within the DPS. This configuration thus demonstrates both exogenous and endogenous process excitation, and in other ways provides a microcosm of a BMD process.

2.2 SUMMARY OF APPENDICES

The Appendices provide a summary of the Requirements Statement Language and a complete development of the TLS requirements statements. A complete description of RSL is provided in the REVS Users Manual (Reference [1]).

Appendix A summarizes the RSL Terminology by providing a copy of the RSL nucleus which defines each element of the language and an illustration of the symbology. Appendix B contains the set of hand-drawn representations of TLS requirements which correspond to the results obtained from application of the methodology defined in Section 3.1. Appendix C represents the TLS kernel which contains the flow and data hierarchies developed as a result of the methodology defined in Section 3.2. Appendix D presents the set of R_NETs and the SUBNET produced by the Calcomp capability of the interactive graphics segment of REVS. Appendix E represents the complete TLS Data Base maintained in the ASSM as extracted by the RADX segment of REVS. Appendix F contains the TLS source specifications from which the TLS requirements were developed.

Appendices C through E have been produced by REVS from the ASSM in much the same manner that the information content of a software specification would be developed. Editing of this information into a specification document would be adapted to the particular needs of a specific program. A sample PPR specification was produced in Reference [2] and the review it elicited has underscored the need for adaptation of the extracted information to the specifics of an application. Therefore, neither REVS nor SREM is designed to produce a specific specification format. This simple final step is left to the discretion of the user.

3.0 FUNCTIONAL REQUIREMENTS

It is possible and practical to view a software requirement as defining either what must be accomplished or how well it must be done. The former is termed a "functional requirement", since it specifies data processing functions; the latter is termed a "performance requirement" since it constrains the quality of performance of the function in the system. In another sense, it is useful to look upon the functional requirement as defining the required output in terms of the available inputs. In a simple case, a program might be named SUMMER and have a functional requirement of generating the sum of a sequence of input numbers (X_i). Defining the output after i inputs to be Y_i , the performance requirement might be that ($Y_{i+1} - Y_i$) be within ϵ of X_{i+1} .

Note that while the functional requirement specifies what is to be done, and the performance requirement constrains how well it must be accomplished, the means of accomplishment is left to process design; since the means of implementation is not specified, the requirements are said to be design-free.

The form of representation of functional requirements has evolved in recent years, and has culminated in Requirements Networks (R-Nets). Originally, verbal descriptions of functions were attempted, but the verbiage was found to be cumbersome and ambiguous. Later, through Engagement Logic and Functional Flow Block Diagrams (FFBD's), diagrams replaced many words (the pictures being worth thousands of words apiece). Unfortunately, much of the ambiguity was retained. Notably, it was difficult in practice to trace required processing paths; data definitions were incomplete; and the mechanism did not lend itself to consistency or completeness analysis.

To avoid the problem of recognizing processing paths, a thread description was attempted; unfortunately, the number of threads in a real system proved so large that the (essentially one-dimensional) representation was almost as hard to use as English text. Conversion to thread trees somewhat reduced the magnitude of the thread problem, but left the other difficulties of undesired specificity (in AND branches), ambiguity (especially in data), and awkwardness for analysis.

The properties preserved in defining R-Nets were:

- graphic representation of functional requirements;
- path orientation for specification of threads;
- design (implementation) independence.

In addition, the use of R-Nets permitted the addition of the following properties:

- unambiguous statement;
- analyzable models;
- explicit data specification.

In effect, those six properties became the top-level specification of the tools and methodology of the SREM functional specification.

It is significant that the properties carried over from previous means of statement are those relating to subjective measures of legibility, utility and design freedom. The added properties are objectively assessable - most readily by demonstration. Thus, a part of the program has been the demonstration of completeness, freedom from ambiguity, and other attributes through static analyzers of the explicit (machine-readable) Requirements Statement Language (RSL). By expressing the functional requirements in machine-readable form, and by using the tools developed on a variety of programs in both industry and academia, it is possible to generate an ultimate test of a functional specification - a functional simulation.

A simulator built without human intervention from a specification is a total demonstration of the consistency, precision, and completeness (in at least a limited sense) of that specification. With a suitable driver, such a simulation provides a useful tool for defining frequency of transaction and examining the gross aspects of system tradeoffs.

Fundamentally, there are three different ways of conceiving of software requirements. The classical approach is functional: what operations are to be performed by the system logic, as embodied in the software. A thread approach is more nearly mechanical: what are the interfaces and the properties of the messages required to be communicated through them. The third concept may be termed philosophical: what are the realities of the world the DPS perceives, and what information about those realities must be manipulated. Clearly, each approach can lead to mechanisms by which

requirements may be generated; SREM uses all three concurrently.

The functional approach is embodied in the concept of a Requirements Network (N-Net), which defines the processing flow required of the software. The mechanical concepts are reflected in the heavy dependence of SREM on definitions of messages through interfaces in establishing the top level of data hierarchies, and philosophy is preserved in the implementation-independent hierarchies defined under entities. The three viewpoints are merged in the simulation-level definitions of requirements as data and executable descriptions; the interrelationships of the three points of view are realized in the simulation itself. Sections 3.1 and 3.2 provide the methodology for generating the highest level of requirement from each perspective, 3.3 carries the definition to the next level and begins to interrelate them through RSL statements, and 3.4 completes the methodology for their realization in the executable description. Sections 3.5 and 3.6 suggest the means for adding descriptive and supportive information to support specification management and documentation. Section 3.7 extends the methodology into analytic modelling.

3.1 PHASE 1 - DEFINITION OF SUBSYSTEM ELEMENTS

There are two different "structural" elements to be defined in the first stage of functional specification. One is the flow connectivity previously represented with Engagement Logic or FFBD's. The other is new with the current methodology, and defines the data hierarchies required. Previously, there was no specific methodology even for the definition of flow connectivity; the approach used was often to lock an appropriate number (typically 3) of the "right people" in a room for a few weeks, and watch the product appear. By adding the data hierarchy to the structures, we have been able to identify a step-by-step mechanism for the top-level development, in which only the areas requiring creativity are left uncontrolled, and those areas are clearly identified.

The first-time SREM user may find that he has to reorient his thought processes in order to effectively use the methodology and the REVS software. In time, he will find that the steps herein form a natural progression for the job to be done. He should always keep in mind that his job is to develop the requirements for software, and not to design the software itself. He should constantly ask himself, "How can I precisely state what the DPS is

required to do in the most general way, so that the designer has the maximum range of choice in deriving the solution?"

A typical specification process usually starts with a notion of the processing steps involved and later considers the data needed to support that processing. SREM partially reverses this order of consideration. The user must first ask two questions: 1) "What data are presented to the DPS for processing?", and 2) "What data are expected from the DPS as output?" From the answers to these questions, the user derives his concepts of the processing steps in between.

It is suggested that the user accomplish the work at each step across the breadth of the project before proceeding to the next step. In large projects, involving many people, the needs of communication and coordination make this mandatory. In smaller projects, especially those involving one or two people, there is often a rush to do all the steps for one small area of the system before considering the next area of concern. This may be possible for a DPS problem where the processing functions are independent, but, at best, many valuable insights into the required operation of the DPS as a whole may be lost. At worst, a significant amount of rework may be involved.

3.1.1 Initial Inputs

The initial inputs required for application of SREM are, typically, a system specification and its companion interface specifications. The objective is to generate a complete specification for the data processing subsystem (DPS) from these basic inputs. Usually, at this early stage of system development the input specifications are incomplete, contain many ambiguities, and leave several issues for future resolution. The SREM user need not wait for all gaps to be filled. Instead, the user should proceed from that which is clearly defined, and use the facilities of the RSL management segment to spotlight issues needing resolution. Assumptions and decisions may be necessary, often based on inadequate data. The SREM user should not avoid these, but should note them in the ASSM using the RSL element DECISION and its attributes. The important thing is to make these entries as they arise. In this way the SREM user not only leaves a traceable record of current

status for others, but leaves a valuable record of the evolution of his thoughts about the subsystem for his own future reference.

3.1.2 Interface Definition

The first RSL entries in the ASSM are concerned with identification of the DPS interfaces which CONNECT with other subsystems. Usually, these are the elements which are most clearly defined in the originating specifications. Also, it has been found that the interfaces, and the messages passing through them, are the key focal points for progressive development of the requirements structure.

Consider the TLS specification (DPSPR) and interface specifications (IFSs) contained in Appendix F. These documents refer to two subsystems which interface with the DPS, namely the Radar and the C² (Command and Communications). We will refer to C² as a subsystem for convenience, even though it is a separate system, external to the TLS. A closer examination of the DPSPR reveals that the DPS is to output data to permanent files. Although not explicitly required by the specification it will later become apparent that it is conceptually useful to define permanent storage as a third subsystem separate from the DPS, even though it is embedded in the DPS. For REVS use we will name the three subsystems SSRADAR, SSC2, and SSPERMFL, respectively.

In the specifications, three separate interfaces between the DPS and the radar are mentioned. Through one input interface the radar sends returns to the DPS; through another it sends clock inputs to the DPS. The DPS issues commands to the radar through an output interface. We will name these interfaces RADAR_IN, RADAR_CLOCK_IN, and RADAR_OUT, respectively. The names are arbitrary, but should be meaningfully related to the specification terminology. Note that an interface is designated as "input" or "output" from the viewpoint of the DPS.

Similarly, the specifications call out one input interface between C² and DPS. We will call this input interface CC_IN. Data recorded by the DPS in permanent storage apparently are never accessed from that source during DPS operation. Therefore, we will link the DPS to our conceptual subsystem SSPERMFL via an output interface, DATA_RECORD.

At this point, the subsystem and interface definitions, and their relationships, can be coded in RSL for entry into the ASSM. Figure 3-1 shows one way of expressing these data in RSL. Note that the management segment attribute, DESCRIPTION, has been used to explain the nature of SSPERMFL. This entry is for example purposes here, and will not be perpetuated in the TLS example. However, notations such as this have value in a real system development project and should be encouraged.

Note that the DPS itself is not entered into the ASSM as a SUBSYSTEM, since it is inherently the object of all software requirements.

3.1.3 Message Definition

Having defined the subsystems, the interfaces, and their connections, the next logical step is to examine the discrete blocks of data, or MESSAGES which are PASSED BY the interfaces. A MESSAGE is MADE BY its contents, whether they be DATA or FILES.

One must be careful to separate the concepts of "message name" or "message category" from that of "message type". In RSL, the identifier associated with MESSAGE refers to the message name or category. MESSAGES are distinguished by differences in their data contents. Thus, two blocks of data with different data contents, which pass through an interface, must be defined as different MESSAGES, hence must have different message names. The message name is not contained in the data, it is an external label.

On the other hand, two packets of data may have identical data content with different values, and may require different processing to be done on the data, within the DPS. In this case one would have two instances of the same MESSAGE, but with different "message type". One would further expect that one of the data elements in the message would be a message type identifier, with a unique value for each type. Otherwise, the DPS could not distinguish between types of messages and perform different processing operations on each type. Messages with different names must also have a unique identifier in order for the DPS to distinguish between messages. It is most economical to require that all messages, whether of different name or type, contain a type identifier as a data element, with a unique value,

SUBSYSTEM: SSPADAR.

CONNECTED TO:

INPUT_INTERFACE: RADAR_IN

INPUT_INTERFACE: RADAR_CLOCK_IN

OUTPUT_INTERFACE: RADAR_OUT.

SUBSYSTEM: SSC2.

CONNECTED TO:

INPUT_INTERFACE: CC_IN.

SUBSYSTEM: SSPERMFL.

CONNECTED TO: OUTPUT_INTERFACE: DATA_RECORD.

DESCRIPTION: "SSPERMFL, ALTHOUGH CONTAINED IN THE DPS,
IS DEFINED HERE AS A SUBSYSTEM FOR CONVENIENCE
IN EXPLANATION AND SIMULATION."

Figure 3-1 RSL Subsystem Entries

different from the message name. This avoids both confusion in thinking, and possible misinterpretation by the RSL translator.

To illustrate these concepts, let us consider a DPS with one input interface called IN. Across this interface pass four distinct types of messages. The first two types consist of a single data element with different values: START, STOP. These command the DPS to start and stop processing, respectively. The second two types consist of a data element with two values: FEET, INCHES. This element defines which units the real number is in. Further, the DPS is to convert the input data to meters for further processing.

One can distinguish between these messages by defining a data element MESSAGE_TYPE with four enumerated values: START, STOP, FEET, INCHES. For the types FEET and INCHES, we can define the other associated data element as LENGTH. Over the four message types we can distinguish two categories, according to data content and function. The first category consists of messages with only one data element, MESSAGE_TYPE, which provides commands to the DPS. The second category consists of messages with two data elements, MESSAGE_TYPE and LENGTH, which provide dimensional input data to the DPS. The first category we can name as a COMMAND message. The second category we can name as a DIMENSION message. Hence, a COMMAND message has two types: START and STOP. A DIMENSION message also has two types: FEET and INCHES. We would not want to name this latter message LENGTH, because that name is assigned to one of the data elements within the message.

The message names, types, and data contents associated with the various interfaces must be extracted from the system and interface specifications, often from widely scattered locations. Sometimes the specifications only define message types, and the requirements engineer must group these into categories which will be named MESSAGES in RSL. For these reasons, some pencil-and-paper preliminary work is usually needed to organize the data before translating them into RSL inputs. One diagram representation of the messages passing an interface, that has proved useful, is shown in Figure 3-2 for the simple example discussed above.

The diagram is merely a tree originating at the interface. Each branch of the tree terminates in a box corresponding to a MESSAGE name. Data elements common to all messages are placed on the tree before the

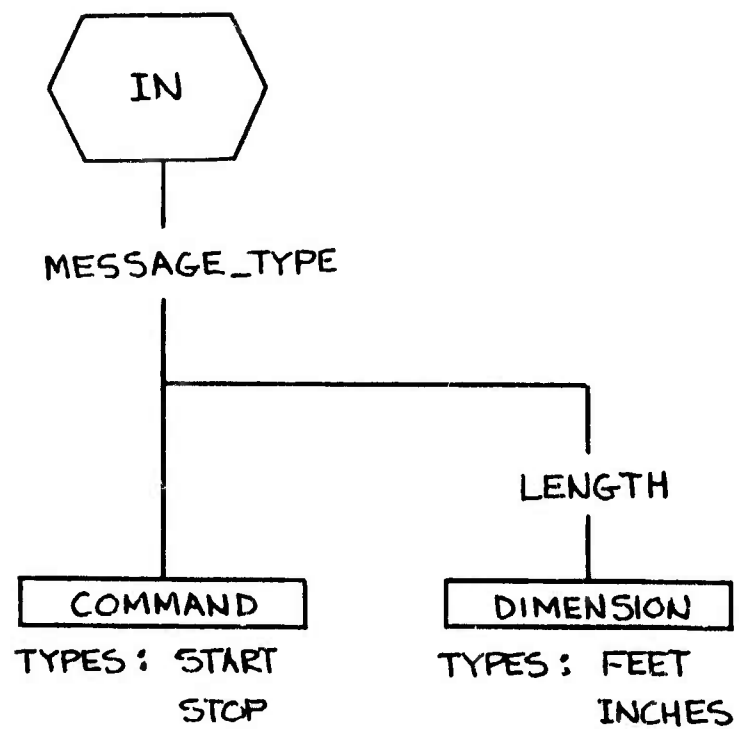


Figure 3-2 An Elementary Interface Hierarchy

first branch point. Data elements unique to a specific message are placed on the branch unique to that message. Elements common to two or more messages, but excluded from others, are placed on an intermediate branch leading only to the appropriate messages. (The TLS example discussed below will illustrate this latter condition.) The data element names may refer to data items, or files. It is useful to note files as such on the diagram. A file is a collection of instances of data items each instance having the same structure. If desired, the SREM user can note the types of each message below the message name. While the diagram format shown has proven useful, it is in no way mandatory. The user is encouraged to adopt whatever notation or format aids him. The important point is: organize the material on paper before writing the RSL inputs.

A useful rule-of-thumb for using SREM is: don't define details until they are needed for the purposes of the moment. At this stage of our analysis, we are interested solely in the elements which are common to, and unique to the various message types. No further detail is needed. For instance, if it is known that an identifiable group of data is unique to a given message, it is only necessary to name the group as a data item on the tree. In practice this is often easy, because the exact composition of the group may be ambiguous long after the group itself is identified. In any case, the composition of a group is easily defined by the RSL relation INCLUDES when that level of detail is needed.

Now, let us consider the messages in the TLS example, starting with those coming from the C^2 "subsystem". Paragraph 3.2 of the TLS C^2 /DPS IFS states that four types of messages are transmitted from the C^2 to the DPS:

- Initiate Engagement Mode
- Terminate Engagement Mode
- Handover Image
- Drop Image Track

Implicitly common to all these message types is some sort of message type identifier. Since these are all common messages we will call this identifier COMMAND_ID. Paragraphs 3.2.1 and 3.2.2 of the IFS imply that there are no other data elements in the first two message types. Both of these types have a common function, namely to command a change in the

operating mode of the DPS. Thus, they form a single MESSAGE category which we will name MODE_CHANGE. Hence, as shown in the diagram of Figure 3-3, we have a message MODE_CHANGE, of two types, with a single data element, COMMAND_ID, which distinguishes the two types.

The remaining two message types contain other data elements, in addition to COMMAND_ID, as stated in paragraphs 3.2.3 and 3.2.4. Common to both is an element called variously "image designation", or "image designator", but which is obviously a single element which we will call HO_ID (shortened from HANDOVER_ID). This additional element completes the definition of the "Drop Image Track" type message. Since the data content of this message is unique, it forms a message category by itself. We will name this message TERMINATION because we wish to reserve DROP_TRACK for use as an enumerated value of COMMAND_ID.

The remaining message type is also in a category by itself, which we will name a HANDOVER message. In addition to COMMAND_ID and HO_ID, paragraph 3.2.3 of the IFS states that this message contains a data element "image estimated state". However, paragraph 1.1.2.1a of the DPSPR refers to an "estimate of state", while paragraph 1.2.2.1g states that "each handoff shall consist of a unique designator, the state vector, and its covariance matrix." At this point we could be content with defining the data element as INITIAL_STATE_ESTIMATE. But, it seems worthwhile to state the main components, since they are not stated in the IFS paragraph where one would expect to find them. Thus, we define two data elements, INITIAL_STATE and INITIAL_COVARIANCE, to represent the vector and matrix, respectively. The prefix "initial" is used to avoid confusion with state data generated by the DPS in the course of subsequent processing. Note that there is no need, at this point, to define the vector components and matrix elements individually. We have now completed, apparently, the definition of messages related to the CC_IN interface, as shown in Figure 3-3. These messages can be defined in RSL as shown in Figure 3-4. Although not shown here, it is useful to use the capabilities of the management segment of RSL to note the source of the state data definitions in the handover message, and to point out that the IFS is incomplete or at variance with the DPSPR.

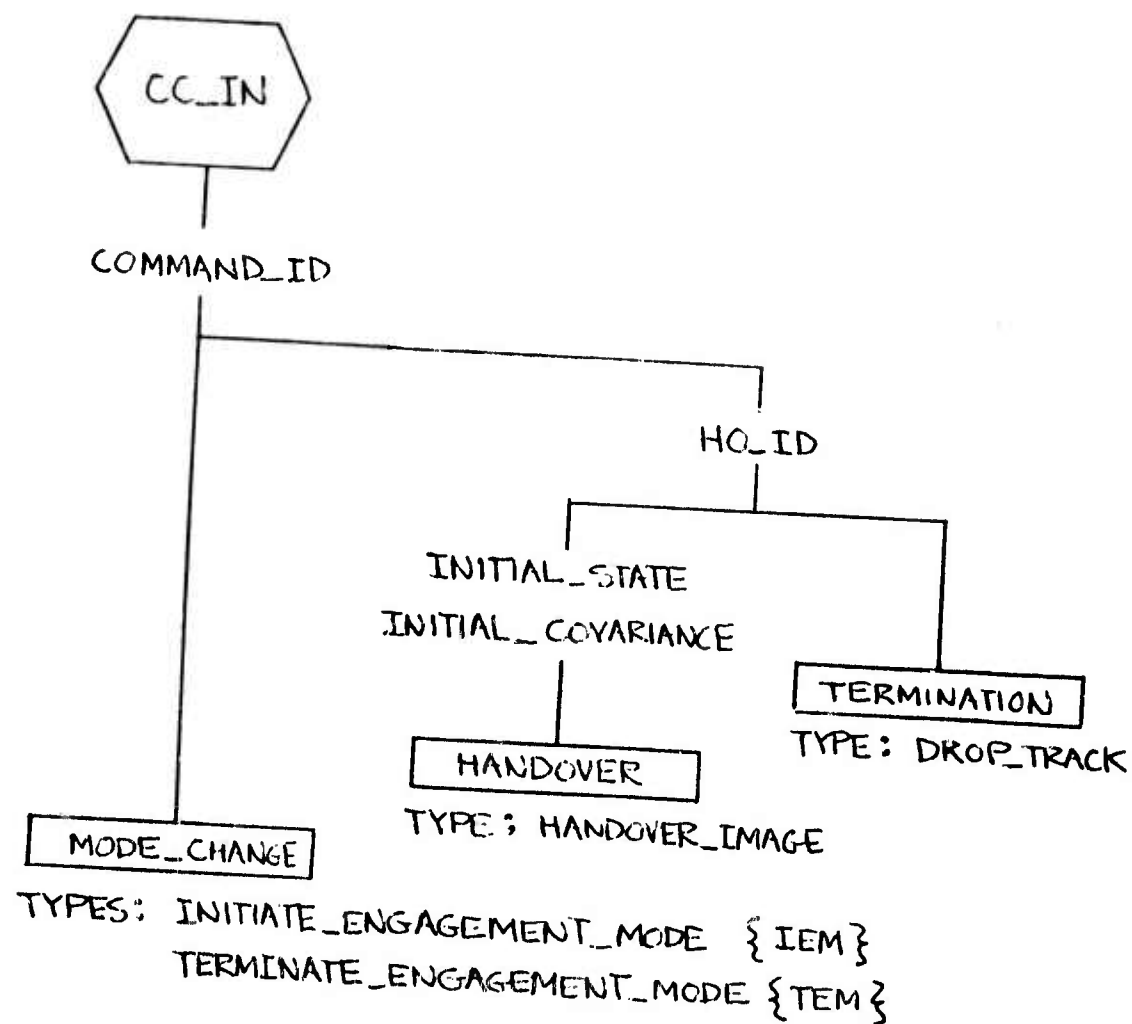


Figure 3-3 C² Input Hierarchy

INPUT_INTERFACE: CC_IN.

PASSES:

MESSAGE: MODE_CHANGE

MESSAGE: HANDOVER

MESSAGE: TERMINATION.

MESSAGE: MODE_CHANGE.

MADE BY:

DATA: COMMAND_ID.

MESSAGE: HANDOVER.

MADE BY:

DATA: COMMAND_ID

DATA: HQ_ID

DATA: INITIAL_STATE

DATA: INITIAL_COVARIANCE.

MESSAGE: TERMINATION.

MADE BY:

DATA: COMMAND_ID

DATA: HQ_ID.

Figure 3-4 RSL Message Entries

3.1.4 The Interface Data Hierarchy

At this point we have partially defined the elements of one of the two major data hierarchy types associated with SREM. This is the "interface data hierarchy" depicted in Figure 3-5.

A SUBSYSTEM is CONNECTED TO either an INPUT_INTERFACE or an OUTPUT_INTERFACE which PASSES blocks of data called MESSAGEs. A MESSAGE is MADE BY individual items of DATA, and/or by a group of DATA which INCLUDES individual DATA items, and/or by FILES. In turn, a FILE CONTAINS individual DATA items. The RSL concept of FILE is more general than the usual software connotation. It is simply a collection of instances of data, each instance having the same content of data items, without regard to the details of storage, and without ordering unless specified.

In general, a data item must have a different DATA name in each hierarchy in which it appears, even though the different names refer to the same information. The exception is that DATA or FILEs may exist in more than one MESSAGE. This is due to the fact that only one MESSAGE can be active in the system at any time. Therefore, the assembly of DATA and FILEs into a MESSAGE is unambiguous regardless of the number of MESSAGEs that may be MADE BY that element. For example, a MESSAGE PASSED THROUGH an INPUT_INTERFACE only exists at the instant of passage (i.e., the enablement of the interface network). An ALPHA accesses not the MESSAGE but the DATA it contains; there can be no ambiguity among those DATA items regardless of the number of MESSAGEs which might contain them, since there can be no more than one MESSAGE entering the system for a given enablement.

SREM is heavily oriented toward an orderly analysis of the interface data hierarchies, in a "top down" direction, as the first step in defining DPS requirements. This is a natural direction, as the interfaces and the messages crossing them are usually the most clearly defined elements of the originating specifications. As the user develops the data definition in progressively greater detail, the definition of specific processing steps, or ALPHAs, begins to emerge, as well as the processing flow STRUCTURE which links the ALPHAs.

For INPUT_INTERFACES, the "top down" consideration of the data hierarchy follows the flow of processing. For OUTPUT_INTERFACES, the "top down"

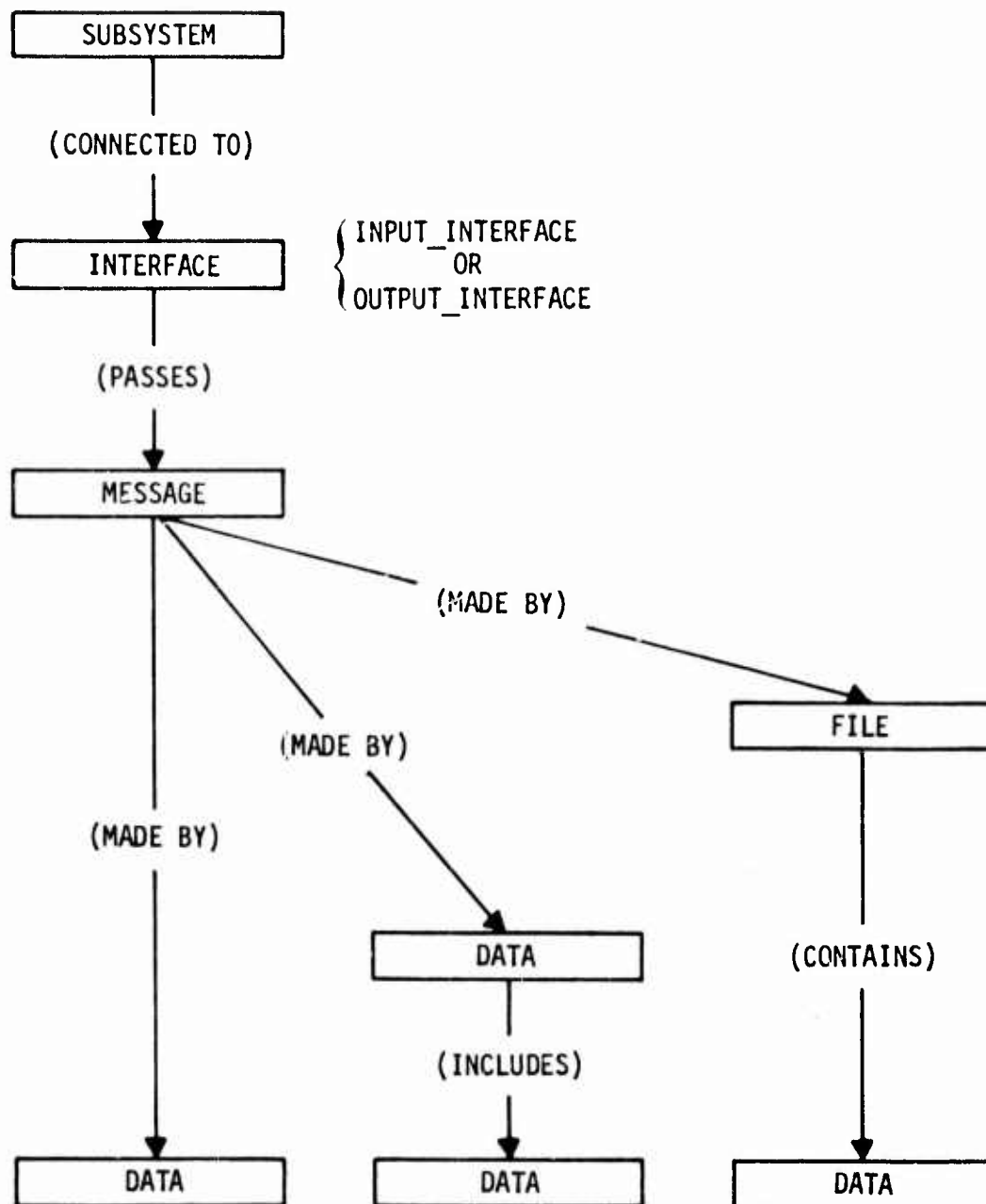


Figure 3-5 Interface Data Hierarchy

direction is opposite to the processing flow. However, backward tracing from the output interface is a valuable tool in constructing the steps necessary to form an output MESSAGE.

Initially, the internal processing required of the DPS may be ill-defined and ambiguous. The requirements engineer will have to draw heavily on experience to synthesize the connections between input and output. As an aid in this creative process, he should continually ask, "What must be done to the data I have in order to provide the output data I need?"

3.1.5 Problems of Definition

"Who wrote this mess?"

That is the question one is tempted to ask when he encounters subparagraphs 3.2.5 and 3.2.6 of the C^2 /DPS IFS. Two innocent subparagraph titles lead to a number of confusing questions, which are typical of preliminary specifications.

The titles of subparagraphs 3.2.1 through 3.2.4 correspond to clearly defined C^2 to DPS message types, and the content of the text addresses those types. The contents of 3.2.5, titled "Message Acknowledgement", and 3.2.6, titled "Error Handling", are TBS (To Be Specified). Do these titles indicate additional input message types, in conflict with the clear definition in paragraph 3.2? If so, what message is being acknowledged by "Message Acknowledgement"? No messages from DPS to C^2 have been defined, and no requirement for a DPS output interface to the C^2 system is indicated. In fact, Figure F-1 of the DPSPR (Appendix F) clearly indicates that message traffic is one-way, from C^2 to DPS.

On the other hand, a requirement for the DPS to acknowledge receipt of any of the four defined C^2 to DPS message types by transmitting a reply to C^2 may be intended. If so, both the DPSPR and IFS must be modified to reflect this, without ambiguity. Similarly, "Error Handling" might refer to either a message type, or to processing in response to erroneous messages. Resolution of these questions is important because major differences in DPS definition result from the alternatives.

The SREM user can either stop work until the problems are resolved, or can proceed tentatively with the assumptions which make the most sense.

If he chooses to proceed, he should take time to note the problem in the ASSM and identify what elements are affected by his assumptions. Figure 3-6 shows one way of doing this, for the message acknowledgement problem, using the RSL management segment. These entries announce the problem and indicate changes needed later if the assumptions are wrong.

The user may object on the grounds that making these entries and recording these questions is tedious and takes up his time. True. However, more of his time would be taken up by other people asking the same questions over and over again. Worse yet, others may make different assumptions or fail to detect the ambiguities. The result is more time spent later on rework. With the information recorded in the ASSM, the answers are available to everyone - even to those who haven't yet thought of the questions.

It is not within the scope of this manual to explore all the aspects of traceability and uses of the RSL management segment. With this brief illustration we will drop the subject. For further development of the TLS example, we will assume that the problems are resolved as follows:

- A message called ACKNOWLEDGEMENT consisting of one data element, COMMAND_ID, is required.
- This message is passed from DPS to C² via a new output interface, CC_OUT.
- "Error Handling" refers to error processing required of the DPS when a message from C² is not identified as one of the four defined types.

3.1.6 R_NET Definition

A Requirements Net, or R_NET is used to describe the required flow of processing in response to a single stimulus which ENABLES the net. This stimulus may be either the passage of a MESSAGE through an INPUT_INTERFACE, or an EVENT defined by arrival at a node on the subject R_NET or on some other R_NET associated with the DPS.

Each INPUT_INTERFACE must enable an R_NET. Otherwise, DATA in a MESSAGE passing the interface could not be processed by the DPS. Hence, there must be at least one R_NET for each INPUT_INTERFACE, since only the processing on an R_NET can distinguish between the arriving messages. Thus, the first logical step in defining the R_NETS of the DPS is to define one for

DECISION: MEANING_OF_MESSAGE_ACKNOWLEDGEMENT.

PROBLEM: "IMPLICATION OF CC TO DPS IFS PARAGRAPH 3-2-5
IS NOT CLEAR. REVISION OF THIS PARAGRAPH AND/OR
DPSR FIGURE 1-1 IS NEEDED."

ALTERNATIVES:

- "1. INTERPRET MESSAGE ACKNOWLEDGEMENT AS A MESSAGE TYPE
FROM CC TO DPS.
2. INTERPRET AS A MESSAGE FROM DPS TO CC IN RESPONSE
TO EACH CC TO DPS MESSAGE. THIS NEEDS A DPS
OUTPUT INTERFACE TO CC."

CHOICE:

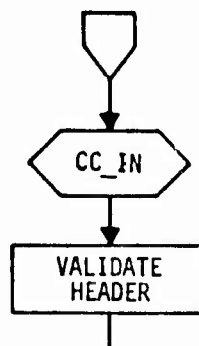
"PROCEED WITH ALTERNATIVE 2 PENDING FORMAL RESOLUTION."

Figure 3-6 RSL Decision Entry

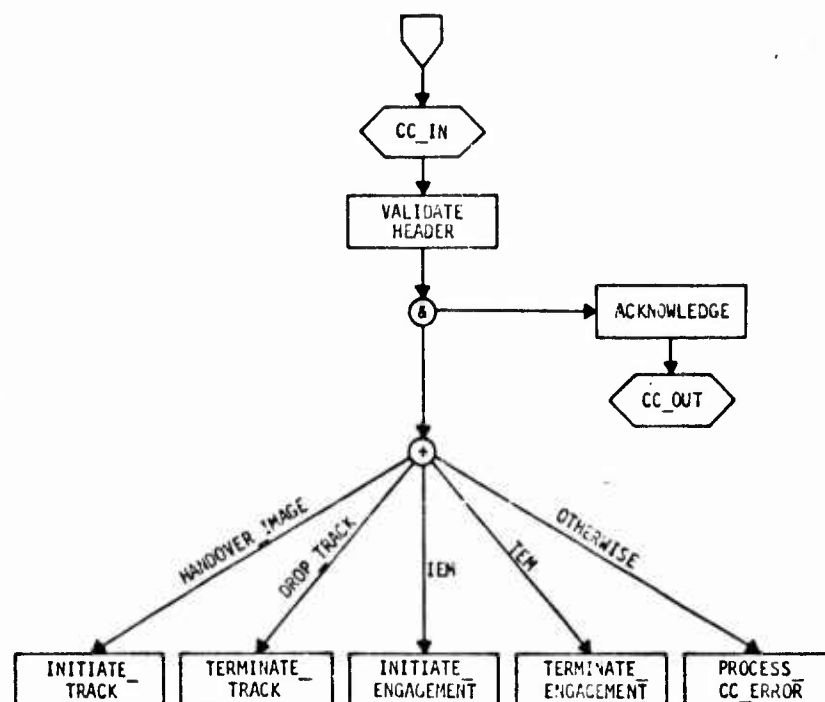
each INPUT_INTERFACE. In the TLS example, three such interfaces have been previously defined, so we need to develop three corresponding R_NETs.

R_NETS may be entered into the ASSM manually from the Anagraph terminal, or via a card deck of RSL statements. No matter which method is used, the net should be diagrammed on paper first to develop the concept fully and minimize revisions.

As a first example, let us consider the R_NET enabled by interface CC_IN. Following the initial node on the net, one draws the INPUT_INTERFACE itself. It is reasonable then, but not mandatory, to provide an ALPHA for common processing of all MESSAGES through that interface, for such purposes as validating data common to all MESSAGES. Thus, we have this typical starting structure.



It was noted in defining the MESSAGES that they are distinguished by the differences in their data contents. Since the input to an ALPHA is fixed, there must be a unique ALPHA for each MESSAGE. There also may be several message types for each MESSAGE, and it is reasonable to expect different processing, hence different ALPHAs, for each type. While not always true, this is usually a profitable assumption at this stage of R_NET development. Further, an additional ALPHA is usually needed for error processing of messages not recognized as one of the valid types. Therefore, it is possible to draw the following skeleton associated with an input interface with the information gained from our preceding analyses of the specifications. (Refer to Appendix A for symbols and allowable structures.)



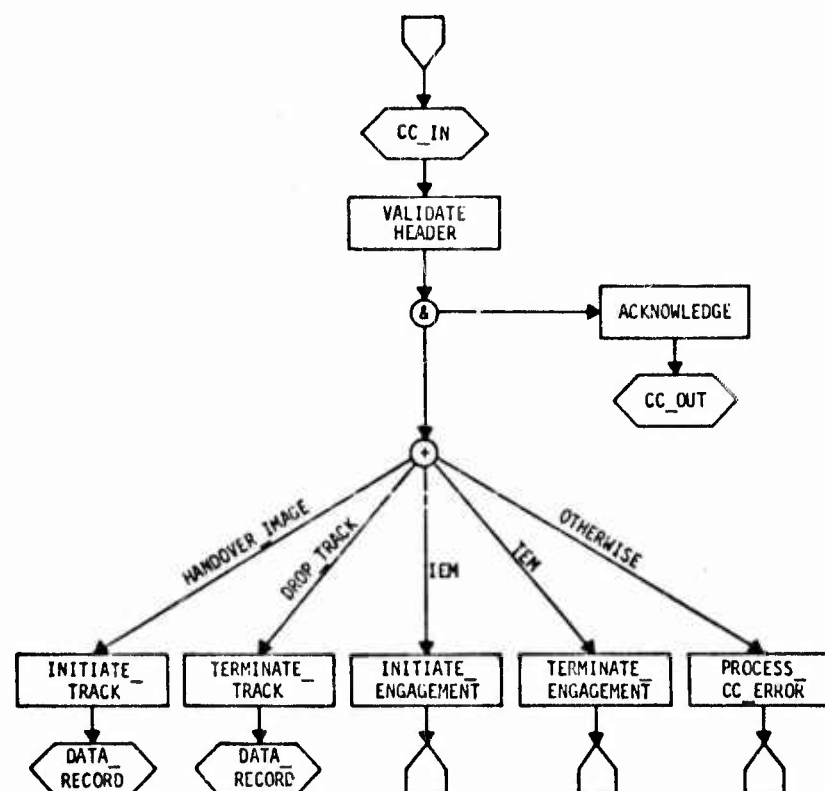
The OR node with multiple branches, as shown, is described in RSL with the aid of the CONSIDER phrase. The object of consideration is a data element with an enumerated set of values, in this case the data element COMMAND_ID. If the value of COMMAND_ID in a particular message does not match the value of any branch, the OTHERWISE branch is invoked.

The ALPHAs defined above reflect all processing required for a given message type. The names chosen reflect a gross conception of the nature of the processing. They are subject to change. As analysis proceeds the definition of the R_NET will be refined. The first tentative ALPHAs may expand, and the branching structure will be modified for all but the simplest systems. Hence, the SREM user should not rush to enter an R_NET into the ASSM at the first opportunity. He should wait until the definition of the net has stabilized and possible relations with other R_NETS are comprehended.

Where the previous effort was purely mechanical, it is now necessary to apply some creativity to complete the interface network. There are two fundamental approaches to that completion from the available documentation: thread tracing and sentential analysis. Both should be used so that completeness of statement is assured.

The first operation is thread tracing. By reading the source specifications, the processing steps required may be traced from an input port to their logical termination. When all paths have been traced, not only for the input networks but also for those developed in the following paragraphs, the set of processing steps (ALPHAs) required should be complete. Sentential analysis provides a cross-check by separating each specification sentence into its nouns (which correspond to system data) and its verbs (which correspond to ALPHAs). The two sets of ALPHAs should be identical; if not, they are made to be through refinement of the diagrams.

The result of specification analysis is completion of the paths from each INPUT_INTERFACE. For example, there is a requirement in the TLS that each MESSAGE received from the CC be acknowledged. Therefore the AND node is added, an ALPHA is provided to FORM the MESSAGE: ACKNOWLEDGEMENT, and the appropriate OUTPUT_INTERFACE: CC_OUT is indicated. Continuing the process, we arrive at the following diagram. It is a complete R_NET for RESPONSE_TO_CC requirements for TLS except that it does not yet reflect inter-network connectivity. Note that each branch ends at either an OUTPUT_INTERFACE, or at a TERMINATE symbol.



In tracing input networks, many of the MESSAGES to be output by the software will have been isolated. However, not all messages for any OUTPUT_INTERFACES, nor indeed any MESSAGE for some of them, may be defined. Thus, there is an inverse operation for output networks which trace back from an OUTPUT_INTERFACE through individual ALPHAs for each possible MESSAGE to the earliest operation required of it in the specifications.

This procedure is not necessary for the network above. All MESSAGES passed by CC_IN require an ACKNOWLEDGEMENT message response to be passed by CC_OUT. All HANDOVER messages clearly require a TRACK_INITIATION message to be passed by DATA_RECORD. The above network completely describes the only conditions where these responses are generated within the DPS. A TRACK_TERMINATION message is passed by DATA_RECORD in response to an input TERMINATION message passed by CC_IN. However, this case represents only one condition where a TRACK_TERMINATION message is generated by the DPS. The remaining conditions will occur on other R_NETs. But, all of these responses have one thing in common. Each is a single MESSAGE passed through a specific OUTPUT_INTERFACE in response to a given MESSAGE or class of MESSAGES passed by a single specific INPUT_INTERFACE. These are examples of "synchronous processing": the input is joined to the output by a direct path of processing steps, and none of the data used are modified by processing performed on any other path.

In the context of R_NETs, logical connectivity is maintained, not only by a continuous path through one R_NET, but perhaps through additional R_NETs, by means of EVENTS. An EVENT is an alternate means for enabling an R_NET. A single logical path is formed by the path leading to the event on the enabling R_NET and continuing on a path on the enabled R_NET. Such paths represent "synchronous processing" only if none of the data used are modified by an independent path.

"Asynchronous processing" is a more complex concept to grasp. This type of processing is implicit when two R_NETs are related by data, but without the "logical connectivity" represented by flowing tokens. An example involving three simple R_NETs will serve to illustrate the basic forms of "asynchronous processing." The example is defined in Figure 3-7.

Whenever a subsystem SS1 passes an XIN message through the DPS interface SS1_IN, the R_NET named X_VALUE is enabled. This R_NET accepts the

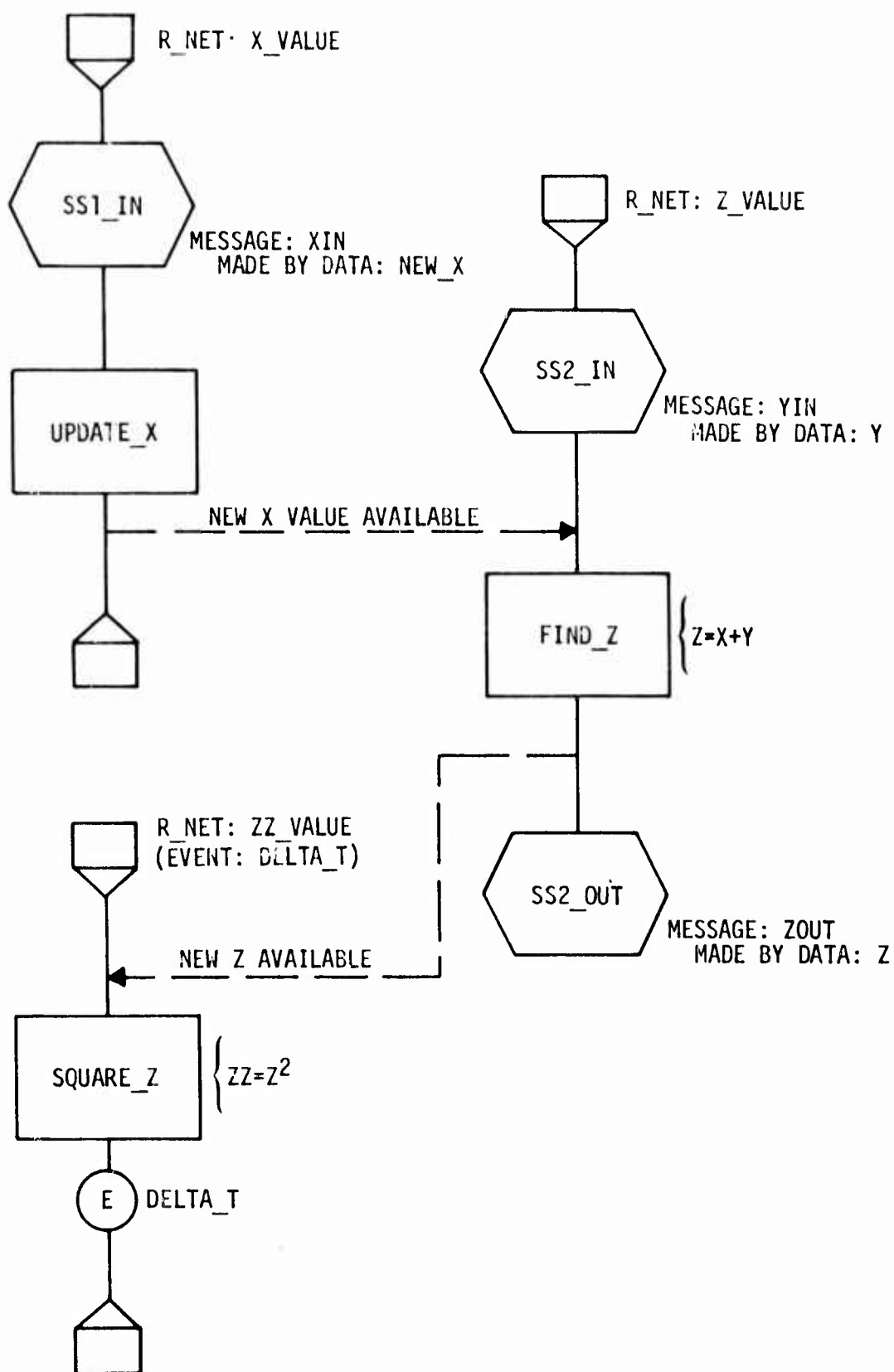


Figure 3-7 Three Asynchronous R_NETs

data NEW_X in the message and, if NEW_X satisfies certain conditions, updates the value of a global variable, X, to the value of NEW_X. Whenever a second subsystem, SS2, passes a YIN message through the interface SS2_IN, the R_NET named Z_VALUE is enabled. This R_NET accepts the data Y in the message and adds it to the current value of X defined in the data base to form Z (defined as a global variable for this example). The value of Z is contained in the ZOUT message sent to SS2 via SS2_OUT, but is also retained in the DPS because Z is defined as global data. Further, a third R_NET named ZZ_VALUE periodically enables itself, to compute Z^2 , by means of an event on its own structure and an associated time delay. The value of Z which is used is, of course, that which is current in the data base. There is no specific order of enablement required of the R_NETs.

When the reader experiments with this system a little, he will quickly realize that the values of Z and Z^2 depend not only on the inputs X and Y (or their previous values), but also on the sequence of enablement of the R_NETs. Thus, within a given time interval, there is not a one-to-one correspondence between the latest values of X and Y and the latest values of Z and Z^2 . For that matter, no such correspondence exists between Z and Z^2 .

A simple example of "asynchronous processing" in the TLS example might be the use of radar clock time. The sole function of the R_NET called RADAR_TIMING is to accept timing inputs from the radar and update the global variable RADAR_CLOCK. If another R_NET needed radar time, it would use the value of RADAR_CLOCK. This value does not reflect the current time, but rather the time at which the radar formed the message which was last accepted by the R_NET called RADAR_TIMING.

The major, and most complicated, example of "asynchronous processing" in the TLS is the relationship between radar returns and the next set of radar commands. Synchronous tracking would require that data from the last radar return from an object be used to produce the next succeeding radar order related to that object. Asynchronous tracking allows use of the last data available in the DPS, even though more current data may be coming soon from the radar. Asynchronous tracking allows less stringent DP response times, better time-line usage, and permits gradually degraded response with increased system load. Synchronous tracking, on the other hand, places

stringent constraints on the DP which lead to saturation and loss of track under relatively light load.

The definition of an asynchronous processing concept is subtle and difficult, and is fraught with traps for the unwary. No cookbook solutions can be offered for this creative process. First, the R_NETs must be traced both forward from the input interfaces and backward from the output interfaces. The data and logical connectivity to fill the gaps between must then be added through an active and creative engineering thought process. Note that SREM does provide a framework of organization which fosters consistency, and ultimately leads to a simulation which can detect errors of concept.

While the SREM user is filling in the gaps in the DPS definition, he must constantly remember that he is not designing software. Rather, he is defining the requirements which that software must satisfy. When he has invented a construct which appears to meet the needs, and which survives simulation, he should view it only as an example which demonstrates that a DPS solution to system requirements is feasible. It is probably not the only valid solution, nor should it be specified as such. The SREM user should constantly reexamine his constructs to ensure that they embody the requirements in the most general statements he can formulate. If he pursues details beyond the point needed to clearly state the required properties of the DPS, he is overly constraining the design.

When the user reaches an impasse in further defining the R_NETs, he will find it profitable to define the "entities" with which the DPS is concerned, and the data hierarchies associated with them. These are discussed in 3.1.7 and 3.1.8. These steps will sharpen his concepts of the data flow within the DPS, and should suggest ways of bridging the gaps in the processing. Then the user can return to complete the R_NETs, and possibly add intervening R_NETs.

3.1.7 Entity Definition

One of the most powerful concepts used in SREM is that of an "entity". This concept allows the user to express more clearly the role of the DPS requirements in the same operation, and the RSL facilities provided tend to enforce that perspective. An "entity" is simply a thing, or category of

things, in the external world about which the DPS must collect, process, and maintain data. Entities are closely tied to the reasons for the existence of the system and its DPS. They are usually implicit in the wording of the originating specifications, although the new SREM user must train himself to recognize those which are of significance.

For instance, the basic purpose of the TLS is to gather data on the position and velocity of designated objects within its detection range in order to predict the position and velocity of those objects at some future time. This suggests that "objects" might be an entity. However, "designated objects" would be a better candidate, because the TLS is not expected to detect any objects other than those the C² System orders it to track.

If we were considering the TLS as a whole, this might be a reasonable choice. But, we are focusing on the DPS. The DPS is not "aware" of objects because it does not perceive them directly. The radar performs the sensor functions. Thus, the DPS is only aware of what the radar perceives as objects and reports to the DPS. The nomenclature of the DPSPR calls these "images". Further reading of the DPSPR shows that much of the DPS processing is concerned with the proper classification of these images, and eventual elimination of those which do not correspond to real objects (ghosts), or which are redundant images of the same object. During the time that an image is considered an "image in track", a certain instance of data items must be maintained in the DPS and be associated with the proper image. When the DPS decides that the image is probably redundant or a ghost, or should drop track on that image for other reasons, the DPS must maintain, for a time, a different set of data about that image.

Thus, we have a notion of a general category of things called "images" which are of importance to the DPS. In SREM, such a category is called an ENTITY_CLASS. Hence, for TLS we will define an ENTITY_CLASS named IMAGE. We are aware of two types of IMAGE, distinguished by the different data associated with each type. Therefore, we will designate two ENTITY_TYPES, called IMAGE_IN_TRACK, and DROPPED_IMAGE. Each IMAGE of which the DPS is aware has an instance of data uniquely associated with it. This instance may be composed of DATA items and FILES. The composition of the instance is a function of ENTITY_TYPE and, by definition, should be different in some manner from at least one other type in the class. However, data common to

all types may be associated with the ENTITY_CLASS itself. Two types may have identical data associated with them. These usually imply a transition from a common earlier state (e.g., an ENTITY_TYPE I is set to either ENTITY_TYPE J or ENTITY_TYPE K depending on some decision in the DPS).

A second ENTITY_CLASS is defined in TLS for each radar PULSE. The pulse is an external phenomenon (an electromagnetic signal) about which the data processor is found to need data, and which exists in multiple copies. Therefore, it satisfies the criteria for consideration as an ENTITY_CLASS. The fact that data must be 'remembered' about each pulse while it is in transit, and the required data themselves, derive from the IFS through the process of defining the functional requirement. Thus, when considering the determination of range to the target, it is necessary to know both the start time of the range gate relative to the start of transmission and the time within the gate that the signal was detected. The IFS asserts that the radar return contains the time within the gate; the start time of the gate must therefore be 'remembered' by the data processor from the command which gave rise to the return. Thus, the data required on a pulse in transit have to do with the transmission parameters relevant to different pulse waveforms. Consideration of data and logical usage differences leads to the definition of four ENTITY_TYPES named T1-T2, T3, RETURNED_PULSE, and LOST_PULSE within the ENTITY_CLASS PULSE.

3.1.8 The Entity Data Hierarchy

The second major data hierarchy associated with SREM is the "entity data hierarchy" depicted in Figure 3-8. The means for manipulating data contained in an entity hierarchy differ from those used with the interface hierarchies of 3.1.4.

An ENTITY_CLASS is COMPOSED OF one or more ENTITY_TYPES. If there are data elements common to all ENTITY_TYPES, then the ENTITY_CLASS ASSOCIATES these DATA items and FILES. For data elements specific to an entity type, the ENTITY_TYPE ASSOCIATES the applicable data elements. Once again, the DATA item is the lowest element in the hierarchy. Each FILE CONTAINS items of DATA.

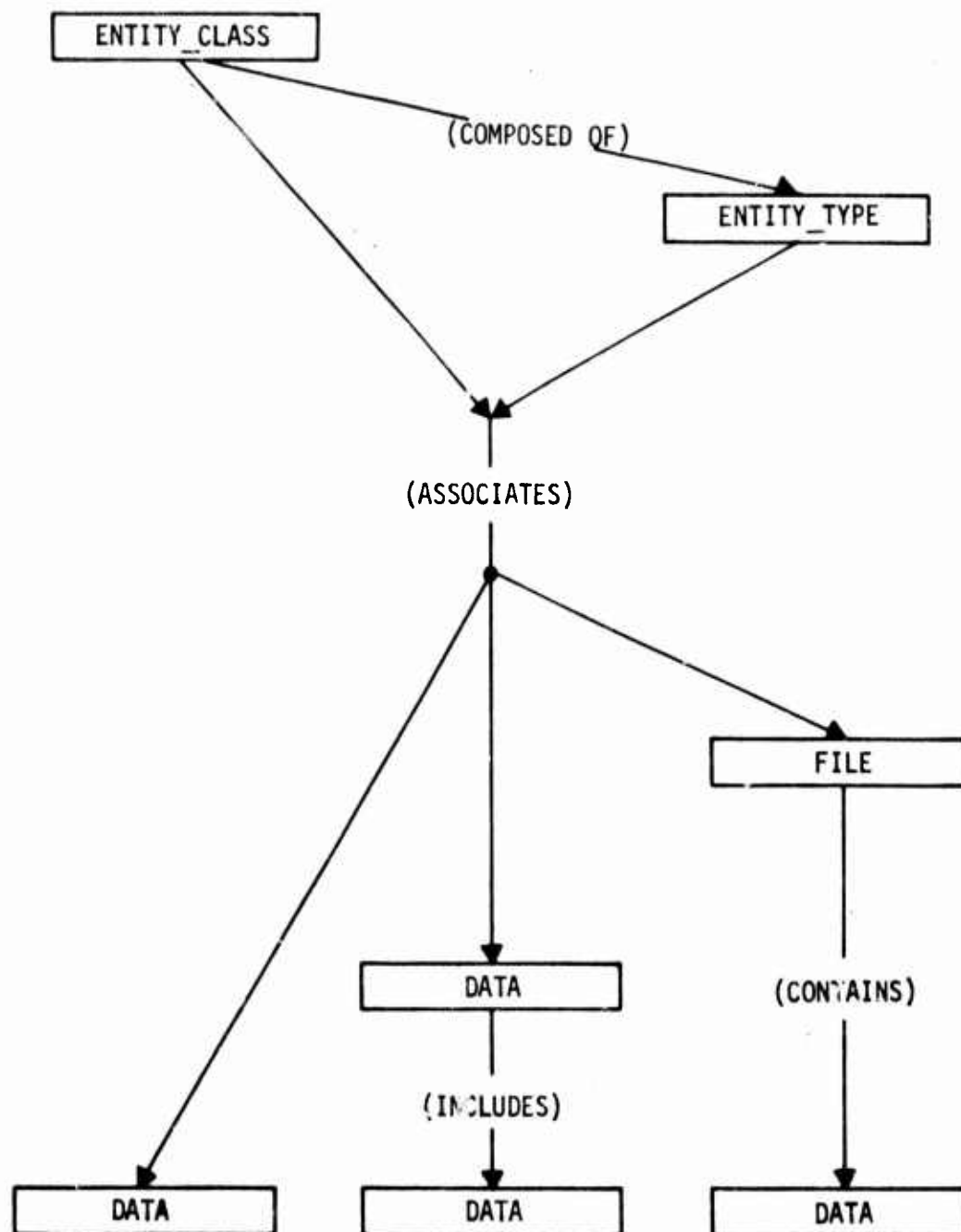


Figure 3-8 Entity Data Hierarchy

The chief characteristic of the entity data hierarchy is the unique method of manipulating data which is implemented in RSL. There is no way, in RSL, for a user to specify that a FILE be created or destroyed by an ALPHA. FILES may only be modified by ALPHAs (although instances in files are created or destroyed in the BETA models). RSL provides a mechanism to require that an ALPHA is to CREATE or DESTROY the knowledge that an instance of an ENTITY_CLASS exists in the environment. When an ALPHA creates a new instance of an ENTITY_CLASS, a new instance of all the common DATA and FILES ASSOCIATED WITH that class is initialized. The ALPHA may then assign proper values to those data elements. These data elements are retained throughout the life of the instance.

The ALPHA can also SET the ENTITY_TYPE. When this is done, an instance of all the specific DATA and FILES ASSOCIATED WITH the ENTITY_TYPE is initialized and can be assigned proper values. When the instance of ENTITY_CLASS is SET to a new ENTITY_TYPE, the specific data elements unique to the old type are destroyed and a new instance of data elements specific to the new type is initialized.

As the data processing system gathers information about an entity, it may first identify the entity as being of one ENTITY_TYPE, and then another. Instances of a class of entities thus evolve from one type to another, but instances of one class (e.g., IMAGES) can never evolve into another class (e.g., PULSES). Each ENTITY_TYPE therefore COMPOSES just one ENTITY_CLASS. An instance belongs to one ENTITY_CLASS after an ALPHA CREATES it, and it belongs to the last ENTITY_TYPE to which an ALPHA SETS it. Eventually, an ALPHA may DESTROY (knowledge of) the instance, and all data associated with the instance vanish. Although the RSL statements for CREATE and DESTROY refer to the name of the ENTITY_CLASS, the operation is applicable only to a single instance.

This concentration of the requirements for creation and destruction of knowledge about external entities, rather than the mechanics of data structures, allows the user to focus on the requirements for the DPS as a part of the system in which it is embedded. Otherwise, the user would tend to stray into process design issues, such as when to set up FILES.

As was done with MESSAGES in 3.1.3, it is useful to diagram the entity data hierarchies before coding them in RSL. Figure 3-9 shows diagrams of the TLS hierarchies associated with the two ENTITY_CLASSES, PULSE and IMAGE. When the user decides that the definitions are stable and ready to enter in the ASSM, the entries can be coded in RSL as shown in Figure 3-10 for IMAGE.

3.1.9 Independent FILES

When the user is defining the processing requirements, he may become aware of the need for data sets which fit in **neither a transient interface** data hierarchy nor a permanent entity data hierarchy. These should have the properties of 1) multiple instances of a data item or data group, and either 2) a need to be retained as GLOBAL data, or 3) a need to be ordered in some particular way, or 4) a need to be temporarily maintained as a class of data meeting some selection criteria. These needs can be satisfied by defining an independent FILE. This FILE exists in the DPS at all times, even though it may be empty. Although instances in the file can be created and destroyed within BETA and GAMMA executable descriptions of ALPHAs, the FILE itself cannot be created or destroyed. It can only be modified. FILES can be INPUT TO ALPHAs or OUTPUT FROM ALPHAs or both.

The distinction between the concepts of ENTITY_CLASS and independent FILE often may appear fuzzy. The key distinction is that a FILE is a set of data, an ENTITY_CLASS is the subject of data.

In the TLS, there exists a file of constants called WAVEFORM_TABLE. This FILE is part of the site-adaptation data for the system, and is independent of real-time input. Two other TLS examples of independent FILES are those called COMMAND and CANDIDATE. These files are dynamic (i.e. change in response to real-time data). Both of these files are used to **select** instances from ENTITY_TYPE IMAGE_IN_TRACK and to order the extracted data to determine new instances of ENTITY_CLASS PULSE. Thus, these files are part of the data bridge between the two ENTITY_CLASSES in TLS.

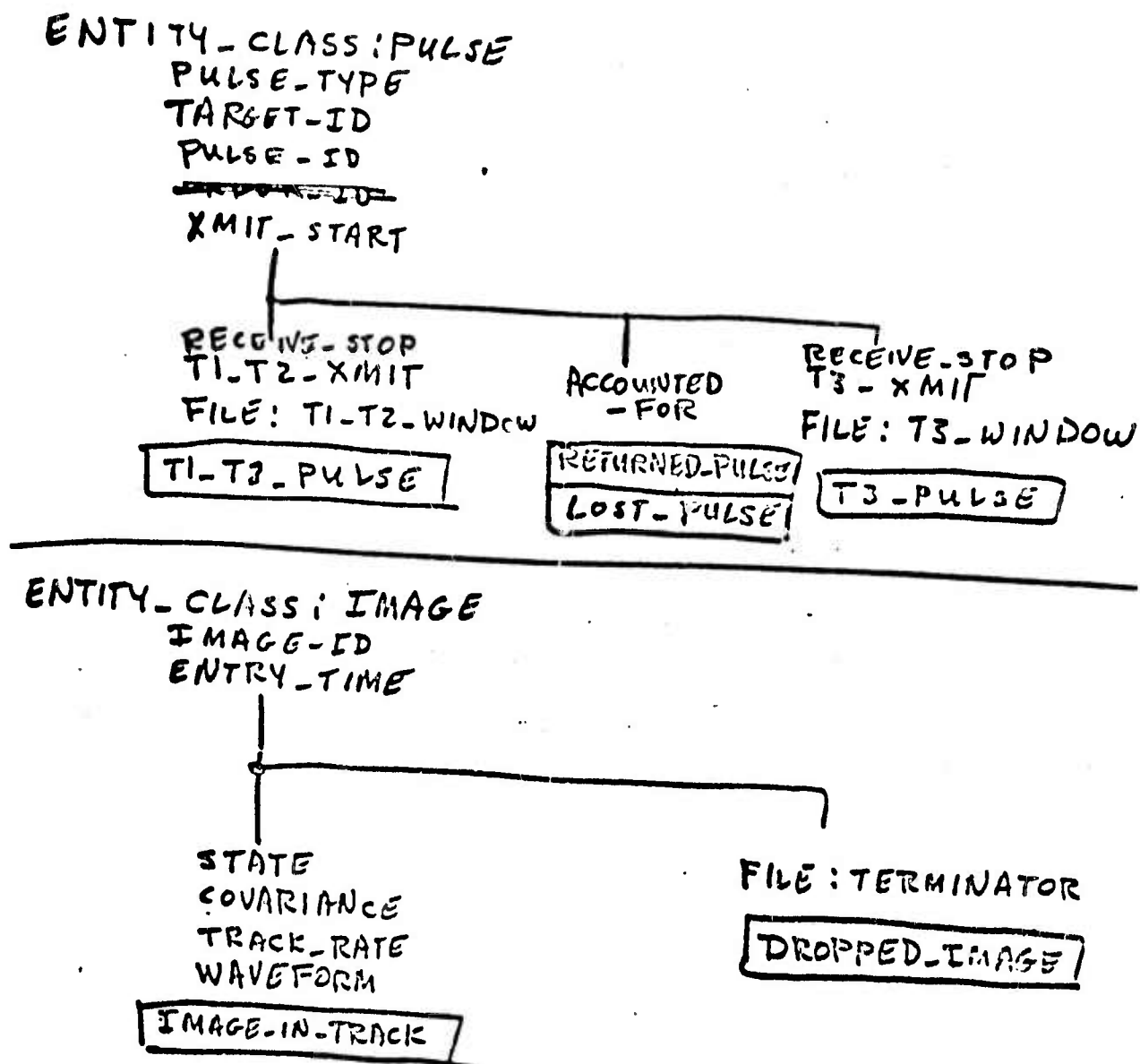


Figure 3-9 Entity Hierarchy

ENTITY_CLASS: IMAGE.
 ASSOCIATES:
 DATA: IMAGE_ID
 DATA: ENTRY_TIME.
 COMPOSED OF:
 ENTITY_TYPE: IMAGE_IN_TRACK
 ENTITY_TYPE: DROPPED_IMAGE.

ENTITY_TYPE: IMAGE_IN_TRACK.
 ASSOCIATES:
 DATA: STATE
 DATA: COVARIANCE
 DATA: TRACK_PATH
 DATA: WAVEFORM.

ENTITY_TYPE: DROPPED_IMAGE.
 ASSOCIATES:
 FILE: TERMINATOR.

Figure 3-10 RSL Entity Entry

3.1.10 Summary of Phase 1

This section has outlined a general procedure for defining the elements needed to specify the requirements for the DPS. At this point many of the definitions are gross and tentative. The constructs may have been entered into the ASSM as they were defined, or they may just exist on paper. This choice is up to the user, and depends upon the size and nature of the DPS problem, and the number of people involved.

The following "top down" sequence of steps has proved to be the most effective:

- 1) Define the subsystems relevant to the DPS.
- 2) Define the interfaces connecting the DPS to the subsystems.
- 3) Establish the messages passing through the interfaces, and define their data contents.
- 4) Develop the R_NETs originating at input interfaces.
- 5) Construct processing steps, tracing backward from the output interfaces when necessary.
- 6) Define the entities of concern to the DPS, and the associated data hierarchies.
- 7) Define independent files as necessary.
- 8) Refine the R_NETs and create new ones, as needed, to link the input and output interfaces.

Some variation on this sequence may be appropriate to certain problems, but the basic principle is to move from known interfaces to internal processing steps, using the data definitions as a vehicle.

The ALPHAs defined at this point will be primitive, and will reflect high level concepts of the nature of the processing. In Phase 3 (Section 3.3) they will be modified, and expanded into subnets, as necessary. But first, Phase 2 (Section 3.2) is needed to consolidate the information developed and to ensure that it forms a consistent basis for detailed development.

3.2 PHASE 2 - EVALUATION OF THE KERNEL

Before completing the definition of the functional requirements structure, it is prudent to enter the information derived in Phase 1 (3.1) into the ASSM and check the results, both manually and with the aid of Requirements Analysis and Data Extraction (RADX) procedures. Therefore, all infor-

mation from the paper constructs that has not been previously entered for convenience should be loaded into the ASSM at this time. This nucleus of information, called the "kernel", constitutes the irreducible minimum needed to document the major elements of the functional requirements definition. In this section we will suggest particular points for the user to check, and will refer him to TLS examples in Appendix C.

3.2.1 Data Naming Conventions

RSL has been designed to force the user into precision of thought, with respect to data definition, at an early stage. A single item of information may require several different DATA names in the course of its existence in the system. These are applied as the usage of the information and its properties of transience or permanence dictate. Naming is a tedious process, and may lead to a variety of awkward names, but it is mandatory in the development of a coherent, unambiguous DPS model. While the RSL translator will detect most common ambiguities or conflicts, the user should continually refine his understanding of the data flows within the DPS in order to catch more subtle errors.

The entity data are the first to display the constraints imposed by RSL naming conventions. In general, it is necessary that a data element exist in only a single hierarchy; the sole exception is that it may exist in multiple MESSAGES. Thus, we find that the identifier of an image being tracked is both the TARGET_ID ASSOCIATED WITH PULSE and the IMAGE_ID ASSOCIATED WITH IMAGE. The two values are the same when the TARGET_ID is assigned, but note that the destruction of an instance in one ENTITY_CLASS is unrelated to the existence of an instance of any other. The meaning of a single identifier, if the IMAGE instance were destroyed while the PULSE was still active, would be indeterminate. It is only to resolve such indeterminacy that the naming rules are imposed; here, the "same" information is given different names for the two occurrences in different hierarchies.

The identifier used for an IMAGE is given to the RESPONSE_TO_CC R_NET as a HO_ID (handover identifier). The same name is applied to a drop-track command when a TERMINATION message is sent, and also any of three MESSAGES through the DATA_RECORD interface, whenever information about that image is updated. However, when the information is to be held in global storage

(such as for an entity), a difference name must be applied (IMAGE_ID for the IMAGE class, TARGET_ID for PULSE). The requirement for renaming to avoid ambiguity is needed in order to construct an executable description (Section 3.4).

3.2.2 Structural Data Definitions

R_NETS can be entered into the ASSM in two ways. Using REVS in the ONLINE mode, the nets can be entered interactively via the Anagraph terminal at the ARC facility. Using REVS in the OFFLINE mode, the nets can be specified by an input deck of RSL statements. If the offline mode is used, certain DATA which are integral to the R_NET structure must be defined with appropriate statements before the set of statements which define the STRUCTURE of the R_NET can be integrated. Prior data definition is not necessary if the Anagraph terminal is used, but should be accomplished within Phase 2 for completeness of the kernel.

Structural DATA elements include those which are referenced in FOR EACH statements, CONSIDER statements associated with an OR node, and conditional expressions associated with an OR node. The latter two types are called "selection variables." Other DATA which should be defined at this point are those which DELAY the occurrence of an EVENT or ORDER a FILE.

The FOR EACH statement may be absolute or conditional. The object upon which the statement operates can be an ENTITY_CLASS, ENTITY_TYPE, or FILE. While the FILE is usually associated with one of the interface or entity hierarchies, assumed to be defined previously, some FILES may exist independently from any hierarchy, as discussed in 3.1.9. The user should verify that all elements used in FOR EACH statements are declared in the ASSM prior to the R_NET which uses them. Elements used in the condition part of a conditional FOR EACH must be defined as discussed below for selection variables.

Selection variables require added definition at this point because of the mechanics of the RSL translation process. In addition to declaration of the DATA name, its TYPE must be identified. Further, if the TYPE is ENUMERATION, the RANGE of values must be defined before the R_NET STRUCTURE can be translated.

While not mandatory, it is recommended that the LOCALITY and USE attributes for selection variables also be defined at this point, and be verified manually. This is because REVS consistency analysis does not include DATA referenced in conditional expressions. Thus, the software cannot detect definition errors until execution of a BETA or GAMMA simulation. Figure 3-11 shows RSL inputs which are typical TLS examples of selection variable definition.

The identification of selection variables within a system is usually straightforward. These are simply the decision parameters whose value determines whether one series of processing steps or an alternative is to be done. When multiple message types pass through an INPUT_INTERFACE, the type identifier contained in the message is, with no known exceptions, a selection variable (e.g., COMMAND_ID in the CC_IN interface hierarchy). Since the data input to an ALPHA is fixed, and since each message type implies either different data content or different processing, there must exist at least one unique ALPHA for each message type.

3.2.3 Entering R_NETs in the ASSM

The required information about each R_NET is its name, enabling mechanism, and structure. In the current version of REVS, names assigned to R_NETs, SUBNETs, and ALPHAs must be unique within the first eight characters. For all other RSL elements, names must be unique over a field of sixty characters, which is the maximum name length allowed.

The enabling mechanism of an R_NET is either a single INPUT_INTERFACE or a single EVENT. If a message passing through an interface is the mechanism, then the first statement in the R_NET STRUCTURE is an INPUT_INTERFACE name declaration. In the case of EVENT, the EVENT name does not appear in the STRUCTURE of the enabled R_NET. However, it appears in the STRUCTURE of the enabling R_NET at the appropriate point.

When the R_NET is entered into the ASSM via card input, data elements which appear in the STRUCTURE must be predefined in the ASSM. These data are discussed in 3.2.2. The R_NET STRUCTURE is discussed in 3.2.4.

DATA: MODE.
TYPE: ENUMERATION.
RANGE: "ENGAGED, STANDBY".
LOCALITY: GLOBAL.
USE: BOTH.

DATA: LAST_PULSE.
TYPE: REAL.
LOCALITY: LOCAL.
USE: BOTH.

DATA: TRACK_RATE.
TYPE: REAL.
LOCALITY: GLOBAL.
USE: BOTH.

DATA: TEOF.
DESCRIPTION:
"TEOF IS THE TIME OF THE END OF THE FRAME. IT IS USED
IN SELECTING CANDIDATES FOR TRANSMISSION IN THIS FRAME."
TYPE: REAL.
LOCALITY: LOCAL.
USE: BOTH.

Figure 3-11 RSL Data Entry

3.2.4 The STRUCTURE of an R_NET

Flows through the system are specified in RSL as Requirements Networks (R_NETs). R_NET flow STRUCTURES consist of nodes, which specify processing operations, and the arcs which connect them. The processing nodes are ALPHAs, which are specifications of functional processing steps, and SUBNETs, which are specifications of processing flows at a lower level in the hierarchy. The processing nodes are single-entry, single-exit.

In addition to the simple sequential flow which may be represented by connecting this type of node, more complex flow situations are expressible in RSL by the use of structured nodes which fan-in and fan-out to specify different processing paths. These nodes are variations of AND and OR nodes, and include an OR with a CONSIDER statement. The REVS Users Manual contains an extensive discussion of these structural elements and should be reviewed by the new user. The TLS examples in the appendices of this volume should also be studied to understand the format of various structures.*

*NOTE: In the version of REVS used to generate this document there were two differences between acceptable STRUCTURE inputs and listed outputs. These concern the CONSIDER and FOR EACH statements. In each case the element-type-names, which are automatically inserted in the output, must be omitted in the input. Where the output list would read:

```
CONSIDER DATA:  MODE
```

```
FOR EACH ENTITY_TYPE:  RETURNED_PULSE
DO      ALPHA:  SUMMARIZE_USAGE END
```

the input statements must read:

```
CONSIDER MODE
```

```
FOR EACH RETURNED_PULSE
DO  SUMMARIZE_USAGE END
```

The current version of REVS will accept inputs in either format.

Indentation is used in the STRUCTURE declaration to facilitate reading. When entering the declaration on cards, using REVS in the offline mode, the user should follow the indentation format typified by the REVS output examples in the appendices. This is not mandatory, but it is strongly recommended in order to check the output against the input. In this way, the user can quickly detect where REVS has interpreted the STRUCTURE in a different way than the user visualized.

In addition to describing the processing flow of an R_NET, the STRUCTURE declaration may also serve to declare the existence of the named ALPHAs and SUBNETs. The STRUCTURE of any SUBNETs should also be declared at this time. Further definition of the attributes of ALPHAs will take place in Phase 3.

If Anagraph or CALCOMP plots of the structures are desired, graphic data must be entered for each R_NET. Currently, this graphics information can only be entered from the Anagraph terminal. The Anagraph plots are useful while working interactively at the terminal. The CALCOMP plots are more useful for permanent retention, and for documentation, as in Appendix D.

3.2.5 Checking the Kernel with the Aid of RADX

As part of the auditing process, it is useful here to define a set of RADX directives which assist the engineer in determining the completeness of his entries into the ASSM at this stage. Two operations are recommended which are simple, but revealing:

- 1) APPEND R_NET STRUCTURE, ENABLED.
LIST R_NET.
- 2) APPEND SUBNET STRUCTURE, REFERRED.
LIST SUBNET.

These directives generate a listing of the R_NETs with their structures and enabling events or interfaces. Clearly, each R_NET requires a structure, and an enabling condition, and any failure of that class is detectable here. Similarly, the correctness (agreement with intention) of EVENT naming may be confirmed.

While it is possible to confirm the structures by inspection of the RADX output, most people find reading the equivalent diagrams (generated by RNETGEN) simpler; since they are equivalent representations, either the CALCOMP illustration or the RADX listing may be employed for the purpose. (Note that viewing the Anagraph output at the terminal is less useful since all element names are truncated to three characters and since branching criteria are not immediately displayed.)

It is in data analysis that the RADX tools are most useful at this stage. Let us define two hierarchies for data output:

HIERARCHY FILES = FILE CONTAINS DATA DATA INCLUDES DATA.
HIERARCHY DATUM = DATA INCLUDES DATA.

Now, we wish to identify the DATA items and the FILES which are not linked with higher data levels (entities, or messages). One way derives from the following definitions:

SET A = ALL WITHOUT ASSOCIATED.
SET B = A WITHOUT MAKES.
SET C = B WITHOUT CONTAINED.
SET D = C WITHOUT INCLUDED.

Now the RCL (RADX) directive: LIST B WITH HIERARCHY FILES will generate both the file names and the data comprising each such file where the file is not associated with an entity and does not make a message. Such a free-standing file should be a conscious product of requirements engineering and not an accident.

Free-standing DATA may be extracted by: LIST D WITH HIERARCHY DATUM. Note that the HIERARCHY DATUM is used here as a convenience to limit the output to the names of the top data level not constituting part of a hierarchy; it does not in fact cause INCLUDED DATA to be output, since the SET from which the extraction is effected (D) has no members which are INCLUDED in any DATA. To complete tracing of the data hierarchies, yet another SET would have to be defined containing the DATA extracted by LIST D WITH HIERARCHY DATUM, and that SET would then be LISTed with HIERARCHY DATUM.

Typically, free-standing DATA and FILES may be local or global constants. Each item extracted by the methods outlined above should be assessed to determine whether in fact it should be isolated or is properly a constituent

of an entity or interface hierarchy.

Finally, the following HIERARCHIES may be defined, and each may be used to complete the RADX directive: LIST ALL WITH HIERARCHY _____.

HIERARCHY ENTITY =

ENTITY_CLASS ASSOCIATES DATA
ENTITY_CLASS ASSOCIATES FILE
ENTITY_CLASS COMPOSED ENTITY_TYPE
ENTITY_TYPE ASSOCIATES DATA
ENTITY_TYPE ASSOCIATES FILE
FILE CONTAINS DATA
DATA INCLUDES DATA.

HIERARCHY INFACE =

INPUT_INTERFACE PASSES MESSAGE
MESSAGE MADE DATA
MESSAGE MADE FILE
FILE CONTAINS DATA
DATA INCLUDES DATA.

HIERARCHY OUTFACE =

OUTPUT_INTERFACE PASSES MESSAGE
MESSAGE MADE DATA
MESSAGE MADE FILE
FILE CONTAINS DATA
DATA INCLUDES DATA.

The result is to extract from the ASSM the complete kernel of the requirements, as illustrated in Appendix C.

3.2.6 Summary of Phase 2

This phase has been a period of consolidation, between the initial definitions of Phase 1, and the completion of those definitions in Phase 3. Nonetheless, the end of Phase 2 is an important milestone in the total effort.

For the first time, the kernel of the requirements construct has been loaded into the ASSM as a whole. Many of the early mistakes and inconsistencies will be detected by the RSL translator during the entry process. More important, the requirements engineer can now use the facilities of the

Requirements Analysis and Data Extractor (RADX) to aid him in checking his work.

The introductory uses of RADX, in 3.2.5, will be built upon and expanded in Phase 3. Eventually, the user will compile his own library of RADX procedures he values. In future efforts he can use these "off-the-shelf" procedures as part of the evaluation sequence which he finds most productive.

3.3 PHASE 3 - COMPLETION OF THE FUNCTIONAL DEFINITION

The work through Phase 2 has provided definition of the higher-level elements of data hierarchies; definition of R_NETs, their STRUCTURES, and their enabling events; and declaration of the existence of ALPHAs and their place in the R_NETs. This information has been entered into the ASSM and has been subjected to some form of checking.

It now remains to complete the definition of these elements to provide a basis for construction of an executable simulation. Each ALPHA must be defined in terms of its data transactions, and definition of all DATA which are known to be INPUT TO or OUTPUT FROM an ALPHA must be completed. In general, the DATA which can be described at this stage are either those global to multiple R_NETs, or those local DATA which make messages. The need for additional local data internal to the R_NETs will be discovered in the development of executable descriptions (Section 3.4). At that time the need for definition of lower levels of the existing data hierarchies may be apparent.

During this process, the original primitive ALPHAs will often need redefinition. Additional ALPHAs can be added, or the original ALPHAs can be redefined as SUBNETs, which preserves traceability and minimizes modification of the R_NETs. Occasionally, the STRUCTURE of the net may change, or new nets may be added, as the processing requirements evolve.

The following phase will be devoted to construction of a functional simulation, using BETAs, which are executable descriptions of the ALPHAs. In preparation, the user should bear in mind that he is going to execute a simulation of the requirements for the DPS software, and not of the software itself. His primary goal in this simulation is to ensure that the requirements are complete and consistent.

Since the major interactions of the R_NETs have been defined in the previous stages (Sections 3.1 and 3.2), it is now practical to partition the work into essentially isolated efforts for several engineers. Each parcel must consist of one or more R_NETs, each considered by a single engineer or group. Since the relationships among R_NETs have already been defined, the interactions among parcels will be restricted to joint agreement on the content of communications (usually individual DATA) between pairs of engineers, and need not be coordinated extensively over the system as a whole. The sole exception to that rule is in naming conventions, where a possibility of conflict exists, and where control is useful.

3.3.1 Data Transactions

The user presumably had some concept of the DATA which were to be INPUT TO and OUTPUT FROM each ALPHA when he named the ALPHA and placed it in the STRUCTURE of an R_NET. After all, the inputs and outputs determine the processing needed to be done and dictate the function of the ALPHA. Now the gross concept must be precisely defined. During this process, additional data definitions or more detailed definition of existing hierarchies may be necessary.

As the user develops the data transactions and the hierarchy transitions discussed in 3.3.3, he will be forced to consider the processing steps within each ALPHA which transform the inputs into the required outputs. He may wish to note his conceptual ideas of the procedure into the ASSM, in English text, using the RSL attribute DESCRIPTION. A structured English description of the processing can later be used for reference during development of the BETAs which are written in PASCAL. If, at this point, it becomes apparent that significant logical branching must occur within the ALPHA, the ALPHA can be redefined as a SUBNET. In this manner the logical structure requirements are made explicit in the STRUCTURE definition of the SUBNET and new ALPHAs are defined to specify the operations within this structure.

As an example of the thought process involved with data transactions, consider the ALPHA named INITIATE_TRACK. This ALPHA is on the CC_RESPONSE R_NET and is on the processing path activated by a HANDOVER message of type HANDOVER_IMAGE (= COMMAND_ID). Since COMMAND_ID was used as a selection

variable in order to reach the ALPHA, it is unlikely that it will be used within the ALPHA. However, the remaining data content of the MESSAGE must be used within the ALPHA because there are no other ALPHAs on this path which could use the unique content of this message type. These DATA are HO_ID, INITIAL_STATE, and INITIAL_COVARIANCE. Thus, these DATA will be defined as INPUT TO the ALPHA.

TRACK_INITIATE must FORM a MESSAGE to be PASSED BY the OUTPUT_INTERFACE named DATA_RECORD. It is obvious that TRACK_INITIATION is the proper message to be formed and passed. Thus, the DATA in that message must be OUTPUT FROM the ALPHA. These are HO_ID, INITIAL_STATE, and TIME_OF_INITIATION. The first two DATA are present in the inputs, but the third is not.

The remaining DATA item for the MESSAGE is the time of arrival of the incoming command at CC_IN. REVS maintains a DATA item called CLOCK_TIME which is set to the simulator time at the initiation of each R_NET, and which is not to be altered by any processing. (Since no time passes during the "execution" of an R_NET, CLOCK_TIME is not updated along a net.) Note that CLOCK_TIME is a DATA item predefined within the nucleus of the ASSM; a second DATA item is maintained by REVS, FOUND, which is TRUE if the last SELECT operation retrieved an instance satisfying the selection criterion. Neither element may ever be OUTPUT by an ALPHA. Either element may be INPUT TO an ALPHA, as CLOCK_TIME is INPUT TO TRACK_INITIATE for assignment as TIME_OF_INITIATION, which is then OUTPUT for the MESSAGE.

What additional processing is needed in TRACK_INITIATE? Its primary function is to use the data provided in the HANDOVER message to initiate track on a new image. Thus, it is logical to assume that the ALPHA CREATES a new instance of ENTITY_CLASS IMAGE, and designates it as ENTITY_TYPE IMAGE_IN_TRACK. Hence, the ALPHA should output the DATA associated with this ENTITY_CLASS and ENTITY_TYPE. These are: IMAGE_ID, ENTRY_TIME, STATE, COVARIANCE, TRACK_RATE, and WAVEFORM.

It is also reasonable to assume that TRACK_INITIATE should assign the values of HO_ID, CLOCK_TIME, INITIAL_STATE, and INITIAL_COVARIANCE to IMAGE_ID, ENTRY_TIME, STATE, and COVARIANCE, respectively. These assignments are unconditional, as no choice criteria are indicated in the specifications. For the moment, we will beg the question of the origin of TRACK_RATE and WAVEFORM. At this stage it is sufficient merely to define

them as OUTPUT FROM the ALPHA. Figure 3-12 shows the declaration, in RSL, of all the data transactions of TRACK_INITIATE. From our concept of the processing, it appears that the original ALPHA is adequate and no additional ALPHAs, or redefinition as a SUBNET, are required.

3.3.2 RADX Evaluation of Data Transactions

In addition to confirming entry of attributes and relationships through RADX directives (e.g., LIST ALPHA.), it is now useful to extract the specific cases which are potential errors, or are at least anomalous to the point where they demand special attention. For example, it is possible for an ALPHA either to have no INPUTS or to have no OUTPUTS, or even to have neither, without its being erroneous; but the normal case is that each ALPHA will have both. Having completed the RSL declarations about the ALPHAs at this stage, it is meaningful to define the following:

- 1) SET A = ALPHA WITHOUT INPUTS.
- 2) SET B = ALPHA WITHOUT OUTPUTS.
- 3) SET C = A WITHOUT OUTPUTS.

Remembering that the declaration of a SET generates a count of its members, we may discover that SET C is empty; that would indicate that each ALPHA has either an INPUTS or an OUTPUTS relationship (or both). However, in TLS, several ALPHAs have neither. In particular, the error-processing elements are required to exist, but have no other specifications; therefore, they have no accesses defined. Similarly, the ALPHA:ACKNOWLEDGE exists solely so that the ACKNOWLEDGEMENT message may be FORMED; since the processing modifies no DATA, the ALPHA has neither INPUTS nor OUTPUTS.

In general, an ALPHA without INPUTS is one which operates on the system as a whole, rather than on information retained by the DPS. For example, ENGAGEMENT_INITIATION and TERM_ENGAGEMENT accomplish their purposes by the occurrence of the appropriate message; only the value of the selection variable (COMMAND_ID) is required to initiate them, and those ALPHAs execute independently of the state of the global data base.

In general, an ALPHA with INPUTS but without OUTPUTS is a signal of a specification problem. Trivially, the only reason for acquisition of DATA for processing is that it may effect some generation of DATA for OUTPUT. We have not yet been able to construct a valid case for an ALPHA which must

ALPHA: TRACK_INITIATE.
INPUTS:
DATA: HO_ID
DATA: INITIAL_STATE
DATA: INITIAL_COVARIANCE
DATA: CLOCK_TIME.
OUTPUTS:
DATA: IMAGE_ID
DATA: STATE
DATA: COVARIANCE
DATA: ENTRY_TIME
DATA: TIME_OF_INITIATION
DATA: TRACK_RATE
DATA: WAVEFORM.

Figure 3-12 RSL Initial ALPHA Entry

accept INPUTS but is not required to provide OUTPUTS.

Since the work described in 3.3.3 and 3.3.4 often proceeds in parallel with that of 3.3.1, we will address those topics at this point. We will return to a more detailed look at RADX evaluation in 3.3.5.

3.3.3 Hierarchy Transitions

In addition to the elementary data relationships, each ALPHA may modify DATA within hierarchies in a variety of ways. Given the engineer's concept of the action of the ALPHA, it is possible to express its action on the hierarchies in terms of the RSL directives: CREATES, DESTROYS, SETS, and FORMS.

There are two fundamental generative operations that an ALPHA may express: CREATES and FORMS. They declare the need for a new instance of a named ENTITY_CLASS, or for a named MESSAGE, respectively. In response to either directive it is required that the specified INITIAL_VALUES for all associated DATA with such values be assigned. This is done automatically when REVS acts upon an INITIAL_VALUE definition. If an INITIAL_VALUE is not defined, REVS will assign a default value according to the TYPE of the DATA. These values are (0, 0.0, FALSE, first defined value in the RANGE) for the respective types (INTEGER, REAL, BOOLEAN, ENUMERATION). Once the DATA associated with the instance are initialized, they may be assigned current values by assignment statements within the BETA.

When an instance of an ENTITY_CLASS is CREATED, it will normally be SET by the generating ALPHA. Otherwise, only the DATA and FILES common to all types in the class can be defined. As the instance persists in the system, its type will evolve (i.e., be SET to a different type). The ALPHAs in which such changes are effected are normally obvious from the R_NET. In each case, such an ALPHA SETS ENTITY_TYPE to the appropriate ENTITY_TYPE name.

For example, the discussion, in 3.3.1, of the ALPHA named TRACK_INITIATE revealed that the ALPHA performs all of the actions above. It FORMS the MESSAGE named TRACK_INITIATION. Also, it first CREATES an instance of ENTITY_CLASS IMAGE, then SETS the instance to ENTITY_TYPE IMAGE_IN_TRACK. Figure 3-13 shows how these declarations are written in RSL.

ALPHA: TRACK_INITIATE.
FORMS:
MESSAGE: TRACK_INITIATION.
CREATES:
ENTITY_CLASS: IMAGE.
SETS:
ENTITY_TYPE: IMAGE_IN_TRACK.

Figure 3-13 RSL Additional ALPHA Entry

When there is no further need for the DPS to retain data related to a specific instance of an ENTITY_CLASS, or to be aware of the existence of the instance in the external world, the (instance of the) ENTITY_CLASS can be DESTROYED BY an ALPHA. The disposition of a PULSE in the TLS depends on whether it is a LOST_PULSE or a RETURNED_PULSE. For a LOST_PULSE, the instance is DESTROYED (no longer needed) after it has been accounted for in the ALPHA named ALLOCATE_AND_CONTROL_RESOURCES. For a RETURNED_PULSE, the instance is not destroyed until it has also been accounted for in the ALPHA named SUMMARIZE_USAGE. Note that, in this case, the instance of PULSE can be DESTROYED in either of the two ALPHAs, but not before it has been accounted for in both. This is because the R_NETs in which the ALPHAs reside execute independently of one another, and no specific sequence is otherwise required.

3.3.4 Further Data Definition

Through the previous work many, if not most, of the DATA in the system have been named, and their relationships with ALPHAs have been established. In order to complete the requirements specification and construct an executable simulation, however, the "attributes" of the DATA must be defined.

The three principal attributes are LOCALITY, TYPE, and USE. In general, the user will have a fair idea of these properties when he first declares the DATA element. His concept is further sharpened when he sees how the ALPHAs utilize the DATA.

3.3.4.1 Locality

DATA and FILES may have different required accessibility in the system. The range of accessibility of an item is denoted by the attribute LOCALITY, which may have values of LOCAL or GLOBAL. Items of DATA or FILES which are LOCAL are associated with the R_NETs in which they are used and are unknown outside of these R_NETs. Implicit in this definition is a concept of permanence: LOCAL DATA exist only during the invocation of the R_NET to which they are LOCAL. They are created when the flow token is generated at R_NET ENABLEment and cease to exist when the flow token leaves that R_NET. ALPHAs which use LOCAL DATA and FILES may appear on more than one R_NET; it is possible for a single DATA item or a FILE to be LOCAL to

more than one R_NET. These are different instances of the DATA or FILE which have no relation to each other, each has a completely separate existence, controlled by the R_NET in question.

GLOBAL DATA and FILES are accessible by more than one R_NET and exist over more than one R_NET invocation. DATA and FILES which are ASSOCIATED with an ENTITY_TYPE or an ENTITY_CLASS are tied to the entity instances to which they belong. They are created when the instance is CREATED and last until the instance is DESTROYED. Items which are not ASSOCIATED with anything are permanently in the global data base, and may exist throughout the duration of the system.

Thus, DATA and FILES which belong to an interface data hierarchy are implicitly LOCAL. DATA and FILES associated with an entity data hierarchy are implicitly GLOBAL. DATA and FILES not associated with either type of hierarchy have no implicit locality. Even if the locality is not explicitly defined, an item exists in the data base at all times, accessible to the R_NETs, but in an "undefined" state. Failure to declare LOCALITY can produce weird consequences later, if the element involved has no implicit locality. On the other hand, declarations are not needed if the locality is implicit. At best they are redundant, and at worst they are misleading because REVS overrides any declaration in the ASSM which conflicts with the implicit locality.

While the user may clearly identify which DATA and FILES need LOCALITY declarations, and which do not, it is easier and less risky to use a RADX procedure to do this. One such procedure is discussed in 3.3.5.

For the remaining items which need a LOCALITY declaration, the user should consider the source of the item. If an independent DATA item or FILE is not OUTPUT FROM some ALPHA on an R_NET previous to its being INPUT TO some other ALPHA on that net, it certainly cannot be LOCAL to that R_NET. If the item is OUTPUT FROM an ALPHA on some other R_NET, the item is certainly GLOBAL unless an error has been made in the INPUTS and OUTPUTS definitions. In many cases the correct locality is obvious. The inability to find a source for an independent item on any of the R_NETs indicates a specification deficiency.

If a DATA item or FILE is OUTPUT FROM an ALPHA on a net other than that which uses the item as input, the indicated LOCALITY is GLOBAL. If no other R_NET generates the item, then the origin of the item is on a previous execution of the R_NET which uses it.

If the ALPHA which OUTPUTS an item does not clearly precede the ALPHA which INPUTS it, care must be taken to ensure that the item has an initial value. The default values assigned by REVS in the absence of an INITIAL_VALUE declaration were described in 3.3.3. If these are unacceptable, the user must declare the correct INITIAL_VALUE. Note that "clearly precede" in the context above means that the ALPHA which OUTPUTS the item is either:

- 1) before the ALPHA which INPUTS it, on the same path within an R_NET, or
- 2) precedes the ALPHA which INPUTS the item on a single path linked by enabling EVENTS.

3.3.4.2 Type and Range

Other details about DATA must be known for purposes of simulation. BETAs and GAMMAS are executable code which are meaningful only if more is known about the DATA than should be stated as a requirement. In addition, a hierarchy of DATA may be stated as a requirement, but a functional simulation (using BETAs) may employ DATA only part way down the hierarchy. That is, the simulation may use one DATA to represent a part of an entire hierarchy. The characteristics of this summarized DATA must be stated for the simulation to execute. Since an analytic emulation (using GAMMAS) might not employ the same DATA, the type information for BETAs may be different from that for GAMMAS. The only DATA that can be used in the BETAs and GAMMAS and have their values communicated between ALPHAs, however, are those whose TYPE has been defined.

The attribute TYPE contains the necessary information for typing of the DATA. This attribute may have values REAL, INTEGER, BOOLEAN, or ENUMERATION. A DATA item with type enumerated corresponds closely to one with a scalar type in PASCAL; that is, it has values which are denoted by identifiers. The legal values for ENUMERATION types are given in the RANGE attribute.

The TYPE declared need not correspond to that of the actual data in the real DPS. Rather, it should be chosen to reflect the purposes of the functional and analytic simulations, and the fidelity required in those simulations. For instance, in the real DPS, unique alphanumeric text strings may be used to identify objects. If these are an open set, we cannot represent them by a DATA element with TYPE: ENUMERATION because that defines a closed set. However, for purposes of simulation, the DPS property of interest is the ability of the DPS to distinguish between unique identifiers. Thus, by defining the identifier as TYPE: INTEGER, and representing each object or class of objects by a unique integer value, we have an adequate representation for our purposes.

Similarly, in TLS, identifiers such as HO_ID, IMAGE_ID, and RADAR_ORDER_ID are represented by integers. In the real TLS some symbolic code convention, which is irrelevant to us at this stage, would be used. In like fashion, message identifiers such as COMMAND_ID which form a closed set are represented as DATA with TYPE: ENUMERATION. The values defined are for explanatory purposes. The values in the actual TLS would certainly be different.

Since the appropriate TYPE for DATA is strongly dependent on its USE, the choice may be deferred until the USE has been determined.

3.3.4.3 Use

Further qualification of the use of a DATA item in the simulation is given by the attribute USE. The value of this attribute may be BETA, GAMMA, or BOTH denoting that the data item is the lowest level in the data hierarchy which will be used in the corresponding simulation.

Frequently, even the first declaration of a DATA item makes clear its USE in the system. For example, if the element is known to correspond to a single unit of information (bit, byte, or word) in implementable software, then it is probably to be used in BOTH beta and gamma models. Normally, a selection variable will be in this class. Similarly, it is likely that an item which INCLUDEs others in the detailed modeling, but which may be treated as a whole for functional modeling will have USE:BETA. It is unlikely that any element with USE restricted to GAMMA will be recognized at this stage of development, since its declaration would be primarily in

support of analytic simulation; the exception would occur if a highly detailed interface specification gave low-level DATA definitions, for which higher levels would suffice in a functional model.

For example, in the TLS definition, INITIAL_STATE is a single DATA element with USE:BETA. In the external world "state" is defined by a position vector, a velocity vector, and perhaps an acceleration vector, each with three components. Such detail is not needed for the functional simulation. However, an analytical simulation has need for these elements. Thus, eventually each of the components will be defined, with USE:GAMMA. Similarly, for the functional simulation, INITIAL_COVARIANCE can be represented by a single element with USE:BETA. Later, its matrix components can be defined with USE:GAMMA when they are needed for analytic simulation.

3.3.4.4 Values

Values are the ultimate object of defining DATA. At the lowest level in a hierarchy of DATA the requirements engineer may specify the attributes UNITS, MAXIMUM_VALUE, MINIMUM_VALUE, INITIAL_VALUE, and RESOLUTION. The attribute UNITS is given separately from the various types of values, both to maintain consistency with their specification and to enable requirements engineers to indicate the minimum possible information about a DATA's value, its UNITS. In nearly all cases an engineer will know whether he is talking about milliseconds or microseconds even if he is unsure of the value. Separating UNITS from the values enables him to provide the best information that he has at the initiation of defining a DATA item.

Since the attributes UNITS, MAXIMUM_VALUE, MINIMUM_VALUE, and INITIAL_VALUE are not vectors, the definition of different UNITS and values of a DATA set above the lowest level in the hierarchy is not possible. RESOLUTION describes the required maximum value of the least significant bit for the DATA in units described in the UNITS attribute.

3.3.5 Evaluation of the ASSM Using RADX

The Requirements Analysis and Data Extraction (RADX) function of REVS is the tool used by the requirements engineer to observe the state of the Abstract System Semantic Model. RADX provides commands that allow the performance of several functions:

- identification and listing of elements in the ASSM that do or do not meet some criterion.
- listing of ASSM elements in such a manner as to be suitable for inclusion in requirements documents.
- listing of RSL element, attribute, and relation definitions.
- analysis of the ASSM to identify requirements that are ambiguous or inconsistent.

The requirements engineer should actively use RADX to extract specific data for analysis and to detect inconsistencies and omissions in the ASSM. This capability is also used by technical managers to evaluate progress of the development activity, and by independent analysts who may be assigned to verify the work of the requirements engineer.

The following subparagraphs discuss typical uses of RADX in evaluating the work done in Phase 3, and in identification of work remaining to be done. These examples are not meant to be comprehensive, nor are they the only way to do the task. A comprehensive catalog of RADX procedures would be a very thick document, and, there are many ways to do a specific extraction task effectively. Other examples can be found in the REVS Users Manual. Our purposes here are merely to suggest some of the uses of RADX, and to encourage the user to creatively use these facilities in his work.

3.3.5.1 RADX Evaluation of Data Origin and Usage

A DATA item may be entered into the data base in any of the following ways:

- 1) It may be OUTPUT BY an ALPHA;
- 2) It may be INCLUDED IN a DATA OUTPUT BY an ALPHA;
- 3) It may be CONTAINED IN A FILE OUTPUT BY an ALPHA; or
- 4) It may MAKE a MESSAGE which is PASSED BY an INPUT_INTERFACE.

(Let us include in the last category DATA INCLUDED in DATA which MAKES such a MESSAGE and DATA CONTAINED IN a FILE which MAKES such a MESSAGE.)

We wish to extract exceptions to the above rules, since they constitute apparent cases of 'creative memory' -- data extractable from the global data base which never need to be entered. An obvious and legitimate case of creative memory is a constant with a defined INITIAL_VALUE; in fact, DATA constants are properly defined in just such a manner. But any non-constant

data item which fits in none of the above categories is an apparent error, and worthy of detailed analysis. (Note that a DATA element which is never INPUT TO an ALPHA and which does not MAKE a MESSAGE which is PASSED BY an OUTPUT_INTERFACE is also worthy of scrutiny, and may be similarly analyzed at this stage. Among the DATA which will properly be detected at this step are those REFERRED by an R_NET-- e.g., selection variables.)

With the aid of the hierarchy definitions in 3.2.5 we can construct RADX procedures to isolate the DATA described above. First we will define RADX commands which are used in both cases:

```
SET INALPH = DATA THAT IS INPUT BY ALPHA.  
SET OUTALPH = DATA THAT IS OUTPUT BY ALPHA.
```

The additional commands below will provide a listing of DATA INPUT TO an ALPHA which does not have an INITIAL_VALUE, and is not in a MESSAGE PASSED BY an INPUT_INTERFACE, and which is not OUTPUT FROM some ALPHA on some R_NET.

```
SET INMSG = ALL IN HIERARCHY INFACE.  
SET INDATA = INMSG AND DATA.  
SET INOUT = INALPH MINUS OUTALPH.  
SET INOUT = INOUT MINUS INDATA.  
SET INOUT = INOUT WITHOUT INITIAL_VALUE.  
LIST INOUT WITH HIERARCHY DATUM.
```

If, instead, we use the commands below, we will get a listing of DATA which is OUTPUT from an ALPHA and not INPUT TO an ALPHA and not in a MESSAGE PASSED BY an OUTPUT_INTERFACE.

```
SET OUTMSG = ALL IN HIERARCHY OUTFACE.  
SET OUTDATA = OUTMSG AND DATA.  
SET OUTIN = OUTALPH MINUS INALPH.  
SET OUTIN = OUTIN MINUS OUTDATA.  
LIST OUTIN WITH HIERARCHY DATUM.
```

Doubtlessly, the imaginative user can invent more efficient and powerful procedures to do the same task, as well as others which investigate other aspects of data usage. The flexibility of RADX command capability allows many valid approaches to the same goal.

3.3.5.2 RADX Evaluation of File Activity

In 3.3.1 we discussed the interpretations to be made when a FILE is INPUT TO and/or OUTPUT FROM an ALPHA. RADX commands can be used to isolate various INPUT/OUTPUT combinations for verification. The following are useful.

SET A = FILE WITH INPUT.
SET B = FILE WITH OUTPUT.
SET C = A OR B.
SET D = FILE MINUS C.
SET E = B MINUS A.
SET F = A MINUS B.
SET G = A AND B.

Set D consists of FILES which are neither INPUT TO nor OUTPUT FROM an ALPHA.

Set E consists of FILES which are only OUTPUT FROM some ALPHA. We can assume that additions were made to the FILE for some purpose. The only valid purpose, other than for INPUT TO some ALPHA, is to MAKE an output MESSAGE. The following RADX commands provide a listing of the remainder which do not MAKE an output MESSAGE and which should be checked for errors.

SET H = E MINUS OUTMSG.
LIST H.

Set F consists of FILES which are only INPUT TO some ALPHA. This indicates that the FILE instances are accessed, and possibly deleted, within the ALPHA. Since the FILE is not OUTPUT FROM an ALPHA, it cannot originate within the DPS. Therefore, it must appear in an input MESSAGE. The following RADX commands provide a listing of the remainder which do not MAKE an input MESSAGE and which should be checked for errors.

SET I = F MINUS INMSG.
LIST I.

Set G consists of FILES which are both INPUT TO and OUTPUT FROM some ALPHAs, not necessarily the same ALPHA. These FILES, together with the ALPHAs which operate on them, can be extracted by the following commands.

APPEND FILE INPUT, OUTPUT.

LIST G.

This listing should be examined to ensure that the FILES are operated upon as intended. Particular attention should be paid to ALPHAs which both INPUT and OUTPUT the same FILE. It is these ALPHAs which either modify the instances within a FILE, or add new instances when an appropriate one cannot be found.

3.3.5.3 RADX Evaluation of Entity Activity

It is also useful to verify that ENTITY_CLASSES and ENTITY_TYPES are manipulated appropriately within the system. Each ENTITY_CLASS must be CREATED BY some ALPHA. Each ENTITY_CLASS should be DESTROYED BY some ALPHA. If it is not destroyed, there must be a valid reason for retaining the associated data. Further, each ENTITY_TYPE within an ENTITY_CLASS must be SET BY some ALPHA. The members of the following sets are apparent deviations and should be examined.

SET A = ENTITY_CLASS WITHOUT CREATED.

SET B = ENTITY_CLASS WITHOUT DESTROYED.

SET C = ENTITY_TYPE WITHOUT SET.

The following hierarchy is useful to determine the actions on each ENTITY_CLASS in the system.

HIERARCHY ENTITY_CHECK =
ENTITY_CLASS CREATED BY ALPHA
ENTITY_CLASS COMPOSED ENTITY_TYPE
ENTITY_TYPE SET BY ALPHA
ENTITY_CLASS DESTROYED BY ALPHA.

The following RADX command will provide a structured listing of each instance of the hierarchy suitable for further analysis.

LIST ALL WITH HIERARCHY ENTITY_CHECK.

3.3.5.4 RADX Evaluation of Data Attributes

Data extraction can be of major support in development of the executable description, although it is not a major contributor to assessment of the result. The fact that RADX cannot penetrate the contents of a BETA to

determine its consistency with declarations of relationships and attributes is of little significance, since the consistency and completeness are thoroughly analyzable with the static analyzers, and since the crucial test of simulation generation is then executable.

One of the key operations at this stage of specification development is defining the LOCALITY, USE, and TYPE of all DATA required for functional simulation. LOCALITY is the easiest of the attributes to specify:

- 1) Using the definitions of 3.2, LIST ALL WITH HIERARCHY ENTITY generates a collection of GLOBAL DATA;
- 2) Using those definitions, LIST ALL WITH HIERARCHY INFACE and LIST ALL WITH HIERARCHY OUTFACE generates a collection of LOCAL DATA;
- 3) LIST B WITH HIERARCHY FILES generates the collection of DATA CONTAINED IN FILES which are not linked to higher levels. Since each such file is inherently either local or global in scope, all CONTAINED DATA have the same LOCALITY as their parent; and
- 4) LIST D WITH HIERARCHY DATUM identifies the top level of DATA which are not linked into higher levels. Each such item is then assigned a LOCALITY, which is also the value assigned to any DATA item INCLUDED in that top level.

Since the system provides an override of any LOCALITY declaration for ASSOCIATED DATA or those which MAKE a MESSAGE, any LOCALITY provided by the user would be at least redundant, and at worst misleading. Therefore, it is recommended that each DATA item and FILE generated by (1) and (2) above be checked to ensure that no LOCALITY is declared. Similarly, it is best to declare the LOCALITY for each FILE derived from (3), and then to omit LOCALITY for each DATA item obtained. Finally, each DATA item obtained in (4) is assigned a LOCALITY, and the same LOCALITY is assigned to each DATA item included in it.

Data extraction also supports assigning USE to the DATA items. Define SET A = DATA WITHOUT INCLUDES. Each item in that set must have USE GAMMA or BOTH. It is given USE BOTH exactly if it is to be a part of the BETA model of some ALPHA. For each item with USE:BOTH, no DATA above it in the hierarchy can have a USE assigned. Note that USE:BOTH and USE:GAMMA may be applied only to the lowest level of DATA defined. Once it is determined that a given lowest-level DATA item will not be included in the functional model, an item above it in the hierarchy must be assigned USE:BETA. That

item may be anywhere above the one with USE:GAMMA, except that it cannot also INCLUDE (either directly or through a chain of INCLUDES) any element with USE:BOTH.

By defining a SET A = DATA WITH USE, the engineer determines the items requiring TYPE. It is not mandatory at this stage to TYPE DATA which are to be used only analytically, so that the subset required for functional simulation can be obtained through SET B = A WITHOUT USE = GAMMA. Through examination of the STRUCTURES and BETAs, the TYPE of each such item is defined, and enumerated DATA are also assigned RANGE.

Note that it is at least misleading, and sometimes likely to induce errors, to define attributes for elements which do not require them. Thus, a LOCALITY:LOCAL declaration for an item ASSOCIATED WITH an ENTITY_TYPE would be overridden by REVS, but would be entered into the ASSM and would appear to the reader of the specification to control the DATA item. Such a condition would clearly degrade legibility of the document, and should be avoided. The data extractor can be used to determine if any over-specification of this sort has been attempted. (Note that the ASSM for TLS as documented in the Appendices includes such cases. At the time of its development, the capabilities of the extractor were more limited than at present, and manual methods were employed to complete the data base.)

3.3.6 Summary of Phase 3

This section has outlined a general procedure for completing the definition of the functional requirements for a DPS. The following steps were done:

- The data transactions of each ALPHA were defined.
- The hierarchy transitions performed by each ALPHA, if any, were defined.
- The user has redefined ALPHAs into SUBNETs, has added ALPHAs, and has modified R_NET STRUCTURES if the Phase 2 baseline was inadequate.
- The user has evolved a clear concept of the processing done within each ALPHA, as a starting point for development of BETAs.
- The definition of all necessary DATA and FILE attributes has been completed, to the extent possible without developing BETAs and GAMMAs.

- The completeness and consistency of the ASSM has been analyzed and verified with the aid of RADX capabilities.

We have reached another milestone. The definition of the functional requirements is hypothetically complete. In Phase 4 we will develop the executable simulation needed to verify that assertion.

3.4 PHASE 4 - DEVELOPMENT OF FUNCTIONAL MODELS

Specification of the functional requirements entails development of functional models which define the outputs of processing in terms of its inputs. In essence, the only things known to a data-processor specification are the contents of its interface messages; all other information should be defined within the specification in terms of mathematical operations on the input data stream. Such a definition is in effect a model of the processing operations required, and with sufficient fidelity would provide the analytic models of Section 4. Where the model reflects only the gross characteristics of the transformation required, it may be said to be functional; such models in REVS are termed BETAS. The BETA and GAMMA representations of the DPS must be driven by a system level simulator which SREM assumes to exist, due to the prior need for such a simulator in deriving the originating specifications. The TLS example presumes existence of a System Environment and Threat Simulator (SETS).

3.4.1 Betas

A BETA is a PASCAL procedure which models the transformation of the DATA and FILES INPUT TO an ALPHA into the DATA and FILES OUTPUT FROM it. The function of a BETA is to implement the data flow required for functional simulation in order to determine the sufficiency of a functional specification. To that end, the fidelity of each BETA is dependent on the capabilities required of the functional simulator, and in particular on the data contents determined for SETS.

Generation of a BETA is more or less creative depending on the fidelity desired. At the simplest level, assignment statements may be employed in place of complex mathematical operations. In TLS, the ALPHA: UPDATE_STATE is modeled by assigning to STATE (a DATA item ASSOCIATED WITH ENTITY_TYPE IMAGE_IN_TRACK) the value of the DATA returned by SETS. In an analytic

model, the equivalent operation would be execution of a high-order Kalman filter. The simple model is appropriate since it follows the logical connectivity of the system data flow, while providing a means for SETS to activate the required branches of the R_NETs.

One of the ALPHAs of TLS is REDUN_DETERMINATION. It embodies the requirements for identifying two images as redundant if their state estimates are close enough (relative to their uncertainties) for them to be considered duplicate detections of the same object. The real processing to implement such a requirement would have to select at least a subset of all images then being tracked for state comparison, then to determine redundancy by an appropriate algorithm. For a functional model, it is sufficient to define the USE of STATE (a multielement vector in reality) as BETA and its TYPE as REAL. Then two instances of IMAGE_IN_TRACK would be redundant if their values of STATE were equal. The corresponding SETS activity is implemented by having the DATA returned in the radar message selected to give the required probability of redundancy.

The BETA for REDUN_DETERMINATION is given in Figure 3-14. It scans all instances of IMAGE_IN_TRACK (the ENTITY_TYPE) to find any value of STATE equal to that of the instance to which the current return is related. If redundancy is found, the REDUNDANT_IMAGE DATA is assigned TRUE; otherwise, it is FALSE.

3.4.2 Local Data

In developing the BETA for each ALPHA, the requirements engineer also pins down attributes about the DATA flow. As already noted, he will define some high-level DATA as being sufficient for functional modeling, and will give them corresponding attributes of TYPE and USE. Similarly, he will specify USE:BOTH for DATA required in a functional model at the same level of fidelity as in an analytic model. In addition, he will identify some DATA required for intercommunication of ALPHAs on a single R_NET during a single transaction (enablement) of that net. For example, the STATE of the IMAGE_IN_TRACK corresponding to the current return must be compared in REDUN_DETERMINATION with that of all other instances. Therefore, a DATA element may be defined as OUTPUT FROM ALPHA:UPDATE_STATE which is the CURRENT_STATE. That definition is needed since the current instance of

ALPHA: REDDOR_DETERMINATION.

BETA:

```
"VAR X: INTEGER;  
  BEGIN X:=0;  
    FOR EACH IMAGE_IN_TRACK DO  
      IF STATE=CURRENT_STATE THEN X:=X+1;  
    ENDFOR EACH;  
    SELECT FIRST FROM IMAGE_IN_TRACK SUCH THAT  
      IMAGE_ID=TARGET_ID;  
    REDUNDANT_IMAGE:=(X>1)  
  END;"
```

Figure 3-14 RSL BETA Entry

IMAGE_IN_TRACK (with which a value of STATE is associated) changes during execution of the ALPHA:REDUN_DETERMINATION. Therefore, the DATA:CURRENT_STATE is declared, and given LOCALITY:LOCAL, TYPE:REAL, and USE:BETA. Since the value of the state vector is also an output for recording, and since a MESSAGE uses only LOCAL DATA, the element may be the same as the one which MAKES the MESSAGE:STATE_UPDATE which is also required from that R_NET.

RADX procedures as discussed in 3.3.5 are of continuing use in analyzing the correctness of the DATA definitions. The user is encouraged to develop his own procedures which are meaningful to his particular project.

3.5 TRACEABILITY

Traceability is a feature of the specification supporting its management, rather than one required for its technical quality per se. Therefore, the requirements for traceability are developed in the management volume, and are addressed here only in terms of their mechanical entry.

3.5.1 Originating Requirements

The originating requirements for a software specification are most commonly contained in higher-level specifications. In general (depending on management decision) each identifiable software requirement in each higher-level specification will be called out as an ORIGINATING_REQUIREMENT in the ASSM. The DESCRIPTION attributed to an ORIGINATING_REQUIREMENT may be a literal excerpt of the source, or may be an interpretation, again at management discretion. Note that ORIGINATING_REQUIREMENTS in REVS do not trace to one another; thus, the lowest level of requirements to which a DECISION or specification element traces should be entered as the ORIGINATING_REQUIREMENT.

3.5.2 References

There may be sources of information which define specifics of the software requirements, but which are not specifications in themselves. For example, a table of constants, or a book defining a standard atmosphere model may be referenced in the source specifications or applied by the requirements engineer in developing the software specification. Such a REFERENCE is also recorded in the ASSM, since it EXPLAINS some feature of

the required processing. Note that a REFERENCE may be changed during development of either the specification or the software; when such a change occurs, its impact on the specification may be followed through the EXPLAINS relationship.

3.5.3 Decisions

As was noted in 3.1.5, the process of developing the DPS requirements may expose system issues which cannot be resolved immediately in a formal manner, but which may have significant impact on the direction of further work. Each such issue is recorded in the ASSM as a DECISION which TRACES FROM the ORIGINATING_REQUIREMENT(s) either directly or through other DECISIONs. As each issue is encountered it should be given a DECISION name, and entered in the ASSM with at least its key attributes: a statement of the PROBLEM, the recognized ALTERNATIVES, and the CHOICE among them which was made to allow work to proceed. When the CHOICE is reviewed by system engineering, its correctness can be confirmed, or the implications of revising it can be assessed through analysis of those elements which TRACE FROM the DECISION. Paragraph 3.1.5 gives an example of the input needed to state one of the TLS issues.

3.5.4 Relating to Sources

The relationships TRACES and EXPLAINS have been defined to link requirements to their sources. Each element of the ASSM should be TRACED TO its ORIGINATING_REQUIREMENT(s) either directly or through DECISIONs which may be TRACED themselves. In general, the ORIGINATING_REQUIREMENTS may be entered into the ASSM before their traceability is developed (i.e., before any of the elements have been defined). As the R_NETs are built and as the other elements are entered, their links with any ORIGINATING_REQUIREMENT may be recorded through the TRACES relationship. Whenever a DERIVATION is encountered, it should be entered and TRACED TO its source.

Chronologically, it is likely that the first entries made into the ASSM will be the ORIGINATING_REQUIREMENTS; some REFERENCES may also precede development of the R_NETs. As elements are developed, DECISIONs and additional REFERENCES will be recorded, and at least some TRACES and EXPLAINS relationships will be provided. Before publication of the specification, management may require element audit to confirm some level of

completeness of tracing; throughout development of both the specification and the resulting software, the linking should be monitored to track, in the specification, evolution of requirements. Note that it is good practice in requirements engineering to explore, through the data extractor, the impact of any projected modifications to requirements. Particularly at the terminal, it is convenient to query the ASSM to the effect: What if the DECISION (name) is modified? It would be accomplished LISTING ALL WITH an appropriate HIERARCHY that TRACED TO that decision, then examining the attributes of each element that was so related. Judicious engineering would then focus on the requirements with the least impact when looking at the set which might be altered to reflect a change in the software environment (e.g., the threat).

3.6 INFORMATIVE MATERIAL

Sections 3.1 through 3.4 have shown the development of the technical content of the ASSM from the entry of the kernel through recording functional models. In addition to such mandatory material, there is a family of supportive information which might be recorded for any element. Characteristic of that family is the DESCRIPTION, an attribute which allows an English-language text to be entered (other languages might be used except for restrictions due to character sets) to explain the intent of the element. Note that informative material is not constraining on the process designer, and is not, in fact, a requirement subject to test; it is merely supportive of communication between the requirements engineer both his peers and the process designer. The need for any informative attribute and the auditing of its entry are options for project management, and are discussed in the management volume.

3.6.1 Description

A text string of arbitrary length may be entered to describe any element of the ASSM under the attribute DESCRIPTION. Even where the element carries a self-explanatory name, or where it has another attribute formally specifying it, a DESCRIPTION is usually valuable. For example, the IMAGE_ID is as simple a concept about an IMAGE as can be promoted. Nevertheless, it would be useful to describe it as being assigned from the HO_ID of the initiation message, and as being embodied also as the TARGET_ID in the ENTITY_

CLASS PULSE. The linkage among the elements is defined in the BETAs of appropriate ALPHAs, of course, and the DESCRIPTION is not binding. But following all of the references to that DATA item to find the meaningful assignments is tedious at best, where the DESCRIPTION is easily absorbed.

3.6.2 Synonym

A synonym may be declared and EQUATED TO any other element for the convenience of the engineer. Frequently, the SYNONYM will be a short name for a frequently referenced item, so that at least some of the references are simplified; occasionally, the naming conventions imposed by the language will prompt the use of a cryptic element name, so that an explanatory SYNONYM is desired. The relationship ABBREVIATES may be regarded as equivalent to EQUATES; it is an artifact of early REVS design.

3.6.3 Authorship

In the current REVS, there is an attribute ENTERED_BY which permits the author of information about an element to record his name and the date of entry. If required, that operation could be automated, so that the attribute would become a log of changes made, with the entry provided by the system whenever a value or relationship was altered.

3.6.4 Complementary Relationships

Each RSL relationship defines a connection between two elements; since it is transitive, it has a complement which simply reverses the subject and object. When information is extracted from the ASSM, both directions of the relationship are derived from a single link; thus, both the relationship and its complement are extracted, and redundant but absolutely consistent information is obtained. Since consistency is assured, redundancy is constructive in informing the user about all relationships on an element under examination.

3.6.5 Structural References

In the structure segment, an R_NET makes use of other elements. Implicitly, it has a relationship to each such element which appears on its STRUCTURE; that relationship is termed REFERS, and is implicit in use of the data extractor (RADX) of REVS. The relationship and its complement

are visible to the user in the RADX output, but are entered automatically and implicitly through STRUCTURE declaration, not through the conventional explicit declarations.

3.7 ANALYTIC MODELS

In order to support analytic simulation, detailed models of data transformations must be provided in the ASSM. For each ALPHA, that model is entered as its GAMMA attribute in the form of a PASCAL procedure.

In general, an analytic model will be an idealized method of performing the operations required by the ALPHA. Typically, it will be unrealizable in a real-time system due to its size or complexity. Since the requirements statement is independent of the machine on which its implementation is to execute, real-time constraints do not apply, and the idealized algorithm is suitable for the existence proof that the analytic simulation is to provide.

Another useful way of looking at an analytic model is to begin with a TEST of a PERFORMANCE_REQUIREMENT. In general, such a TEST will be a function of the outputs of processing (MESSAGES and DATA entered into the global data base) relative to DPS inputs (MESSAGES). In a mathematical sense, the TEST might be inverted to provide an algorithm for converting the inputs to the required outputs. If that inversion is partitioned into the ALPHAs along the processing path, it might become the set of GAMMAs needed at this stage.

For example, if the requirement is that a specific differential equation is to be satisfied, the equation itself serves as the TEST (see Section 4), while the GAMMAs for the appropriate ALPHAs provide one means of solving the equations. If we did not require that a set of GAMMAs be generated and validated, we could not confirm the correctness of the specification. Thus, it is simple to specify a perpetual-motion machine (the power available at the output port equals or exceeds that supplied at all input ports), but very difficult to provide a proof of one's existence.

The GAMMAs may, in principle, be generated in exactly the same way as the BETAs. In practice, only the simplest will be treated in so light a manner; the majority of the ALPHAs will require modelling by a team of experts in the particular disciplines required for that processing. For

example, the ALPHA:UPDATE_STATE has a very simple BETA model. Its GAMMA would probably be a high-order Kalman filter requiring months of development. Time and cost limitations have precluded generation of GAMMAs for any ALPHAs of TLS.

4.0 PERFORMANCE REQUIREMENTS

Performance requirements specify how well the functional requirements must be implemented within the real-time process. While the set of functional requirements specifies the meaning and integrity of data to be output by the process, each performance requirement constrains one or more mathematical expressions over a collection of these outputs and internal data. In general, the system specification will establish performance criteria at a level higher than that of the functional criteria; consequently the methodology provides for performance requirements to be specified as constraints on a collection of data produced from a series or combination of functional requirements, thereby indirectly constraining each embodied functional requirement. This provision within the methodology does not preclude the requirements engineer from decomposing the system specification performance criteria into component performance requirements for allocation to separate elements; however, it does support flexibility to enhance design freedom for the process designer.

An example of a simple performance requirement might be a specified accuracy of an output relative to a defined precision of the input from which the output is determined. Similarly, in a real-time process it is often necessary to constrain the interval of time between arrival of a stimulus at an input port and the appearance of a response at an output port. Most often, a performance requirement establishes the criteria for a mathematical expression which must be satisfied when applied to the data in the process, and frequently applies over the course of an engagement rather than at any single point in time. Consequently, performance requirements in a system specification tend to be global in nature and are defined at the subsystem level.

Historically, performance requirements have been stated in the English language and, as a consequence, were subject to interpretation. Frequently, these interpretations were not unique and the meaning and understanding achieved by the developer did not coincide with that intended by the specifier. As an example, in the track loop problem the specification established a constraint on the total energy to be allocated to an image. Within the

Track Loop System construct, there exist three possible applications of this performance requirement depending on the particular interpretation of the developer: if (1) the requirement is applied at the output of the allocator, allowance for scheduling conflicts or for radar subsystem performance is not included; if (2) the requirement is applied at the output to the radar subsystem, provisions for scheduling conflicts would be included; however, if (3) the requirement is applied at the point where radar returns are input and processed, then scheduling conflicts and radar preemptions are accounted for and, in addition, the non-DPS term which corresponds to radar processing between transmission of the signal and receipt of the return at the processor interface is also included.

In an effort to eliminate ambiguities in the statement of performance requirements to the process designer, SREM employs a machine-readable language (RSL) with clearly defined elements, structures, relationships and attributes. With RSL, the requirements engineer can specifically and discretely specify performance requirements to the process designer¹. Within the validation segment of REVS, the requirements engineer states the performance constraint as a procedure which operates on data collected at specified points, VALIDATION_POINTS, along the R_NET STRUCTURE. For each constraint an unambiguous pass/fail decision is determined for the explicit TEST which the requirements engineer formulates and asserts is sufficient to declare that the PERFORMANCE_REQUIREMENT is satisfied.

In the track loop example mentioned above, where the test to assure satisfaction of the performance constraint is applied at the output to the radar subsystem, the collection of data to be operated on by the test procedure includes those data which make the outgoing radar message and the remembered copy of the data from which the outgoing message was formed and which is ASSOCIATED WITH the ENTITY_CLASS PULSE. By locating the point for data measurement (VALIDATION_POINT) at the output to the radar subsystem, the specific DATA and FILES required by the test procedure are accessed and retrieved such that the particular test can be exercised, thereby eliminating ambiguity in the performance requirement statement. Inclusion of the VALIDATION_PATH descriptors for connectivity and traceability to the ORIGINATING_REQUIREMENT and any DECISIONS involved in

interpretations of the performance constraint completes the definition and clarifies the requirement.

Determination of consistency between the derived performance requirements and the performance specifications depends heavily on simulation as does determination of consistency between the functional requirements and the functional specifications. Performance requirements include two ranges of constraints - those local to the process and those of a global nature which apply to the system as a whole; functional requirements are local to the process. Performance requirements, taken as a whole, can be said to be consistent (both with one another and with the laws of nature) when there is an existence proof of a simultaneous solution to both the local and global requirements. Therefore, to demonstrate that the performance specification and the derived performance requirements are achievable in some process, it is sufficient to find a set of algorithms which, when executed as specified by the R_NET STRUCTURE and the data connectivity, satisfies the collection of performance requirements. The candidate algorithms mechanized for this execution are written as PASCAL procedures and are attributed to the ALPHAs which each analytically models. These PASCAL procedures (GAMMA models) are linked together by the REVS simulation generation process to form an analytical simulator in exactly the same sense that the functional models (BETA models) are linked to form a functional simulator. To the user, the principal difference between the functional and performance requirements simulators is that the performance results from analytic simulation are assessed against the requirements defined in each PERFORMANCE_REQUIREMENT TEST to determine sufficiency of the solution.

The precision of the methodology developed for specifying functional requirements is not readily available in specifying performance requirements due primarily to the local and global range inherent in the performance specification. That is, although the capabilities within the methodology provide for an explicit and precise statement of each performance requirement, translation from the English representation of the specifications in the source material to the particular PERFORMANCE_REQUIREMENT depends heavily on the judgement of the requirements engineer and consequently can be treated as "methodical" only in a limited sense. Therefore, the following sections, describing the approach to developing performance requirements,

will tend to be more illustrative than definitive and represent the thought process to be conducted by the requirements engineer rather than specific rules to be followed.

The following steps outline the guidelines developed in subsequent sections to specify performance requirements.

- Step 1. The system specification is analyzed and each performance requirement is identified, named and entered into the ASSM data base. This activity is continued until all PERFORMANCE REQUIREMENTS have been inserted along with the ORIGINATING REQUIREMENT or DECISION that each PERFORMANCE_REQUIREMENT is TRACED_FROM.
- Step 2. Each PERFORMANCE_REQUIREMENT is considered in conjunction with the R_NET or SUBNET STRUCTURES to identify the point along the net at which the test for satisfaction of the requirement most appropriately applies. This point establishes the location of the VALIDATION-POINT that determines the termination of the VALIDATION_PATH that is to be CONSTRAINED BY the PERFORMANCE_REQUIREMENT. (It should be noted that in certain cases a single VALIDATION_POINT may be all that is required to implement a TEST of the PERFORMANCE_REQUIREMENT.) This activity is continued until a point of application for the TEST for each PERFORMANCE_REQUIREMENT has been located along the R_NET or SUBNET STRUCTURES and a VALIDATION_POINT has been named and entered into the ASSM data base as an element-type and located on the STRUCTURE.
- Step 3. An initial TEST which satisfies each PERFORMANCE_REQUIREMENT is formulated and analyzed based on data at the VALIDATION_POINT. In general, analysis of the initial TEST formulation will result in the determination that additional data, not available at the single VALIDATION_POINT, will be required to support the TEST. Consequently, additional VALIDATION_POINTS are identified, named and appropriately located on the R_NET and SUBNET STRUCTURES within the ASSM. This activity is continued until the necessary VALIDATION_POINTS have been defined such that the collection of data required for each TEST has been made available from the DATA and FILES accessible on the nets. At this time, DATA and FILES required from each VALIDATION_POINT are declared to be INPUT TO the respective VALIDATION_POINT and entered into the ASSM. The VALIDATION_POINTS are then considered collectively and VALIDATION_PATHS are then named.
- Step 4. An initial PATH structure for each VALIDATION_PATH is entered into the ASSM considering the VALIDATION_POINTS and EVENTS required for each initial TEST formulation. This activity is continued until a PATH structure has been declared for each VALIDATION_PATH constrained by each PERFORMANCE_REQUIREMENT.

- Step 5. The requirements engineer is now in a position to finalize each TEST formulation and refine the initial declaration of VALIDATION_POINTS and VALIDATION_PATHs for each PERFORMANCE_REQUIREMENT TEST. Each TEST is coded as a PASCAL procedure and entered into the ASSM as an attribute of the respective PERFORMANCE_REQUIREMENT.
- Step 6. The requirements engineer now identifies the port-to-port response times required by the system specification. For each performance specification a PERFORMANCE_REQUIREMENT TEST is defined including the attendant VALIDATION_POINTS and their RECORDS relationships, and the constrained VALIDATION_PATH with PATH structure and with MAXIMUM_TIME, MINIMUM_TIME and UNITS attributes in the same manner that other PERFORMANCE_REQUIREMENT statements are developed. These descriptors are entered into the ASSM to complete the initial definition of the performance requirements statement.

At this point the requirements engineer has completed the initial activities of SREM necessary to explicitly and unambiguously state PERFORMANCE_REQUIREMENTS. The remaining activities involve confirmation that the statements are consistent and complete. These activities employ the static and dynamic checking features of REVS and the execution of the analytical simulation built by REVS.

4.1 LOCATE TEST POINTS

A PERFORMANCE_REQUIREMENT is stated relative to data collected at specific VALIDATION_POINTS located on the R_NET and SUBNET STRUCTURES. Even in simple cases, identification of the data-collection points is critical to the interpretation of a system performance constraint. The content and context of the specification in the source material is used by the requirements engineer to determine the system engineer's intent and from which the test point is defined.

As an example, consider the specification statement in the TLS DPSPR (Appendix F) which requires that "The DPS shall allocate radar commands so that not more than (TBD) joules are commanded per image, . . ." At least three interpretations can be associated with this apparently explicit requirement.

- 1) The allocator shall assign track rates such that the accumulated sum of the energy for each image over the engagement (the product of the allocated pulse rate, the energy per pulse and the duration of the allocation) shall not exceed the specified limits;

- 2) The energy required by the radar commands transmitted across the interface to the radar shall not exceed the specified limit; or
- 3) The energy required by the radar commands acted upon by the radar, as detected by the DPS in the radar return messages, shall not exceed the specified limit.

Since not all of the pulses allocated per image may be scheduled and since some of the scheduled pulses may be pre-empted by the radar, the three interpretations lead to different performance requirement definitions. The first point in the methodology at which the interpretation becomes important is during the process of locating the VALIDATION_POINT on the R_NET or SUBNET at which the TEST for compliance with the system specification is to be made. For the first interpretation, the test point should be located at the output of the allocator; for the second interpretation, the test point should be located at the input to the OUTPUT_INTERFACE RADAR_OUT; for the third interpretation, the test point should be located at the output of the INPUT_INTERFACE RADAR_IN. In the Track Loop example, development of the PERFORMANCE_REQUIREMENT was based on the second interpretation and is TRACED FROM a DECISION which delineated the development process, through use of the DECISION attributes, the PROBLEM, the ALTERNATIVES and the CHOICE.

The VALIDATION_POINT at which the final data is collected on which the PERFORMANCE_REQUIREMENT TEST operates in most cases is associated with the point in processing along the path at which the test is relevant. In general, selection of the appropriate R_NET or SUBNET is critical to the requirements definition; however, the particular location along the net may offer some flexibility depending on the type of data involved. Since, in REVS, execution of a net occurs in zero time, the selected location may be any point at which the required data are available. As an example, the energy per image constraint defined by the third interpretation could be applied at a point on the RESPONSE_TO_RADAR R_NET either preceding or following the ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.

At this point in the methodology, the ASSM should contain the following performance related RSL statements, described in the order in which they would generally be entered.

- 1) Each ORIGINATING_REQUIREMENT will have been defined.
- 2) Each PERFORMANCE_REQUIREMENT will have been identified including traceability to the ORIGINATING_REQUIREMENT.
- 3) Each DECISION involved in deriving each PERFORMANCE_REQUIREMENT based on interpretation of the ORIGINATING_REQUIREMENT will have been defined. Explicit declaration of the DECISION attributes PROBLEM, ALTERNATIVES and CHOICE will be included.
- 4) The VALIDATION_POINT at which each PERFORMANCE_REQUIREMENT TEST is to be applied will have been identified and uniquely named.
- 5) The appropriate R-NET and SUBNET STRUCTURES will have been updated to include the location of each VALIDATION_POINT node.

The RSL statements and R_NET STRUCTURE are provided in Figure 4-1 for the energy-per-image constraint.

4.2 DEFINE DATA AND TESTS

A VALIDATION_POINT is precisely a port through which simple DATA are accessed and through which DATA and FILES ASSOCIATED WITH ENTITY_CLASSES and ENTITY_TYPES are extracted, all of which is RECORDED for post-processing by the PERFORMANCE_REQUIREMENT TEST in the REVS validation segment. These DATA are obtained during simulation from the global data base defined by the functional requirements specifications. Simple DATA (not in an entity, file, or message hierarchy) which must be recorded at a VALIDATION_POINT are related to that point via RECORDS. Message DATA are similarly RECORDED BY a named VALIDATION_POINT on the R_NET to which they are LOCAL. If an entire FILE is to be recorded, the relationship RECORDS may be applied to it as well.

For DATA which are CONTAINED IN a FILE, or are ASSOCIATED WITH an ENTITY_CLASS or ENTITY_TYPE, a problem may arise in defining the instance of the repeated data for which the DATA are to be RECORDED. In general, the instance desired will have been selected earlier on the R_NET. For example, the PULSE which is relevant for the PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE is the one which was CREATED BY the ALPHA: PICK_COMMAND. Since no intervening activity could modify the selection, the DATA corresponding to that PULSE are available to the VALIDATION_POINT used for the

ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_R.

DESCRIPTION:

(DPSPR PARAGRAPH 3.2.4(B), RESOURCE CONTROL, STATES THAT ("THE DPS SHALL ALLOCATE RADAR COMMANDS SO THAT NOT MORE THAN (TRD) JOULES ARE COMMANDED PER IMAGE, NOR MORE THAN (TRD) KILOWATTS OR (TRD) PULSES/SECOND FOR ALL IMAGES IN TRACK.")[.

PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.

TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_B.

PERFORMANCE_REQUIREMENT: PULSES_PER_SECOND.

TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_B.

PERFORMANCE_REQUIREMENT: RADIATED_POWER.

TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_B.

DECISION: RADAR_RESOURCE_CONTROL_B1.

PROBLEM:

(DPSPR PARAGRAPH 3.2.4(B), STATEMENT ("THE DPS SHALL ALLOCATE RADAR COMMANDS SO THAT NOT MORE THAN (TRD) JOULES ARE COMMANDED PER IMAGE,...") ALLOWS FOR THREE POSSIBLE INTERPRETATIONS IN DETERMINING THE POINT AT WHICH THE PERFORMANCE_REQUIREMENT TEST IS APPLIED.[.

ALTERNATIVES:

- [1. THE ALLOCATOR SHALL ASSIGN TRACK RATES SUCH THAT THE CUMULATIVE SUM OF THE ENERGY FOR EACH IMAGE OVER THE ENGAGEMENT (THE PRODUCT OF ALLOCATED PULSE RATE, ENERGY PER PULSE, AND DURATION OF ALLOCATION) SHALL NOT EXCEED (TRD) JOULES.
2. THE ENERGY REQUIRED BY THE RADAR COMMANDS TRANSMITTED ACROSS THE INTERFACE TO THE RADAR SHALL NOT EXCEED (TRD) JOULES.
3. THE ENERGY REQUIRED BY THE RADAR COMMANDS ACTED UPON BY THE RADAR AS DETECTED BY THE DPS IN THE RETURN MESSAGES, SHALL NOT EXCEED (TRD) JOULES.[.

CHOICE:

- [2. THE ENERGY REQUIRED BY THE RADAR COMMANDS TRANSMITTED ACROSS THE INTERFACE TO THE RADAR SHALL NOT EXCEED (TRD) JOULES SHALL BE TESTED FOR COMPLIANCE AT THE INPUT TO THE OUTPUT INTERFACE: RADAR_OUT.[.

TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_B.

TRACES TO: PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.

VALIDATION_POINT: RADAR_COMMAND_OUTPUT_POINT.

TRACED FROM: DECISION: RADAR_RESOURCE_CONTROL_R1.

Figure 4-1 Performance Requirements Statements Representation
at the Completion of SREM Step - Locate Test Points

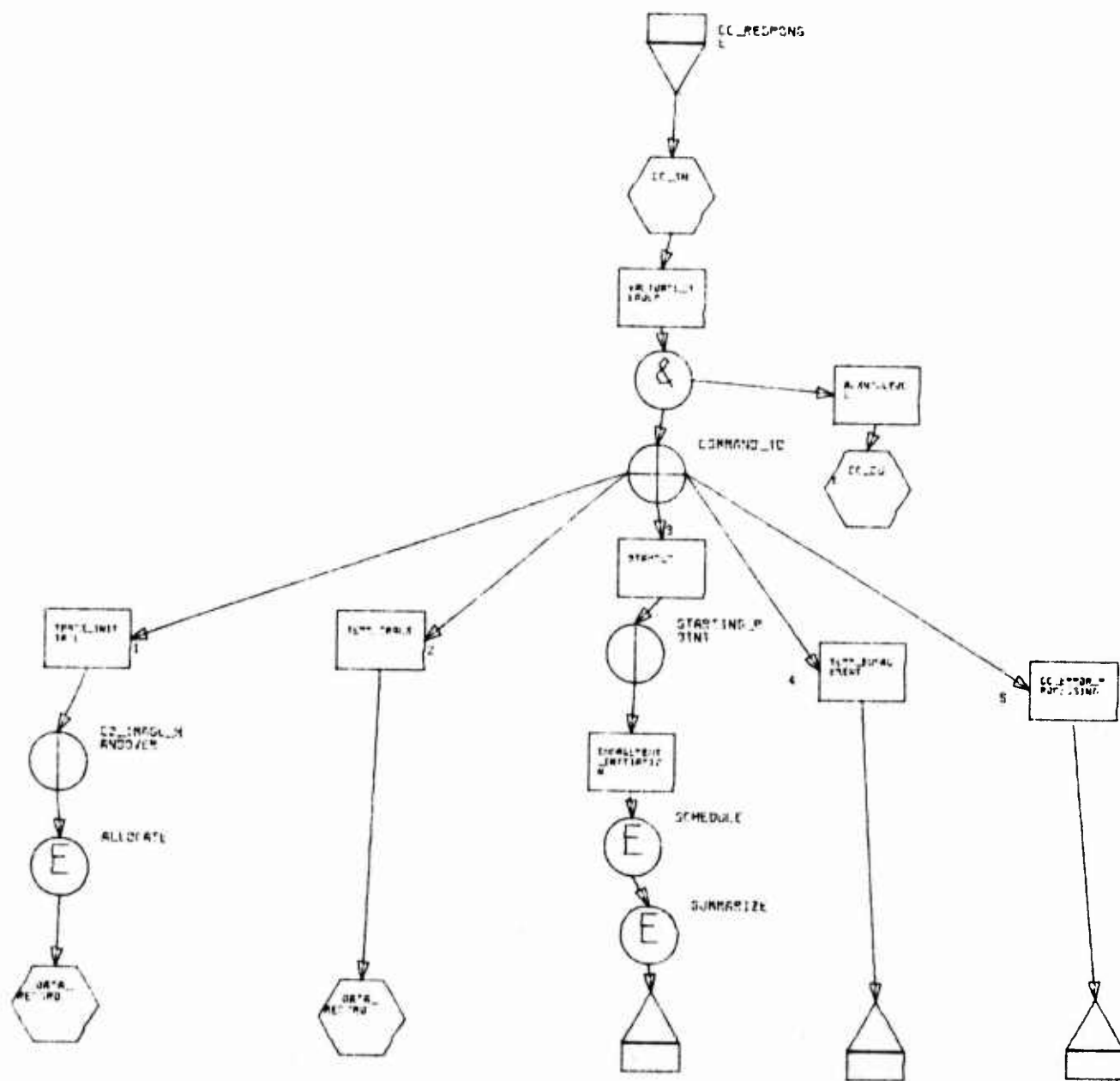


Figure 4-1 Performance Requirements Statements Representation
at the Completion of SREM Step - Locate Test Points
(Continued)

requirement. However, the energy of that command, which is also needed for the TEST, is not so simply available.

The energy of the pulse is a part of the WF_CHARACTERISTICS in the FILE: WAVEFORM_TABLE. The link between the PULSE and the WAVEFORM_TABLE is the identifier of the kind of transmission, the PULSE_TYPE and WF_NAME, respectively. (Similarly, the waveform identifier is the RADAR_TYPE of the outgoing command MESSAGE.) There are three obvious ways of obtaining the required information.

- 1) RECORD either the PULSE_TYPE or the RADAR_TYPE from XMIT_R; RECORD the WAVEFORM_TABLE once (say after ALPHA: STARTER, where it was created); co-relate the information in the TEST.
- 2) Provide an ALPHA with ARTIFICIALITY: VALIDATION before the VALIDATION_POINT to select the WF_NAME; assign the WF_CHARACTERISTICS in that ALPHA to a new LOCAL DATA item; and RECORD that DATA item.
- 3) Provide a GLOBAL DATA item at some earlier processing point with ARTIFICIALITY: VALIDATION and pass it along appropriately to the VALIDATION_POINT where it is to be RECORDED.

In TLS, the third choice is attractive because there is a DATA item left from early thinking on the problem which has the required properties. The DATA: COMMAND_ENERGY was once thought to be useful on R_NET: XMIT_R. Consequently, it was included in each record of the FILE: COMMAND. But development of the models made it clear that the element was not needed for XMIT_R, so it might be deleted. However, it has exactly the right properties for an ARTIFICIAL element, since its value is the energy of the COMMAND which would be selected by PICK_COMMAND for this transmission.

The choice among the options available is up to the requirements engineer, and the process designer should recognize that there are degrees of freedom in the selection which can be reexamined if the design would benefit. For TLS, the first option is particularly attractive: the FILE to be recorded contains only constants, so that correlation of data in the TEST poses no problem. The third option may also be acceptable, since it ensures that the DATA RECORDED for the PERFORMANCE_REQUIREMENT are clearly those which are needed. The second option, which modifies the R_NET with an ARTIFICIAL element is esthetically less desirable than either of the other choices. For the purposes of this document, the first option is

selected, and an additional VALIDATION_POINT: STARTING_POINT is identified immediately following the ALPHA: STARTER to RECORD the FILE: WAVEFORM_TABLE.

A TEST is a PASCAL procedure attributed to a PERFORMANCE_REQUIREMENT which CONSTRAINS one or more VALIDATION_PATHs. The TEST executes, in a post-processing environment, on those DATA which have been RECORDED BY one or more VALIDATION_POINTS required to collect the DATA to be tested and which appear as nodes on the PATH structure of a VALIDATION_PATH that is CONSTRAINED BY the PERFORMANCE_REQUIREMENT. Recall that data required for the TEST are RECORDED on an output data set by means of the VALIDATION_POINT relationships and attributes and each data record is labeled in the output data set by the VALIDATION_POINT name. During post-process TEST execution, the TEST must access the appropriate data record in the output data set and does so with use of the RETRIEVE operator.

The requirements engineer may choose to define the VALIDATION_PATHs, the PERFORMANCE_REQUIREMENTS and the TESTs concurrently; however, in doing so he minimizes the effectiveness of REVS capabilities which support this activity through static checking of the RSL statements. It is recognized that the requirements engineer must have a concept of the TEST configuration at the time the VALIDATION_POINTS and VALIDATION_PATHs are defined. Keep in mind, however, that it is mandatory that the TEST be configured based on the data available on the R_NET and SUBNET STRUCTURES. It is only after all attempts have failed to produce a valid TEST configuration, under these constraints, that the requirements engineer should redefine the R_NET and SUBNET STRUCTURES or introduce STRUCTURES and element types with attribute ARTIFICIALITY in order to accomplish the TEST definition. Consequently, it is recommended that the requirements engineer approach the PERFORMANCE_REQUIREMENTS definition in a top-down, step-wise manner. That is,

- 1) Examine each PERFORMANCE_REQUIREMENT to determine the functional requirement, defined by the R_NET and SUBNET STRUCTURES, to which the PERFORMANCE_REQUIREMENT applies.
- 2) Examine the DATA available along the R_NET and SUBNET STRUCTURES to a) determine a point of optimal location (the terminal VALIDATION_POINT) at which a TEST should be performed and b) provide insight into formulation of the TEST.
- 3) Declare the necessary DATA and FILES at this point as RECORDED BY the terminal VALIDATION_POINT and formulate the initial TEST.

- 4) Define and locate additional VALIDATION POINTs based on data requirements for the TEST formulation and declare the necessary DATA and FILEs as RECORDED BY these additional VALIDATION POINTs.
- 5) Define and locate the initial VALIDATION POINT, that is, the earliest occurring VALIDATION POINT appearing along the R_NET and SUBNET processing flow that defines the functional requirement to which the PERFORMANCE_REQUIREMENT applies.
- 6) Name the VALIDATION_PATHs of the nets.
- 7) Relate the VALIDATION_PATHs to the PERFORMANCE_REQUIREMENT which CONSTRAINS them.

Application of these steps to the definition of PERFORMANCE_REQUIREMENT statements yields an orderly development process and provides maximum utility of the features provided in the REVS RADX and VALIDATION segments. Exceptions to this step-wise development process may occur when a single PERFORMANCE_REQUIREMENT applies to more than one functional requirement defined by R_NETs and SUBNETs that are not connected by EVENT enablements. Under these circumstances, the general solution is effected by allowing the PERFORMANCE_REQUIREMENT to constrain multiple VALIDATION_PATHs. Alternatively, the requirements engineer may choose to decompose the single PERFORMANCE_REQUIREMENT into multiple requirements which apply explicitly to the functional requirements defined by each non-connective R_NET and SUBNET STRUCTURE.

In mechanically applying the step-wise process, the requirements engineer would read the system specification and enumerate the statements of performance as ORIGINATING_REQUIREMENTS. Each requirement is then located on the R_NETs and SUBNETs as one or more VALIDATION POINTs. Each PERFORMANCE_REQUIREMENT and VALIDATION_PATH is then named. A PERFORMANCE_REQUIREMENT is logically named for its content, such as ENERGY_PER_IMAGE which is used for the track loop example discussed previously. Similarly, where the second interpretation of the ORIGINATING_REQUIREMENT was implemented, the VALIDATION_PATH is named RADAR_COMMAND_OUTPUT which signifies both the location of the test and the data to which the test applies. In this particular case the PERFORMANCE_REQUIREMENT CONSTRAINS a degenerate VALIDATION_PATH (i.e., only one VALIDATION POINT is required); consequently, the VALIDATION_PATH declaration serves only to establish continuity within the RSL statements. The single VALIDATION POINT is appropriately named RADAR_COMMAND_OUTPUT_POINT.

The requirements engineer next considers each requirement in the specification separately, and identifies for each VALIDATION_POINT the information which must be extracted to support the TEST for compliance. Note that a single VALIDATION_POINT may be used to collect data for multiple PERFORMANCE_REQUIREMENTS; consequently, it may appear in the PATH structures of several VALIDATION_PATHS. Therefore, the DATA and FILES declared as RECORDED BY a VALIDATION_POINT will be the logical OR of the data to be collected for each of the PERFORMANCE_REQUIREMENTS that use the VALIDATION_POINT. Subsequently, each specified requirement is translated into a PASCAL procedure which is written as the TEST for that PERFORMANCE_REQUIREMENT.

At this point in the methodology for developing performance requirements, the ASSM data base should contain the following additional performance-related RSL statements, described in the order in which they would generally be entered.

- 1) The DATA and FILES accessible from those appearing on the nets will have been declared as input to each terminal VALIDATION_POINT through use of the RECORDS relationship.
- 2) Additional VALIDATION_POINTS required by each PERFORMANCE_REQUIREMENT will have been defined and appropriately located as nodes on the net STRUCTURES. DATA and FILES RECORDED BY these VALIDATION_POINTS will have been declared.
- 3) Each VALIDATION_PATH CONSTRAINED BY each PERFORMANCE_REQUIREMENT will have been defined and a PATH structure for each VALIDATION_PATH will have been defined by declaration of the VALIDATION_POINTS and EVENTS appearing on the net STRUCTURES between the initial and terminal VALIDATION_POINTS.
- 4) A TEST will have been written for each PERFORMANCE_REQUIREMENT.

An example of the RSL statements in the ASSM at this point is provided in Figure 4-2 for the energy-per-image constraint discussed in preceding sections.

4.3 DEFINE SUPPLEMENTAL VALIDATION POINTS AND DATA

A VALIDATION_POINT not only defines a point in the processing flow at which data are collected for evaluation against a PERFORMANCE_REQUIREMENT, but also may identify the point along the nets at which the measurement is made for compliance with the system performance specification.

ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_R.

DESCRIPTION:

"DPSR PARAGRAPH 3.2.4(B), RESOURCE CONTROL, STATES THAT
("THE DPS SHALL ALLOCATE RADAR COMMANDS SO THAT NOT MORE
THAN (TBD) JOULES ARE COMMANDED PER IMAGE, NOR MORE THAN
(TBD) KILOWATTS OR (TBD) PULSES/SECOND FOR ALL IMAGES IN
TRACK.")".

PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.

TEST: (* THE TBD BELOW MUST BE REPLACED BEFORE EXECUTION*)

"CONST

ENERGY_LIMIT=(TBD);

VAR

IMAGE_ENERGY: REAL;

BEGIN

ENERGY_PER_IMAGE:=TRUE;

FOR EACH C2_IMAGE_HANOVER RECORDING

DO

IMAGE_ENERGY:=0.0

FOR EACH RADAR_COMMAND_OUTPUT_POINT RECORDING

SUCH THAT (RADAR_COMMAND_OUTPUT_POINT.TARGET_ID=

C2_IMAGE_HANOVER.HO_ID)

DO

SELECT FIRST FROM STARTING_POINT.WAVEFORM_TABLE

SUCH THAT (RADAR_COMMAND_OUTPUT_POINT.RADAR_TYPE=

STARTING_POINT.WF_NAME);

IF FOUND THEN

IMAGE_ENERGY:=IMAGE_ENERGY+

STARTING_POINT.WF_CHARACTERISTICS;

END;

ENDFOREACH;

IF (IMAGE_ENERGY>ENERGY_LIMIT) THEN

ENERGY_PER_IMAGE:=FALSE;

ENDFOREACH

END;"

TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_R.

PERFORMANCE_REQUIREMENT: PULSES_PER_SECOND.

TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_R.

PERFORMANCE_REQUIREMENT: RADIATED_POWER.

TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_R.

Figure 4-2 Performance Requirements Statements Representation at
the Completion of SREM Step - Define Data and Tests

DECISION: RADAR_RESOURCE_CONTROL_B1.

PROBLEM:

"DPSPR PARAGRAPH 3.2.4(B), STATEMENT ("THE DPS SHALL ALLOCATE RADAR COMMANDS SO THAT NOT MORE THAN (TRD) JOULES ARE COMMANDED PER IMAGE,...") ALLOWS FOR THREE POSSIBLE INTERPRETATIONS IN DETERMINING THE POINT AT WHICH THE PERFORMANCE_REQUIREMENT TEST IS APPLIED."

ALTERNATIVES:

- "1. THE ALLOCATOR SHALL ASSIGN TRACK RATES SUCH THAT THE CUMULATIVE SUM OF THE ENERGY FOR EACH IMAGE OVER THE ENGAGEMENT (THE PRODUCT OF ALLOCATED PULSE RATE, ENERGY PER PULSE, AND DURATION OF ALLOCATION) SHALL NOT EXCEED (TRD) JOULES.
2. THE ENERGY REQUIRED BY THE RADAR COMMANDS TRANSMITTED ACROSS THE INTERFACE TO THE RADAR SHALL NOT EXCEED (TRD) JOULES.
3. THE ENERGY REQUIRED BY THE RADAR COMMANDS ACTED UPON BY THE RADAR AS DETECTED BY THE DPS IN THE RETURN MESSAGES, SHALL NOT EXCEED (TRD) JOULES."

CHOICE:

- "2. THE ENERGY REQUIRED BY THE RADAR COMMANDS TRANSMITTED ACROSS THE INTERFACE TO THE RADAR SHALL NOT EXCEED (TRD) JOULES SHALL BE TESTED FOR COMPLIANCE AT THE INPUT TO THE OUTPUT INTERFACE: RADAR_OUT."

TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_B.
TRACES TO: PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.

VALIDATION_POINT: RADAR_COMMAND_OUTPUT_POINT.

RECORDS:

DATA: TARGET_ID,
DATA: RADAR_TYPE.

TRACED FROM: DECISION: RADAR_RESOURCE_CONTROL_B1.

VALIDATION_POINT: C2_IMAGE_HANOVER.

RECORDS: DATA: HO_ID.

TRACED FROM:

ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_B.
PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.

VALIDATION_POINT: STARTING_POINT.

RECORDS: FILE: WAVEFORM_TABLE.

TRACED FROM: PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.

Figure 4-2 Performance Requirements Statements Representation at the Completion of SREM Step - Define Data and Tests (Continued)

VALIDATION_PATH: RADAR_COMMAND_OUTPUT.
CONSTRAINED BY: PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.
TRACED FROM:
 ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_B,
 DECISION: RADAR_RESOURCE_CONTROL_R1.
PATH:
 VALIDATION_POINT: RADAR_COMMAND_OUTPUT_POINT
 END.

VALIDATION_PATH: C2_HANOVER_COMMAND_INPUT.
CONSTRAINED BY: PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.
TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_R.
PATH:
 VALIDATION_POINT: C2_IMAGE_HANOVER
 END.

VALIDATION_PATH: RADAR_WAVEFORM_PROPERTIES.
CONSTRAINED BY: PERFORMANCE_REQUIREMENT: ENERGY_PER_IMAGE.
TRACED FROM: ORIGINATING_REQUIREMENT: RADAR_RESOURCE_CONTROL_B.
PATH:
 VALIDATION_POINT: STARTING_POINT
 END.

Figure 4-2 Performance Requirements Statements Representation at
the Completion of SREM Step - Define Data and Tests
(Continued)

Through identification of the initial VALIDATION_POINT on a VALIDATION_PATH (i.e., the earliest VALIDATION_POINT along the R-NET and SUBNET STRUCTURES representing the functional requirement to which the performance constraint applies), the requirements engineer may define the PERFORMANCE_REQUIREMENT in terms of performance between the initial and terminal VALIDATION_POINTS. Processing of synchronous threads (i.e., elementary, stimulus-response processing) will frequently employ such paths, as will requirements which reflect the decomposition and partitioning of system performance specifications to a level below that where the constraint is defined to be applicable to the DPS treated as a "black-box." For example, in a complete BMD system, constraints on discrimination may be specified in terms of target classification relative to the object state vector obtained at the output of the tracking function. In such cases, a VALIDATION_POINT would be located at the entry point of the net which defines the functional requirements for discrimination to identify the starting point of the PERFORMANCE_REQUIREMENT TEST.

Early identification of a specific TEST for a PERFORMANCE_REQUIREMENT within the development process implies a capability to define, early in development, data needed for testing that has no counterpart in the functional requirement description of the system. On a synchronous thread, these data are commonly collected at an early VALIDATION_POINT; however, on an asynchronous thread, it may be inconvenient to collect and correlate data from two VALIDATION_POINTS appearing on disjoint paths or net structures. Therefore, the requirements engineer defines DATA items, element types and in extreme cases R-NET or SUBNET STRUCTURES with attributes ARTIFICIALITY: VALIDATION to convey the needed DATA to a VALIDATION_POINT at which it may be extracted for a TEST. For example, in Track Loop suppose that a performance constraint establishes a maximum delay time between the arrival of a track return message at the INPUT_INTERFACE RADAR_IN and the time at which the data contained in the return message is incorporated in the data which makes the command message that passes to the radar through the OUTPUT_INTERFACE RADAR_OUT. The connection between the two interfaces is asynchronous due to scheduling operations and use of the IMAGE_IN_TRACK ENTITY_TYPE in R-NET SKED_R. Since no one-to-one correlation exists between the return received and the command issued, a validation DATA

item, RETURN_TIME, can be defined and ASSOCIATED WITH the ENTITY_TYPE IMAGE_IN_TRACK. This DATA item would be OUTPUT FROM the UPDATE_STATE ALPHA, where it would be set equal to the current value of engagement time, and would be RECORDED BY a VALIDATION_POINT located on the R_NET XMIT_R immediately preceding the OUTPUT_INTERFACE RADAR_OUT.

Note that information conveyed by an element type with ARTIFICIALITY: VALIDATION must be provided in the real-time process when it is being used to validate the process design against the system performance specification. In practice, element types so defined represent a counterpart to the hardware test point, and like their equivalent may be retained in the fielded system by management directive.

At this point the requirements engineer has completed the steps of the methodology for developing performance requirements. The ASSM data base has been built with liberal use of the TRANSLATOR and RADX segments of REVS to insure the accuracy and completeness of each PERFORMANCE_REQUIREMENT description. The requirements engineer is now in a position to begin the compilation debug process and execution of the analytic simulator for verification and refinement of the PERFORMANCE_REQUIREMENT statements.

PART II - MANAGEMENT APPROACH

5.0 INTRODUCTION

One of the major benefits in using the SREM tools and techniques discussed in Part I to develop software requirements is that the process is inherently manageable. The technical approach consists of specific activities which have well defined beginnings and endings, and the high degree of automation provided by REVS allows a degree of management visibility which is ordinarily not attainable in the specification development process.

The three sections which follow discuss the three important aspects of managing software requirements engineering:

- Defining Measurable Milestones
- Planning
- Management Control.

The emphasis in these sections is to describe the management considerations which are unique to the application of SREM. Just as Part I will not make good requirements engineers out of technicians, Part II will not make good managers out of clerks. Even good managers, however, are helpless if they do not have the means to establish visibility and control. Part II is intended to give the experienced manager an understanding of how the tools and techniques of SREM can be used to establish and maintain a sound management plan for the specification of software requirements.

6.0 DEFINING MEASURABLE MILESTONES

The key to successful management of software requirements engineering is the establishment of meaningful, clear, and measurable milestones. There is a tendency to treat requirements generation as a level of effort problem, since the job of milestone definition is not easy and requires a technical understanding of the work to be performed. Establishing milestones such as "Processing Performance Specification - First Draft", "Processing Performance Specification - Second Draft", etc., may provide clarity and a superficial measurability but does not establish a meaningful or effective management tool.

Management of an activity using the Software Requirements Engineering Methodology (SREM) can be extremely effective because the discipline inherent in SREM permits segmenting the effort into activities which have meaningful and measurable terminations. An example approach to defining SREM related milestones is illustrated in this section. The milestones are divided into two groups:

Group 1 - Functional software requirements development.

Group 2 - Functional software requirements validation.

An overview of the SREM activities described in Section 3 is shown in Figure 6-1. Simply, software requirements are generated by transforming corresponding system requirements. The initial activity is a preliminary evaluation and organization of the source specifications. This is accomplished by drawing the R-Nets. Once this activity is complete, parallel development of the segmented requirements can proceed. As the requirements segments are developed, they are entered into the REVS data base (ASSM) from which they are drawn for static evaluation. When the ASSM entry for all software requirements is complete, a dynamic simulation (functional and/or analytic) can be generated and a dynamic evaluation performed to complete the requirements validation. At various points in this process, problems with the source specifications can be identified and fed back to the system engineer.

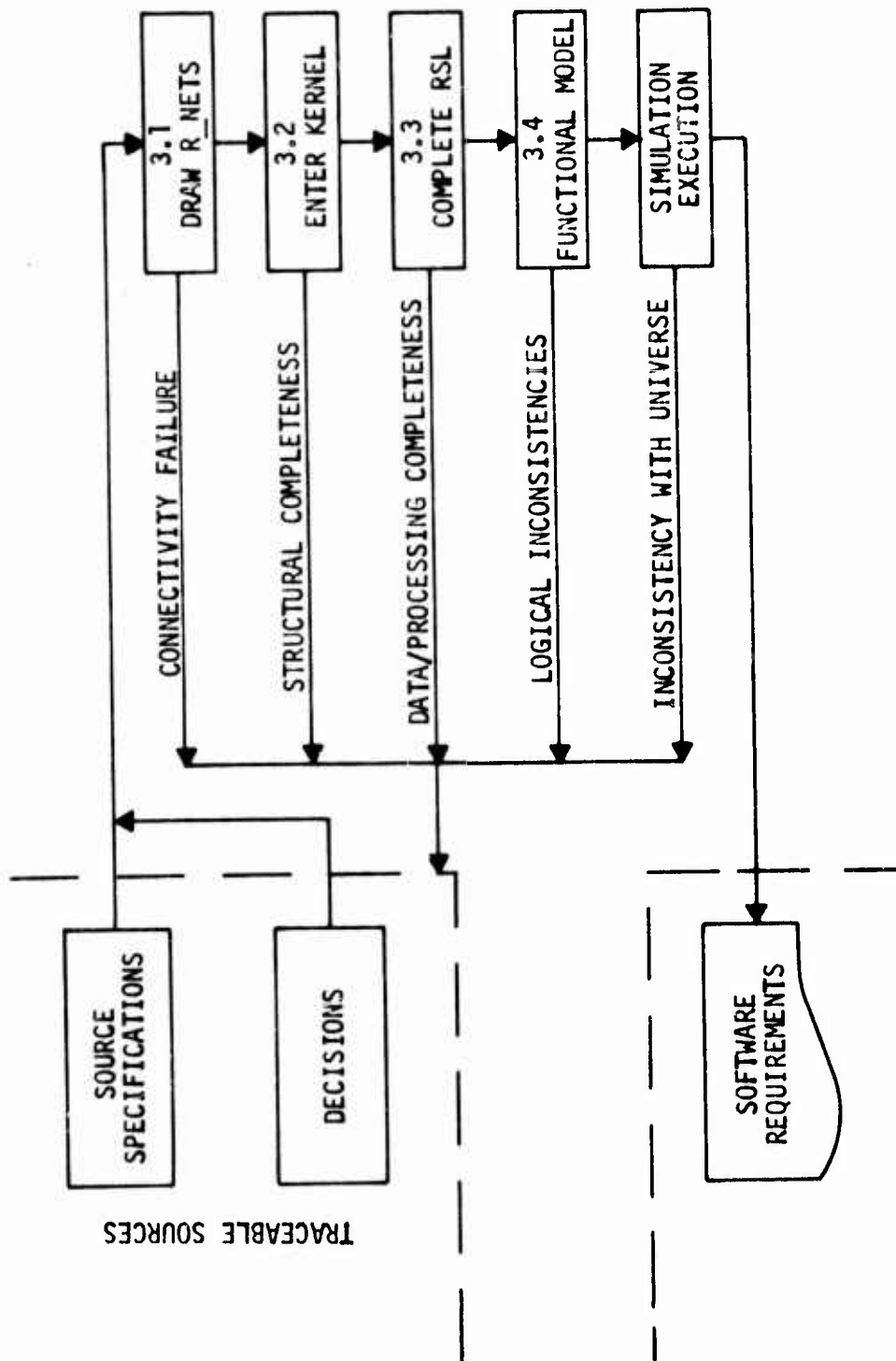


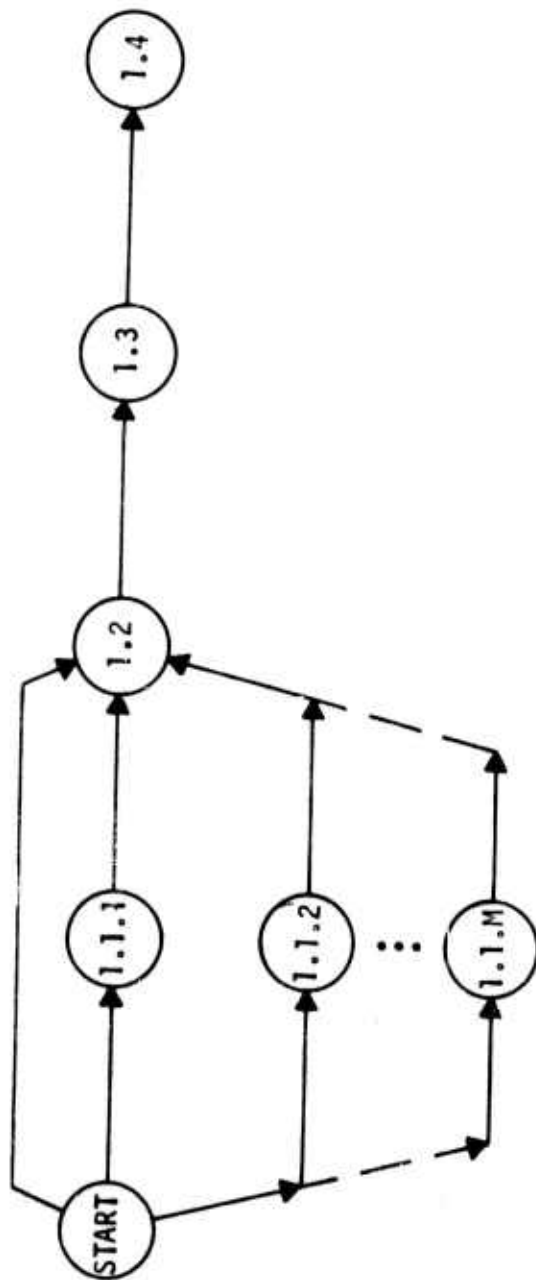
Figure 6-1 Overview of SREM Activities
(Development and Validation of Functional Requirements)

6.1 SOFTWARE REQUIREMENTS DEVELOPMENT

Starting with the initial set of milestones, the natural beginning of software requirements development is the acceptance of source specifications. This is not a final acceptance. That can't take place until the software requirements are validated. Rather, the software requirements manager must establish that the source specifications provide a sufficient base from which the requirements engineering can proceed. Ideally this base is complete, clear, and correct. In complex systems, such as those required for ballistic missile defense, establishing an ideal base is a goal seldom achieved. The best that can be hoped for is to identify those places where the source specifications are incomplete, unclear, or of questionable correctness, and then proceed with risk -- but under control. Immediate feedback is provided to system engineering on the deficiencies found in the source specifications so that answers can be developed.

The initial set of milestones shown in Figure 6-2 is designed to accomplish two objectives. The obvious purpose of specification review is to establish specification acceptance and identify exceptions to and conditions on this acceptance. The second objective is to begin the actual software requirements development. Drawing the R-Nets is used to organize the source specifications and to establish a basis for commitment of the individuals responsible for writing the software requirements. It is this commitment which is the best measure of progress through the acceptance of source specifications.

As discussed in Section 3, the R-Nets typically require some creative engineering before they can be completed. Therefore, this initial review will not normally result in fully complete R-Nets. It should, however identify what information or assumptions are needed for completion. These are described in RSL as DECISIONs as shown in Figure 6-3. At this point, only the statement of the PROBLEM and the TRACES TO attributes are entered (and maybe the ALTERNATIVES). Each DECISION is reviewed with system engineering to verify that the source specifications have been correctly interpreted before the ALTERNATIVES and CHOICE are completed.



MILESTONE DESCRIPTIONS

START	1.1.1 ---- 1.1.M	1.2	1.3	1.4
SOURCE SPECIFICATIONS BASELINED BY SYSTEM ENGINEERING CCB	R-NET #1, R-NET #2, ETC. COMPLETE TO ALPHA LEVEL	DATA BASE APPROVAL AT ALPHA LEVEL	DECISIONS APPROVED BY SRE CCB	SOFTWARE REQUIREMENTS INTERNALLY BASELINED BY SRE CCB

Figure 6-2 Sample Activity Network for Software Requirements Development

Criteria for this initial review should include:

- Can you, the reviewer, develop software requirements from the information contained in the source specifications?
- What changes or additional information would make your job easier?
- Are the performance requirements present and understandable?
- Do the source requirements appear to be overly restrictive (i.e., at too low a level of detail)?

For each milestone of Figure 6-3 the Software Requirements Manager or his Configuration Control Board (CCB) is responsible for recognizing the milestone completion. On a very large software development project a hierarchy of secondary milestones may be necessary. These lower level milestones should be under the cognizance of lower level managers and are represented to the Software Requirements manager only in aggregate as higher level milestones.

The first effort is aimed at completing each R-NET to the ALPHA level. Most of this activity can be segmented to the R-NET and SUBNET levels. However, it is advisable to assign one individual the responsibility to coordinate the data base. This is a third party action to bring about agreement on interfaces between R-NETS and SUBNETS. In addition, this data base approval activity should include analysis of the data base to eliminate dead (unused) or orphaned (unset) data entities.

To a very large project, the Requirements Manager may also wish to have an independent group transcribe the source specifications into ORIGINATING REQUIREMENTS and enter them into the ASSM and then place them under configuration control. Since this step is critical to the validity of traceability audits later on, this independent entry may be justified. On smaller projects, this may not be appropriate.

Again, an opportunity exists for the software requirements engineer to feed back problems to the system engineer via DECISIONs. While there is a tendency to identify and communicate problems in a continuous manner, this destroys the baseline and gives the entire project a drifting feeling. It is therefore necessary to set up specific milestones for DECISIONs. The

DECISION : RADAR_SCHEDULER_PRIORITIZATION.

ALTERNATIVES:

- "1. SCHEDULE PULSE_BY_PULSE. THIS WOULD SIMPLIFY THE NETS BUT WOULD ORBIATE OPTIMIZATION:
 2. OPTIMIZE OVER THE ENTIRE FRAME. TAKING THE FRAME AS A WHOLE GIVES BEST RESULTS BUT REQUIRES WEIGHTING FACTORS FOR PULSE ENSEMBLES.
 3. PRIORITIZE PULSES SUCH THAT ANY PULSE OF HIGH PRIORITY BEATS ALL PULSES OF LOWER. THIS IS SUBOPTIMAL, BUT REALIZABLE BOTH IN THE SPEC AND IN THE SOFTWARE DESIGN. NO A PRIORI WEIGHTS NEEDED."
- "OPTION 3. PRIORITIZED PULSES".

CHOICE:

PROBLEM:

"OPTIMIZATION OF RADAR USAGE REQUIRES A FINITE RADAR FRAME. THIS IMPLIES A PRIORITIZATION SCHEME FOR INTENDED ORDERS."

TRACES TO:

R_NET: SKED_R

R_NET: XMIT_R.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_4_B_FUNCTIONAL.

DECISION : SYNCHRONOUS_VS_ASYNCHRONOUS_TRACK.

ALTERNATIVES:

- "1. SYNCHRONOUS TRACKING (OR RESPONSIVE) REQUIRES THE LAST RADAR RETURN ON AN IMAGE BE USED TO PRODUCE THE NEXT RADAR ORDER.
2. ASYNCHRONOUS TRACKING "OR AUTOGENIC" ALLOWS A TRACK PULSE TO BE SENT USING WHAT EVER STATE IS IN THE DATA BASE."

CHOICE:

"ASYNCHRONOUS TRACKING IS SELECTED TO MAXIMIZE THE ALLOWED DP TIME RESPONSE FOR PROCESSING RADAR RETURNS. THIS DOES NOT PROHIBIT A RESPONSIVE TRACKING IMPLEMENTATION."

PROBLEM:

"TRACKING CAN BE EXPRESSED AS SYNCHRONOUS OR ASYNCHRONOUS."

TRACES TO:

ALPHA: PICK_CANDIDATES.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_A_FUNCTIONAL.

Figure 6-3 Sample Decisions from Track Loop

ones recommended here seem to be a minimum set and can be augmented to tailor the process to a specific project.

After documenting all known source specification problems the software requirements can proceed with internal baselining. This baselining culminates with formal approval of the Software Requirements Engineering (SRE) Configuration Control Board (CCB). The use of the Requirements Engineering and Validation System (REVS) enforces quality control. Because of this, the CCB baselining can address higher level issues of whether or not the software requirements adequately reflect the intent of the source specifications and to what extent the software requirements provide an appropriate base for process design. The former issue can be resolved by analyzing the DECISIONs against the source specification. These can be categorized into those with major effect on software requirements and those with minor effect on the software requirements. Baselining may be deferred until certain major problems are resolved and the number of minor problems is reduced to an acceptable level. The visible act of deferring a major milestone can bring considerable pressure to work problems expeditiously.

6.2 SOFTWARE REQUIREMENTS VALIDATION

The major effort in software requirements engineering is in the validation phase. Validation begins with static evaluation which is a check of data consistency. Validation also includes the development of BETA (functional) models and simulator to test the software requirements for dynamic consistency. This is primarily a test to verify that the specified logic is correct in a dynamic sense. With appropriate functional models (BETAs), a BETA-level simulation can also be used to examine and predict system level performance as a function of data processing performance.

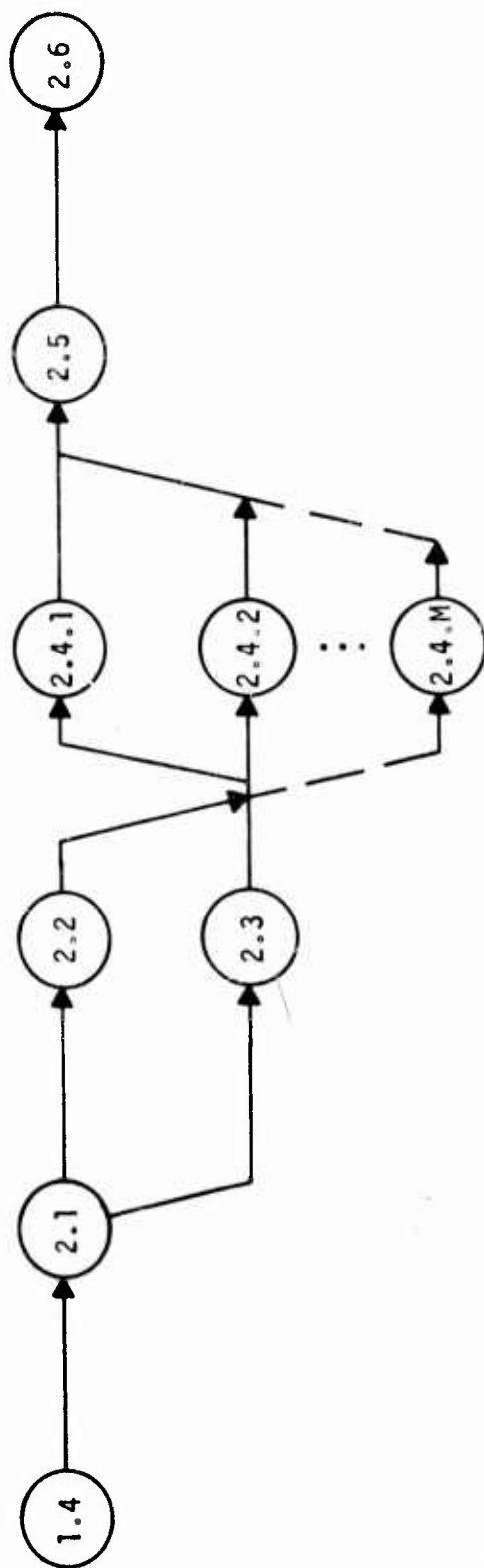
Validation may include the development of GAMMA (analytic) models and simulator. GAMMA models are non-real-time algorithms which input and output the specified data at its elemental level. The purpose of the GAMMA simulator is to demonstrate that a design solution to the software requirements exists at least if the real time constraint is relaxed.

A sample activity network for requirements validation is shown in Figure 6-4. The first effort establishes that the completed R-Nets at the ALPHA level are fully complete and consistent. When this is established, the DECISIONs related to the source specifications and the resulting software requirements are reviewed and approved. Following this, the requirements are completed to BETA level as described in Section 3, and the BETA models and the BETA data base are reviewed and approved. When the BETA-level dynamic performance evaluation is completed and all DECISIONs have been approved (milestone 2.11), then a new requirements baseline is established at milestone 2.12 and the functional requirements are complete.

In a project in which there are well known design solutions available for all the processing specified, the functional requirements validation would end with milestone 2.12, the updated baseline. However, in BMD systems, and in fact in most modern large scale systems, there are algorithm issues which present considerable development risk. If this is the case, the requirements validation is not complete until the computational feasibility of the requirements is established. This is called the analytic feasibility demonstration. The purpose of this phase is to demonstrate through simulation that it is possible to process the specified input data to obtain the specified results. The sequence of activities for this phase of the effort begins with milestone 2.13 and ends with a second baseline release of the requirements at milestone 2.22. The intervening milestones are similar to those for functional validation discussed earlier. The final event is the formal release of the internally-approved specification (milestone 2.23).

6.3 SUMMARY

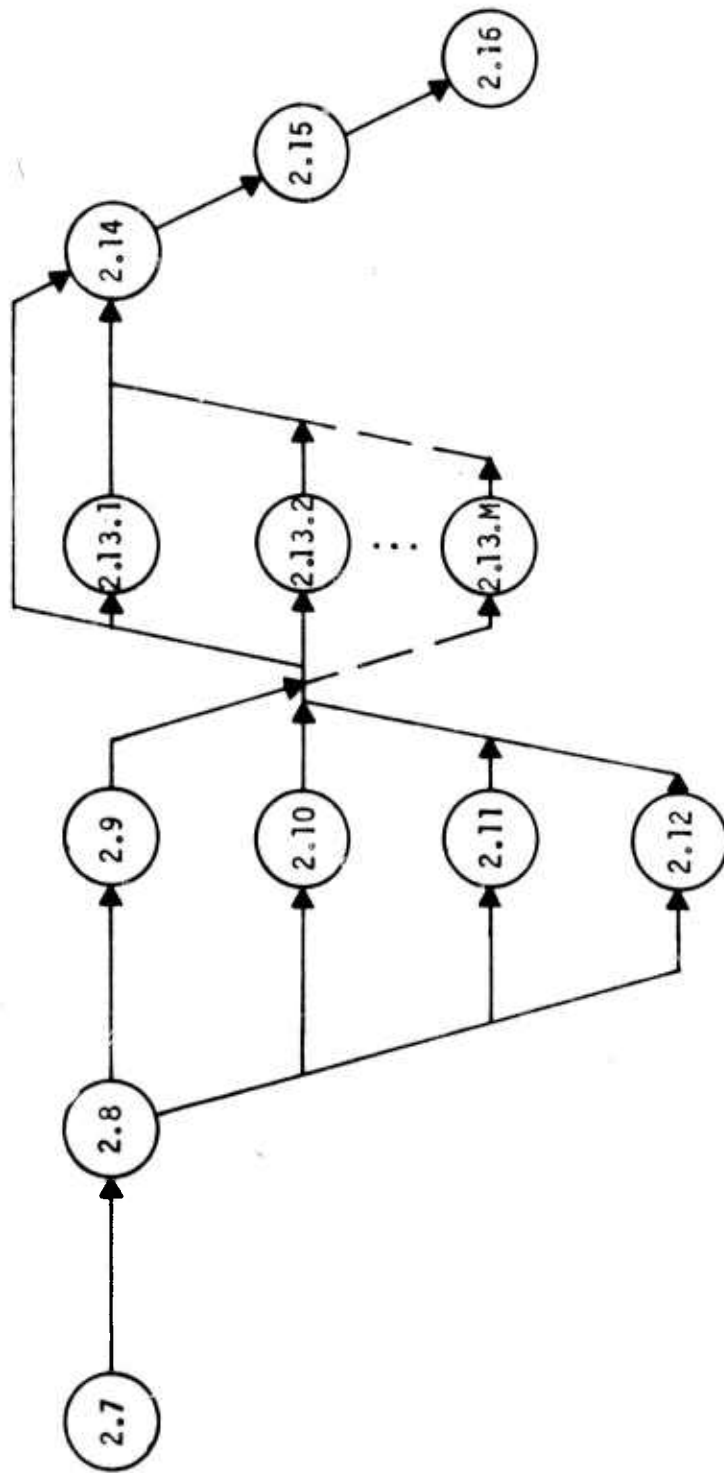
The foregoing is an example of how the SREM activities described in Section 3 can be related to measurable and meaningful milestones. An identical approach can be used for the activities described in Section 4. The activity networks presented here are not intended to be a universal SREM management plan in which one can fill in dates and names and be done. Every program is different as is each engineering organization. Therefore, there cannot be a universal management plan any more than there can be a



MILESTONE DESCRIPTIONS

2.1	2.2	2.3	2.4.1----2.4.M	2.5	2.6
STATIC EVALUATION	DECISIONS RELATED TO SOURCE SPECIFICATION APPROVED BY SRE CCB	DECISIONS RELATED TO SOFTWARE REQ. APPROVED BY SRE CCB	R-NET #1, R-NET #2, ETC. COMPLETE TO BETA LEVEL	DATA BASE APPROVAL AT BETA LEVEL	BETA LEVEL INTERNALLY BASELINED BY SRE CCB

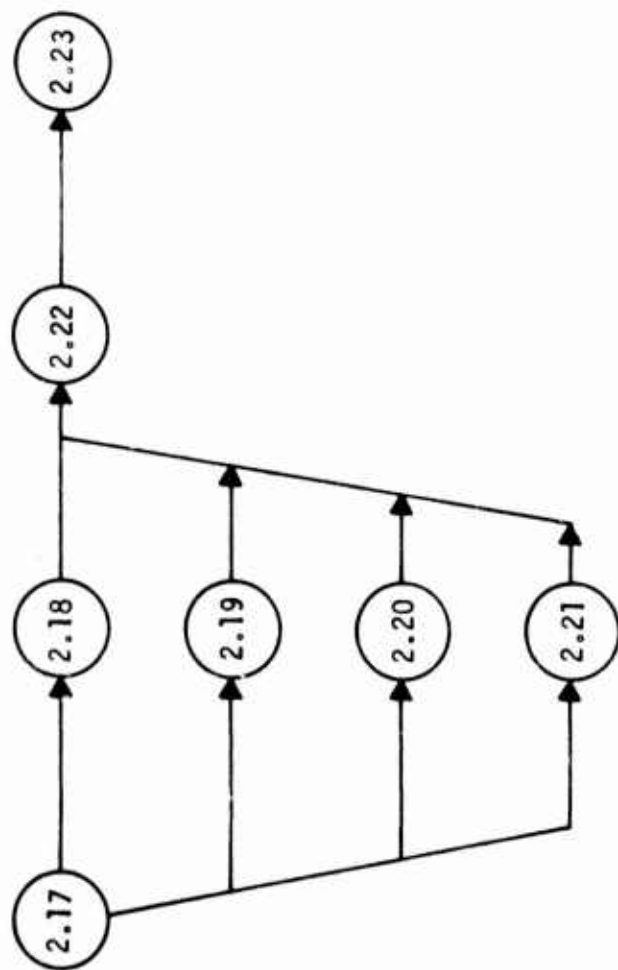
Figure 6-4 Sample Activity Network for Software Requirements Validation



MILESTONE DESCRIPTIONS

2.7	2.8	2.9	2.10	2.11
BETA SIMULATOR GENERATED	DYNAMIC EVALUATION	DECISIONS RELATED TO SOURCE SPECIFICATION APPROVED BY SRE CCB	DECISIONS RELATED TO SOFTWARE REQ. APPROVED BY SRE CCB	DECISIONS RELATED TO BETA LEVEL APPROVED BY SRE CCB
2.12	2.13.1---3.13.M	2.14	2.15	2.16
FIRST MAJOR SOFTWARE REQ. UPDATE BASELINED BY SRE CCB	R-NET #1, R-NET #2, ETC. COMPLETE TO GAMMA LEVEL	DATA BASE APPROVAL AT GAMMA LEVEL	GAMMA LEVEL INTERNALLY BASELINED BY SRE CCB	GAMMA SIMULATOR GENERATED

Figure 6-4 Sample Activity Network for Software Requirements Validation (Continued)



MILESTONE DESCRIPTIONS

2.17	2.18	2.19	2.20
ANALYTIC FEASIBILITY DEMONSTRATION	DECISIONS RELATED TO SOURCE SPECIFICATIONS APPROVED BY SRE CCB	DECISIONS RELATED TO SOFTWARE REQ. APPROVED BY SRE CCB	DECISIONS RELATED TO BETA LEVEL APPROVED BY SRE CCB

2.21	2.22	2.23
DECISIONS RELATED TO GAMMA LEVEL APPROVED BY SRE CCB	SECOND MAJOR SOFTWARE REQ. UPDATE BASELINED BY SRE CCB	SOFTWARE REQUIREMENTS RELEASE

Figure 6-4 Sample Activity Network for Software Requirements Validation (Continued)

universal R-Net. The point is that the SREM approach to software requirements development and validation is structured and formalized such that meaningful milestones can be readily defined -- milestones that are specific and measurable.

7.0 PLANNING

Planning for software requirements engineering using SREM is unique in two respects. First, several of the automated features of REVS can have a strong effect on scheduling. For example, the automatic simulation generation produces a simulation in much less time than conventional manual techniques. Second, very little data exists on the cost and schedule aspects of implementing SREM.

With these two thoughts in mind it seems appropriate to identify those planning characteristics peculiar to SREM, organize them into a simple model and evaluate that model. This approach has the promise of producing some quantitative measures for SREM planning.

To date, there is no usable data on which to base a firm cost and schedule model for SREM. The Track Loop System used as the example problem in Sections 3 and 4 was an experimental model used to test and refine the methodology steps, RSL, and the REVS software. Consequently, the TLS software requirements were developed several times and the effect of prior knowledge cannot be factored out of the data.

7.1 PRELIMINARY GUIDELINES

From the TLS development experience, some preliminary guidelines can be established. These guidelines form the basis of a cost and schedule estimation technique.

- 1) The overall technical coordination of the activity must be the responsibility of one person. In a small problem such as TLS, this person can also be a "working" engineer. In a large project, this technical management function is a full time job by itself. This is analogous to the chief of a chief programmer team.
- 2) The initial R-Net development effort should not be broken down beyond the point of assigning one engineer to each R-Net. The number of R-Nets can be initially estimated to equal the number of input interfaces plus the number of independent (asynchronous) output interfaces.
- 3) The development of BETA and GAMMA models and performance TESTS is essentially an engineering modeling and programming effort. These can be estimated by conventional software estimation techniques except that the integration effort is greatly reduced through the use of the automatic simulation feature of REVS.

While these guidelines are very preliminary and general, they form a better basis for estimating than the current practice of allocating 10 to 20 percent of the estimated software cost (derived by cost-per-instruction times estimated-number-of-instructions) to the requirements development.

7.2 COST MODEL

The formalism of the RSL expression of software requirements and the methodical approach of developing and validating them using SREM suggests that a formal cost model can be developed. A proposed starting point for such a model follows the general form of

$$K \times \sum_{i=1}^n C_i \cdot D_i \cdot N_i$$

where

- K = a weighting factor equal to the cost per typical element (equivalent to dollars-per-instruction in software development).
- n = number of elements (ALPHAs, etc.).
- C_i = the relative complexity of the i^{th} element.
- D_i = the relative newness (state-of-the-art) of the i^{th} element.
- N_i = the size of the i^{th} element.

It appears that the cost of the requirements is sensitive to the following elements:

- Overall logic and data-flow complexity which can be reasonably represented by the ALPHAs. ALPHAs are chosen because:
 - 1) their INPUT and OUTPUT relationships are a direct indicator of the data flow complexity, and
 - 2) there is almost a one-to-one relationship between an ALPHA and a logic node or node-branch on an R-Net.
- Functional Models - BETAs.
- Analytic Models - GAMMAs.

- Performance Allocations - VALIDATION PATHs.
- Performance Measures - TESTs.
- Simulator/Driver Integration - MESSAGEs.

One additional factor is the firmness of the source requirements which directly effects the number of times the requirements engineering work will have to be redone.

A model of the cost of developing and validating the software requirements is then given by the following:

$$\begin{aligned}
 H = S_R \left[K_A \sum_{i=1}^n C_{A_i} \cdot D_{A_i} \cdot N_{A_i} + K_B \sum_{i=1}^n C_{B_i} \cdot D_{B_i} \cdot N_{B_i} \right. \\
 + K_G \sum_{i=1}^n C_{G_i} \cdot D_{G_i} \cdot N_{G_i} + K_P \sum_{i=1}^p C_{P_i} \cdot D_{P_i} \\
 \left. + K_T \sum_{i=1}^t C_{T_i} \cdot D_{T_i} \cdot N_{T_i} + K_{S^D S^M} \right].
 \end{aligned}$$

The parameters in this model are defined in Table 7.1.

This model estimates the cost of direct man-hours only. Costs for management, ODC (Other Direct Charge, such as travel), and computer time must be added.

Not only can such a cost model be used to estimate costs before SRE actually starts, it can also be used during SRE to project cost to complete. This is done by updating the estimated parameters with real values as they become available. Cost control is discussed in Section 8.

7.3 SCHEDULING

Having completed a top down cost estimate, that result can be used to obtain a gross schedule of activities. The cost model must first be partitioned into six parts.

$$H_A = S_R K_A \sum_{i=1}^n C_{A_i} D_{A_i} N_{A_i}$$

Table 7.1 Definition of Symbols Used in Cost Model

<u>SYMBOL</u>	<u>DEFINITION</u>
C_{A_i}	Complexity factor for i^{th} ALPHA. (Value of minimum complexity is unity.)
C_{B_i}	Complexity factor for i^{th} BETA. (Value of minimum complexity is unity.)
C_{G_i}	Complexity factor for i^{th} GAMMA. (Value of minimum complexity is unity.)
C_{P_i}	Complexing factor i^{th} PATH decomposition. (One-to-one correspondence between a performance requirement in the source specification to a PATH is unity; more complex relationships are higher.)
C_T	Complexity factor of i^{th} TEST. (Value of minimum complexity is unity.)
D_{A_i}	Newness factor for i^{th} ALPHA. (Value for completely off-the-shelf is 0, completely new is 1.)
D_{B_i}	Newness factor for i^{th} BETA. (Value for completely off-the-shelf is 0, completely new is 1.)
D_{G_i}	Newness factor for i^{th} GAMMA. (Value for completely off-the-shelf is 0, completely new is 1.)
D_{P_i}	Newness factor of i^{th} PATH decomposition. (Known decompositions are unity; if trade-off analyses are required, the factor is higher.)
D_{T_i}	Newness factor of i^{th} TEST. (Completely off-the-shelf is 0, completely new is unity.)
D_S	Newness factor for SETS. (A fully tested, well documented, and previously used SETS is 1; others are higher.)
K_A	Weighting factor for ALPHA development.
K_B	Weighting factor for BETA development.
K_G	Weighting factor for GAMMA development.
K_P	Weighting factor for PATH decomposition.
K_T	Weighting factor for TEST development.

Table 7.1 Definition of Symbols Used in Cost Model (Continued)

<u>SYMBOL</u>	<u>DEFINITION</u>
K_S	Weighting factor for simulation integration.
M	Number of I/O messages.
N_{A_i}	Size of i^{th} ALPHA in lines of RSL.
n	Number of ALPHAs (BETAs and GAMMAs).
p	Number of VALIDATION PATHs.
t	Number of TESTs.
N_{A_i}	Size of i^{th} ALPHA in lines of RSL.
N_{B_i}	Size of i^{th} BETA in lines of RSL.
N_{G_i}	Size of i^{th} GAMMA in lines of RSL.
N_{T_i}	Size of i^{th} TEST in lines of RSL.
S_R	Softness of system requirements. (Firm is 1, soft is greater than 1.)

$$H_B = S_R K_B \sum_{i=1}^n C_{B_i} D_{B_i} N_{B_i}$$

$$H_G = S_R K_G \sum_{i=1}^n C_{G_i} D_{G_i} N_{G_i}$$

$$H_P = S_R K_P \sum_{i=1}^p C_{P_i} \cdot D_{P_i}$$

$$H_T = S_R K_T \sum_{i=1}^t C_{T_i} \cdot D_{T_i} \cdot N_{T_i}$$

$$H_S = S_R K_S D_S M$$

Then four estimates are extracted as follows:

$$H_1 = H_A$$

$$H_2 = H_B + \left[\frac{H_B}{H_B + H_G} \right] H_S$$

$$H_3 = H_G + \left[\frac{H_G}{H_B + H_G} \right] H_S$$

$$H_4 = H_P + H_T$$

Note that $H = H_1 + H_2 + H_3 + H_4$.

Looking at the milestones in Section 5 these are grouped into three sets. The first is associated with the R-NET development and includes milestones 1.1 through 1.4. The second set, 2.1 through 2.6, is related to BETA modeling. The final set includes the remaining milestones, 2.7 through 2.23, and is roughly related to GAMMA modeling. Figure 7-1 shows the three

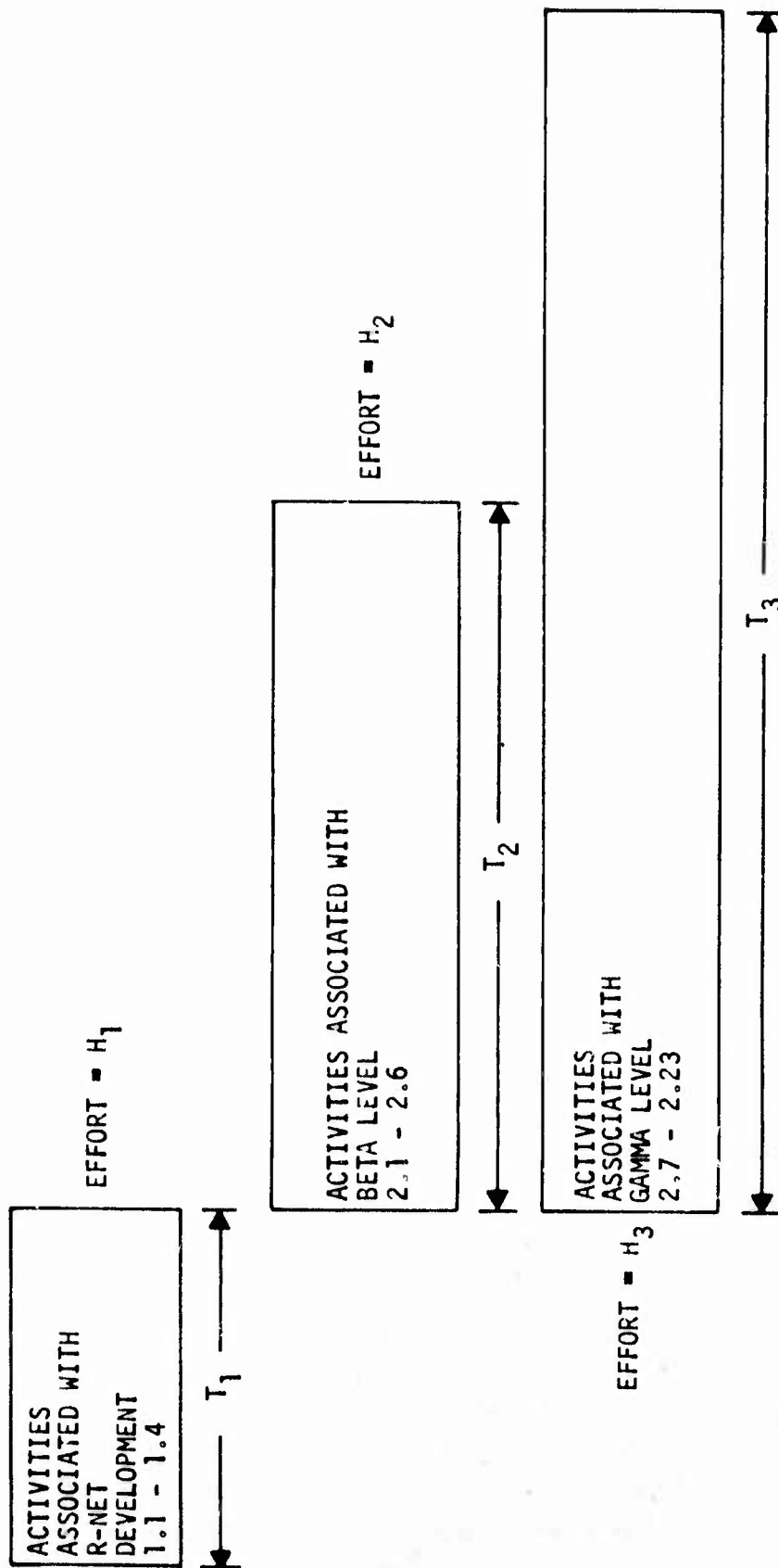


Figure 7-1 Rough Spread of Activities by Level

sets of activities related to these milestones crudely spread over time periods T_1 , T_2 and T_3 expressed in work hours. Then effort divided by time is equivalent to manloading.

For activities related to R-NET development the effort in hours is approximately H_1 . Assuming a constant, flat-loaded activity the loading then is H_1/T_1 . A small group should be working this phase of the requirements development. Even on a very large project no more than five engineers should be directly involved in the R-NET development. Assuming 10 percent for direct support and 10 percent for management support this translates into a constraint:

$$T_1 \geq H_1/6$$

Schedule times T_2 and T_3 can also be roughly estimated by examining values for H_2/T_2 and A_3/T_3 noting that these two activities should overlap. For a small SRE project the following might apply:

$$H_1 = 80 \text{ man hours}$$

$$H_2 = 400 \text{ man hours}$$

$$H_3 = 800 \text{ man hours}$$

Then, the planning might select

$$T_1 = 40 \text{ work hours}$$

$$T_2 = 80 \text{ work hours}$$

$$T_3 = 160 \text{ work hours}$$

giving a manloading as shown in the top part of Figure 7-2. Since it is usually unrealistic to expect adequate performance from short assignments, a better approach is to smooth the total manpower curve somewhat. This is done in the bottom part of Figure 7-2.

Having obtained a rough manpower curve it is now necessary to establish a detailed milestone schedule. For a small project such as the one just discussed it is probably better to lump some of the serial milestones so that

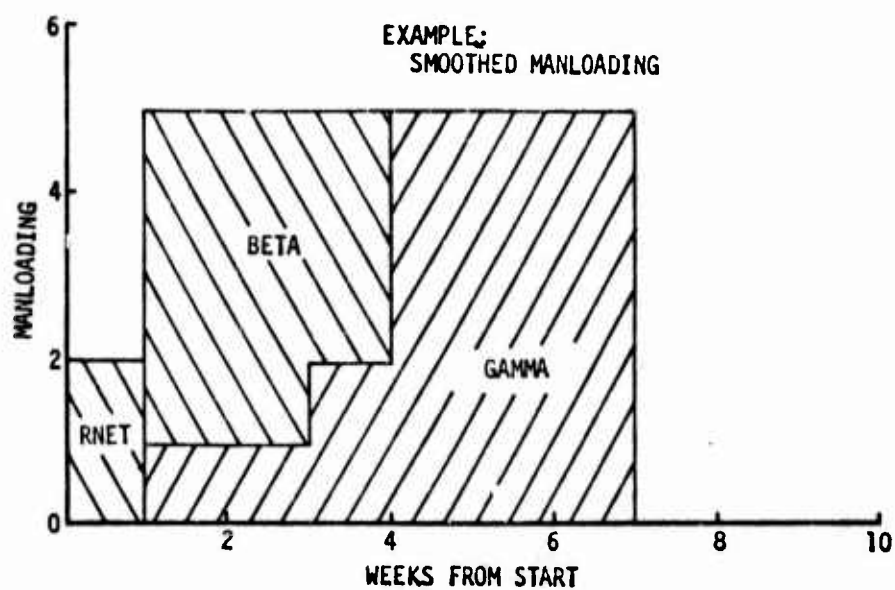
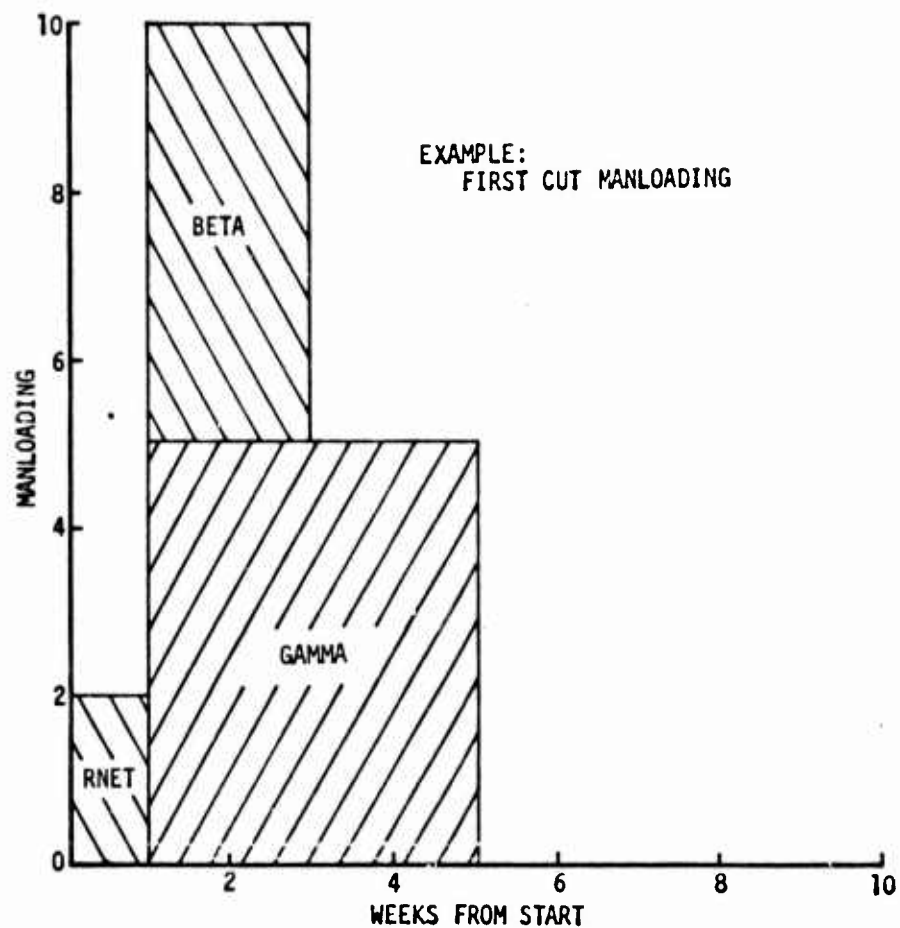


Figure 7-2 Rough Spread of Manpower for Example

a week-by-week measurement is adequate. For the sake of demonstration the complete set of milestones will be laid out on the seven-week schedule shown in Figure 7-2 (bottom). Figure 7-3 shows the detail milestone schedule. The two major activities are the BETA and GAMMA modeling which occur in parallel. For a small project many of the review and approval milestones can be accomplished simultaneously.

The schedule in Figure 7-3 shows no slack time. An end-to-end schedule with no slack time is dangerous even on a small project. Since the SRE activities are laid out on a seven-week or thirty-five day period with no slack, then 10 percent or about 4 days is a conservative amount of slack. The slack should be inserted after the most vulnerable milestones so that the overall schedule is minimally perturbed if a milestone is missed. Figure 7-4 shows a revised schedule with slack time (dashed) protecting key milestones. With slack time on the parallel BETA and GAMMA branches additional flexibility is achieved since manpower not required during one slack activity can be shifted to the other branch.

Note that the schedules do not provide the typical periods for document preparation, review, and publication. The automatic documentation features of REVS reduce the need for these activities. The technical and management review of the requirements is accomplished using REVS generated reports. Once approved, the generation of the final documentation is simply one more computer run using the appropriate RADX directives with REVS.

The planning process has been illustrated using the milestones defined in Section 5 for the activities discussed in Section 3. The planning of the Section 4 activities is accomplished in a similar manner using the H_4 term in the cost breakdown.

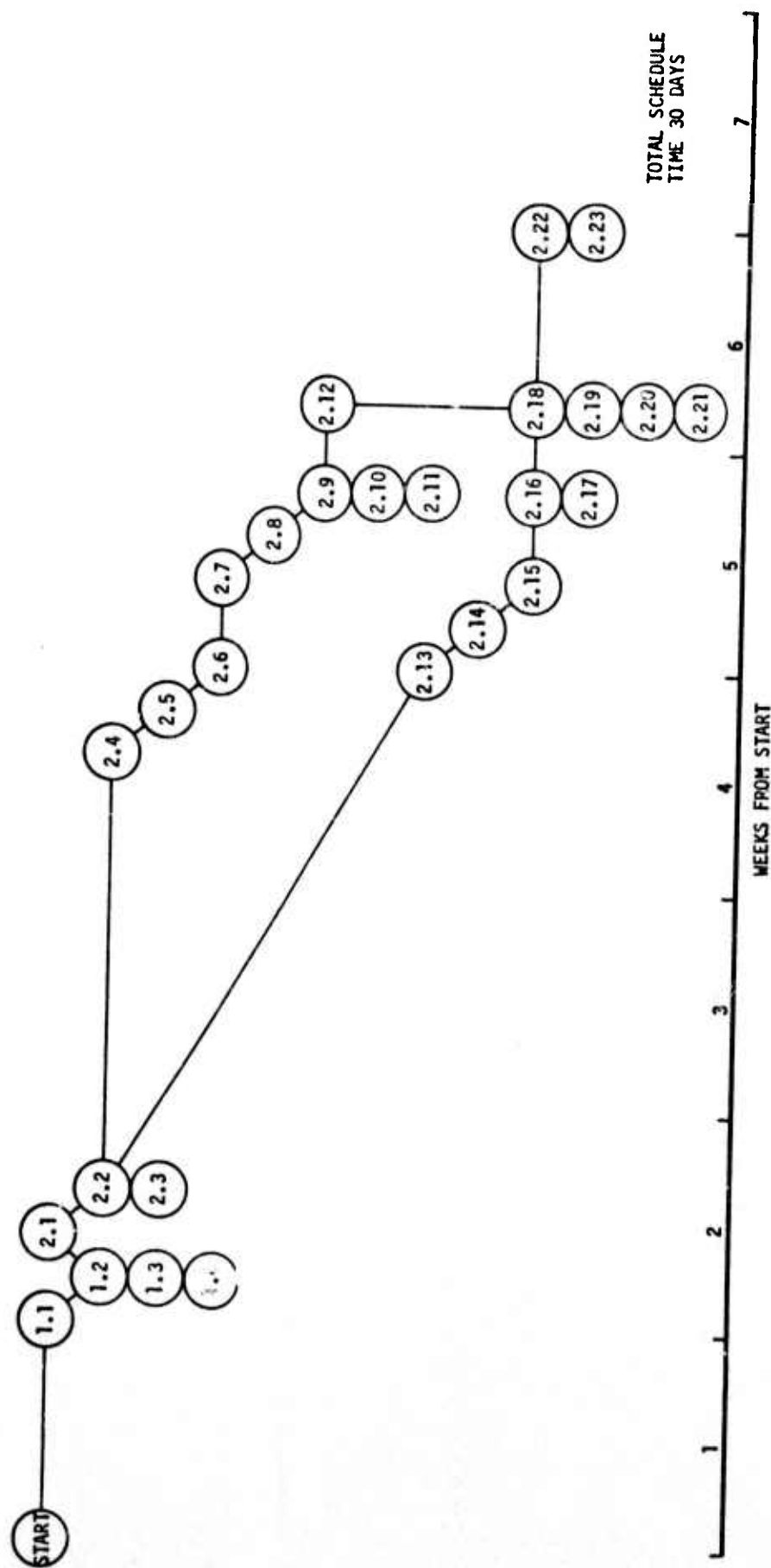


Figure 7-3 Sample Milestone Schedule for Small Project

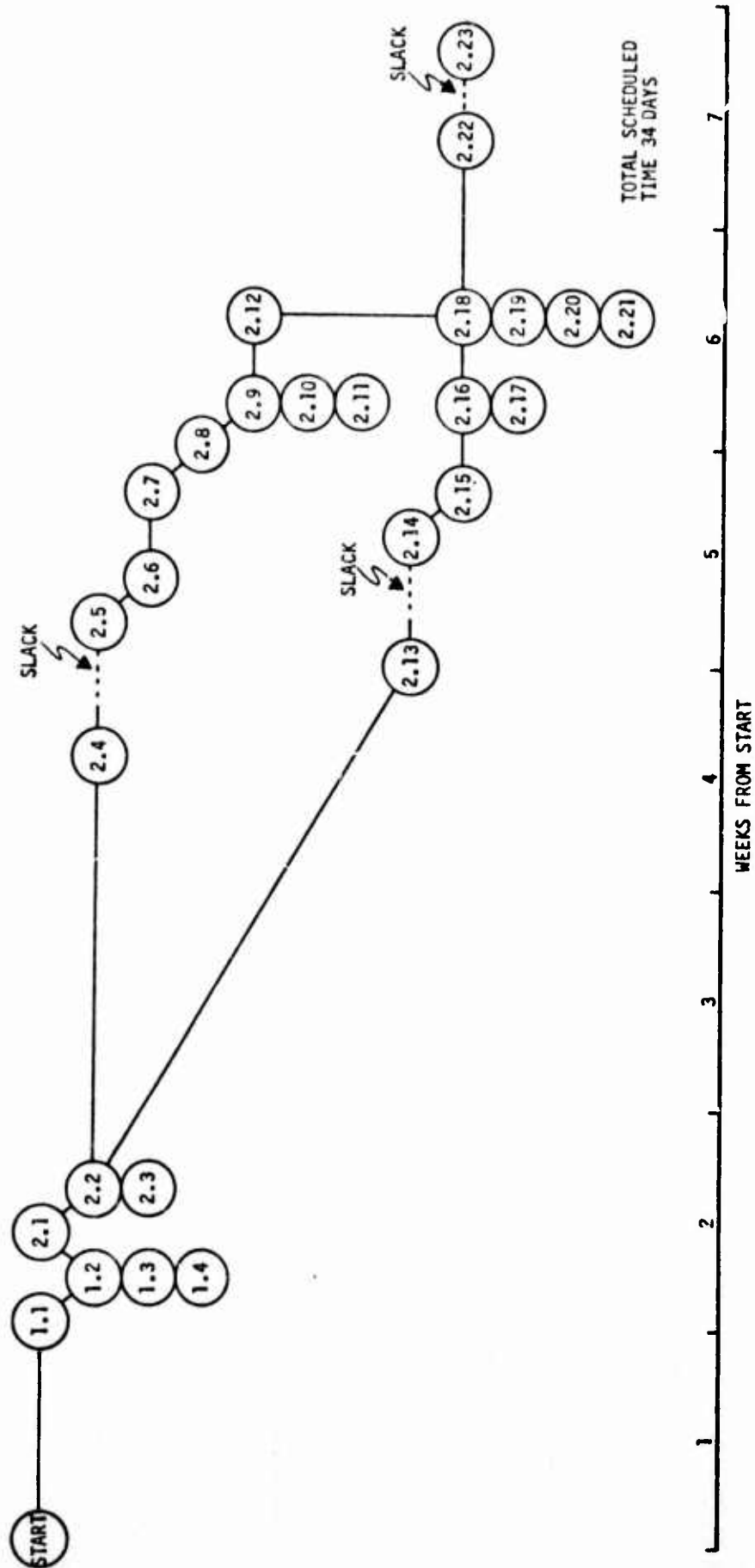


Figure 7-4 Sample Milestone Schedule of Figure 7-3 With Slack Time Added

8.0 MANAGEMENT CONTROL

The Software Requirements Engineering activity must interface with other activities during the system development phase. Figure 8-1 explains the major interests shared by SRE with these other activities. The key interactions may be summarized as follows:

- SRE supports SE in system definition by accepting the specifications and pointing out deficiencies.
- SRE may question the subsystem allocation because of implementation problems.
- SRE participates in the overall system cost/schedule planning and control by providing status data to SE and visibility to Process Design.
- SRE must work with the other subsystem engineering activities to define interfaces to a functional level. Process Design can work to design the detail interfaces. When a referee is needed the SE must decide interface issues.
- SRE defines the processing via software requirements which may be modified if real-time implementation is a problem.

8.1 CONTROL MECHANISMS

Management of SREM can be viewed as a two-level process. REVS provides certain direct, automatic control of the product under development (the software requirements). As described in Table 8.1 this frees management to focus on higher level issues and the implementation of REVS. With a validated REVS, implementation of its control features is completely mechanical and can be delegated with confidence. In particular, in Table 8.1, we note the following observations:

- At each level the manager can focus on intent, softness of decisions, level of detail and schedule performance.
- At the BETA, GAMMA and PATH/PERFORMANCE levels, cost performance becomes important. Overkill must be avoided by maintaining pressure to use the simplest, cheapest models which will do the job. (Thus level of detail is an issue here also.)
- At the GAMMA level the manager must assess real-time feasibility even though the GAMMA models are not real-time.

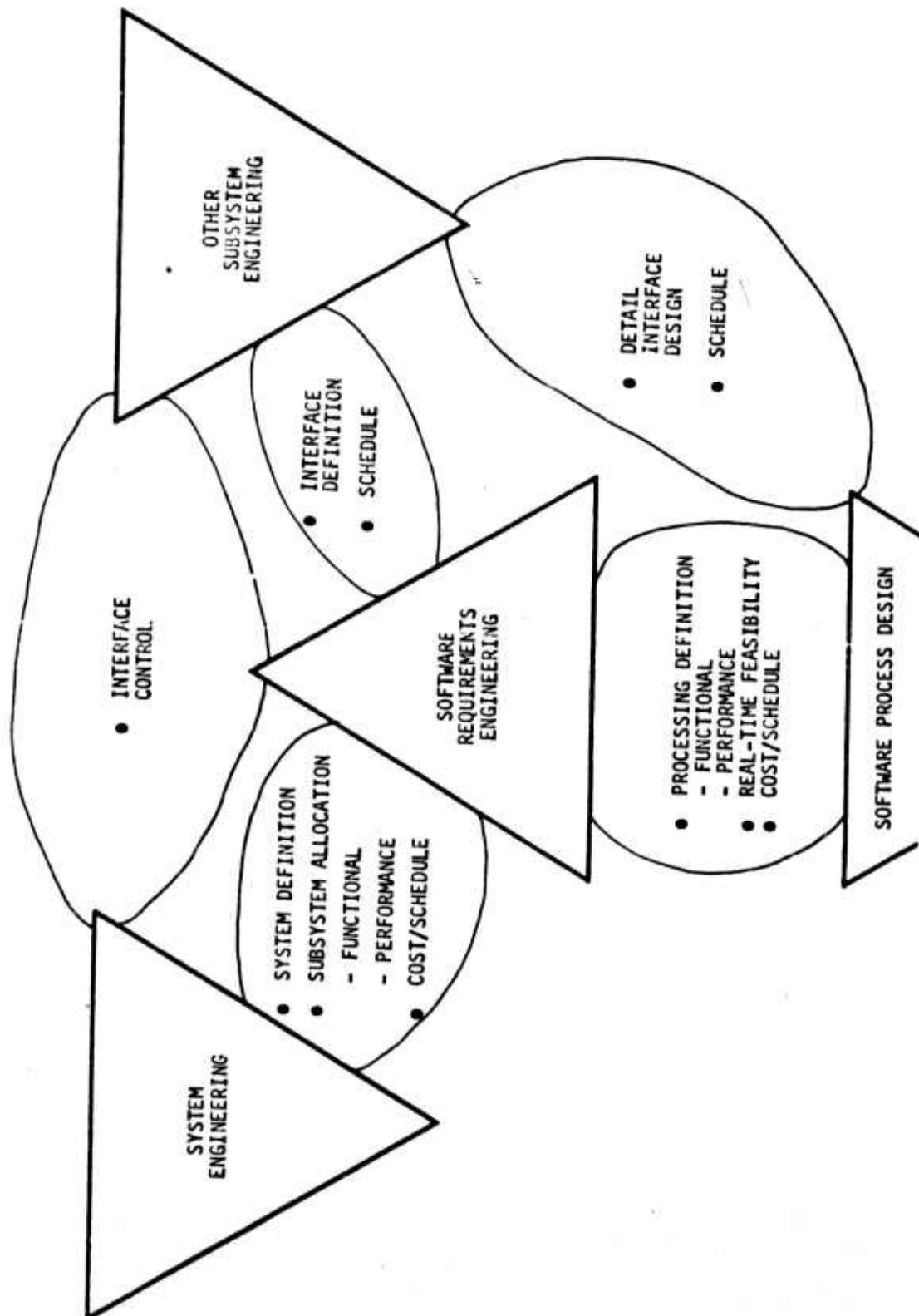


Figure 8-1 Interests Shared by SRE

Table 8.1 SREM Focus of Management Control on Substantive Issues

CONTROL PROVIDED BY REVS	MAJOR MANAGERIAL ISSUES
ALPHA: • TRACEABILITY • DECISIONS • INTERNAL CONSISTENCY - STATIC • STRUCTURAL COMPLETENESS BETA: • ALL OF THE ABOVE • INTERNAL CONSISTENCY - DYNAMIC • EXTERNAL CONSISTENCY • UPWARD CONSISTENCY WITH ALPHA LEVEL GAMMA: • ALL OF THE ABOVE • NON-REAL-TIME FEASIBILITY • UPWARD CONSISTENCY WITH BETA LEVEL VALIDATION PATHS/PERFORMANCE REQUIREMENTS: • ALL OF THE ABOVE • UPWARD CONSISTENCY WITH GAMMA LEVEL	ALPHA: • INTENT OF SOURCE SPECS REFLECTED • SOFTNESS OF INTERIM DECISIONS • LEVEL OF DETAIL • SCHEDULE PERFORMANCE BETA: • ALL OF THE ABOVE • COST PERFORMANCE GAMMA: • ALL OF THE ABOVE • REAL-TIME FEASIBILITY VALIDATION PATHS/PERFORMANCE REQUIREMENTS: • SAME AS BETA • IMPLEMENTATION OF REVS

- Note all of the control which REVS gives cheaply and quickly. Previously, managers had to strain to get this kind of visibility.

The major managerial issues identified in Table 8.1 require mechanisms outside of REVS. These are the more widely used managerial controls, and are much more effective when based on the timely, complete, and organized information provided by REVS. The kinds of control mechanisms used for these issues are shown in Table 8.2 and are summarized below:

- Internal reviews can be used to establish confidence in the software requirements by addressing items checked in the Table.
- Starting with a parametric cost model, projected cost can be estimated as actual parameter values are determined (e.g., number and complexity of ALPHAs).
- Earned value as a measure of progress toward each milestone can be used in the C-SPEC sense to evaluate cost/schedule performance in a continuous fashion. Earned value can be quantified by using number of ALPHAs, BETAs, GAMMAs, PATHs, and TESTs weighted by complexity factors.
- The milestone schedule is a standard schedule performance control best implemented by public display.
- Reviews with the system engineer should help understand questions of intent and softness of decisions plus adequacy of level of detail.
- The process designer should be involved in reviews of level of detail and real-time feasibility (includes storage capacity).

8.2 CHANGE CONTROL

The process of change control is a critical one on a large, complex system development activity. Figure 8-2 shows how collected decisions are folded into the baseline as scheduled updates. The updates are inserted at points in the schedule where they are likely to have significant inputs from within the SRE activity and from System Engineering and Process Design. They also follow specific measures of the software requirements, namely the initial dynamic evaluation and the initial feasibility demonstration.

Table 8.2 Control Mechanisms for Major Managerial Control Issues

MAJOR MANAGERIAL CONTROL ISSUES	CONTROL MECHANISM					
	INTERNAL REVIEWS	COST PARAMETER MEASUREMENT	EARNED VALUE	MILESTONE SCHEDULE	SYSTEM ENGINEER REVIEW	PROCESS DESIGN REVIEW
SOURCE SPEC INTENT	X				X	
DECISION SOFTNESS	X				X	
LEVEL OF DETAIL	X				X	X
COST PERFORMANCE		X	X			
SCHEDULE PERFORMANCE			X	X		
REAL-TIME FEASIBILITY	X					X

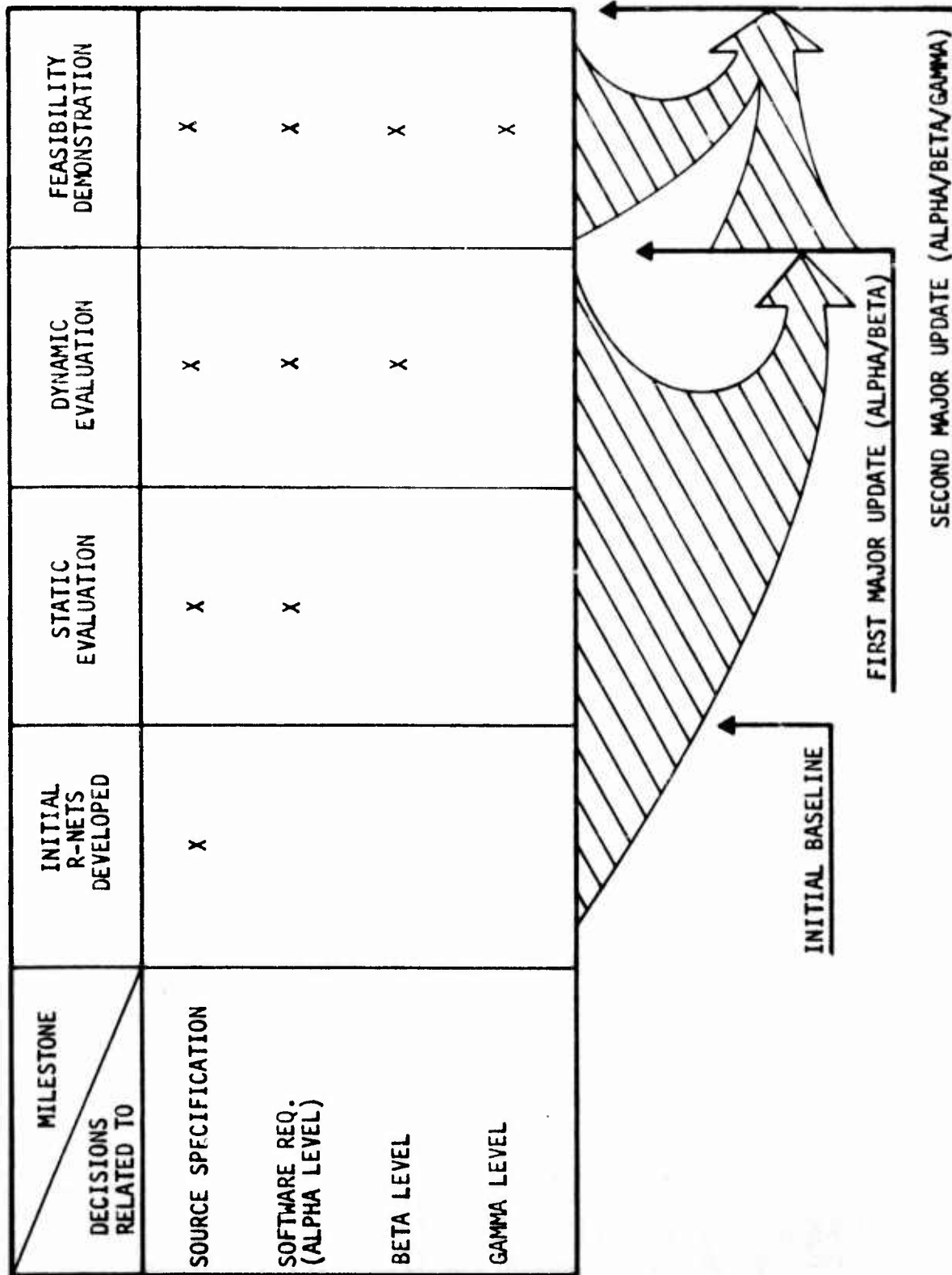


Figure 8-2 Change Flow into Update of Baseline

8.3 SELLING THE SOFTWARE REQUIREMENTS

Both the System Engineer and Process Designer must "buy off" the software requirements. SREM possesses key features which should assist the SRE manager in effecting the buy-off. These are explained in Table 8.3 in terms of answers to concerns of both System Engineering and Process Design.

Table 8.3 Selling Software Requirements

SYSTEM ENGINEER CONCERNS	SRE ANSWER	PROCESS DESIGNER CONCERNS
1. DO SOFTWARE REQUIREMENTS FAITHFULLY TRANSLATE SOURCE SPECS	• REVS TRACEABILITY • INTENT REVIEWS WITH SYSTEM ENGINEER • QUICK RESPONSE • CONTROLLED BASELINE	
2. WHAT CHANGES IN SOURCE SPECS ARE NECESSARY FOR IMPLEMENTATION?	• DOCUMENTED DECISIONS • STATIC EVALUATION • DYNAMIC EVALUATION • CONTROLLED BASELINE	
3. WHAT IS COST/SCHEDULE DELTA FOR A SYSTEM CHANGE?	• QUICK RESPONSE TO DEFINE SOFTWARE REQUIREMENTS CHANGES FOR COST/SCHEDULE EVALUATION	
	• SRE METHODOLOGY • LEVEL OF DETAIL REVIEWS	1. IS LEVEL OF DETAIL TOO CONSTRAINING? ADEQUATE?
	• RSL • REVS	2. REQUIREMENTS CLEAR, COMPLETE AND CONSISTENT?
	• FEASIBILITY DEMO • REAL-TIME REVIEWS WITH PROCESS DESIGNER	3. IMPLEMENTATION FEASIBLE?
	• REVS TEST DEFINITION	4. WHAT IS SOFTWARE ACCEPTANCE CRITERIA?
	• BASELINE CONTROL • DECISION TRACE	5. ARE REQUIREMENTS FIRM?

9.0 CONCLUSIONS

This manual has attempted to explain both the technical and management considerations in the development and validation of software requirements using the tools and techniques of SREM. The engineering and management principles explained here are not new -- they are the result of hundreds of man-years of experience in large-scale real-time software development. The discipline of RSL and the power of REVS are new, as is the formalization of the detailed steps to be followed in their use.

This manual will not make instant engineers or managers out of inexperienced people. It will, however, guide the experienced engineer and manager in the application of RSL, REVS, and SREM techniques to obtain a software requirements specification which is superior in terms of the qualities of a good specification discussed in the opening Section.

APPENDIX A
RSL TERMINOLOGY

A-1

PRECEDING PAGE BLANK NOT FILMED

ELEMENT_TYPE: ALPHA
 (* A PROCESSING STEP IN THE FUNCTIONAL REQUIREMENTS DOMAIN. *).
 STRUCTURE APPLICABILITY: NET.

ELEMENT_TYPE: DATA
 (* A SINGLE ITEM OR SET OF DATA THAT IS SPECIFIED AND THAT WILL EITHER BE REQUIRED IN THE REAL-TIME SOFTWARE OR IS NEEDED FOR DESCRIPTIVE PURPOSES. *).

ELEMENT_TYPE: DECISION
 (* THE DECISION THAT HAS BEEN MADE TO ENABLE REQUIREMENTS TO BE TAKEN FROM THE OFSPR TO THE PPR. THIS MEANS THAT THE REQUIREMENTS ARE NOT SIMPLY ALLOCATED, BUT HAVE BEEN SUBJECTED TO DERIVATION. *).

ELEMENT_TYPE: ENTITY_CLASS
 (* A GENERAL CLASS OF "OBJECTS" IN THE REAL WORLD OUTSIDE THE DATA PROCESSING SYSTEM AND WHICH IS IMPORTANT TO IT. FOR EXAMPLE, AN ENTITY_CLASS MIGHT BE RVS OR INTERCEPTORS. THE ENTITY_TYPES MIGHT BE DETECTION, POTENTIAL_RV, IDENTIFIED_RV, ETC. *).

ELEMENT_TYPE: ENTITY_TYPE
 (* A SPECIFIC TYPE OF "OBJECT" IN THE REAL WORLD OUTSIDE THE DATA PROCESSING SYSTEM AND WHICH IS OF IMPORTANCE TO IT. WHEN A SPECIFIC TYPE OF "OBJECT" IS DETERMINED TO EXIST IN AN ENTITY_CLASS, FILES AND DATA MAY BE TEMPORARILY CREATED TO MAINTAIN INFORMATION ABOUT IT. *).

ELEMENT_TYPE: EVENT
 (* AN IDENTIFIED POINT THAT EXISTS IN THE PROCESSING OF ONE OR MORE R_NETS OR SUBNETS AND WHICH MAY CAUSE THE ENABLEMENT OF AN R_NET. *).
 STRUCTURE APPLICABILITY: NET.
 STRUCTURE APPLICABILITY: PATH.

ELEMENT_TYPE: FILE
 (* AN AGGREGATION OF INSTANCES OF DATA, EACH INSTANCE OF WHICH IS TREATED IN THE SAME MANNER. *).

ELEMENT_TYPE: INPUT_INTERFACE
 (* A "PORT" BETWEEN THE DATA PROCESSING SYSTEM AND THE REST OF THE BMD SYSTEM WHICH ACCEPTS DATA FROM THE OTHER SYSTEM (E.G. THE RADAR_RETURNS). *).
 STRUCTURE APPLICABILITY: NET.

ELEMENT_TYPE: MESSAGE
 (* AN AGGREGATION OF DATA AND FILES THAT PASS THROUGH AN INTERFACE AS A LOGICAL UNIT. *).

ELEMENT_TYPE: ORIGINATING_REQUIREMENT
 (* THE HIGHER LEVEL (DPSR) REQUIREMENT FROM WHICH LOWER LEVEL REQUIREMENTS (THE ONES DESCRIBED IN THE RSL) ARE TRACEABLE. *).

ELEMENT_TYPE: OUTPUT_INTERFACE
 (* A "PORT" BETWEEN THE DATA PROCESSING SYSTEM AND THE REST OF THE BMD SYSTEM WHICH TRANSMITS DATA TO THE OTHER SYSTEM (E.G. THE RADAR_COMMANDS). *).
 STRUCTURE APPLICABILITY: NET.

ELEMENT_TYPE: PERFORMANCE_REQUIREMENT
 (* AN ANALYTIC PERFORMANCE REQUIREMENT OR
 NON-STIMULUS-RESPONSE TIMING REQUIREMENT
 WHICH IS TO BE MET BY THE REAL-TIME SOFTWARE. *).

ELEMENT_TYPE: R_NET
 (* THE ORDER OF LOGICAL PROCESSING STEPS THAT MUST BE
 PERFORMED. AN R_NET MAY CONTAIN ANDS, ORS, AND
 FOR EACH NODES; IT MUST BE ENABLED AND TERMINATED.
 THE PROCESSING STEPS ARE ALPHAS OR SUBNETS WHICH
 MAY BE EXPANDED INTO LOWER LEVELS OF DETAIL. AN
 R_NET MAY ALSO CONTAIN VALIDATION_POINTS, EVENTS,
 AND INTERFACES. *).

ELEMENT_TYPE: REFERENCE
 (* SOURCE MATERIAL FOR REQUIREMENTS. *).

ELEMENT_TYPE: SUBNET
 (* THE ORDER OF LOGICAL PROCESSING STEPS THAT MUST BE
 PERFORMED IN ORDER TO PERFORM THE REQUIREMENTS OF
 THE PROCESSING STEP THAT REPRESENTS IT AT THE NEXT
 HIGHER LEVEL. *).

STRUCTURE APPLICABILITY: NET.

ELEMENT_TYPE: SUBSYSTEM
 (* A PART OF THE BMD SYSTEM (SUCH AS RADAR) WHICH
 COMMUNICATES WITH THE DATA PROCESSING SYSTEM. *).

ELEMENT_TYPE: SYNONYM
 (* A SYNONYM IS MERELY AN ALTERNATE NAME THAT
 CAN BE USED IN PLACE OF THE PRIME NAME. IT
 IS USED AS AN ABBREVIATION IN MOST CASES,
 BUT MAY BE USED FOR OTHER REASONS ALSO.
 NOTE: IN AN <ELEMENT TYPE LIST>, "ALL"
 ALWAYS IMPLIES "ALL EXCEPT SYNONYM". *).

ELEMENT_TYPE: UNSTRUCTURED_REQUIREMENT
 (* A REQUIREMENT THAT MUST BE PASSED TO THE DESIGNER OF
 THE REAL-TIME CODE BUT THAT DOES NOT FIT INTO THE
 STRUCTURED FRAMEWORK PROVIDED BY RSL. THIS ELEMENT
 MIGHT BE USED BECAUSE THE REQUIREMENT IN QUESTION IS
 TOO UNIQUE TO JUSTIFY DEFINITION OF A NEW TYPE OF
 ELEMENT, A NEW RELATIONSHIP, OR A NEW ATTRIBUTE. IT
 ALSO MIGHT BE USED BECAUSE THE REQUIREMENTS ANALYST
 DOES NOT CARE TO DESIGN A NEW CONCEPT TO FIT THE
 REQUIREMENT, OR BECAUSE THE REQUIREMENT IS CLEARLY
 A DESIGN LIMITATION THAT SHOULD BE DESCRIBED
 IN ENGLISH TEXT. (AN EXAMPLE OF THE LAST
 REASON MIGHT BE PRECLUSION OF USING A
 MULTIPROCESSOR WITH ASSOCIATIVE MEMORY.) *).

ELEMENT_TYPE: VALIDATION_PATH
 (* THE PATH OF PROCESSING OVER WHICH THE QUANTITATIVE
 VALIDATION TESTING WILL BE PERFORMED. *).

ELEMENT_TYPE: VALIDATION_POINT
 (* A LOGICAL POINT IN THE PROCESSING AT WHICH TIMING,
 VALUE, OR PRESENCE DATA MUST BE OBTAINABLE IN THE
 REAL-TIME SOFTWARE IN ORDER TO VALIDATE THAT THE
 REQUIREMENTS HAVE BEEN FULFILLED. *).

STRUCTURE APPLICABILITY: NET.
 STRUCTURE APPLICABILITY: PATH.

ELEMENT_TYPE: VERSION
 (* THE AGGREGATION OF REQUIREMENTS THAT ARE TO
 APPLY AS A UNIT TO THE DATA PROCESSING
 SYSTEM AT A PARTICULAR TIME. LOOP_1,
 LOOP_2, ETC., ARE VERSIONS, AS IS AN IOC
 SYSTEM. *).

RELATIONSHIP: ASSOCIATES
 (* TELLS WHICH DATA AND FILES
 COME INTO EXISTENCE WHEN THE DATA
 PROCESSING SYSTEM (SOME ALPHA) CREATES AN
 INSTANCE OF AN ENTITY_CLASS OR AN R_NET IS
 ENABLED. *).
 COMPLEMENTARY RELATIONSHIP: ASSOCIATED ("WITH").
 SUBJECT: ENTITY_TYPE, ENTITY_CLASS.
 OBJECT: DATA, FILE.

RELATIONSHIP: COMPOSES
 (* TELLS WHICH ENTITY_TYPES ARE MEMBERS OF AN
 ENTITY_CLASS. *).
 COMPLEMENTARY RELATIONSHIP: COMPOSED ("OF").
 SUBJECT: ENTITY_TYPE.
 OBJECT: ENTITY_CLASS.

RELATIONSHIP: CONNECTS ("TO")
 (* TELLS WHICH SUBSYSTEM THE INPUT_INTERFACE OR
 OUTPUT_INTERFACE COMMUNICATES WITH. *).
 COMPLEMENTARY RELATIONSHIP: CONNECTED ("TO").
 SUBJECT: INPUT_INTERFACE, OUTPUT_INTERFACE.
 OBJECT: SUBSYSTEM.

RELATIONSHIP: CONTAINS
 (* TELLS THE IDENTITY OF EACH CONSTITUENT PART OF EACH
 INSTANCE IN A FILE. A DIRECT IMPLEMENTATION IN
 SOFTWARE WOULD USE THIS RELATIONSHIP TO GIVE THE
 MAKE-UP OF RECORDS IN FILES. *).
 COMPLEMENTARY RELATIONSHIP: CONTAINED ("IN").
 SUBJECT: FILE.
 OBJECT: DATA.

RELATIONSHIP: CONSTRAINS
 (* IDENTIFIES TO WHICH VALIDATION_PATH(S) THE
 PERFORMANCE_REQUIREMENT APPLIES. *).
 COMPLEMENTARY RELATIONSHIP: CONSTRAINED ("BY").
 SUBJECT: PERFORMANCE_REQUIREMENT.
 OBJECT: VALIDATION_PATH.

RELATIONSHIP: CREATES
 (* TELLS WHICH PROCESSING STEPS CREATE THE INSTANCE
 OF AN ENTITY_CLASS. *).
 COMPLEMENTARY RELATIONSHIP: CREATED ("BY").
 SUBJECT: ALPHA.
 OBJECT: ENTITY_CLASS.

RELATIONSHIP: DELAYS
 (* THE ENABLEMENT OF R_NETS BY THE OBJECT EVENT IS
 POSTPONED FOR THE AMOUNT OF TIME SPECIFIED BY
 THE DATA. *).
 COMPLEMENTARY RELATIONSHIP: DELAYED ("BY").
 SUBJECT: DATA.
 OBJECT: EVENT.

RELATIONSHIP: DESTROYS
 (* TELLS WHICH PROCESSING STEPS DESTROY AN
 INSTANCE OF THE ENTITY_CLASS. *).
 COMPLEMENTARY RELATIONSHIP: DESTROYED ("BY").
 SUBJECT: ALPHA.
 OBJECT: ENTITY_CLASS.

RELATIONSHIP: ENABLES
 (* WHEN THE EVENT(S) IS (ARE) PASSED THROUGH BY THE
 CONTROL FLOW ON AN R_NET, OR WHEN DATA ARE
 AVAILABLE AT AN INTERFACE (AS DEFINED IN THE
 INTERFACE DEFINITION), THE FUNCTIONAL PROCESSING
 INDICATED ON THE R_NET CAN BE BEGUN. *).
 COMPLEMENTARY RELATIONSHIP: ENABLED ("BY").
 SUBJECT: EVENT, INPUT_INTERFACE.
 OBJECT: R_NET.

RELATIONSHIP: EQUATES ("TO")

(* DEFINES AN ALTERNATE NAME FOR AN ELEMENT. THE OBJECT IS CALLED THE PRIME NAME. THE SUBJECT NAME CAN BE USED FOR INPUT TO THE ASSM, BUT ALL RELATIONSHIPS, ATTRIBUTES, AND STRUCTURES AS DEFINED ARE ACTUALLY CHARACTERISTICS OF THE PRIME NAME. *).

COMPLEMENTARY RELATIONSHIP: EQUATED ("TO").

SUBJECT: SYNONYM.

OBJECT: ALL.

RELATIONSHIP: EXPLAINS

(* THE REFERENCE EXPLAINS THE BACKGROUND INFORMATION ABOUT THE OBJECT ELEMENTS. *).

COMPLEMENTARY RELATIONSHIP: EXPLAINED ("BY").

SUBJECT: REFERENCE.

OBJECT: ALL EXCEPT REFERENCE.

RELATIONSHIP: FORMS

(* INDICATES THE ALPHA WHICH DEFINES A MESSAGE TO BE PASSED THROUGH AN OUTPUT_INTERFACE. *).

COMPLEMENTARY RELATIONSHIP: FORMED ("BY").

SUBJECT: ALPHA.

OBJECT: MESSAGE.

RELATIONSHIP: IMPLEMENTS

(* TELLS THE VERSIONS TO WHICH THE ELEMENT, AS DESCRIBED, APPLIES. *).

COMPLEMENTARY RELATIONSHIP: IMPLEMENTED ("BY").

SUBJECT: ALL EXCEPT VERSION.

OBJECT: VERSION.

RELATIONSHIP: INCLUDES

(* INDICATES THE HIERARCHICAL RELATIONSHIP BETWEEN DATA. IF A INCLUDES B, THEN OBTAINING A WILL OBTAIN B. *).

COMPLEMENTARY RELATIONSHIP: INCLUDED ("IN").

SUBJECT: DATA.

OBJECT: DATA.

RELATIONSHIP: INPUTS

(* INDICATES THAT THE NAMED ELEMENT INPUTS THE OBJECT ELEMENT(S). *).

COMPLEMENTARY RELATIONSHIP: INPUT ("TO").

SUBJECT: ALPHA, VALIDATION_POINT.

OBJECT: DATA, FILE.

RELATIONSHIP: MAKES

(* TELLS THE IDENTITY OF DATA AND FILES THAT MAKE UP A MESSAGE. *).

COMPLEMENTARY RELATIONSHIP: MADE ("BY").

SUBJECT: DATA, FILE.

OBJECT: MESSAGE.

RELATIONSHIP: ORDERS

(* THE ELEMENT ON WHICH THE INSTANCES ARE ORDERED IN A FILE. *).

COMPLEMENTARY RELATIONSHIP: ORDERED ("BY").

SUBJECT: DATA.

OBJECT: FILE.

RELATIONSHIP: OUTPUTS

(* INDICATES THE NAMED ELEMENT OUTPUTS THE OBJECT ELEMENT(S). *).

COMPLEMENTARY RELATIONSHIP: OUTPUT ("FROM").

SUBJECT: ALPHA.

OBJECT: DATA, FILE.

RELATIONSHIP: PASSES

(* INDICATES THE INFORMATION WHICH PASSES THROUGH THE INTERFACE. *).

COMPLEMENTARY RELATIONSHIP: PASSED ("THROUGH").

SUBJECT: INPUT_INTERFACE, OUTPUT_INTERFACE.

OBJECT: MESSAGE.

RELATIONSHIP: SETS

(* TELLS WHICH ALPHA SETS THE ENTITY_TYPE OF AN INSTANCE IN AN ENTITY_CLASS. *).

COMPLEMENTARY RELATIONSHIP: SET ("BY").

SUBJECT: ALPHA.

OBJECT: ENTITY_TYPE.

RELATIONSHIP: TRACES ("TO")

(* TELLS THE HIGHER LEVEL (ORIGINATING) REQUIREMENT FROM WHICH THE ELEMENT WAS TRACED (ALLOCATED). A DEFAULT ORIGINATING REQUIREMENT IS "NONE", WHICH INDICATES THAT THE ELEMENT IS DERIVED. WE WOULD EXPECT THAT THE ATTRIBUTE "ARTIFICIALITY" WOULD HAVE VALUE "ARTIFICIAL" IF THE ELEMENT IS TRACED FROM "NONE". HOWEVER, THIS MAY BE UNTRUE IN CASES WHERE AN ARBITRARY HEURISTIC MUST BE EMPLOYED. *).

COMPLEMENTARY RELATIONSHIP: TRACED ("FROM").

SUBJECT: ORIGINATING_REQUIREMENT, DECISION.

OBJECT: ALL EXCEPT ORIGINATING_REQUIREMENT.

ATTRIBUTE: ALTERNATIVES

(* THE ALTERNATIVES THAT HAVE BEEN ENVISIONED TO RESOLVE THE PROBLEM. *).

APPLICABLE TO: DECISION.

VALUE: TEXT.

ATTRIBUTE: ARTIFICIALITY.

APPLICABLE TO: ALL.

VALUE: ARTIFICIAL

(* THE ELEMENT HAS BEEN DEFINED FOR EXPLANATORY OR EXECUTABILITY PURPOSES IN THE REQUIREMENTS STATEMENT AND NEED NOT BE PRESENT IN THE REAL-TIME SOFTWARE. *).

VALUE: IMPLEMENT_PRECISELY

(* THE ELEMENT MUST BE IMPLEMENTED IN THE REAL-TIME SOFTWARE EXACTLY AS DEFINED; NO CHANGES SHOULD BE CONSIDERED UNTIL PERMISSION IS OBTAINED FROM THE REQUIREMENTS ANALYST. *).

VALUE: IMPLEMENT_APPROXIMATELY

(* THE ELEMENT MUST BE IMPLEMENTED IN THE REAL-TIME SOFTWARE, BUT THE PRECISE IMPLEMENTATION IS LEFT TO THE PROCESS DESIGNER. OF COURSE, THE DETAILED IMPLEMENTATION MUST BE VALIDATED BY THE REQUIREMENTS ANALYST. *).

VALUE: VALIDATION

(* THE ELEMENT IS NECESSARY FOR DESCRIBING PERFORMANCE REQUIREMENTS BUT IS NOT REQUIRED IN THE REAL-TIME SOFTWARE. *).

ATTRIBUTE: BETA

(* THIS PROVIDES THE PROCEDURAL CODE (WHICH IS NOT INTERPRETED BY THE RSL PROCESSORS) FOR FUNCTIONAL MODELS. IT IS PASSED TO THE SIMULATION GENERATOR AND, SUBSEQUENTLY, TO THE COMPILER. *).

APPLICABLE TO: ALPHA.

VALUE: TEXT.

ATTRIBUTE: CHOICE

(* THE DECISION FROM AMONG THE ALTERNATIVES WITH THE RATIONALE FOR THE DECISION, *).

APPLICABLE TO: DECISION.

VALUE: TEXT.

ATTRIBUTE: COMPLETENESS.

APPLICABLE TO: ALL.

VALUE: INCOMPLETE

(* THE ELEMENT'S DESCRIPTION IS KNOWN TO BE INCOMPLETE. THEREFORE, READERS SHOULD BE AWARE THAT, EVEN IF ALL RELATIONSHIPS, ATTRIBUTES, AND STRUCTURES ARE STATED, THE ELEMENT IS STILL INCOMPLETE. INFORMATION ABOUT THE ELEMENT SHOULD BE EMPLOYED ONLY AT THE USER'S OWN RISK, *).

VALUE: CHANGEABLE

(* ALTHOUGH ALL RELATIONSHIPS, ATTRIBUTES, AND STRUCTURES MAY BE DEFINED FOR THE ELEMENT, SOME OF THEM WILL PROBABLY BE CHANGED. INFORMATION ABOUT THE ELEMENT IS BELIEVED TO BE CORRECT, BUT IT IS SUBJECT TO CHANGE, *).

VALUE: COMPLETE

(* THE ELEMENT'S DESCRIPTION SHOULD BE ASSUMED TO BE COMPLETE AND WILL PROBABLY NOT CHANGE, *).

ATTRIBUTE: DESCRIPTION

(* TEXTUAL DESCRIPTION, *).

APPLICABLE TO: ALL.

VALUE: TEXT.

ATTRIBUTE: ENTERED_BY.

APPLICABLE TO: ALL.

VALUE: TEXT

(* THE IDENTITY OF THE LAST PERSON TO ENTER INFORMATION ABOUT THE ELEMENT, *).

ATTRIBUTE: EXTRACTOR

(* PROCEDURAL CODE IDENTIFYING FOR WHICH INSTANCES OF FILES AND ENTITIES THE DATA INPUT TO THE VALIDATION_POINT IS TO BE EXTRACTED IN THE REAL-TIME SOFTWARE, *).

APPLICABLE TO: VALIDATION_POINT.

VALUE: TEXT.

ATTRIBUTE: GAMMA

(* THIS PROVIDES THE PROCEDURAL CODE (WHICH IS NOT INTERPRETED BY RSL PROCESSORS) FOR ANALYTIC MODELS. IT IS PASSED TO THE SIMULATION GENERATOR AND, SUBSEQUENTLY, TO THE COMPILER, *).

APPLICABLE TO: ALPHA.

VALUE: TEXT.

ATTRIBUTE: INITIAL_VALUE

(* THE INITIAL_VALUE A DATA ITEM IS REQUIRED TO HAVE IN THE IMPLEMENTED SOFTWARE. THIS VALUE WILL BE ASSUMED BY THE DATA ITEM WHEN IT COMES INTO EXISTENCE IN A SIMULATION, *).

APPLICABLE TO: DATA.

VALUE: NUMERIC.

VALUE: TEXT.

ATTRIBUTE: LOCALITY.
APPLICABLE TO: DATA, FILE.
VALUE: GLOBAL

(* GLOBAL DATA AND FILES MAY BE ASSOCIATED WITH ENTITY_TYPES OR ENTITY-CLASSES OR MAY BE IN THE GLOBAL DATA BASE. *).

VALUE: LOCAL

(* LOCAL DATA AND FILES ARE CREATED AND INITIALIZED FOR EACH ENABLEMENT OF AN R_NET. *).

ATTRIBUTE: MAXIMUM_TIME.
APPLICABLE TO: VALIDATION_PATH.
VALUE: NUMERIC

(* THE MAXIMUM TIME THAT CAN BE TAKEN TO TRAVERSE THE VALIDATION PATH. *).

ATTRIBUTE: MAXIMUM_VALUE

(* THE MAXIMUM_VALUE APPLIES TO DATA VALUES AND EMPLOYS THE UNITS STATED IN THE UNITS ATTRIBUTE. *).

APPLICABLE TO: DATA.
VALUE: NUMERIC.

ATTRIBUTE: MINIMUM_TIME

(* THE MINIMUM TIME THAT CAN BE TAKEN TO TRAVERSE THE VALIDATION PATH. *).

APPLICABLE TO: VALIDATION_PATH.
VALUE: NUMERIC.

ATTRIBUTE: MINIMUM_VALUE

(* THE MINIMUM_VALUE APPLIES TO DATA VALUES AND EMPLOYS THE UNITS IN THE UNITS ATTRIBUTE. *).

APPLICABLE TO: DATA.
VALUE: NUMERIC.

ATTRIBUTE: PROBLEM

(* THE PROBLEM THAT HAS LED TO THE NEED FOR A DECISION. *).

APPLICABLE TO: DECISION.
VALUE: TEXT.

ATTRIBUTE: RANGE

(* THE RANGE OF THE DATA IS ENUMERATED HERE. IT IS MEANINGFUL ONLY IF ENUMERATION IS THE VALUE OF TYPE. *).

APPLICABLE TO: DATA.
VALUE: TEXT

(* THE ALLOWED VALUES ARE SEPARATED BY COMMAS. *).

ATTRIBUTE: RESOLUTION

(* DESCRIBES THE REQUIRED MAXIMUM VALUE OF THE LEAST SIGNIFICANT BIT FOR THE DATA IN UNITS DESCRIBED IN THE UNITS ATTRIBUTE. *).

APPLICABLE TO: DATA.
VALUE: NUMERIC.

ATTRIBUTE: TEST

(* PROCEDURAL CODE WHICH DEFINES THE COMPUTATIONS NECESSARY TO TEST THE SATISFACTION OF A PERFORMANCE_REQUIREMENT USING DATA INPUT TO VALIDATION_POINTS. *).

APPLICABLE TO: PERFORMANCE_REQUIREMENT.
VALUE: TEXT.

ATTRIBUTE: TYPE

(* THE TYPE FOR THE DATA ITEMS WHICH ARE REFERENCED ON P-NETS OR ARE USED IN BETA OR GAMMA SIMULATIONS, *).

APPLICABLE TO: DATA.

VALUE: REAL.

VALUE: INTEGER.

VALUE: BOOLEAN.

VALUE: ENUMERATION

(* THE ALLOWED VALUES MUST BE SPECIFIED IN THE RANGE ATTRIBUTE. *).

ATTRIBUTE: UNITS

(* THE CONFIGURATION MANAGEMENT MANAGER MUST CREATE THE ATTRIBUTE VALUES THAT MAY BE EMPLOYED IN DESCRIBING THE REQUIREMENTS FOR THE DATA PROCESSING SYSTEM UNDER CONSIDERATION, *).

APPLICABLE TO: DATA, VALIDATION_PATH.

VALUE: NAMED.

ATTRIBUTE: USE

(* THE APPLICABILITY OF TYPE AND RANGE ARE SPECIFIED IN THE USE, *).

APPLICABLE TO: DATA.

VALUE: BETA.

VALUE: GAMMA.

VALUE: BOTH

(* BOTH BETA AND GAMMA *).

DATA: CLOCK_TIME

(* A PREDEFINED DATA ITEM WHICH INCREMENTS AT THE SAME RATE AS ENGAGEMENT TIME. EXCEPT FOR ITS INITIAL_VALUE WHICH IS ARBITRARY, CLOCK_TIME MAY BE REGARDED AS ENGAGEMENT TIME. IT HAS NO CLOCK ERROR, *).

LOCALITY: GLOBAL.

TYPE: REAL.

UNITS: SECONDS.

USE: BOTH.

DATA: FOUND

(* A PREDEFINED DATA ITEM WHICH IS SET TO EITHER TRUE OR FALSE AFTER EACH SELECT IN A BETA OR GAMMA. FOUND IS SET TO TRUE IF AN INSTANCE SATISFYING THE SELECTION CRITERION IS LOCATED. OTHERWISE, FOUND IS ASSIGNED THE VALUE FALSE, *).

LOCALITY: GLOBAL.

TYPE: BOOLEAN.

USE: BOTH.

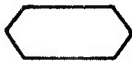
<u>SYMBOL</u>	<u>DEFINITION</u>
	R_NET INITIATION
	SUBNET INITIATION
	INPUT_INTERFACE OR OUTPUT_INTERFACE
	ALPHA REFERENCE
	SUBNET REFERENCE
	OR-NODE
	AND-NODE
	EVENT-NODE
	FOR EACH-NODE
	VALIDATION_POINT-NODE
	SUBNET TERMINATION
	R_NET TERMINATION

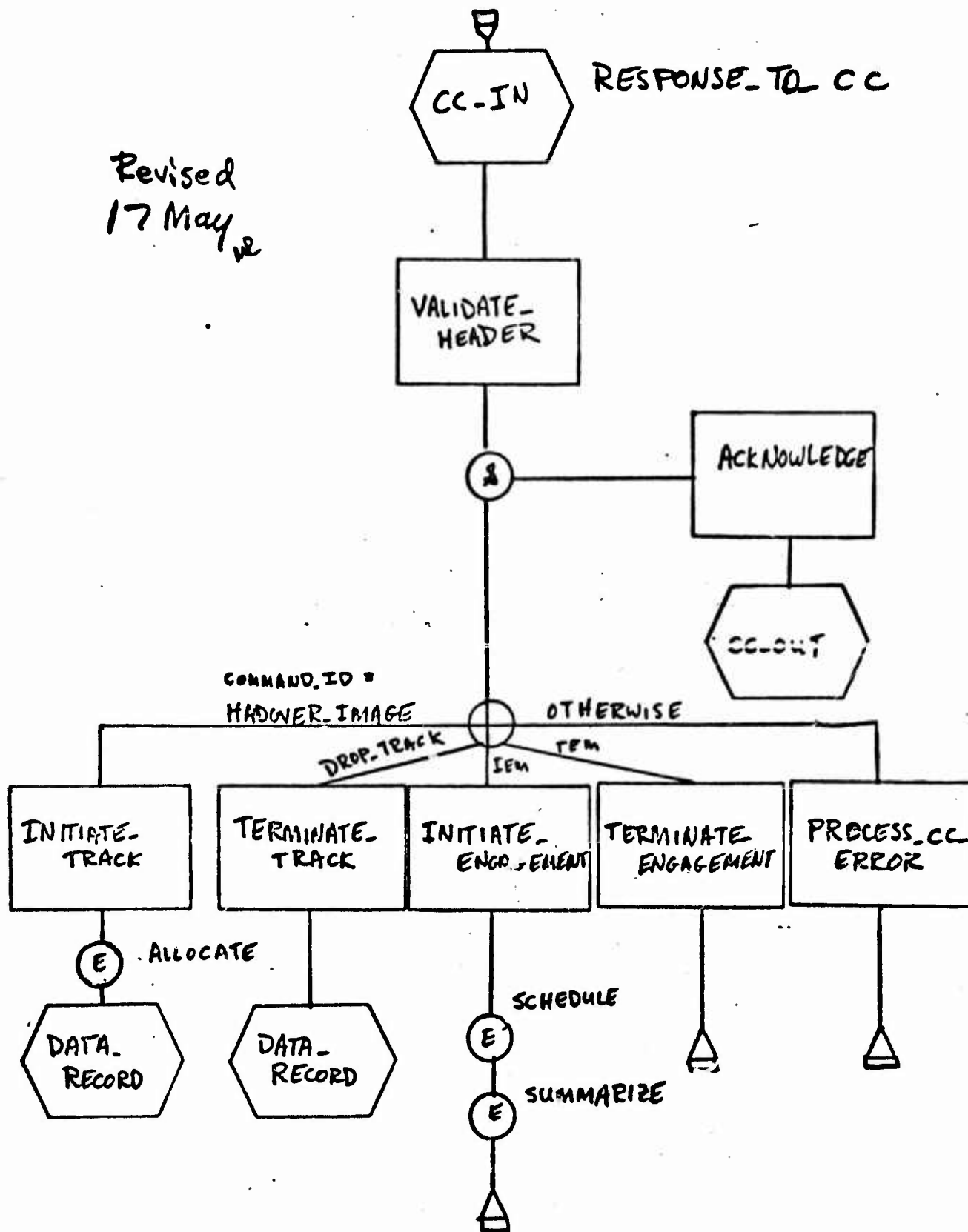
Figure A-1 RSL Symbols

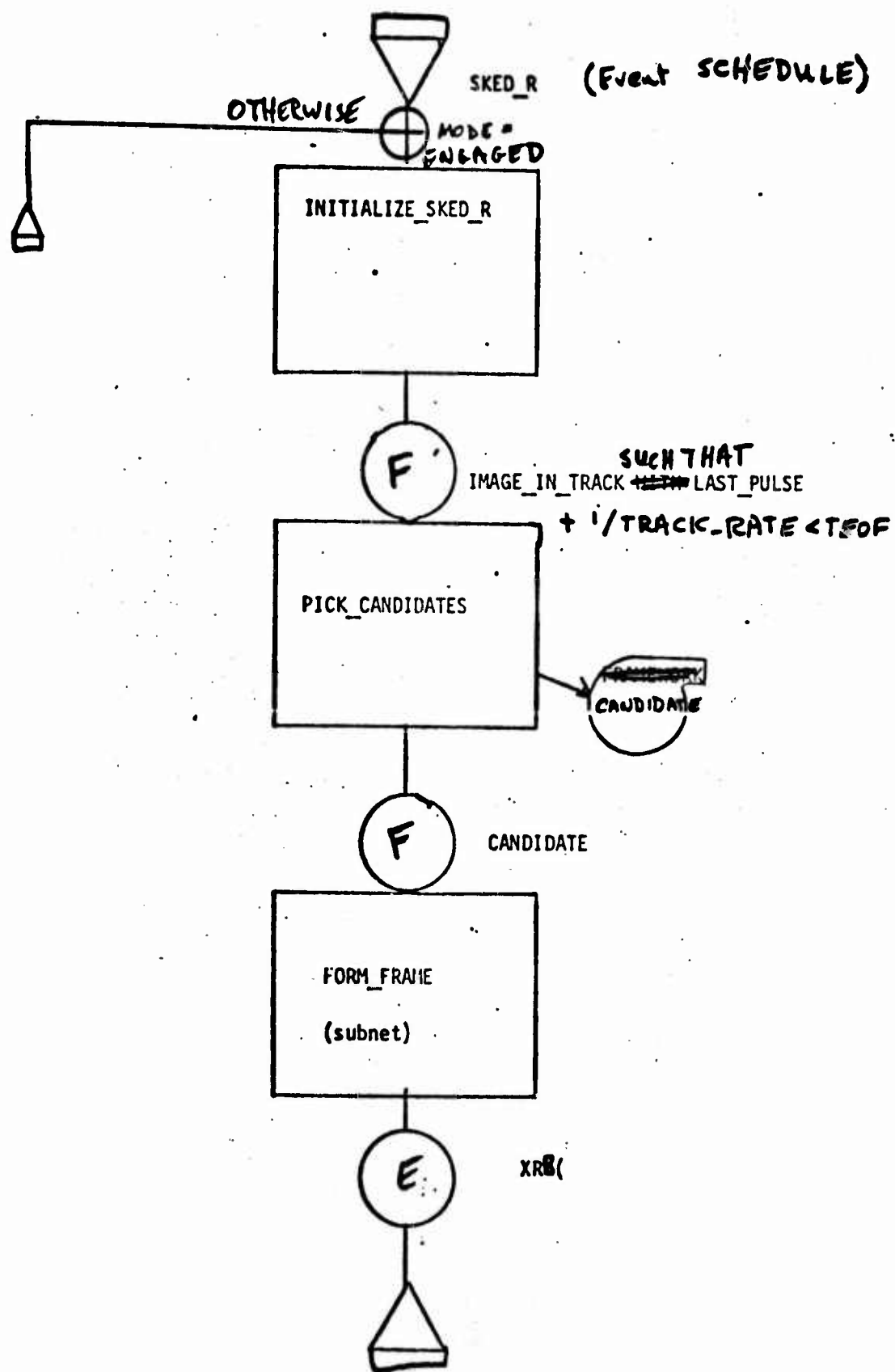
APPENDIX B
DRAFT REPRESENTATION OF THE KERNEL OF TLS

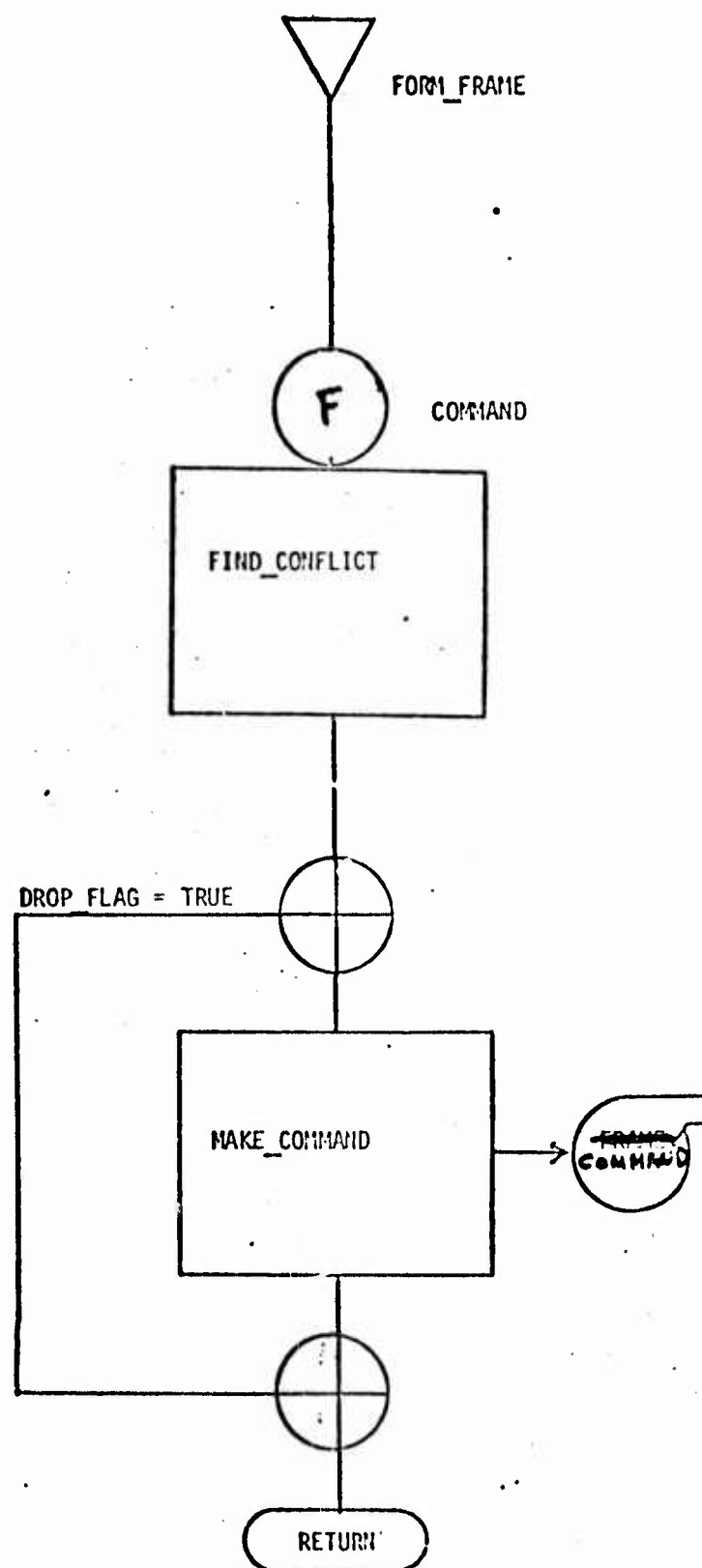
B-1

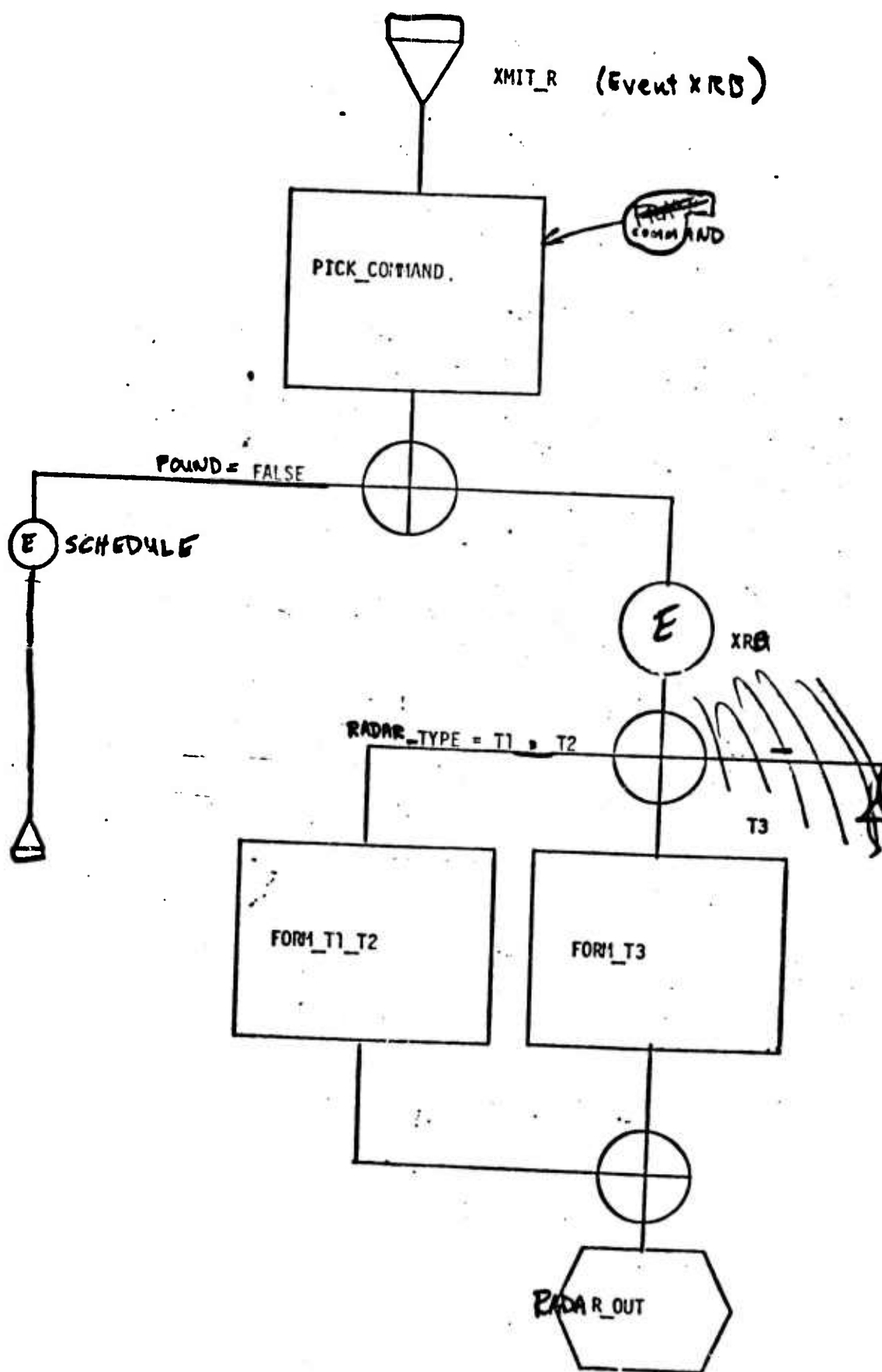
PRECEDING PAGE BLANK NOT FILMED

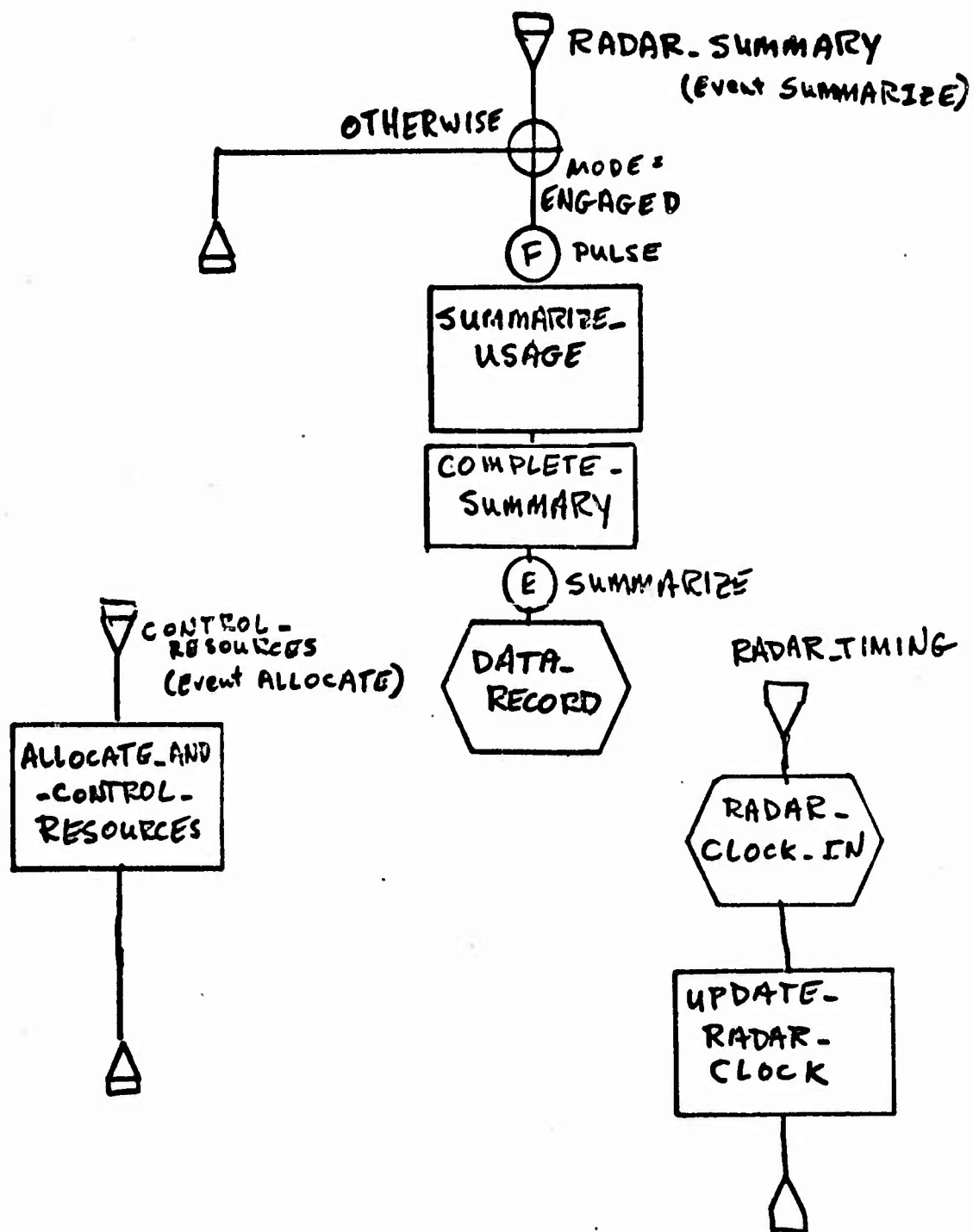
Revised
17 May₇₂

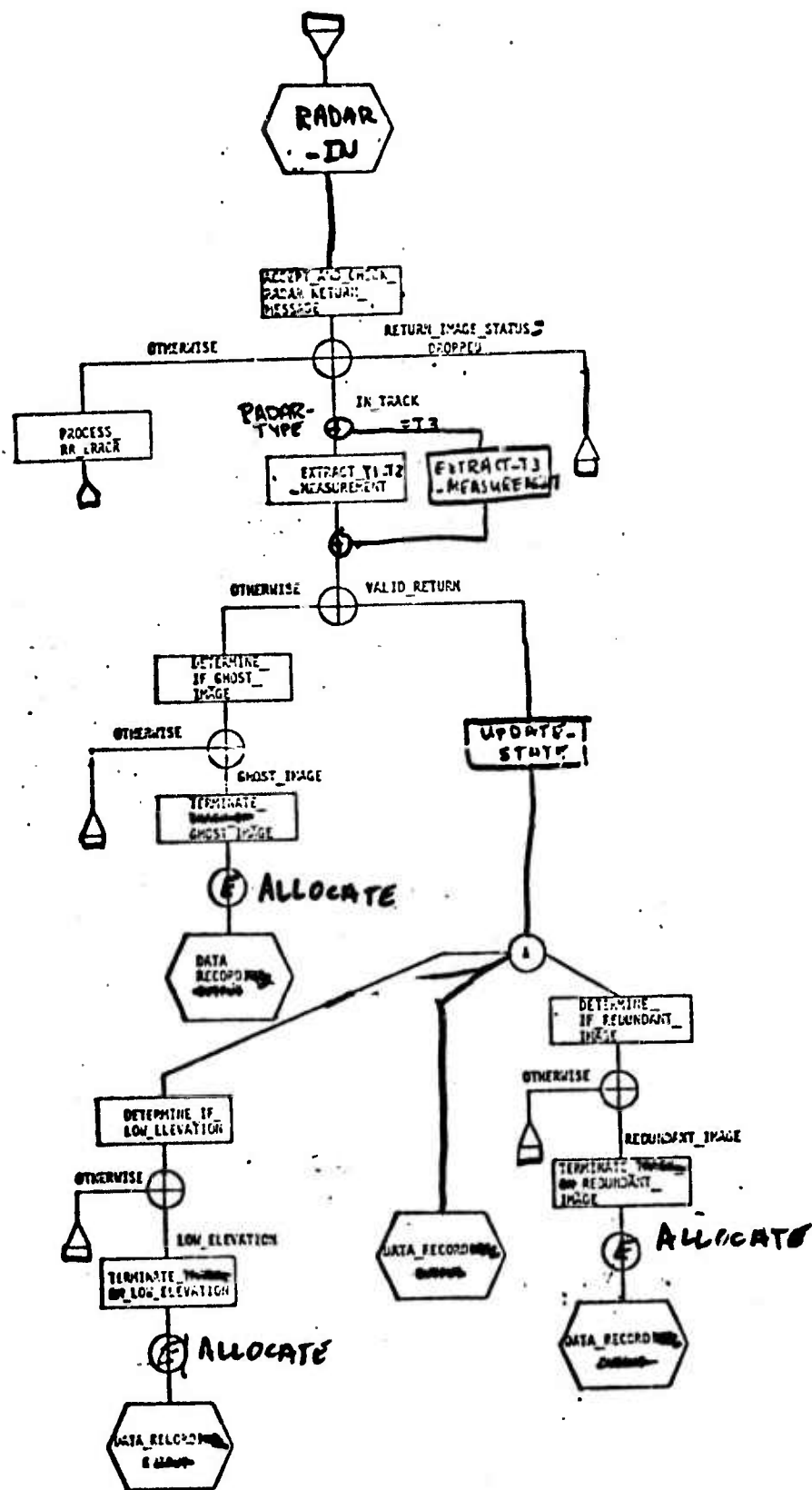












RR-IN

RR-ORDER-ID
RADAR-TYPE

T1-T2-RECEIVE
FILE: T1-T2-DATA
T1-T2-RETURN

T3-RECEIVE
FILE: T3-DATA
T3-RETURN

RR-OUT

RR-ORDER-ID
RADAR-TYPE
TRANSMIT-START

T1-T2-TRANSMIT
FILE: T1-T2-GATE
T1-T2-COMMAND

T3-TRANSMIT
FILE: T3-GATE
T3-COMMAND

RADAR-CLOCK-IN

RADAR-CLOCK-TIME
R-CLOCK-MESSAGE

CC-IN

COMMAND-ID

MODE-CHANGE

HO-ID

INITIAL-STATE
INITIAL-COVARIANCE

HANDOVER

TERMINATION

CC-OUT

~~MESSAGE-TYPE~~

COMMAND-ID

ACKNOWLEDGEMENT

DATA-RECORD

ENGAGEMENT-TIME
RESOURCES

RADAR-USAGE

HO-ID

CURRENT-STATE

STATE-UPDATE

TIME-OF-INITIATION

INITIAL-STATE

TRACK-INITIATION

REASON-FOR-DROP

TIME-OF-DROP

TRACK-TERMINATION

Global Files

CANDIDATE CANDIDATE-IMAGE-ID
PRIORITY (Orders)
CANDIDATE-WAVEFORM
CANDIDATE-ENERGY

COMMAND COMMAND-IMAGE-ID
WINDOW
COMMAND-WAVEFORM
COMMAND-ENERGY
START-TIME (Orders)

WAVEFORM_TABLE WF.NAME
(constant) WF-CHARACTERISTICS

Global Data

FRAME-RATE DELAYS XRA
SUMMARY-RATE DELAYS RSA
MODE

RADAR-MODEL-DATA
ENERGY-BOUND

~~ELEVATION-LIMIT~~

~~ELEVATION-LIMIT~~
ELEVATION-LIMIT
RADAR-CLOCK

(.01 sec)

(.3 sec)

(ENGAGED, STANDBY)

ENTITY-CLASS: PULSE

PULSE-TYPE

TARGET-ID

PULSE-ID

~~RECEIVED-ID~~

XMIT-START

RECEIVED-STOP

T1-T2-XMIT

FILE: T1-T2-WINDOW

T1-T2-PULSE

ACCOUNTED
-FOR

RETURNED-PULSE

LOST-PULSE

RECEIVED-STOP

T3-XMIT

FILE: T3-WINDOW

T3-PULSE

ENTITY-CLASS: IMAGE

IMAGE-ID

ENTRY-TIME

STATE

COVARIANCE

TRACK-RATE

WAVEFORM

IMAGE-IN-TRACK

FILE: TERMINATOR

DROPPED-IMAGE

APPENDIX C

TLS KERNEL

C-1

PRECEDING PAGE BLANK-NOT FILMED

```

R_NET : CC_RESPONSE.
STRUCTURE:
  INPUT_INTERFACE : CC_IN
  ALPHA : VALIDATE_HEADER
  DO
    ALPHA : ACKNOWLEDGE
    OUTPUT_INTERFACE : CC_OUT
  AND
    CONSIDER DATA : COMMAND_ID
    DO
      (HANDOVER_IMAGE)
      ALPHA : TRACK_INITIATE
      EVENT : ALLOCATE
      OUTPUT_INTERFACE : DATA_RECORD
    OR
      (DROP_TRACK)
      ALPHA : TERM_TRACK
      OUTPUT_INTERFACE : DATA_RECORD
    OR
      (INITIATE_ENGAGEMENT_MODE)
      ALPHA : STARTER
      ALPHA : ENGAGEMENT_INITIATION
      EVENT : SCHEDULE
      EVENT : SUMMARIZE
      TERMINATE
    OR
      (TERMINATE_ENGAGEMENT_MODE)
      ALPHA : TERM_ENGAGEMENT
      TERMINATE
    OTHERWISE
      ALPHA : CC_ERROR_PROCESSING
      TERMINATE
    END
  END
END .

R_NET : CONTROL_RESOURCES.
STRUCTURE:
  ALPHA : ALLOCATE_AND_CONTROL_RESOURCES
  TERMINATE
END .

R_NET : RADAR_SUMMARY.
STRUCTURE:
  CONSIDER DATA : MODE
  DO
    (ENGAGED)
    FOR EACH ENTITY_TYPE : RETURNED_PULSE
    DO
      ALPHA : SUMMARIZE_USAGE
    END
    ALPHA : COMPLETE_SUMMARY
    EVENT : SUMMARIZE
    OUTPUT_INTERFACE : DATA_RECORD
  OTHERWISE
    TERMINATE
  END
END .

```



```
R_NET : RADAR_TIMING.  
  STRUCTURE:  
    INPUT_INTERFACE : RADAR_CLOCK_IN  
    ALPHA : UPDATE_RADAR_CLOCK  
    TERMINATE  
  END .
```

```

R_NET : RESPONSE_TO_RADAR.
STRUCTURE:
  INPUT_INTERFACE : RADAR_IN
  ALPHA : ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE
  CONSIDER DATA : RETURN_IMAGE_STATUS
DO
  (IN_TRACK)
  CONSIDER DATA : RADAR_TYPE
DO
  (T3)
  OTHERWISE
  ALPHA : T1_T2_MEASUREMENT_EXTRACTION
END
DO
  (VALID_RETURN)
  ALPHA : UPDATE_STATE
DO
  OUTPUT_INTERFACE : DATA_RECORD
AND
  ALPHA : REDUN_DETERMINATION
DO
  (REDUNDANT_IMAGE)
  ALPHA : REDUN_TERMINATION
  EVENT : ALLOCATE
  OUTPUT_INTERFACE : DATA_RECORD
  OTHERWISE
  TERMINATE
END
AND
  ALPHA : LOW_ELEVATION_DETERMINATION
DO
  (LOW_ELEVATION)
  ALPHA : LOW_TERMINATION
  EVENT : ALLOCATE
  OUTPUT_INTERFACE : DATA_RECORD
  OTHERWISE
  TERMINATE
END
END
OTHERWISE
  ALPHA : GHOST_DETERMINATION
DO
  (GHOST_IMAGE)
  ALPHA : GHOST_TERMINATION
  EVENT : ALLOCATE
  OUTPUT_INTERFACE : DATA_RECORD
  OTHERWISE
  TERMINATE
END
END
OR
  (DROPPED)
  TERMINATE
OTHERWISE
  ALPHA : RR_ERROR_PROCESSING
  TERMINATE
END
END .

```

```

R_NET : SKED_R.
STRUCTURE:
  CONSIDER DATA : MODE
  DO
    (ENGAGED)
    ALPHA : INITIALIZE_SKED_R
    FOR EACH ENTITY_TYPE : IMAGE_IN_TRACK SUCH THAT
      (LAST_PULSE*(1.0/TRACK_RATE)<TEOF)
    DO
      ALPHA : PICK_CANDIDATES END
    FOR EACH FILE : CANDIDATE
    DO
      SUBNET : FORM_FRAME END
    EVENT : XRB
    TERMINATE
  OTHERWISE
    TERMINATE
  END
END .

```

```

R_NET : XMIT_R.
STRUCTURE:
  ALPHA : PICK_COMMAND
  DO
    (FOUND)
    EVENT : XRB
    CONSIDER DATA : RADAR_TYPE
    DO
      (T3)
      ALPHA : FORM_T3
    OTHERWISE
      ALPHA : FORM_T1_T2
    END
    OUTPUT_INTERFACE : RADAR_OUT
  OTHERWISE
    EVENT : SCHEDULE
    TERMINATE
  END
END .

```

```
SUBNET : FORM_FRAME.  
  STRUCTURE:  
    ALPHA : FIND_CONFLICT  
    DO  
      (NOT DROP_FLAG)  
      ALPHA : MAKE_COMMAND  
    OTHERWISE  
    END  
  RETURN  
END .
```

ENTITY_CLASS : IMAGE
 ASSOCIATES
 DATA : ENTRY_TIME
 DATA : IMAGE_ID
 COMPOSED
 ENTITY_TYPE : DROPPED_IMAGE
 ASSOCIATES
 FILE : TERMINATOR
 CONTAINS
 DATA : DROP_REASON
 DATA : DROP_TIME
 ENTITY_TYPE : IMAGE_IN_TRACK
 ASSOCIATES
 DATA : COVARIANCE
 DATA : LAST_PULSE
 DATA : STATE
 DATA : TRACK_RATE
 DATA : WAVEFORM

ENTITY_CLASS : PULSE
 ASSOCIATES
 DATA : PULSE_ID
 DATA : PULSE_TYPE
 DATA : TARGET_ID
 DATA : XMIT_START
 COMPOSED
 ENTITY_TYPE : LOST_PULSE
 ASSOCIATES
 DATA : ACCOUNTED_FOR
 ENTITY_TYPE : RETURNED_PULSE
 ASSOCIATES
 DATA : ACCOUNTED_FOR
 ENTITY_TYPE : T1_T2_PULSE
 ASSOCIATES
 DATA : RECEIVE_STOP
 DATA : T1_T2_XMIT
 ASSOCIATES
 FILE : T1_T2_WINDOW
 CONTAINS
 DATA : T1_T2_WINDOW_DATA
 ENTITY_TYPE : T3_PULSE
 ASSOCIATES
 DATA : RECEIVE_STOP
 DATA : T3_XMIT
 ASSOCIATES
 FILE : T3_WINDOW
 CONTAINS
 DATA : T3_WINDOW_DATA

INPUT_INTERFACE : CC_IN
 PASSES
 MESSAGE : HANDOVER
 MADE
 DATA : COMMAND_ID
 DATA : HO_ID
 DATA : INITIAL_COVARIANCE
 DATA : INITIAL_STATE
 MESSAGE : MODE_CHANGE
 MADE
 DATA : COMMAND_ID
 MESSAGE : TERMINATION
 MADE
 DATA : COMMAND_ID
 DATA : HO_ID

INPUT_INTERFACE : RADAR_CLOCK_IN
 PASSES
 MESSAGE : R_CLOCK_MESSAGE
 MADE
 DATA : RADAR_CLOCK_TIME

INPUT_INTERFACE : RADAR_IN

PASSES

MESSAGE : T1_T2_RETURN

MADE

DATA : RADAR_TYPE

DATA : RR_ORDER_ID

DATA : T1_T2_RECEIVE

INCLUDES

DATA : ALPHA_ERROR

DATA : BETA_ERROR

DATA : T1T2RTN_ERROR_REPORT

INCLUDES

DATA : REASON_FOR_TRANSMISSION_FAILURE

DATA : WAKE_ARRAY

INCLUDES

DATA : AVERAGE_SIGNAL_POWER

DATA : THRESHOLD_DOWN_CROSSING_TIME

DATA : THRESHOLD_UP_CROSSING_TIME

MADE

FILE : T1_T2_DATA

CONTAINS

DATA : T1_T2_RECORD

INCLUDES

DATA : NOISE_LEVEL

DATA : RANGE_MARK_INFORMATION

INCLUDES

DATA : RANGE_MARK_TIME

DATA : SIGNAL_AMPLITUDE

MESSAGE : T3_RETURN

MADE

DATA : RADAR_TYPE

DATA : RR_ORDER_ID

DATA : T3_RECEIVE

INCLUDES

DATA : ALPHA_ERROR

DATA : BETA_ERROR

DATA : T3RTN_ERROR_REPORT

INCLUDES

DATA : REASON_FOR_TRANSMISSION_FAILURE

DATA : WAKE_ARRAY

INCLUDES

DATA : AVERAGE_SIGNAL_POWER

DATA : THRESHOLD_DOWN_CROSSING_TIME

DATA : THRESHOLD_UP_CROSSING_TIME

MADE

FILE : T3_DATA

CONTAINS

DATA : T3_RECORD

INCLUDES

DATA : NOISE_LEVEL

DATA : RANGE_MARK_INFORMATION

INCLUDES

DATA : RANGE_MARK_TIME

DATA : SIGNAL_AMPLITUDE

OUTPUT_INTERFACE : CC_OUT
PASSES
MESSAGE : ACKNOWLEDGEMENT
MADE
DATA : COMMAND_ID

OUTPUT_INTERFACE : DATA_RECORD
PASSES
MESSAGE : RADAR_USAGE
MADE
DATA : DATA_RECORD_TYPE
DATA : ENGAGEMENT_TIME
DATA : RESOURCES
MESSAGE : STATE_UPDATE
MADE
DATA : CURRENT_STATE
DATA : DATA_RECORD_TYPE
DATA : HO_ID
MESSAGE : TRACK_INITIATION
MADE
DATA : DATA_RECORD_TYPE
DATA : HO_ID
DATA : INITIAL_STATE
DATA : TIME_OF_INITIATION
MESSAGE : TRACK_TERMINATION
MADE
DATA : DATA_RECORD_TYPE
DATA : HO_ID
DATA : REASON_FOR_DROP
DATA : TIME_OF_DROP

OUTPUT_INTERFACE : RADAR_OUT

PASSES

MESSAGE : T1_T2_COMMAND

MADE

DATA : RADAR_TYPE

DATA : RR_ORDER_ID

DATA : TRANSMIT_START

DATA : T1_T2_TRANSMIT

INCLUDES

DATA : ALPHA_PHASE_TAPER

DATA : BETA_PHASE_TAPER

DATA : RECEIVE_INFORMATION

INCLUDES

DATA : LENGTH_OF_RECEIVE

DATA : RECEIVE_START_TIME

DATA : RECEIVER_GAIN_SETTING

MADE

FILE : T1_T2_GATE

CONTAINS

DATA : T1_T2_GATE_DATA

INCLUDES

DATA : ACCEPTANCE_THRESHOLD

DATA : GATE_LENGTH

DATA : GATE_START_TIME

DATA : RANGE_MARK_GENERATION_TECHNIQUE

DATA : SIGNAL_PROCESSING_MODE

DATA : THRESHOLD_TYPE

MESSAGE : T3_COMMAND

MADE

DATA : RADAR_TYPE

DATA : RR_ORDER_ID

DATA : TRANSMIT_START

DATA : T3_TRANSMIT

INCLUDES

DATA : ALPHA_PHASE_TAPER

DATA : BETA_PHASE_TAPER

DATA : RECEIVE_INFORMATION

INCLUDES

DATA : LENGTH_OF_RECEIVE

DATA : RECEIVE_START_TIME

DATA : RECEIVER_GAIN_SETTING

MADE

FILE : T3_GATE

CONTAINS

DATA : T3_GATE_DATA

INCLUDES

DATA : ACCEPTANCE_THRESHOLD

DATA : GATE_LENGTH

DATA : GATE_START_TIME

DATA : RANGE_MARK_TECHNIQUE

DATA : SIGNAL_PROCESSING_MODE

DATA : THRESHOLD_TYPE

FILE : CANDIDATE

CONTAINS

DATA : CANDIDATE_ENERGY
DATA : CANDIDATE_IMAGE_ID
DATA : CANDIDATE_WAVEFORM
DATA : PRIORITY

FILE : COMMAND

CONTAINS

DATA : COMMAND_ENERGY
DATA : COMMAND_IMAGE_ID
DATA : COMMAND_WAVEFORM
DATA : START_TIME
DATA : WINDOW

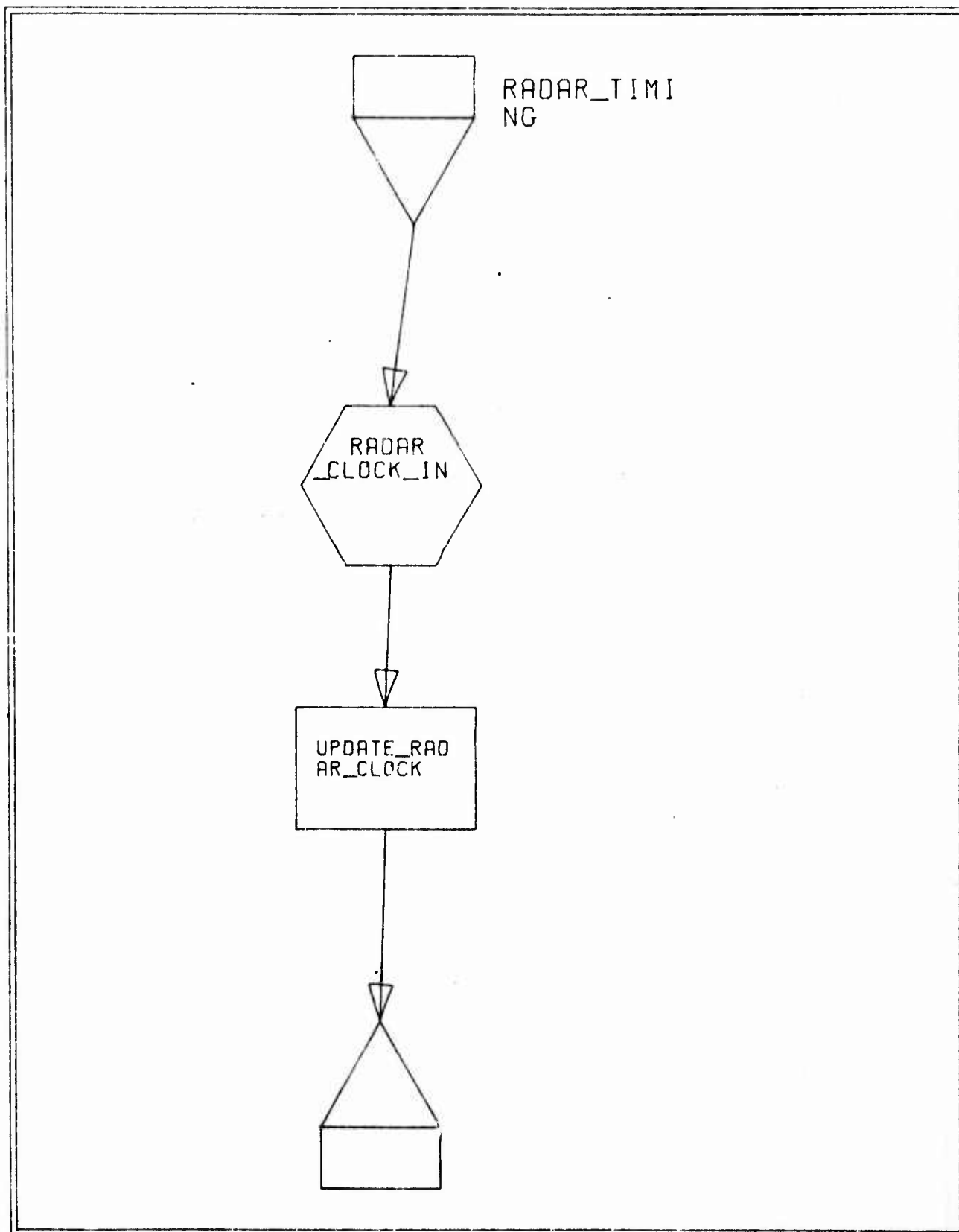
FILE : WAVEFORM_TABLE

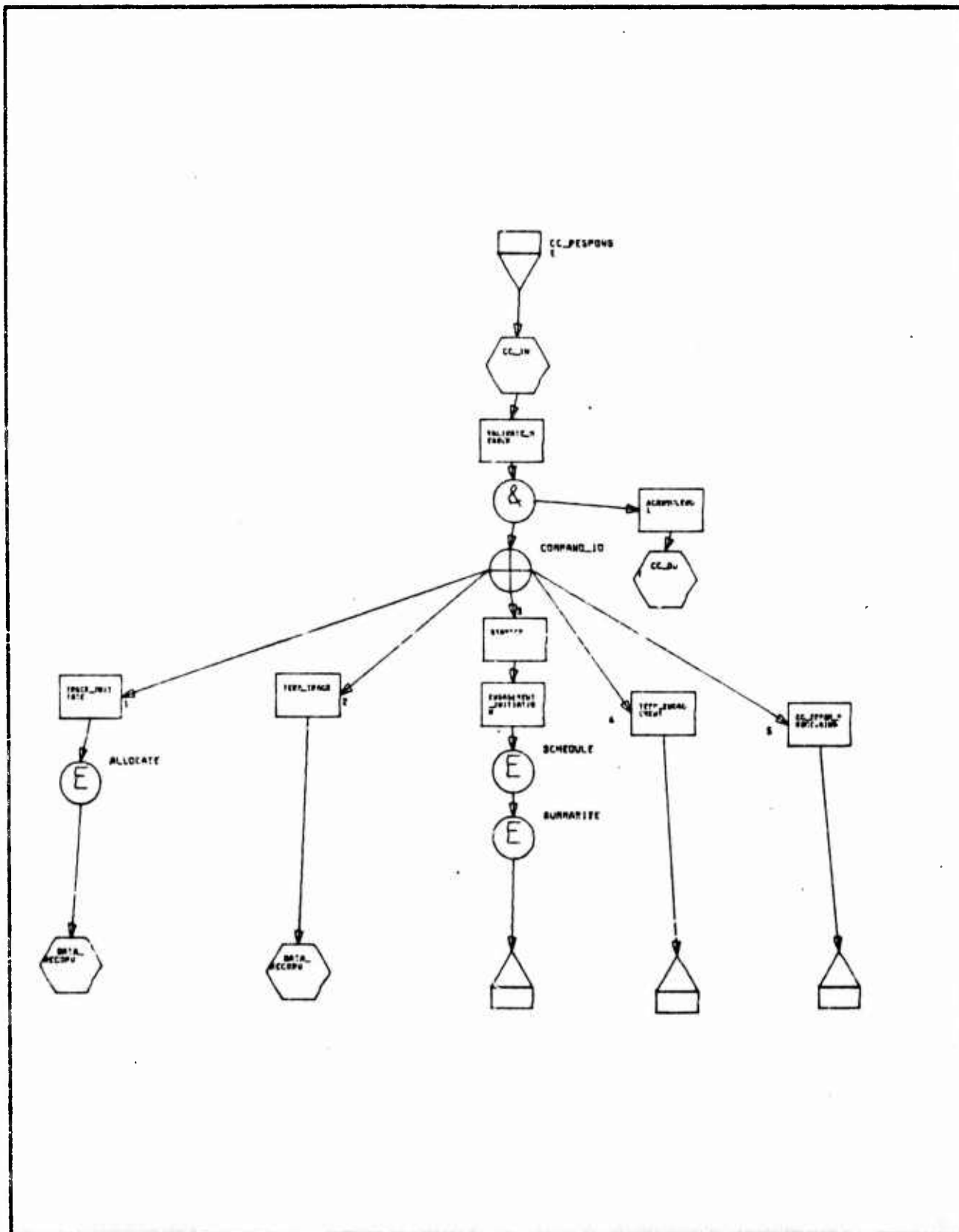
CONTAINS

DATA : WF_CHARACTERISTICS
DATA : WF_NAME

DATA : CLOCK_TIME
DATA : DELTAT
DATA : DROP_FLAG
DATA : ELEVATION_LIMIT
DATA : ENERGY_BOUND
DATA : FOUND
DATA : FRAME_RATE
DATA : GHOST_IMAGE
DATA : IST
DATA : LAST_ALLOCATE
DATA : LOW_ELEVATION
DATA : MODE
DATA : RADAR_CLOCK
DATA : RADAR_MEASUREMENT
DATA : RADAR_MODEL_DATA
DATA : REDUNDANT_IMAGE
DATA : RETURN_IMAGE_STATUS
DATA : RR_ORDER_IDC
DATA : SUMMARY_RATE
DATA : TEOF
DATA : VALID_RETURN

APPENDIX D
TLS REQUIREMENTS NETWORKS



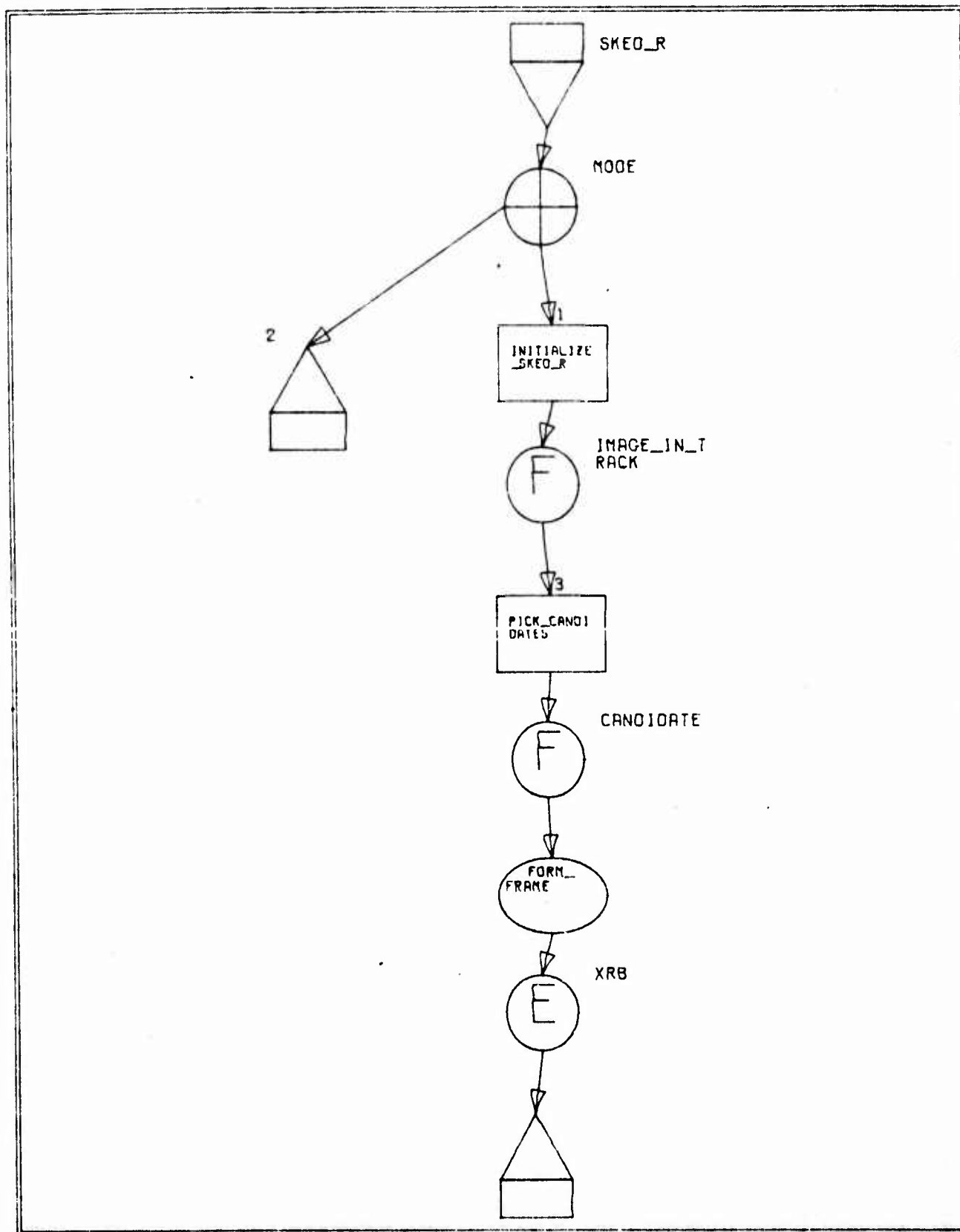


CC_RESPONSE
BRANCH LEGEND

BRANCH NO.	ORDINAL VALUE	CONDITIONAL EXPRESSION
1		(HANDOVER_IMAGE)
2		(DROP_TRACK)
3		(INITIATE_ENGAGEMENT_MODE)
4		(TERMINATE_ENGAGEMENT_MODE)
5		OTHERWISE

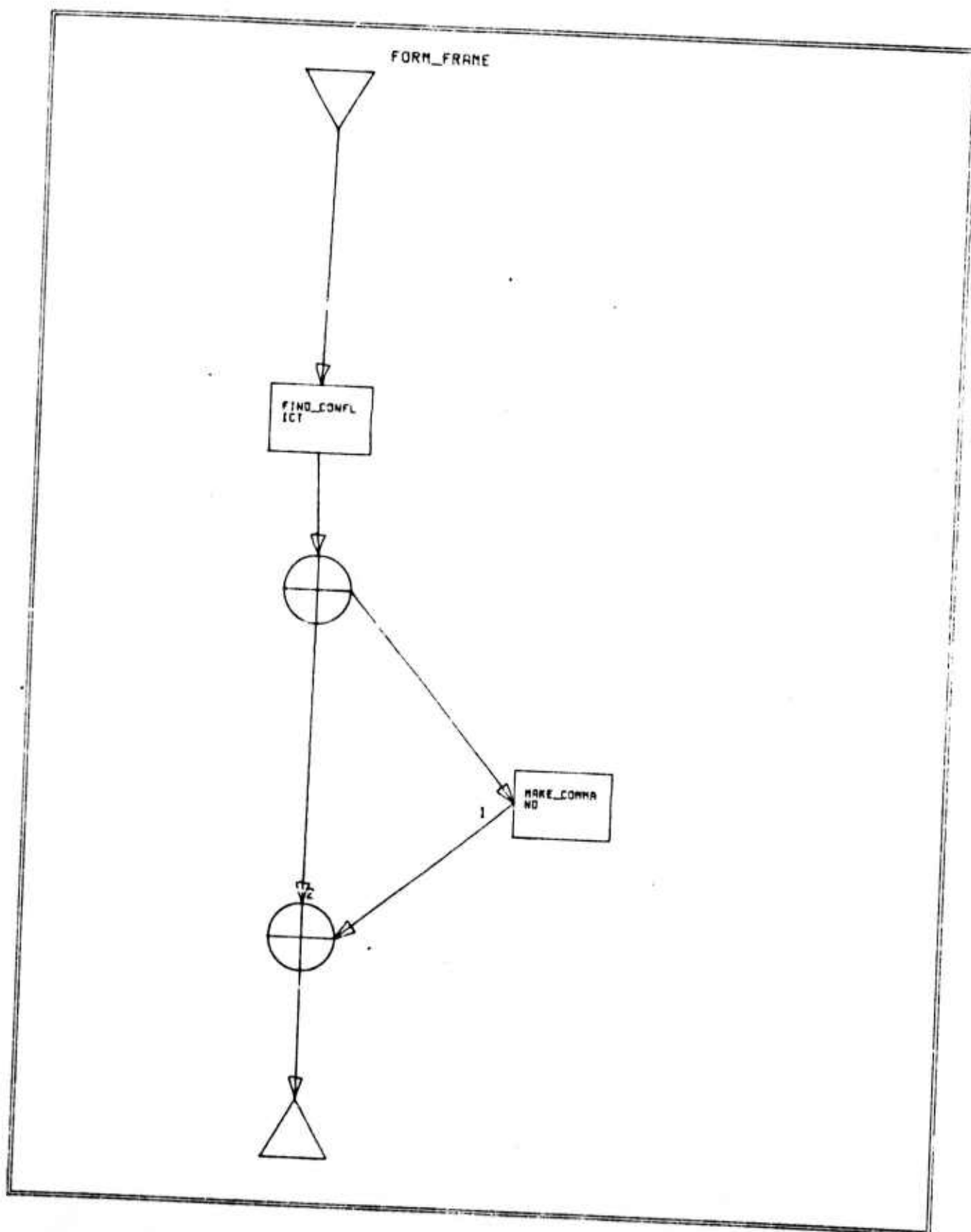
RESPONSE_TO_RAORR
BRANCH LEGEND

BRANCH NO.	ORDINAL VALUE	CONCITIONAL EXPRESSION
1		(IN_TRACK)
2		(DROPPED)
3		OTHERWISE
4		(T3)
5		OTHERWISE
6		(VALID_RETURN)
7		OTHERWISE
8		(REDUNDANT_IMAGE)
9		OTHERWISE
10		(LOW_ELEVATION)
11		OTHERWISE
12		(GHOST_IMAGE)
13		OTHERWISE



SKED_R
BRANCH LEGEND

BRANCH NO.	ORDINAL VALUE	CONDITIONAL EXPRESSION
1		(ENGAGED)
2		OTHERWISE
3		(LAST_PULSE+(1.0/TRACK_RATE)<TEOF)



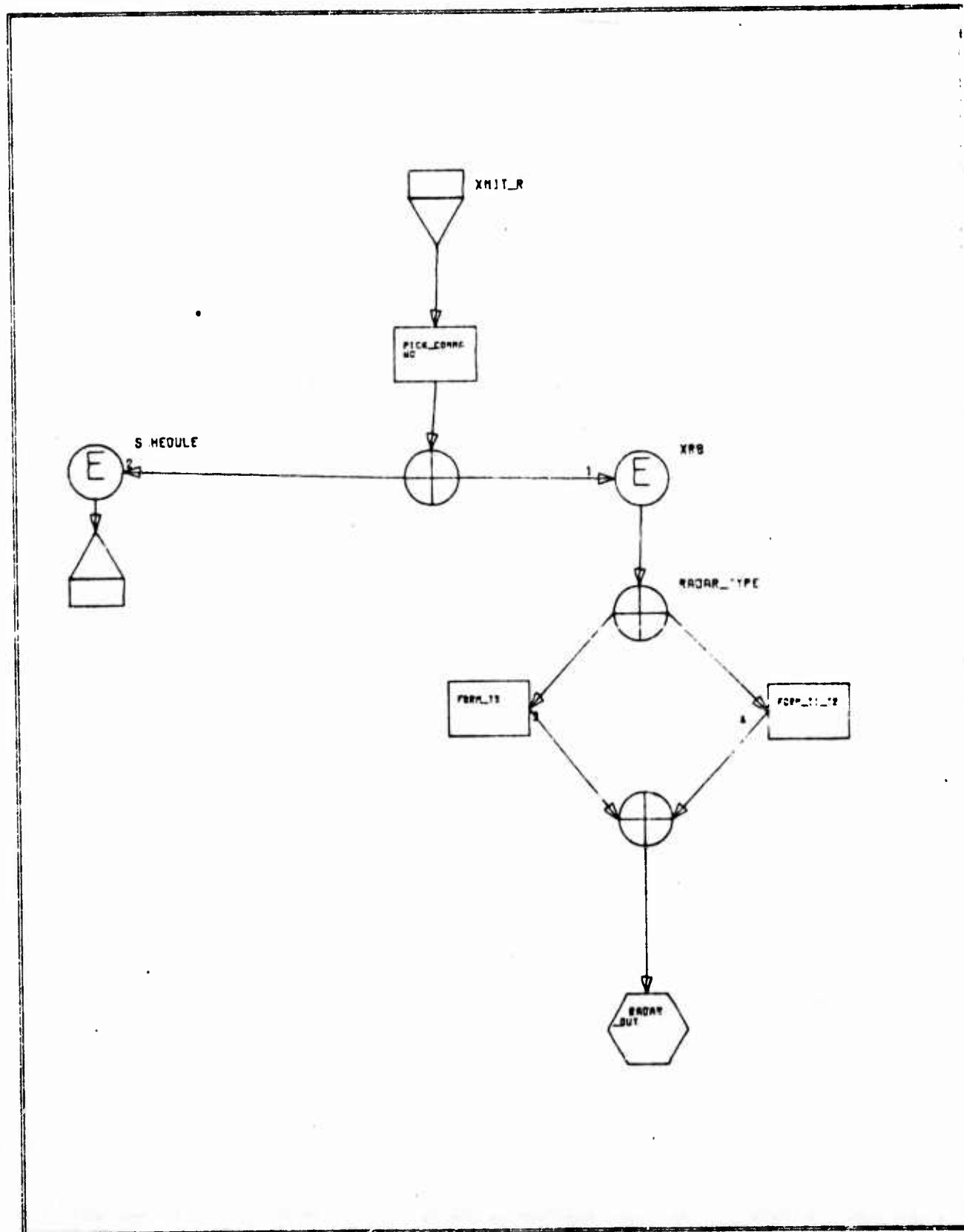
FORM FRAME
BRANCH LEGEND

BRANCH
NO. ORDINAL
 VALUE

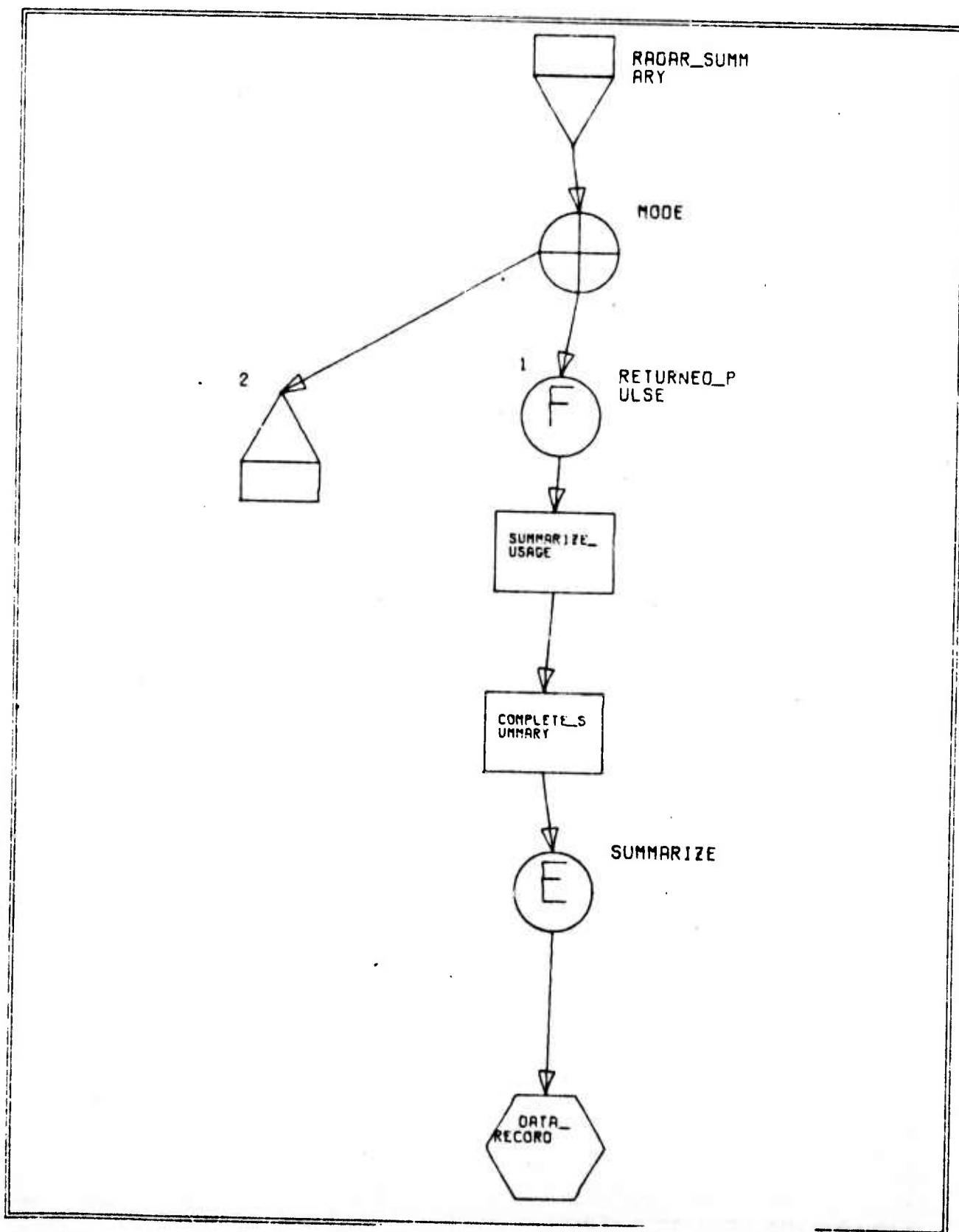
CONDITIONAL EXPRESSION

1
2

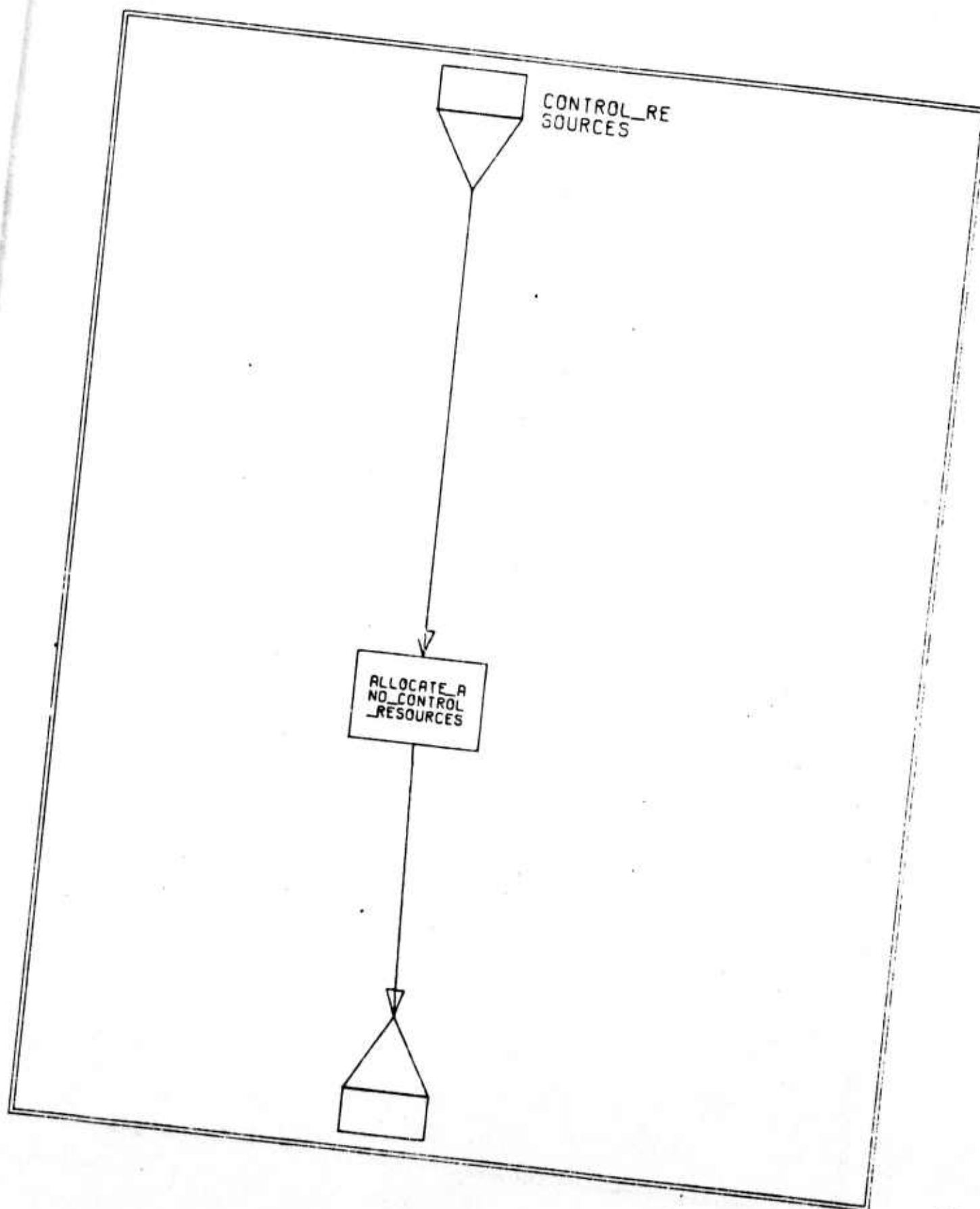
(NOT DROP_FLAG)
OTHERWISE



XMIT_R BRANCH LEGEND		
BRANCH NO.	ORDINAL VALUE	CONDITIONAL EXPRESSION
1		(FOUND)
2		OTHERWISE
3		(T3)
4		OTHERWISE



RADAR SUMMARY BRANCH LEGEND		
BRANCH NO.	ORIGINAL VALUE	CONDITIONAL EXPRESSION
1		(ENGAGED)
2		OTHERWISE



APPENDIX E
COMPLETE TLS DATA BASE

ALPHA : ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.

BETA:

"VAR X:REAL;

BEGIN SELECT FIRST FROM PULSE SUCH THAT

(PULSE_ID=RR_ORDER_ID);

IF NOT FOUND THEN RETURN_IMAGE_STATUS:=NO_ORDER

ELSE IF (RADAR_TYPE<>PULSE_TYPE) THEN

RETURN_IMAGE_STATUS:=WRONG_ORDER

ELSE BEGIN SET RETURNED_PULSE;

ACCOUNTED_FOR:=NEITHER;

SELECT FIRST FROM IMAGE_IN_TRACK

SUCH THAT (IMAGE_ID=TARGET_ID);

IF NOT FOUND THEN RETURN_IMAGE_STATUS:=DROPPED

ELSE BEGIN RETURN_IMAGE_STATUS:=IN_TRACK;

X:=RECEIVE_STOP;

FOR EACH PULSE SUCH THAT

(RECEIVE_STOP<X AND (PULSE_TYPE=T1 OR

PULSE_TYPE=T2 OR PULSE_TYPE=T3)) DO

SET LOST_PULSE;

ACCOUNTED_FOR:=NEITHER;

ENDFOREACH

END

END

END;".

INPUTS:

DATA: IMAGE_ID .

DATA: PULSE_ID

DATA: PULSE_TYPE

DATA: RADAR_TYPE

DATA: RECEIVE_STOP

DATA: RR_ORDER_ID

DATA: TARGET_ID.

OUTPUTS:

DATA: RETURN_IMAGE_STATUS.

SETS:

ENTITY_TYPE: LOST_PULSE

ENTITY_TYPE: RETURNED_PULSE.

EQUATED TO:

SYNONYM: CKRADMES.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

ALPHA : ACKNOWLEDGE.

BETA:

"BEGIN

FORM ACKNOWLEDGEMENT

END;"

FORMS:

MESSAGE: ACKNOWLEDGEMENT.

REFERRED BY:

R_NET : CC_RESPONSE.

DECISION: TRACK_PERFORMANCE_ALLOCATION

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_8_PERFORMANCE

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_0_FUNCTIONAL

ORIGINATING_REQUIREMENT: DPSPR_3_2_4_8_FUNCTIONAL.

REFERRED BY:

R_NET : CONTROL_RESOURCES.

ALPHA : CC_ERROR_PROCESSING.

BETA:

"BEGIN

END;"

REFERRED BY:

R_NET : CC_RESPONSE.

ALPHA : ALLOCATE_AND_CONTROL_RESOURCES.

BETA:

"VAR E,PWR,DELTA_TIME:REAL;J:INTEGER;

BEGIN

E:=0.0;

FOR EACH RETURNED_PULSE DO

IF ACCOUNTED_FOR=SUMMED THEN

BEGIN

SELECT FIRST FROM WAVEFORM_TABLE SUCH THAT
(WF_NAME=PULSE_TYPE);

IF FOUND THEN E:=E+WF_CHARACTERISTICS;

DESTROY PULSE;

END

ELSE IF ACCOUNTED_FOR= NEITHER THEN

BEGIN

SELECT FIRST FROM WAVEFORM_TABLE SUCH THAT
(WF_NAME=PULSE_TYPE);

IF FOUND THEN E:=E+WF_CHARACTERISTICS;

ACCOUNTED_FOR:=COUNTED;

END;

ENDFOREACH;

FOR EACH LOST_PULSE DO

SELECT FIRST FROM WAVEFORM_TABLE SUCH THAT
(WF_NAME=PULSE_TYPE);

IF FOUND THEN E:=E+WF_CHARACTERISTICS;

DESTROY PULSE;

ENDFOREACH;

DELTA_TIME:=CLOCK_TIME-LAST_ALLOCATE;

LAST_ALLOCATE:=CLOCK_TIME;

IF E>0.0 THEN

BEGIN

PWR:=E/DELTA_TIME;

J:=0;

FOR EACH IMAGE_IN_TRACK DO J:=J+1;ENDFOREACH;

FOR EACH IMAGE_IN_TRACK DO

SELECT FIRST FROM WAVEFORM_TABLE SUCH THAT
(WF_NAME=WAVEFORM);

IF FOUND THEN TRACK_RATE:=(PWR/J)/WF_CHARACTERISTICS;

ENDFOREACH;

END

END;".

DESTROYS:

ENTITY_CLASS: PULSE.

INPUTS:

DATA: ACCOUNTED_FOR

DATA: CLOCK_TIME

DATA: IMAGE_ID

DATA: LAST_ALLOCATE

DATA: PULSE_TYPE

DATA: WAVEFORM

FILE: WAVEFORM_TABLE.

OUTPUTS:

DATA: ACCOUNTED_FOR

DATA: LAST_ALLOCATE

DATA: TRACK_RATE.

TRACED FROM:

ALPHA : COMPLETE_SUMMARY.

BETA:

"BEGIN FORM RADAR_USAGE;

DATA_RECORD_TYPE:=RADAR_USAGE_REPORT;

ENGAGEMENT_TIME:=CLOCK_TIME

END;".

FORMS:

MESSAGE: RADAR_USAGE.

INPUTS:

DATA: CLOCK_TIME.

OUTPUTS:

DATA: DATA_RECORD_TYPE

DATA: ENGAGEMENT_TIME.

REFERRED BY:

R_NET : RADAR_SUMMARY.

ALPHA : ENGAGEMENT_INITIATION.

BETA:

"CONST X=ENGAGED;

BEGIN MODE:=X

END;".

OUTPUTS:

DATA: MODE.

REFERRED BY:

R_NET : CC_RESPONSE.

ALPHA : FIND_CONFLICT.

BETA:

"BEGIN

DROP_FLAG:=FALSE;

FOR EACH COMMAND DO

IF START_TIME>TEOF THEN BEGIN

DROP_FLAG:=TRUE;

DESTROY CANDIDATE END;

ENDFOREACH;

END;".

DESCRIPTION:

"COMPARES TRANSMIT RECEIVE WINDOW OF THE CANDIDATE WITH
THOSE OF THE THEN_CURRENT COMMAND FOR CONFLICT. IF
A CONFLICT IS FOUND DROP_FLAG IS SET TRUE.".

INPUTS:

DATA: START_TIME

DATA: TEOF.

OUTPUTS:

DATA: DROP_FLAG.

TRACED FROM:

DECISION: TRACK_PERFORMANCE_ALLOCATION

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_PERFORMANCE

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_E_FUNCTIONAL.

REFERRED BY:

SUBNET : FORM_FRAME.

ALPHA : FORM_T1_T2.

BETA:

"BEGIN

FORM T1_T2_COMMAND;

T1_T2_TRANSMIT:=0.0;

CREATE T1_T2_GATE;

T1_T2_GATE_DATA:=0.0;

SET T1_T2_PULSE;

RECEIVE_STOP:=START_TIME;

T1_T2_XMIT:=0.0;

CREATE T1_T2_WINDOW;

T1_T2_WINDOW_DATA:=0.0;

END;"

FORMS:

MESSAGE: T1_T2_COMMAND.

INPUTS:

FILE: COMMAND.

OUTPUTS:

DATA: RECEIVE_STOP

DATA: T1_T2_TRANSMIT

DATA: T1_T2_XMIT

FILE: T1_T2_GATE

FILE: T1_T2_WINDOW.

SETS:

ENTITY_TYPE: T1_T2_PULSE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_C_FUNCTIONAL.

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_E_FUNCTIONAL.

REFERRED BY:

R_NET : XMIT_R.

ALPHA : FORM_T3.

BETA:

"BEGIN

FORM T3_COMMAND;

T3_TRANSMIT:=0.0;

CREATE T3_GATE;

T3_GATE_DATA:=0.0;

SET T3_PULSE;

T3_XMIT:=0.0;

RECEIVE_STOP:=START_TIME;

CREATE T3_WINDOW;

T3_WINDOW_DATA:=0.0;

END;"

FORMS:

MESSAGE: T3_COMMAND.

INPUTS:

FILE: COMMAND.

OUTPUTS:

DATA: RECEIVE_STOP

DATA: T3_TRANSMIT

DATA: T3_XMIT

FILE: T3_GATE

FILE: T3_WINDOW.

SETS:

ENTITY_TYPE: T3_PULSE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_C_FUNCTIONAL.

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_E_FUNCTIONAL.

REFERRED BY:

R_NET : XMIT_R.

ALPHA : GHOST_DETERMINATION.

BETA:

"BEGIN GHOST_IMAGE:= (RADAR_MEASUREMENT>=10.0) END!";

INPUTS:

DATA: RADAR_MEASUREMENT.

OUTPUTS:

DATA: GHOST_IMAGE.

TRACED FROM:

DECISION: TRACK_PERFORMANCE_ALLOCATION

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_FUNCTIONAL

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_PERFORMANCE

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_C_FUNCTIONAL.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

ALPHA : GHOST_TERMINATION.

BETA:

"BEGIN SET DROPPED_IMAGE;

FORM TRACK_TERMINATION;

CREATE TERMINATOR;

HO_ID:=IMAGE_ID;

REASON_FOR_DROP:=GHOST;

DROP_REASON:=GHOST;

TIME_OF_DROP:=CLOCK_TIME;

DATA_RECORD_TYPE:=TRACK_TERMINATION_REPORT;

DROP_TIME:=CLOCK_TIME

END!";

FORMS:

MESSAGE: TRACK_TERMINATION.

INPUTS:

DATA: CLOCK_TIME

DATA: IMAGE_ID.

OUTPUTS:

DATA: DATA_RECORD_TYPE

DATA: HO_ID

DATA: REASON_FOR_DROP

DATA: TIME_OF_DROP

FILE: TERMINATOR.

SETS:

ENTITY_TYPE: DROPPED_IMAGE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_FUNCTIONAL.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

ALPHA : INITIALIZE_SKED_R.

BETA:

"BEGIN

TEOF:=CLOCK_TIME*FRAME_RATE; IST:=CLOCK_TIME

END;"

DESCRIPTION:

"COMPUTES THE TIME OF THE END OF THE CURRENT FRAME."

INPUTS:

DATA: CLOCK_TIME

DATA: FRAME_RATE.

OUTPUTS:

DATA: IST

DATA: TEOF.

REFERRED BY:

R_NET : SKED_R.

ALPHA : LOW_ELEVATION_DETERMINATION.

BETA:

"BEGIN LOW_ELEVATION:=(ENTRY_TIME-CLOCK_TIME>ELEVATION_LIMIT)

END;"

INPUTS:

DATA: CURRENT_STATE

DATA: ELEVATION_LIMIT.

OUTPUTS:

DATA: LOW_ELEVATION.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_D_FUNCTIONAL.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

ALPHA : LOW_TERMINATION.

BETA:

"BEGIN SET DROPPED_IMAGE;

FORM TRACK_TERMINATION;

CREATE TERMINATOR;

HO_ID:=IMAGE_ID;

REASON_FOR_DROP:=LOW;

DROP_REASON:=LOW;

TIME_OF_DROP:=CLOCK_TIME;

DATA_RECORD_TYPE:=TRACK_TERMINATION_REPORT;

DROP_TIME:=CLOCK_TIME

END;"

FORMS:

MESSAGE: TRACK_TERMINATION.

INPUTS:

DATA: CLOCK_TIME

DATA: IMAGE_ID.

OUTPUTS:

DATA: DATA_RECORD_TYPE

DATA: HO_ID

DATA: REASON_FOR_DROP

DATA: TIME_OF_DROP

FILE: TERMINATOR.

SETS:

ENTITY_TYPE: DROPPED_IMAGE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_D_FUNCTIONAL.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

ALPHA : MAKE_COMMAND.

BETA:

"BEGIN

CREATE COMMAND;

COMMAND_IMAGE_ID:=CANDIDATE_IMAGE_ID;

COMMAND_WAVEFORM:=CANDIDATE_WAVEFORM;

COMMAND_ENERGY:=CANDIDATE_ENERGY;

START_TIME:=IST;

IST:=IST+DELTAT;

SELECT FIRST FROM IMAGE_IN_TRACK SUCH THAT
(IMAGE_ID=CANDIDATE_IMAGE_ID);

LAST_PULSE:=START_TIME;

DESTROY CANDIDATE

END;".

INPUTS:

DATA: CANDIDATE_ENERGY

DATA: CANDIDATE_IMAGE_ID

DATA: CANDIDATE_WAVEFORM

DATA: DELTAT

DATA: IMAGE_ID

DATA: IST.

OUTPUTS:

DATA: COMMAND_ENERGY

DATA: COMMAND_IMAGE_ID

DATA: COMMAND_WAVEFORM

DATA: IST

DATA: LAST_PULSE

DATA: START_TIME.

REFERRED BY:

SUBNET : FORM_FRAME.

ALPHA : PICK_CANDIDATES.

BETA:

"BEGIN

CREATE CANDIDATE;

CANDIDATE_IMAGE_ID:=IMAGE_ID;

IF WAVEFORM=T1 THEN PRIORITY:=1.0

ELSE IF WAVEFORM=T2 THEN PRIORITY:=2.0

ELSE PRIORITY:=3.0;

CANDIDATE_WAVEFORM:=WAVEFORM;

SELECT FIRST FROM WAVEFORM_TABLE

SUCH THAT WF_NAME=WAVEFORM;

IF NOT FOUND THEN CANDIDATE_ENERGY:=0.0

ELSE CANDIDATE_ENERGY:=WF_CHARACTERISTICS;

END;"

DESCRIPTION:

"EACH IMAGE_IN_TRACK WHICH MIGHT GENERATE A MESSAGE THIS
FRAME HAS ITS TRANSMIT AND RECEIVE START AND STOP TIMES
EXTRACTED, ITS ENERGY DEMAND DETERMINED AND ITS
PRIORITY ESTABLISHED.".

INPUTS:

DATA: IMAGE_ID

DATA: WAVEFORM

(*INSTANCES OF ENTITY_TYPE IMAGE_IN_TRACK*)

FILE: WAVEFORM_TABLE.

OUTPUTS:

DATA: CANDIDATE_ENERGY

DATA: CANDIDATE_IMAGE_ID

DATA: CANDIDATE_WAVEFORM

DATA: PRIORITY.

TRACED FROM:

DECISION: SYNCHRONOUS_VS_ASYNCHRONOUS_TRACK.

REFERRED BY:

R_NET : SKED_R.

```

ALPHA : PICK_COMMAND.
BETA:
"BEGIN
  SELECT FIRST FROM COMMAND;
  IF FOUND THEN
    BEGIN
      TRANSMIT_START:=START_TIME;
      RADAR_TYPE:=COMMAND_WAVEFORM;
      RR_ORDER_IDC:=RR_ORDER_IDC+1; RR_ORDER_ID:=RR_ORDER_IDC;
      CREATE PULSE;
      PULSE_TYPE:=COMMAND_WAVEFORM;
      TARGET_ID:=COMMAND_IMAGE_ID;
      PULSE_ID:=RR_ORDER_IDC;
      XMIT_START:=START_TIME;
      DESTROY COMMAND;
    END
  END;"
DESCRIPTION:
  "PICK_COMMAND SELECTS NEXT COMMAND.".
CREATES:
  ENTITY_CLASS: PULSE.
INPUTS:
  DATA: RR_ORDER_IDC
  DATA: START_TIME
  FILE: COMMAND.
OUTPUTS:
  DATA: FOUND
  DATA: PULSE_ID
  DATA: PULSE_TYPE
  DATA: RADAR_TYPE
  DATA: RR_ORDER_ID
  DATA: RR_ORDER_IDC
  DATA: TARGET_ID
  DATA: TRANSMIT_START
  DATA: XMIT_START.
REFERRED BY:
  R_NET : XMIT_R.

```

ALPHA : REDUN_DETERMINATION.

BETA:

"VAR X:INTEGER;

BEGIN X:=0;

FOR EACH IMAGE_IN_TRACK DO

IF STATE=CURRENT_STATE THEN X:=X+1;

ENDFOREACH;

SELECT FIRST FROM IMAGE_IN_TRACK SUCH THAT

IMAGE_ID=TARGET_ID;

REDUNDANT_IMAGE:=(X>1)

END;".

INPUTS:

DATA: CURRENT_STATE

DATA: STATE.

OUTPUTS:

DATA: REDUNDANT_IMAGE.

TRACED FROM:

DECISION: TRACK_PERFORMANCE_ALLOCATION

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_A_FUNCTIONAL

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_PERFORMANCE

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_B_FUNCTIONAL.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

ALPHA : REDUN_TERMINATION.

BETA:

"BEGIN SET DROPPED_IMAGE;

FORM TRACK_TERMINATION;

CREATE TERMINATOR;

HO_ID:=IMAGE_ID;

REASON_FOR_DROP:=REDUNDANT;

DROP_REASON:= REDUNDANT;

TIME_OF_DROP:=CLOCK_TIME;

DATA_RECORD_TYPE:=TRACK_TERMINATION_REPORT;

DROP_TIME:=CLOCK_TIME

END;".

FORMS:

MESSAGE: TRACK_TERMINATION.

INPUTS:

DATA: CLOCK_TIME

DATA: IMAGE_ID.

OUTPUTS:

DATA: DATA_RECORD_TYPE

DATA: HO_ID

DATA: REASON_FOR_DROP

DATA: TIME_OF_DROP

FILE: TERMINATOR.

SETS:

ENTITY_TYPE: DROPPED_IMAGE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_A_FUNCTIONAL.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

ALPHA : RR_ERROR_PROCESSING.
BETA: "BEGIN END;".
REFERRED BY:
R_NET : RESPONSE_TO_RADAR.

ALPHA : STARTER.
ARTIFICIALITY: ARTIFICIAL.
BETA:
"BEGIN CREATE WAVEFORM_TABLE;
WF_NAME:=T1;
WF_CHARACTERISTICS:=1.0;
CREATE WAVEFORM_TABLE;
WF_NAME:=T2;
WF_CHARACTERISTICS:=2.0;
CREATE WAVEFORM_TABLE;
WF_NAME:=T3;
WF_CHARACTERISTICS:=3.0

END;".
DESCRIPTION: "THIS ELEMENT INITIALIZES WAVEFORM_TABLE".
OUTPUTS:
FILE: WAVEFORM_TABLE.
REFERRED BY:
R_NET : CC_RESPONSE.

ALPHA : SUMMARIZE_USAGE.
BETA:
"BEGIN IF ACCOUNTED_FOR<>SUMMED THEN BEGIN
SELECT FIRST FROM WAVEFORM_TABLE SUCH THAT
(WF_NAME=PULSE_TYPE);
RESOURCES:=RESOURCES+WF_CHARACTERISTICS;
IF ACCOUNTED_FOR=COUNTED THEN DESTROY PULSE
ELSE ACCOUNTED_FOR:=SUMMED

END
END;".
DESTROYS:
ENTITY_CLASS: PULSE.
INPUTS:
DATA: ACCOUNTED_FOR
DATA: PULSE_TYPE
FILE: WAVEFORM_TABLE.
OUTPUTS:
DATA: ACCOUNTED_FOR
DATA: RESOURCES.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_4_A_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_5_D_FUNCTIONAL.
REFERRED BY:
R_NET : RADAR_SUMMARY.

ALPHA : TERM_ENGAGEMENT.

BETA:

"CONST X= STANDBY ;

BEGIN MODE:=X

END;".

OUTPUTS:

DATA: MODE.

REFERRED BY:

R_NET : CC_RESPONSE.

ALPHA : TERM_TRACK.

BETA:

"LABEL 1001;

CONST X=CC_COMMAND_TO_DROP;

BEGIN

SELECT FIRST FROM IMAGE_IN_TRACK SUCH THAT (IMAGE_ID=HO_ID);

IF FOUND THEN SET DROPPED_IMAGE

ELSE

BEGIN

SELECT FIRST FROM DROPPED_IMAGE SUCH THAT

(IMAGE_ID=HO_ID);

IF NOT FOUND THEN GOTO 1001

END;

FORM TRACK_TERMINATION;

CREATE TERMINATOR;

DROP_TIME:=CLOCK_TIME;

DROP_REASON:=X;

DATA_RECORD_TYPE:=TRACK_TERMINATION_REPORT;

REASON_FOR_DROP:=X;

TIME_OF_DROP:=DROP_TIME;

1001;

END;".

FORMS:

MESSAGE: TRACK_TERMINATION.

INPUTS:

DATA: CLOCK_TIME

DATA: HO_ID.

OUTPUTS:

DATA: DATA_RECORD_TYPE

DATA: REASON_FOR_DROP

DATA: TIME_OF_DROP

FILE: TERMINATOR.

SETS:

ENTITY_TYPE: DROPPED_IMAGE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_E_FUNCTIONAL

ORIGINATING_REQUIREMENT: DPSPR_3_2_5_B_FUNCTIONAL.

REFERRED BY:

R_NET : CC_RESPONSE.

ALPHA : TRACK_INITIATE.

BETA:

"BEGIN

FORM TRACK_INITIATION;
CREATE IMAGE;
SET IMAGE_IN_TRACK;
TIME_OF_INITIATION:=CLOCK_TIME;
IMAGE_ID:=HO_ID;
STATE:=INITIAL_STATE;
COVARIANCE:=INITIAL_COVARIANCE;
TRACK_RATE:=20.0;
WAVEFORM:=T1;
DATA_RECORD_TYPE:=TRACK_INITIATION_REPORT;
ENTRY_TIME:=CLOCK_TIME

END;"

CREATES:

ENTITY_CLASS: IMAGE.

FORMS:

MESSAGE: TRACK_INITIATION.

INPUTS:

DATA: CLOCK_TIME
DATA: HO_ID
DATA: INITIAL_COVARIANCE
DATA: INITIAL_STATE.

OUTPUTS:

DATA: COVARIANCE
DATA: DATA_RECORD_TYPE
DATA: ENTRY_TIME
DATA: IMAGE_ID
DATA: STATE
DATA: TIME_OF_INITIATION
DATA: TRACK_RATE
DATA: WAVEFORM.

SETS:

ENTITY_TYPE: IMAGE_IN_TRACK.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_1_A_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_5_A_FUNCTIONAL.

REFERRED BY:

R_NET : CC_RESPONSE.

ALPHA : T1_T2_MEASUREMENT_EXTRACTION.

BETA:

"BEGIN VALID_RETURN:=TRUE;
RADAR_MEASUREMENT :=T1_T2_RECEIVE

END;"

INPUTS:

DATA: T1_T2_RECEIVE
FILE: T1_T2_DATA.

OUTPUTS:

DATA: RADAR_MEASUREMENT
DATA: VALID_RETURN.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

```

ALPHA : T3_MEASUREMENT_EXTRACTION.
  BETA:
  "BEGIN
    VALID_RETURN:=ODD(TRUNC(T3_RECEIVE*0.1));
    RADAR_MEASUREMENT:=T3_RECEIVE
  END!".
  INPUTS:
    DATA: T3_RECEIVE
    FILE: T3_DATA.
  OUTPUTS:
    DATA: RADAR_MEASUREMENT
    DATA: VALID_RETURN.
  REFERRED BY:
    R_NET : RESPONSE_TO_RADAR.

ALPHA : UPDATE_RADAR_CLOCK.
  BETA:
  "BEGIN
    RADAR_CLOCK:=RADAR_CLOCK_TIME
  END!".
  INPUTS:
    DATA: RADAR_CLOCK_TIME.
  OUTPUTS:
    DATA: RADAR_CLOCK.
  REFERRED BY:
    R_NET : RADAR_TIMING.

ALPHA : UPDATE_STATE.
  BETA:
  "BEGIN IF WAVEFORM=T1 THEN WAVEFORM:=T2 ELSE
    IF WAVEFORM=T2 THEN WAVEFORM:=T3 ELSE
      WAVEFORM:=T1;
    FORM STATE_UPDATE;
    HO_ID:=IMAGE_ID;
    STATE:= RADAR_MEASUREMENT;
    CURRENT_STATE:=STATE;
    DATA_RECORD_TYPE:=STATE_UPDATE_REPORT;
    COVARIANCE:=CLOCK_TIME
  END!".
  FORMS:
    MESSAGE: STATE_UPDATE.
  INPUTS:
    DATA: CLOCK_TIME
    DATA: IMAGE_ID
    DATA: RADAR_MEASUREMENT
    DATA: WAVEFORM.
  OUTPUTS:
    DATA: COVARIANCE
    DATA: CURRENT_STATE
    DATA: DATA_RECORD_TYPE
    DATA: HO_ID
    DATA: STATE
    DATA: WAVEFORM.
  TRACED FROM:
    DECISION: TRACK_PERFORMANCE_ALLOCATION
    ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_PERFORMANCE
    ORIGINATING_REQUIREMENT: DPSPR_3_2_2_D_FUNCTIONAL
    ORIGINATING_REQUIREMENT: DPSPR_3_2_3_A_FUNCTIONAL.
  REFERRED BY:
    R_NET : RESPONSE_TO_RADAR.

```

ALPHA : VALIDATE_HEADER.

BETA:

"BEGIN

END;".

INPUTS:

DATA: COMMAND_ID.

OUTPUTS:

DATA: COMMAND_ID.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_1_A_FUNCTIONAL.

REFERRED BY:

R_NET : CC_RESPONSE.

DATA : ACCEPTANCE_THRESHOLD.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_GATE_DATA
DATA: T3_GATE_DATA.

DATA : ACCOUNTED_FOR.
LOCALITY: GLOBAL.
RANGE: "NEITHER,COUNTED,SUMMED".
TYPE: ENUMERATION.
USE: BOTH.
ASSOCIATED WITH:
ENTITY_TYPE: LOST_PULSE
ENTITY_TYPE: RETURNED_PULSE.
INPUT TO:
ALPHA: ALLOCATE_AND_CONTROL_RESOURCES
ALPHA: SUMMARIZE_USAGE.
OUTPUT FROM:
ALPHA: ALLOCATE_AND_CONTROL_RESOURCES
ALPHA: SUMMARIZE_USAGE.

DATA : ALPHA_ERROR.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_RECEIVE
DATA: T3_RECEIVE.

DATA : ALPHA_PHASE_TAPER.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_TRANSMIT
DATA: T3_TRANSMIT.

DATA : AVERAGE_SIGNAL_POWER.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: WAKE_ARRAY.

DATA : BETA_ERROR.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_RECEIVE
DATA: T3_RECEIVE.

DATA : BETA_PHASE_TAPER.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_TRANSMIT
DATA: T3_TRANSMIT.

DATA : CANDIDATE_ENERGY.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
CONTAINED IN:
FILE: CANDIDATE.
INPUT TO:
ALPHA: MAKE_COMMAND.
OUTPUT FROM:
ALPHA: PICK_CANDIDATES.

DATA : CANDIDATE_IMAGE_ID.
LOCALITY: GLOBAL.
TYPE: INTEGER.
USE: BOTH.
CONTAINED IN:
FILE: CANDIDATE.
INPUT TO:
ALPHA: MAKE_COMMAND.
OUTPUT FROM:
ALPHA: PICK_CANDIDATES.

DATA : CANDIDATE_WAVEFORM.
LOCALITY: GLOBAL.
RANGE: "T1,T2,T3".
TYPE: ENUMERATION.
USE: BOTH.
CONTAINED IN:
FILE: CANDIDATE.
INPUT TO:
ALPHA: MAKE_COMMAND.
OUTPUT FROM:
ALPHA: PICK_CANDIDATES.

DATA : CLOCK_TIME
(* A PREDEFINED DATA ITEM WHICH INCREMENTS AT THE
SAME RATE AS ENGAGEMENT TIME. EXCEPT FOR ITS
INITIAL_VALUE WHICH IS ARBITRARY, CLOCK_TIME MAY
BE REGARDED AS ENGAGEMENT TIME. IT HAS NO CLOCK
ERROR. *).

LOCALITY: GLOBAL.
TYPE: REAL.
UNITS: SECONDS.
USE: BOTH.
INPUT TO:
ALPHA: ALLOCATE_AND_CONTROL_RESOURCES
ALPHA: COMPLETE_SUMMARY
ALPHA: GHOST_TERMINATION
ALPHA: INITIALIZE_SKED_R
ALPHA: LOW_TERMINATION
ALPHA: REDUN_TERMINATION
ALPHA: TERM_TRACK
ALPHA: TRACK_INITIATE
ALPHA: UPDATE_STATE.

DATA : COMMAND_ENERGY.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
CONTAINED IN:
FILE: COMMAND.
OUTPUT FROM:
ALPHA: MAKE_COMMAND.

DATA : COMMAND_ID.
LOCALITY: LOCAL.
RANGE:
"HANDOVER_IMAGE,DROP_TRACK,INITIATE_ENGAGEMENT_MODE,
TERMINATE_ENGAGEMENT_MODE".
TYPE: ENUMERATION.
USE: BOTH.
MAKES:
MESSAGE: ACKNOWLEDGEMENT
MESSAGE: HANDOVER
MESSAGE: MODE_CHANGE
MESSAGE: TERMINATION.
INPUT TO:
ALPHA: VALIDATE_HEADER.
OUTPUT FROM:
ALPHA: VALIDATE_HEADER.
REFERRED BY:
R_NET : CC_RESPONSE.

DATA : COMMAND_IMAGE_ID.
LOCALITY: GLOBAL.
TYPE: INTEGER.
USE: BOTH.
CONTAINED IN:
FILE: COMMAND.
OUTPUT FROM:
ALPHA: MAKE_COMMAND.

DATA : COMMAND_WAVEFORM.
LOCALITY: GLOBAL.
RANGE: "T1,T2,T3".
TYPE: ENUMERATION.
USE: BOTH.
CONTAINED IN:
FILE: COMMAND.
OUTPUT FROM:
ALPHA: MAKE_COMMAND.

DATA : COVARIANCE.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BETA.
ASSOCIATED WITH:
ENTITY_TYPE: IMAGE_IN_TRACK.
OUTPUT FROM:
ALPHA: TRACK_INITIATE
ALPHA: UPDATE_STATE.

DATA : CURRENT_STATE.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BETA.
MAKES:
 MESSAGE: STATE_UPDATE.
INPUT TO:
 ALPHA: LOW_ELEVATION_DETERMINATION
 ALPHA: REDUN_DETERMINATION.
OUTPUT FROM:
 ALPHA: UPDATE_STATE.

DATA : DATA_RECORD_TYPE.
LOCALITY: LOCAL.
RANGE:
 "RADAR_USAGE_REPORT, STATE_UPDATE_REPORT,
 TRACK_TERMINATION_REPORT, TRACK_INITIATION_REPORT".
TYPE: ENUMERATION.
USE: BOTH.
MAKES:
 MESSAGE: RADAR_USAGE
 MESSAGE: STATE_UPDATE
 MESSAGE: TRACK_INITIATION
 MESSAGE: TRACK_TERMINATION.
OUTPUT FROM:
 ALPHA: COMPLETE_SUMMARY
 ALPHA: GHOST_TERMINATION
 ALPHA: LOW_TERMINATION
 ALPHA: REDUN_TERMINATION
 ALPHA: TERM_TRACK
 ALPHA: TRACK_INITIATE
 ALPHA: UPDATE_STATE.

DATA : DELTAT.
DESCRIPTION:
 "MINIMUM PULSE SPACING FOR BEAM SWITCHING.".
INITIAL_VALUE: 3.0E-6.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BOTH.
INPUT TO:
 ALPHA: MAKE_COMMAND.

DATA : DROP_FLAG.
LOCALITY: LOCAL.
TYPE: BOOLEAN.
USE: BOTH.
OUTPUT FROM:
 ALPHA: FIND_CONFLICT.
REFERRED BY:
 SUBNET : FORM_FRAME.

DATA : DROP_REASON.
LOCALITY: GLOBAL.
RANGE: "GHOST,REDUNDANT,LOW,CC_COMMAND_TO_DROP".
TYPE: ENUMERATION.
USE: BOTH.
CONTAINED IN:
FILE: TERMINATOR.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_A_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_R_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_O_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_E_FUNCTIONAL.

DATA : DROP_TIME.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
CONTAINED IN:
FILE: TERMINATOR.

DATA : ELEVATION_LIMIT.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
INPUT TO:
ALPHA: LOW_ELEVATION_DETERMINATION.

DATA : ENERGY_BOUND.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: GAMMA.

DATA : ENGAGEMENT_TIME.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BOTH.
MAKES:
MESSAGE: RADAR_USAGE.
OUTPUT FROM:
ALPHA: COMPLETE_SUMMARY.

DATA : ENTRY_TIME.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
ASSOCIATED WITH:
ENTITY_CLASS: IMAGE.
OUTPUT FROM:
ALPHA: TRACK_INITIATE.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_E_FUNCTIONAL.

DATA : FOUND

(* A PREDEFINED DATA ITEM WHICH IS SET TO EITHER TRUE OR FALSE AFTER EACH SELECT IN A BETA OR GAMMA. FOUND IS SET TO TRUE IF AN INSTANCE SATISFYING THE SELECTION CRITERION IS LOCATED. OTHERWISE, FOUND IS ASSIGNED THE VALUE FALSE. *).

LOCALITY: GLOBAL.
TYPE: BOOLEAN.
USE: BOTH.
OUTPUT FROM:
ALPHA: PICK_COMMAND.
REFERRED BY:
R_NET : XMIT_R.

DATA : FRAME_RATE.

INITIAL_VALUE: 0.01.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
DELAYS:
EVENT: SCHEDULE.
INPUT TO:
ALPHA: INITIALIZE_SKED_R.

DATA : GATE_LENGTH.

LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_GATE_DATA
DATA: T3_GATE_DATA.

DATA : GATE_START_TIME.

LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_GATE_DATA
DATA: T3_GATE_DATA.

DATA : GHOST_IMAGE.

LOCALITY: LOCAL.
TYPE: BOOLEAN.
USE: BOTH.
OUTPUT FROM:
ALPHA: GHOST_DETERMINATION.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_3_C_FUNCTIONAL.
REFERRED BY:
R_NET : RESPONSE_TO_RADAR.

DATA : HO_ID.
 LOCALITY: LOCAL.
 TYPE: INTEGER.
 USE: BOTH.
 MAKES:
 MESSAGE: HANDOVER
 MESSAGE: STATE_UPDATE
 MESSAGE: TERMINATION
 MESSAGE: TRACK_INITIATION
 MESSAGE: TRACK_TERMINATION.
 INPUT TO:
 ALPHA: TERM_TRACK
 ALPHA: TRACK_INITIATE.
 OUTPUT FROM:
 ALPHA: GHOST_TERMINATION
 ALPHA: LOW_TERMINATION
 ALPHA: REDUN_TERMINATION
 ALPHA: UPDATE_STATE.

DATA : IMAGE_ID.
 LOCALITY: GLOBAL.
 TYPE: INTEGER.
 USE: BOTH.
 ASSOCIATED WITH:
 ENTITY_CLASS: IMAGE.
 INPUT TO:
 ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE
 ALPHA: ALLOCATE_AND_CONTROL_RESOURCES
 ALPHA: GHOST_TERMINATION
 ALPHA: LOW_TERMINATION
 ALPHA: MAKE_COMMAND
 ALPHA: PICK_CANDIDATES
 ALPHA: REDUN_TERMINATION
 ALPHA: UPDATE_STATE.
 OUTPUT FROM:
 ALPHA: TRACK_INITIATE.

DATA : INITIAL_COVARIANCE.
 LOCALITY: LOCAL.
 TYPE: REAL.
 USE: BOTH.
 MAKES:
 MESSAGE: HANDOVER.
 INPUT TO:
 ALPHA: TRACK_INITIATE.

DATA : INITIAL_STATE.
 LOCALITY: LOCAL.
 TYPE: REAL.
 USE: BOTH.
 MAKES:
 MESSAGE: HANDOVER
 MESSAGE: TRACK_INITIATION.
 INPUT TO:
 ALPHA: TRACK_INITIATE.

DATA : IST.
 DESCRIPTION: "INITIAL START TIME FOR THE FRAME."
 LOCALITY: LOCAL.
 TYPE: REAL.
 USE: BOTH.
 INPUT TO:
 ALPHA: MAKE_COMMAND.
 OUTPUT FROM:
 ALPHA: INITIALIZE_SKED_R
 ALPHA: MAKE_COMMAND.

DATA : LAST_ALLOCATE.
 LOCALITY: GLOBAL.
 TYPE: REAL.
 USE: BOTH.
 INPUT TO:
 ALPHA: ALLOCATE_AND_CONTROL_RESOURCES.
 OUTPUT FROM:
 ALPHA: ALLOCATE_AND_CONTROL_RESOURCES.

DATA : LAST_PULSE.
 LOCALITY: GLOBAL.
 TYPE: REAL.
 USE: BOTH.
 ASSOCIATED WITH:
 ENTITY_TYPE: IMAGE_IN_TRACK.
 OUTPUT FROM:
 ALPHA: MAKE_COMMAND.
 REFERRED BY:
 P_NET : SKED_R.

DATA : LENGTH_OF_RECEIVE.
 LOCALITY: LOCAL.
 TYPE: REAL.
 USE: GAMMA.
 INCLUDED IN:
 DATA: RECEIVE_INFORMATION.

DATA : LOW_ELEVATION.
 LOCALITY: LOCAL.
 TYPE: BOOLEAN.
 USE: BOTH.
 OUTPUT FROM:
 ALPHA: LOW_ELEVATION_DETERMINATION.
 TRACED FROM:
 ORIGINATING_REQUIREMENT: DPSPR_3_2_2_C_FUNCTIONAL.
 REFERRED BY:
 R_NET : RESPONSE_TO_RADAR.

DATA : MODE.
 LOCALITY: GLOBAL.
 RANGE: "ENGAGED,STANDBY".
 TYPE: ENUMERATION.
 USE: BOTH.
 OUTPUT FROM:
 ALPHA: ENGAGEMENT_INITIATION
 ALPHA: TERM_ENGAGEMENT.
 REFERRED BY:
 R_NET : RADAR_SUMMARY
 R_NET : SKED_R.

DATA : NOISE_LEVEL.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_RECORD
DATA: T3_RECORD.

DATA : PRIORITY.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
ORDERS:
FILE: CANDIDATE.
FILE: CANDIDATE.
OUTPUT FROM:
ALPHA: PICK_CANDIDATES.
TRACED FROM:
DECISION: TRACK_PERFORMANCE_ALLOCATION
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_PERFORMANCE.

DATA : PULSE_ID.
LOCALITY: GLOBAL.
TYPE: INTEGER.
USE: BOTH.
ASSOCIATED WITH:
ENTITY_CLASS: PULSE.
INPUT TO:
ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.
OUTPUT FROM:
ALPHA: PICK_COMMAND.

DATA : PULSE_TYPE.
LOCALITY: GLOBAL.
RANGE: "T1,T2,T3".
TYPE: ENUMERATION.
USE: BOTH.
ASSOCIATED WITH:
ENTITY_CLASS: PULSE.
INPUT TO:
ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE
ALPHA: ALLOCATE_AND_CONTROL_RESOURCES
ALPHA: SUMMARIZE_USAGE.
OUTPUT FROM:
ALPHA: PICK_COMMAND.

DATA : RADAR_CLOCK.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
OUTPUT FROM:
ALPHA: UPDATE_RADAR_CLOCK.

DATA : RADAR_CLOCK_TIME.
 LOCALITY: LOCAL.
 RESOLUTION: 6.25E-9.
 TYPE: REAL.
 UNITS: SECONDS.
 USE: BOTH.
 MAKES:
 MESSAGE: R_CLOCK_MESSAGE.
 INPUT TO:
 ALPHA: UPDATE_RADAR_CLOCK.
 TRACED FROM:
 ORIGINATING_REQUIREMENT: RADAR_DPS_IFS_3_2_9_FUNCTIONAL.

DATA : RADAR_MEASUREMENT.
 LOCALITY: LOCAL.
 TYPE: REAL.
 USE: BETA.
 INPUT TO:
 ALPHA: GHOST_DETERMINATION
 ALPHA: UPDATE_STATE.
 OUTPUT FROM:
 ALPHA: T1_T2_MEASUREMENT_EXTRACTION
 ALPHA: T3_MEASUREMENT_EXTRACTION.

DATA : RADAR_MODEL_DATA.
 LOCALITY: GLOBAL.
 TYPE: REAL.
 USE: GAMMA.

DATA : RADAR_TYPE.
 LOCALITY: LOCAL.
 RANGE: "T1,T2,T3".
 TYPE: ENUMERATION.
 USE: BOTH.
 MAKES:
 MESSAGE: T1_T2_COMMAND
 MESSAGE: T1_T2_RETURN
 MESSAGE: T3_COMMAND
 MESSAGE: T3_RETURN.
 INPUT TO:
 ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.
 OUTPUT FROM:
 ALPHA: PICK_COMMAND.
 REFERRED BY:
 R_NET : RESPONSE_TO_RADAR
 R_NET : XMIT_R.

DATA : RANGE_MARK_GENERATION_TECHNIQUE.
 LOCALITY: LOCAL.
 TYPE: INTEGER.
 USE: GAMMA.
 INCLUDED IN:
 DATA: T1_T2_GATE_DATA.

DATA : RANGE_MARK_INFORMATION.
 INCLUDES:
 DATA: RANGE_MARK_TIME
 DATA: SIGNAL_AMPLITUDE.
 INCLUDED IN:
 DATA: T1_T2_RECORD
 DATA: T3_RECORD.

DATA : RANGE_MARK_TECHNIQUE.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T3_GATE_DATA.

DATA : RANGE_MARK_TIME.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: RANGE_MARK_INFORMATION.

DATA : REASON_FOR_DROP.
LOCALITY: LOCAL.
RANGE: "GHOST,REDUNDANT,LOW,CC_COMMAND_TO_DROP".
TYPE: ENUMERATION.
USE: BOTH.
MAKES:
MESSAGE: TRACK_TERMINATION.
OUTPUT FROM:
ALPHA: GHOST_TERMINATION
ALPHA: LOW_TERMINATION
ALPHA: REDUN_TERMINATION
ALPHA: TERM_TRACK.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_A_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_D_FUNCTIONAL
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_E_FUNCTIONAL.

DATA : REASON_FOR_TRANSMISSION_FAILURE.
LOCALITY: LOCAL.
RANGE:
"PRE_EMPTED_TRANSMISSION,
RECEIVE_WINDOW_OVERLAP,
TRANSMIT_WINDOW_OVERLAP,
INSUFFICIENT_TRANSMISSION_TIME,
RADAR_COMMAND_INCONSISTENCY,
TRANSMIT_START_TIME_EXCEEDED".
TYPE: ENUMERATION.
INCLUDED IN:
DATA: T1T2RTN_ERROR_REPORT
DATA: T3RTN_ERROR_REPORT.

DATA : RECEIVE_INFORMATION.
INCLUDES:
DATA: LENGTH_OF_RECEIVE
DATA: RECEIVE_START_TIME
DATA: RECEIVER_GAIN_SETTING.
INCLUDED IN:
DATA: T1_T2_TRANSMIT
DATA: T3_TRANSMIT.

DATA : RECEIVE_START_TIME.

LOCALITY: LOCAL.

TYPE: REAL.

USE: GAMMA.

INCLUDED IN:

DATA: RECEIVE_INFORMATION.

DATA : RECEIVE_STOP.

LOCALITY: GLOBAL.

TYPE: REAL.

USE: BOTH.

ASSOCIATED WITH:

ENTITY_TYPE: T1_T2_PULSE

ENTITY_TYPE: T3_PULSE.

INPUT TO:

ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.

OUTPUT FROM:

ALPHA: FORM_T1_T2

ALPHA: FORM_T3.

DATA : RECEIVER_GAIN_SETTING.

LOCALITY: LOCAL.

TYPE: REAL.

USE: GAMMA.

INCLUDED IN:

DATA: RECEIVE_INFORMATION.

DATA : REDUNDANT_IMAGE.

LOCALITY: LOCAL.

TYPE: BOOLEAN.

USE: BOTH.

OUTPUT FROM:

ALPHA: REDUN_DETERMINATION.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_A_FUNCTIONAL

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_B_FUNCTIONAL.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

DATA : RESOURCES.

LOCALITY: LOCAL.

TYPE: REAL.

USE: BETA.

MAKES:

MESSAGE: RADAR_USAGE.

OUTPUT FROM:

ALPHA: SUMMARIZE_USAGE.

DATA : RETURN_IMAGE_STATUS.

LOCALITY: LOCAL.

RANGE: "IN_TRACK,DROPPED,NO_ORDER,WRONG_ORDER".

TYPE: ENUMERATION.

USE: BOTH.

OUTPUT FROM:

ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.

REFERRED BY:

R_NET : RESPONSE_TO_RADAR.

DATA : RR_ORDER_ID.
LOCALITY: LOCAL.
TYPE: INTEGER.
USE: BOTH.
MAKES:
MESSAGE: T1_T2_COMMAND
MESSAGE: T1_T2_RETURN
MESSAGE: T3_COMMAND
MESSAGE: T3_RETURN.
INPUT TO:
ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.
OUTPUT FROM:
ALPHA: PICK_COMMAND.

DATA : RR_ORDER_IDC.
INITIAL_VALUE: 0.
LOCALITY: GLOBAL.
TYPE: INTEGER.
USE: BOTH.
INPUT TO:
ALPHA: PICK_COMMAND.
OUTPUT FROM:
ALPHA: PICK_COMMAND.

DATA : SIGNAL_AMPLITUDE.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: RANGE_MARK_INFORMATION.

DATA : SIGNAL_PROCESSING_MODE.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: T1_T2_GATE_DATA
DATA: T3_GATE_DATA.

DATA : START_TIME.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
ORDERS:
FILE: COMMAND.
CONTAINED IN:
FILE: COMMAND.
INPUT TO:
ALPHA: FIND_CONFLICT
ALPHA: PICK_COMMAND.
OUTPUT FROM:
ALPHA: MAKE_COMMAND.

DATA : STATE.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: RETA.
ASSOCIATED WITH:
 ENTITY_TYPE: IMAGE_IN_TRACK.
INPUT TO:
 ALPHA: REDUN_DETERMINATION.
OUTPUT FROM:
 ALPHA: TRACK_INITIATE
 ALPHA: UPDATE_STATE.
TRACED FROM:
 * ORIGINATING_REQUIREMENT: DPSPR_3_2_3_A_FUNCTIONAL.

DATA : SUMMARY_RATE.
INITIAL_VALUE: 0.3.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
DELAYS:
 EVENT: SUMMARIZE.

DATA : TARGET_ID.
LOCALITY: GLOBAL.
TYPE: INTEGER.
USE: BOTH.
ASSOCIATED WITH:
 ENTITY_CLASS: PULSE.
INPUT TO:
 ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.
OUTPUT FROM:
 ALPHA: PICK_COMMAND.

DATA : TEOF.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BOTH.
INPUT TO:
 ALPHA: FIND_CONFLICT.
OUTPUT FROM:
 ALPHA: INITIALIZE_SKED_R.
REFERRED BY:
 R_NET : SKED_R.

DATA : THRESHOLD_DOWN_CROSSING_TIME.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
 DATA: WAKE_ARRAY.

DATA : THRESHOLD_TYPE.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
 DATA: T1_T2_GATE_DATA
 DATA: T3_GATE_DATA.

DATA : THRESHOLD_UP_CROSSING_TIME.
LOCALITY: LOCAL.
TYPE: REAL.
USE: GAMMA.
INCLUDED IN:
DATA: WAKE_ARRAY.

DATA : TIME_OF_DROP.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BOTH.
MAKES:
MESSAGE: TRACK_TERMINATION.
OUTPUT FROM:
ALPHA: GHOST_TERMINATION
ALPHA: LOW_TERMINATION
ALPHA: REDUN_TERMINATION
ALPHA: TERM_TRACK.

DATA : TIME_OF_INITIATION.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BOTH.
MAKES:
MESSAGE: TRACK_INITIATION.
OUTPUT FROM:
ALPHA: TRACK_INITIATE.

DATA : TRACK_RATE.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
ASSOCIATED WITH:
ENTITY_TYPE: IMAGE_IN_TRACK.
OUTPUT FROM:
ALPHA: ALLOCATE_AND_CONTROL_RESOURCES
ALPHA: TRACK_INITIATE.
REFERRED BY:
R_NET : SKED_R.

DATA : TRANSMIT_START.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BOTH.
MAKES:
MESSAGE: T1_T2_COMMAND
MESSAGE: T3_COMMAND.
OUTPUT FROM:
ALPHA: PICK_COMMAND.

DATA : T1_T2_GATE_DATA.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BETA.
INCLUDES:
 DATA: ACCEPTANCE_THRESHOLD
 DATA: GATE_LENGTH
 DATA: GATE_START_TIME
 DATA: RANGE_MARK_GENERATION_TECHNIQUE
 DATA: SIGNAL_PROCESSING_MODE
 DATA: THRESHOLD_TYPE.
CONTAINED IN:
 FILE: T1_T2_GATE.

DATA : T1_T2_RECEIVE.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BOTH.
INCLUDES:
 DATA: ALPHA_ERROR
 DATA: BETA_ERROR
 DATA: T1T2RTN_ERROR_REPORT
 DATA: WAKE_ARRAY.
MAKES:
 MESSAGE: T1_T2_RETURN.
INPUT TO:
 ALPHA: T1_T2_MEASUREMENT_EXTRACTION.

DATA : T1_T2_RECORD.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BETA.
INCLUDES:
 DATA: NOISE_LEVEL
 DATA: RANGE_MARK_INFORMATION.
CONTAINED IN:
 FILE: T1_T2_DATA.

DATA : T1_T2_TRANSMIT.
TYPE: REAL.
USE: BETA.
INCLUDES:
 DATA: ALPHA_PHASE_TAPER
 DATA: BETA_PHASE_TAPER
 DATA: RECEIVE_INFORMATION.
MAKES:
 MESSAGE: T1_T2_COMMAND.
OUTPUT FROM:
 ALPHA: FORM_T1_T2.

DATA : T1_T2_WINDOW_DATA.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
CONTAINED IN:
 FILE: T1_T2_WINDOW.

DATA : T1_T2_XMIT.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
ASSOCIATED WITH:
 ENTITY_TYPE: T1_T2_PULSE.
OUTPUT FROM:
 ALPHA: FORM_T1_T2.

DATA : T1T2RTN_ERROR_REPORT.
INCLUDES:
 DATA: REASON_FOR_TRANSMISSION_FAILURE.
INCLUDED IN:
 DATA: T1_T2_RECEIVE.

DATA : T3_GATE_DATA.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BETA.
INCLUDES:
 DATA: ACCEPTANCE_THRESHOLD
 DATA: GATE_LENGTH
 DATA: GATE_START_TIME
 DATA: RANGE_MARK_TECHNIQUE
 DATA: SIGNAL_PROCESSING_MODE
 DATA: THRESHOLD_TYPE.
CONTAINED IN:
 FILE: T3_GATE.

DATA : T3_RECEIVE.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BOTH.
INCLUDES:
 DATA: ALPHA_ERROR
 DATA: BETA_ERROR
 DATA: T3RTN_ERROR_REPORT
 DATA: WAKE_ARRAY.
MAKES:
 MESSAGE: T3_RETURN.
INPUT TO:
 ALPHA: T3_MEASUREMENT_EXTRACTION.

DATA : T3_RECORD.
LOCALITY: LOCAL.
TYPE: REAL.
USE: BETA.
INCLUDES:
 DATA: NOISE_LEVEL
 DATA: RANGE_MARK_INFORMATION.
CONTAINED IN:
 FILE: T3_DATA.

DATA : T3_TRANSMIT.
 TYPE: REAL.
 USE: BETA.
 INCLUDES:
 DATA: ALPHA_PHASE_TAPER
 DATA: BETA_PHASE_TAPER
 DATA: RECEIVE_INFORMATION.
 MAKES:
 MESSAGE: T3_COMMAND.
 OUTPUT FROM:
 ALPHA: FORM_T3.

DATA : T3_WINDOW_DATA.
 LOCALITY: GLOBAL.
 TYPE: REAL.
 USE: BETA.
 CONTAINED IN:
 FILE: T3_WINDOW.

DATA : T3_XMIT.
 LOCALITY: GLOBAL.
 TYPE: REAL.
 USE: BOTH.
 ASSOCIATED WITH:
 ENTITY_TYPE: T3_PULSE.
 OUTPUT FROM:
 ALPHA: FORM_T3.

DATA : T3PTN_ERROR_REPORT.
 INCLUDES:
 DATA: REASON_FOR_TRANSMISSION_FAILURE.
 INCLUDED IN:
 DATA: T3_RECEIVE.

DATA : VALID_RETURN.
 LOCALITY: LOCAL.
 TYPE: BOOLEAN.
 USE: BOTH.
 OUTPUT FROM:
 ALPHA: T1_T2_MEASUREMENT_EXTRACTION
 ALPHA: T3_MEASUREMENT_EXTRACTION.
 REFERRED BY:
 R_NET : RESPONSE_TO_RADAR.

DATA : WAKE_ARRAY.
 INCLUDES:
 DATA: AVERAGE_SIGNAL_POWER
 DATA: THRESHOLD_DOWN_CROSSING_TIME
 DATA: THRESHOLD_UP_CROSSING_TIME.
 INCLUDED IN:
 DATA: T1_T2_RECEIVE
 DATA: T3_RECEIVE.

DATA : WAVEFORM.
LOCALITY: GLOBAL.
RANGE: "T1,T2,T3".
TYPE: ENUMERATION.
USE: BOTH.
ASSOCIATED WITH:
ENTITY_TYPE: IMAGE_IN_TRACK.
INPUT TO:
ALPHA: ALLOCATE_AND_CONTROL_RESOURCES
ALPHA: PICK_CANDIDATES
(*INSTANCES OF ENTITY_TYPE IMAGE_IN_TRACK*)
ALPHA: UPDATE_STATE.
OUTPUT FROM:
ALPHA: TRACK_INITIATE
ALPHA: UPDATE_STATE.

DATA : WF_CHARACTERISTICS.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
CONTAINED IN:
FILE: WAVEFORM_TABLE.

DATA : WF_NAME.
LOCALITY: GLOBAL.
RANGE: "T1,T2,T3".
TYPE: ENUMERATION.
USE: BOTH.
CONTAINED IN:
FILE: WAVEFORM_TABLE.

DATA : WINDOW.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: GAMMA.
CONTAINED IN:
FILE: COMMAND.

DATA : XMIT_START.
LOCALITY: GLOBAL.
TYPE: REAL.
USE: BOTH.
ASSOCIATED WITH:
ENTITY_CLASS: PULSE.
OUTPUT FROM:
ALPHA: PICK_COMMAND.

DECISION : RADAR_SCHEDULER_PRIORITIZATION.

ALTERNATIVES:

- "1. SCHEDULE PULSE_BY_PULSE. THIS WOULD SIMPLIFY THE NETS BUT WOULD OBVIATE OPTIMIZATION;
2. OPTIMIZE OVER THE ENTIRE FRAME. TAKING THE FRAME AS A WHOLE GIVES BEST RESULTS BUT REQUIRES WEIGHTING FACTORS FOR PULSE ENSEMBLES.
3. PRIORITIZE PULSES SUCH THAT ANY PULSE OF HIGH PRIORITY BEATS ALL PULSES OF LOWER. THIS IS SUBOPTIMAL, BUT REALIZABLE BOTH IN THE SPEC AND IN THE SOFTWARE DESIGN. NO A PRIORI WEIGHTS NEEDED."

CHOICE:

"OPTION 3. PRIORITIZED PULSES".

PROBLEM:

"OPTIMIZATION OF RADAR USAGE REQUIRES A FINITE RADAR FRAME. THIS IMPLIES A PRIORITIZATION SCHEME FOR INTENDED ORDERS."

TRACES TO:

R_NET: SKED_R

R_NET: XMIT_R.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_4_B_FUNCTIONAL.

DECISION : SYNCHRONOUS_VS_ASYNCHRONOUS_TRACK.

ALTERNATIVES:

- "1. SYNCHRONOUS TRACKING (OR RESPONSIVE) REQUIRES THE LAST RADAR RETURN ON AN IMAGE BE USED TO PRODUCE THE NEXT RADAR ORDER.
2. ASYNCHRONOUS TRACKING "OR AUTOGENIC" ALLOWS A TRACK PULSE TO BE SENT USING WHAT EVER STATE IS IN THE DATA BASE."

CHOICE:

"ASYNCHRONOUS TRACKING IS SELECTED TO MAXIMIZE THE ALLOWED DP TIME RESPONSE FOR PROCESSING RADAR RETURNS. THIS DOES NOT PROHIBIT A RESPONSIVE TRACKING IMPLEMENTATION."

PROBLEM:

"TRACKING CAN BE EXPRESSED AS SYNCHRONOUS OR ASYNCHRONOUS."

TRACES TO:

ALPHA: PICK_CANDIDATES.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_3_A_FUNCTIONAL.

DECISION : TRACK_PERFORMANCE_ALLOCATION.

CHOICE:

- "1. ALLOCATION WILL BE PERFORMED TO CONSTRAIN
IMAGE STATES AND RATES AT ALL TIMES.
2. PULSE SCHEDULING WILL BE CONSTRAINED
BY A REALTIONSHIP BETWEEN IMAGE
STATES, ITS TRACK RATE,
AND THE RADAR CONSTRAINTS,
3. DEGHOSTING WILL BE A FUNCTION OF
RADAR MEASUREMENTS ONLY.
4. "UPDATE STATE" WILL BE CONSTRAINED
BY ITS DIFFERENCE
IN BETA AND CEP FROM A
"PERFECT FILTER".
5. REDUNDANT IMAGE ELIMINATION
PERFORMANCE
WILL BE EXPRESSED IN TERMS OF
STATES ONLY."

PROBLEM:

"TRACK ACCURACY IS A JOINT FUNCTION OF
THE TRACK RATE, SUCCESSFUL SCHEDULING,
ACCURATE PULSE COMMANDS, AND ACCURATE
PROCESSING OF THE RADAR RETURN".

TRACES TO:

ALPHA: ALLOCATE_AND_CONTROL_RESOURCES
ALPHA: FIND_CONFLICT
ALPHA: GHOST_DETERMINATION
ALPHA: REDUN_DETERMINATION
ALPHA: UPDATE_STATE
DATA: PRIORITY.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_A_PERFORMANCE
ORIGINATING_REQUIREMENT: DPSPR_3_2_2_B_PERFORMANCE
ORIGINATING_REQUIREMENT: DPSPR_3_2_3_A_PERFORMANCE
ORIGINATING_REQUIREMENT: DPSPR_3_2_3_B_PERFORMANCE
ORIGINATING_REQUIREMENT: DPSPR_3_2_3_C_PERFORMANCE
ORIGINATING_REQUIREMENT: DPSPR_3_2_3_D_PERFORMANCE.

ENTITY_CLASS : IMAGE.

ASSOCIATES:

DATA: ENTRY_TIME

DATA: IMAGE_ID.

COMPOSED OF:

ENTITY_TYPE: DROPPED_IMAGE

ENTITY_TYPE: IMAGE_IN_TRACK.

CREATED BY:

ALPHA: TRACK_INITIATE.

ENTITY_CLASS : PULSE.

ASSOCIATES:

DATA: PULSE_ID

DATA: PULSE_TYPE

DATA: TARGET_ID

DATA: XMIT_START.

COMPOSED OF:

ENTITY_TYPE: LOST_PULSE

ENTITY_TYPE: RETURNED_PULSE

ENTITY_TYPE: T1_T2_PULSE

ENTITY_TYPE: T3_PULSE.

CREATED BY:

ALPHA: PICK_COMMAND.

DESTROYED BY:

ALPHA: ALLOCATE_AND_CONTROL_RESOURCES

ALPHA: SUMMARIZE_USAGE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_4_A_FUNCTIONAL.

ENTITY_TYPE : DROPPED_IMAGE.
ASSOCIATES:
FILE: TERMINATOR.
COMPOSES:
ENTITY_CLASS: IMAGE.
SET BY:
ALPHA: GHOST_TERMINATION
ALPHA: LOW_TERMINATION
ALPHA: REDUN_TERMINATION
ALPHA: TERM_TRACK.

ENTITY_TYPE : IMAGE_IN_TRACK.
ASSOCIATES:
DATA: COVARIANCE
DATA: LAST_PULSE
DATA: STATE
DATA: TRACK_RATE
DATA: WAVEFORM.
COMPOSES:
ENTITY_CLASS: IMAGE.
SET BY:
ALPHA: TRACK_INITIATE.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_3_A_FUNCTIONAL.
REFERRED BY:
R_NET : SKED_R.

ENTITY_TYPE : LOST_PULSE..
ASSOCIATES:
DATA: ACCOUNTED_FOR.
COMPOSES:
ENTITY_CLASS: PULSE.
SET BY:
ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.

ENTITY_TYPE : RETURNED_PULSE.
ASSOCIATES:
DATA: ACCOUNTED_FOR.
COMPOSES:
ENTITY_CLASS: PULSE.
SET BY:
ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.
REFERRED BY:
R_NET : RADAR_SUMMARY.

ENTITY_TYPE : T1_T2_PULSE.
ASSOCIATES:
DATA: RECEIVE_STOP
DATA: T1_T2_XMIT
FILE: T1_T2_WINDOW.
COMPOSES:
ENTITY_CLASS: PULSE.
SET BY:
ALPHA: FORM_T1_T2.

ENTITY_TYPE : T3_PULSE.
ASSOCIATES:
DATA: RECEIVE_STOP
DATA: T3_XMIT
FILE: T3_WINDOW.
COMPOSES:
ENTITY_CLASS: PULSE.
SET BY:
ALPHA: FORM_T3.

EVENT : ALLOCATE.

ENABLES:

R_NET: CONTROL_RESOURCES.

REFERRED BY:

R_NET : CC_RESPONSE

R_NET : RESPONSE_TO_RADAR.

EVENT : SCHEDULE.

ENABLES:

R_NET: SKED_P.

DELAYED BY:

DATA: FRAME_RATE.

REFERRED BY:

R_NET : CC_RESPONSE

R_NET : XMIT_R.

EVENT : SUMMARIZE.

ENABLES:

R_NET: RADAR_SUMMARY.

DELAYED BY:

DATA: SUMMARY_RATE.

REFERRED BY:

R_NET : CC_RESPONSE

R_NET : RADAR_SUMMARY.

EVENT : XRB.

DESCRIPTION:

"TURNS ON R_NET XMIT_R FOR THE CURRENT FRAME."

ENABLES:

R_NET: XMIT_R.

REFERRED BY:

R_NET : SKED_R

R_NET : XMIT_R.

FILE : CANDIDATE.

CONTAINS:

DATA: CANDIDATE_ENERGY
DATA: CANDIDATE_IMAGE_ID
DATA: CANDIDATE_WAVEFORM
DATA: PRIORITY.

ORDERED BY:

DATA: PRIORITY.

REFERRED BY:

R_NET : SKED_R.

FILE : COMMAND.

CONTAINS:

DATA: COMMAND_ENERGY
DATA: COMMAND_IMAGE_ID
DATA: COMMAND_WAVEFORM
DATA: START_TIME
DATA: WINDOW.

INPUT TO:

ALPHA: FORM_T1_T2
ALPHA: FORM_T3
ALPHA: PICK_COMMAND.

ORDERED BY:

DATA: START_TIME.

FILE : TERMINATOR.

CONTAINS:

DATA: DROP_REASON
DATA: DROP_TIME.

ASSOCIATED WITH:

ENTITY_TYPE: DROPPED_IMAGE.

OUTPUT FROM:

ALPHA: GHOST_TERMINATION
ALPHA: LOW_TERMINATION
ALPHA: REDUN_TERMINATION
ALPHA: TERM_TRACK.

FILE : T1_T2_DATA.

CONTAINS:

DATA: T1_T2_RECORD.

MAKES:

MESSAGE: T1_T2_RETURN.

INPUT TO:

ALPHA: T1_T2_MEASUREMENT_EXTRACTION.

FILE : T1_T2_GATE.

CONTAINS:

DATA: T1_T2_GATE_DATA.

MAKES:

MESSAGE: T1_T2_COMMAND.

OUTPUT FROM:

ALPHA: FORM_T1_T2.

FILE : T1_T2_WINDOW.

CONTAINS:

DATA: T1_T2_WINDOW_DATA.

ASSOCIATED WITH:

ENTITY_TYPE: T1_T2_PULSE.

OUTPUT FROM:

ALPHA: FORM_T1_T2.

FILE : T3_DATA.

CONTAINS:

DATA: T3_RECORD.

MAKES:

MESSAGE: T3_RETURN.

INPUT TO:

ALPHA: T3_MEASUREMENT_EXTRACTION.

FILE : T3_GATE.

CONTAINS:

DATA: T3_GATE_DATA.

MAKES:

MESSAGE: T3_COMMAND.

OUTPUT FROM:

ALPHA: FORM_T3.

FILE : T3_WINDOW.

CONTAINS:

DATA: T3_WINDOW_DATA.

ASSOCIATED WITH:

ENTITY_TYPE: T3_PULSE.

OUTPUT FROM:

ALPHA: FORM_T3.

FILE : WAVEFORM_TABLE.

CONTAINS:

DATA: WF_CHARACTERISTICS

DATA: WF_NAME.

INPUT TO:

ALPHA: ALLOCATE_AND_CONTROL_RESOURCES

ALPHA: PICK_CANDIDATES

ALPHA: SUMMARIZE_USAGE.

OUTPUT FROM:

ALPHA: STARTER.

INPUT_INTERFACE : CC_IN.
CONNECTS TO:
SUBSYSTEM: SSC2.
ENABLES:
R_NET: CC_RESPONSE.
PASSES:
MESSAGE: HANDOVER
MESSAGE: MODE_CHANGE
MESSAGE: TERMINATION.
REFERRED BY:
R_NET : CC_RESPONSE.

INPUT_INTERFACE : RADAR_CLOCK_IN.
CONNECTS TO:
SUBSYSTEM: SSRADAR.
ENABLES:
R_NET: RADAR_TIMING.
PASSES:
MESSAGE: R_CLOCK_MESSAGE.
EXPLAINED BY:
REFERENCE: TLS_RADAR_DPS_INTERFACE_SPECIFICATION.
REFERRED BY:
R_NET : RADAR_TIMING.

INPUT_INTERFACE : RADAR_IN.
DESCRIPTION:

"THE RADIN INTERFACE PROVIDES THE MECHANISM THROUGH WHICH THE DPS RECEIVES RADAR SUBSYSTEM RETURNS IN RESPONSE TO RADAR COMMANDS ISSUED BY THE DPS THROUGH THE RADOUT INTERFACE. RADAR SUBSYSTEM RETURN MESSAGES SHALL COMPLY WITH REQUIREMENTS SPECIFIED IN THE TLS RADAR_DPS INTERFACE SPECIFICATION, PARAGRAPH 3_2_6.".
"H.A.HELTON, MAY. 3, 1976."

ENTERED BY:
CONNECTS TO:
SUBSYSTEM: SSRADAR.
ENABLES:
R_NET: RESPONSE_TO_RADAR.
IMPLEMENTS:
VERSION: ORIGINAL_PUBLICATION_DATED_AUGUST_1975.
PASSES:
MESSAGE: T1_T2_RETURN
MESSAGE: T3_RETURN.
ABBREVIATED BY:
SYNONYM: RADIN.
EXPLAINED BY:
REFERENCE: TLS_RADAR_DPS_INTERFACE_SPECIFICATION.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_PARAGRAPH_3_2.
REFERRED BY:
R_NET : RESPONSE_TO_RADAR.

MESSAGE : ACKNOWLEDGEMENT.
FORMED BY:
ALPHA: ACKNOWLEDGE.
MADE BY:
DATA: COMMAND_ID.
PASSED THROUGH:
OUTPUT_INTERFACE: CC_OUT.

MESSAGE : HANDOVER.
MADE BY:
DATA: COMMAND_ID
DATA: HO_ID
DATA: INITIAL_COVARIANCE
DATA: INITIAL_STATE.
PASSED THROUGH:
INPUT_INTERFACE: CC_IN.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_1_A_FUNCTIONAL.

MESSAGE : MODE_CHANGE.
MADE BY:
DATA: COMMAND_ID.
PASSED THROUGH:
INPUT_INTERFACE: CC_IN.

MESSAGE : R_CLOCK_MESSAGE.
MADE BY:
DATA: RADAR_CLOCK_TIME.
PASSED THROUGH:
INPUT_INTERFACE: RADAR_CLOCK_IN.

MESSAGE : RADAR_USAGE.
FORMED BY:
ALPHA: COMPLETE_SUMMARY.
MADE BY:
DATA: DATA_RECORD_TYPE
DATA: ENGAGEMENT_TIME
DATA: RESOURCES.
PASSED THROUGH:
OUTPUT_INTERFACE: DATA_RECORD.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_5_D_FUNCTIONAL.

MESSAGE : STATE_UPDATE.
FORMED BY:
ALPHA: UPDATE_STATE.
MADE BY:
DATA: CURRENT_STATE
DATA: DATA_RECORD_TYPE
DATA: HO_ID.
PASSED THROUGH:
OUTPUT_INTERFACE: DATA_RECORD.
TRACED FROM:
ORIGINATING_REQUIREMENT: DPSPR_3_2_5_C_FUNCTIONAL.

MESSAGE : TERMINATION.

MADE BY:

DATA: COMMAND_ID

DATA: HO_ID.

PASSED THROUGH:

INPUT_INTERFACE: CC_IN.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_2_E_FUNCTIONAL.

MESSAGE : TRACK_INITIATION.

FORMED BY:

ALPHA: TRACK_INITIATE.

MADE BY:

DATA: DATA_RECORD_TYPE

DATA: HO_ID

DATA: INITIAL_STATE

DATA: TIME_OF_INITIATION.

PASSED THROUGH:

OUTPUT_INTERFACE: DATA_RECORD.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_5_A_FUNCTIONAL.

MESSAGE : TRACK_TERMINATION.

FORMED BY:

ALPHA: GHOST_TERMINATION

ALPHA: LOW_TERMINATION

ALPHA: REDUN_TERMINATION

ALPHA: TERM_TRACK.

MADE BY:

DATA: DATA_RECORD_TYPE

DATA: HO_ID

DATA: REASON_FOR_DROP

DATA: TIME_OF_DROP.

PASSED THROUGH:

OUTPUT_INTERFACE: DATA_RECORD.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_5_B_FUNCTIONAL.

MESSAGE : T1_T2_COMMAND.

EQUATED TO:

SYNONYM: T1T2CMD.

EXPLAINED BY:

REFERENCE: TLS_RADAR_DPS_INTERFACE_SPECIFICATION.

FORMED BY:

ALPHA: FORM_T1_T2.

MADE BY:

DATA: RADAR_TYPE

DATA: RR_ORDER_ID

DATA: TRANSMIT_START

DATA: T1_T2_TRANSMIT

FILE: T1_T2_GATE.

PASSED THROUGH:

OUTPUT_INTERFACE: RADAR_OUT.

MESSAGE : T1_T2_RETURN.

EQUATED TO:

SYNONYM: T1T2RTN.

EXPLAINED BY:

REFERENCE: TLS_RADAR_DPS_INTERFACE_SPECIFICATION.

MADE BY:

DATA: RADAR_TYPE

DATA: RR_ORDER_ID

DATA: T1_T2_RECEIVE

FILE: T1_T2_DATA.

PASSED THROUGH:

INPUT_INTERFACE: RADAR_IN.

MESSAGE : T3_COMMAND.

EQUATED TO:

SYNONYM: T3CMD.

EXPLAINED BY:

REFERENCE: TLS_RADAR_DPS_INTERFACE_SPECIFICATION.

FORMED BY:

ALPHA: FORM_T3.

MADE BY:

DATA: RADAR_TYPE

DATA: RR_ORDER_ID

DATA: TRANSMIT_START

DATA: T3_TRANSMIT

FILE: T3_GATE.

PASSED THROUGH:

OUTPUT_INTERFACE: RADAR_OUT.

MESSAGE : T3_RETURN.

EQUATED TO:

SYNONYM: T3RTN.

EXPLAINED BY:

REFERENCE: TLS_RADAR_DPS_INTERFACE_SPECIFICATION.

MADE BY:

DATA: RADAR_TYPE

DATA: RR_ORDER_ID

DATA: T3_RECEIVE

FILE: T3_DATA.

PASSED THROUGH:

INPUT_INTERFACE: RADAR_IN.

ORIGINATING_REQUIREMENT : DPSPR_PARAGRAPH_3_2.

TRACES TO:

INPUT_INTERFACE: RADAR_IN

OUTPUT_INTERFACE: RADAR_OUT.

ORIGINATING_REQUIREMENT : DPSPR_3_2_1_A_FUNCTIONAL.

DESCRIPTION:

"ACTIONS: ACCEPT C2 MESSAGE, INITIATE TRACK ON IMAGE,
SEND RADAR ORDER

INFORMATION: C2 MESSAGE "INITIATE TRACK COMMAND",
HANDOVER IMAGE, RADAR ORDER".

TRACES TO:

ALPHA: TRACK_INITIATE

ALPHA: VALIDATE_HEADER

MESSAGE: HANDOVER.

ORIGINATING_REQUIREMENT : DPSPR_3_2_2_A_FUNCTIONAL.

DESCRIPTION:

" ACTION: SEND RADAR ORDER

INFORMATION: RADAR, REDUNDANT IMAGE.".

TRACES TO:

ALPHA: REDUN_DETERMINATION

ALPHA: REDUN_TERMINATION

DATA: DROP_REASON

DATA: REASON_FOR_DROP

DATA: REDUNDANT_IMAGE.

ORIGINATING_REQUIREMENT : DPSPR_3_2_2_A_PERFORMANCE.

TRACES TO:

DECISION: TRACK_PERFORMANCE_ALLOCATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_2_B_FUNCTIONAL.

DESCRIPTION:

" ACTIONS: SEND RADAR ORDER

INFORMATION: GHOST IMAGE, RADAR ORDER.".

TRACES TO:

ALPHA: GHOST_DETERMINATION

ALPHA: GHOST_TERMINATION

DATA: DROP_REASON

DATA: GHOST_IMAGE

DATA: REASON_FOR_DROP.

ORIGINATING_REQUIREMENT : DPSPR_3_2_2_B_PERFORMANCE.

TRACES TO:

ALPHA: ALLOCATE_AND_CONTROL_RESOURCES

ALPHA: FIND_CONFLICT

ALPHA: GHOST_DETERMINATION

ALPHA: REDUN_DETERMINATION

ALPHA: UPDATE_STATE

DATA: PRIORITY

DECISION: TRACK_PERFORMANCE_ALLOCATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_2_C_FUNCTIONAL.

DESCRIPTION:

" ACTION: SEND RADAR ORDER
INFORMATION: RADAR ORDER, ELEVATION OF RADAR ORDER".

TRACES TO:

ALPHA: FORM_T1_T2
ALPHA: FORM_T3
DATA: LOW_ELEVATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_2_D_FUNCTIONAL.

DESCRIPTION:

" ACTION: SEND RADAR ORDER, DETERMINE_IMAGE_ELEVATION
INFORMATION: IMAGE, ELEVATION OF IMAGE, RADAR ORD. 2,
TRANSMISSION TIME OF RADAR ORDER".

TRACES TO:

ALPHA: LOW_ELEVATION_DETERMINATION
ALPHA: LOW_TERMINATION
ALPHA: UPDATE_STATE
DATA: DROP_REASON
DATA: REASON_FOR_DROP.

ORIGINATING_REQUIREMENT : DPSPR_3_2_2_E_FUNCTIONAL.

DESCRIPTION:

" ACTION: DROP TRACK ON HANDOVER IMAGE, SEND RADAR ORDER
INFORMATION: DROP TRACK C2 MESSAGE, RADAR OR
DER, TRANSMISSION
TIME OF RADAR ORDER.".

TRACES TO:

ALPHA: TERM_TRACK
DATA: DROP_REASON
DATA: ENTRY_TIME
DATA: REASON_FOR_DROP
MESSAGE: TERMINATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_A_FUNCTIONAL.

DESCRIPTION:

" ACTION: MAINTAIN TRACK ON IMAGE
INFORMATION: IMAGE"ESTIMATE OF STATE", RADAR RETURN,
TIME OF LAST PROCESSED RETURN.".

TRACES TO:

ALPHA: UPDATE_STATE
DATA: STATE
DECISION: SYNCHRONOUS_VS_ASYNCHRONOUS_TRACK
ENTITY_TYPE: IMAGE_IN_TRACK.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_A_PERFORMANCE.

TRACES TO:

DECISION: TRACK_PERFORMANCE_ALLOCATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_B_FUNCTIONAL.

DESCRIPTION:

" ACTION: DROP IMAGE"REDUNDANT"
INFORMATION: REDUNDANT IMAGE.".

TRACES TO:

ALPHA: REDUN_DETERMINATION
DATA: REDUNDANT_IMAGE.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_B_PERFORMANCE.
TRACES TO:

DECISION: TRACK_PERFORMANCE_ALLOCATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_C_FUNCTIONAL.
DESCRIPTION:

" ACTION: DROP IMAGE "GHOST"
INFORMATION: GHOST IMAGE."

TRACES TO:

ALPHA: GHOST_DETERMINATION

DATA: GHOST_IMAGE.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_C_PERFORMANCE.

TRACES TO:

DECISION: TRACK_PERFORMANCE_ALLOCATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_D_FUNCTIONAL.
DESCRIPTION:

" ACTION: SEND RADAR ORDER
INFORMATION: RADAR ORDER."

TRACES TO:

ALPHA: ALLOCATE_AND_CONTROL_RESOURCES.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_D_PERFORMANCE.

TRACES TO:

DECISION: TRACK_PERFORMANCE_ALLOCATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_3_E_FUNCTIONAL.
DESCRIPTION:

"ACTION: SEND RADAR ORDER
INFORMATION: RADAR ORDER."

TRACES TO:

ALPHA: FIND_CONFLICT

ALPHA: FORM_T1_T2

ALPHA: FORM_T3.

ORIGINATING_REQUIREMENT : DPSPR_3_2_4_A_FUNCTIONAL.
DESCRIPTION:

"ACTION: MAINTAIN ESTIMATE OF RADAR RESOURCES
INFORMATION: RADAR RESOURCE USAGE ESTIMATE, RADAR
ENERGY ESTIMATE, UPPER BOUND ESTIMATE."

TRACES TO:

ALPHA: SUMMARIZE_USAGE

ENTITY_CLASS: PULSE.

ORIGINATING_REQUIREMENT : DPSPR_3_2_4_B_FUNCTIONAL.
DESCRIPTION:

"ACTION: ALLOCATE RADAR ORDERS
INFORMATION: RADAR ORDERS, IMAGE."

TRACES TO:

ALPHA: ALLOCATE_AND_CONTROL_RESOURCES

DECISION: RADAR_SCHEDULER_PRIORITIZATION

R_NET: CONTROL_RESOURCES.

ORIGINATING_REQUIREMENT : DPSPR_3_2_5_A_FUNCTIONAL.

DESCRIPTION:

"ACTION: OUTPUT TO PERMANENT FILE
INFORMATION: TIME OF APPEARANCE OF C2 MESSAGE,
HANDOVER IMAGE (ESTIMATED STATE).".

TRACES TO:

ALPHA: TRACK_INITIATE
MESSAGE: TRACK_INITIATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_5_B_FUNCTIONAL.

DESCRIPTION:

"ACTION: OUTPUT TO PERMANENT FILE
INFORMATION: TIME OF DROP TRACK, REASON FOR
DROP TRACK.".

TRACES TO:

ALPHA: TERM_TRACK
MESSAGE: TRACK_TERMINATION.

ORIGINATING_REQUIREMENT : DPSPR_3_2_5_C_FUNCTIONAL.

DESCRIPTION:

"ACTION: OUTPUT TO PERMANENT FILE, UPDATE STATE
INFORMATION: IMAGE STATE ESTIMATE.".

TRACES TO:

MESSAGE: STATE_UPDATE
R_NET: RESPONSE_TO_RADAR.

ORIGINATING_REQUIREMENT : DPSPR_3_2_5_D_FUNCTIONAL.

DESCRIPTION:

"ACTION: OUTPUT TO PERMANENT FILE
INFORMATION: RADAR RESOURCE USAGE.".

TRACES TO:

ALPHA: SUMMARIZE_USAGE
MESSAGE: RADAR_USAGE
R_NET: RADAR_SUMMARY.

ORIGINATING_REQUIREMENT : RADAR_DPS_IFS_3_2_9_FUNCTIONAL.

TRACES TO:

DATA: RADAR_CLOCK_TIME.

OUTPUT_INTERFACE : CC_OUT.

CONNECTS TO:

SUBSYSTEM: SSC2.

PASSES:

MESSAGE: ACKNOWLEDGEMENT.

REFERRED BY:

R_NET : CC_RESPONSE.

OUTPUT_INTERFACE : DATA_RECORD.

CONNECTS TO:

SUBSYSTEM: SSPERMFL.

PASSES:

MESSAGE: RADAR_USAGE

MESSAGE: STATE_UPDATE

MESSAGE: TRACK_INITIATION

MESSAGE: TRACK_TERMINATION.

REFERRED BY:

R_NET : CC_RESPONSE

R_NET : RADAR_SUMMARY

R_NET : RESPONSE_TO_RADAR.

OUTPUT_INTERFACE : RADAR_OUT.

DESCRIPTION:

"THE RADOUT INTERFACE PROVIDES THE MECHANISM THROUGH WHICH THE DPS ISSUES COMMANDS TO THE RADAR SUBSYSTEM. THE RADAR SUBSYSTEM WILL EXECUTE ONLY THE COMMANDS ISSUED BY THE DPS AND WILL TRANSMIT ONE PULSE FOR EACH COMMAND WHICH SATISFIES THE RADAR SUBSYSTEM AND INTERFACE CONSTRAINTS. THE RADAR SUBSYSTEM WILL EXECUTE THE COMMANDS IN THE ORDER RECEIVED AND WILL BEGIN EXECUTION AFTER RECEIPT OF END_OF_TRANSMISSION."

ENTERED BY:

"H.A.HELTON, APR. 30, 1976."

CONNECTS TO:

SUBSYSTEM: SSRADAR.

IMPLEMENTS:

VERSION: ORIGINAL_PUBLICATION_DATED_AUGUST_1975.

PASSES:

MESSAGE: T1_T2_COMMAND

MESSAGE: T3_COMMAND.

ABBREVIATED BY:

SYNONYM: RADOUT.

EXPLAINED BY:

REFERENCE: TLS_RADAR_DPS_INTERFACE_SPECIFICATION.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_PARAGRAPH_3_2.

REFERRED BY:

R_NET : XMIT_R.

R_NET : CC_RESPONSE.

REFERS TO:

ALPHA : ACKNOWLEDGE
ALPHA : CC_ERROR_PROCESSING
ALPHA : ENGAGEMENT_INITIATION
ALPHA : STARTER
ALPHA : TERM_ENGAGEMENT
ALPHA : TERM_TRACK
ALPHA : TRACK_INITIATE
ALPHA : VALIDATE_HEADER
DATA : COMMAND_ID
EVENT : ALLOCATE
EVENT : SCHEDULE
EVENT : SUMMARIZE
INPUT_INTERFACE : CC_IN
OUTPUT_INTERFACE : CC_OUT
OUTPUT_INTERFACE : DATA_RECORD.

ENABLED BY:

INPUT_INTERFACE: CC_IN.

STRUCTURE:

INPUT_INTERFACE : CC_IN
ALPHA : VALIDATE_HEADER

DO

ALPHA : ACKNOWLEDGE
OUTPUT_INTERFACE : CC_OUT

AND

CONSIDER DATA : COMMAND_ID
DO

(HANDOVER_IMAGE)

ALPHA : TRACK_INITIATE
EVENT : ALLOCATE
OUTPUT_INTERFACE : DATA_RECORD

OR

(DROP_TRACK)

ALPHA : TERM_TRACK
OUTPUT_INTERFACE : DATA_RECORD

OR

(INITIATE_ENGAGEMENT_MODE)

ALPHA : STARTER
ALPHA : ENGAGEMENT_INITIATION
EVENT : SCHEDULE
EVENT : SUMMARIZE
TERMINATE

OR

(TERMINATE_ENGAGEMENT_MODE)

ALPHA : TERM_ENGAGEMENT
TERMINATE

OTHERWISE

ALPHA : CC_ERROR_PROCESSING
TERMINATE

END

END

END .

R_NET : CONTROL_RESOURCES.

DESCRIPTION:

"THE TLS_DPS SHALL IMPLEMENT THE REQUIREMENTS SPECIFIED IN THE DPSPR, REFERENCE 2.2, ASSOCIATED WITH MANAGEMENT AND CONTROL OF TLS RESOURCES AND SHALL PERFORM THE FUNCTIONS HEREIN DEFINED AND DIAGRAMMED IN THE CONTRES R_NET.

THE DPS SHALL ASSIGN TRACK RATES AND ENERGY ALLOWANCES TO EACH IMAGE HANDED OVER FROM COMMAND AND CONTROL AND SHALL DETERMINE ENERGY BOUNDS AND TRACK RATE ENVELOPES FOR EACH HANDOVER IMAGE WHICH REMAINS IN TRACK STATUS. THE ENERGY BOUNDS AND TRACK RATE ENVELOPES SHALL BE MAINTAINED WITHIN AN ACCURACY AND RESOLUTION TOLERANCE SUFFICIENT TO MEET THE SPECIFIED REQUIREMENTS FOR DETERMINATION OF TARGET INTERCEPT CONDITIONS.

THE DPS SHALL ASSESS STATUS OF THE TLS RESOURCES AND SHALL ALLOCATE TLS RESOURCES TO EACH HANDOVER IMAGE BASED ON RADAR SUBSYSTEM PERFORMANCE CAPABILITIES AND SHALL MAINTAIN A GRACEFUL DEGRADATION POSTURE WHILE UNDER OVERLOAD CONDITIONS.

THE DPS SHALL GENERATE TLS RESOURCE UTILIZATION PROFILES AND SHALL COMMIT THESE DATA TO PERMANENT FILE THROUGH THE DATA RECORD OUTPUT INTERFACE."

REFERS TO:

ALPHA : ALLOCATE_AND_CONTROL_RESOURCES.

ABBREVIATED BY:

SYNONYM: CONTRES.

ENABLED BY:

EVENT: ALLOCATE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_4_B_FUNCTIONAL.

STRUCTURE:

ALPHA : ALLOCATE_AND_CONTROL_RESOURCES

TERMINATE

END .

R_NET : RADAR_SUMMARY.

REFERS TO:

ALPHA : COMPLETE_SUMMARY

ALPHA : SUMMARIZE_USAGE

DATA : MODE

ENTITY_TYPE : RETURNED_PULSE

EVENT : SUMMARIZE

OUTPUT_INTERFACE : DATA_RECORD.

ENABLED BY:

EVENT: SUMMARIZE.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_5_D_FUNCTIONAL.

STRUCTURE:

CONSIDER DATA : MODE

DO

(ENGAGED)

FOR EACH ENTITY_TYPE : RETURNED_PULSE

DO ALPHA : SUMMARIZE_USAGE END

ALPHA : COMPLETE_SUMMARY

EVENT : SUMMARIZE

OUTPUT_INTERFACE : DATA_RECORD

OTHERWISE

TERMINATE

END

END .

R_NET : RADAR_TIMING.

DESCRIPTION:

"RADAR_TIMING MAINTAINS A RECORD OF THE
RADAR_CLOCK_TIME."

REFERS TO:

ALPHA : UPDATE_RADAR_CLOCK

INPUT_INTERFACE : RADAR_CLOCK_IN.

ENABLED BY:

INPUT_INTERFACE: RADAR_CLOCK_IN.

STRUCTURE:

INPUT_INTERFACE : RADAR_CLOCK_IN

ALPHA : UPDATE_RADAR_CLOCK

TERMINATE

END .

R_NET : RESPONSE_TO_RADAR.

DESCRIPTION:

"THE TLS_DPS SHALL IMPLEMENT THE REQUIREMENTS SPECIFIED IN THE TLS_RADAR_DPS_INTERFACE_SPECIFICATION ASSOCIATED WITH PROCESSING RADAR SUBSYSTEM RESPONSES TO COMMANDS, AND SHALL PERFORM THE FUNCTIONS HEREIN DEFINED AND DIAGRAMMED IN THE RESPRAD R_NET.

THE DPS SHALL RECEIVE AND PROCESS RADAR MESSAGES TRANSMITTED BY THE TLS RADAR SUBSYSTEM AND SHALL INTERROGATE EACH MESSAGE FOR ERRORS AND SHALL PROCESS ALL DETECTED MESSAGE ERRORS.

THE DPS SHALL, UPON RECEIPT OF ERROR_FREE RADAR MESSAGES, DETECT AND PROCESS REDUNDANT_IMAGES, GHOST_IMAGES, AND LOW_ELEVATION_IMAGES, AND SHALL UPDATE STATE_PARAMETERS FOR EACH IMAGE_IN_TRACK.

THE DPS SHALL TERMINATE TRACK ON EACH IMAGE WHICH IS DETERMINED TO BE EITHER A REDUNDANT OR GHOST IMAGE OR WHICH IS FOUND TO EXCEED THE LOW_ELEVATION CONSTRAINTS AND SHALL MAINTAIN TRACK ON EACH IMAGE DETERMINED TO BE REAL.

THE DPS SHALL CONSTRUCT AND MAINTAIN DESCRIPTIVE DATA FILES FOR EACH IMAGE WHICH IS EITHER MAINTAINED IN TRACK OR DROPPED FROM TRACK AND SHALL PERIODICALLY COMMIT THESE DATA TO PERMANENT RECORDS THROUGH THE DATA RECORDS INTERFACE."

REFERS TO:

ALPHA : ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE

ALPHA : GHOST_DETERMINATION

ALPHA : GHOST_TERMINATION

ALPHA : LOW_ELEVATION_DETERMINATION

ALPHA : LOW_TERMINATION

ALPHA : REDUN_DETERMINATION

ALPHA : REDUN_TERMINATION

ALPHA : RR_ERROR_PROCESSING

ALPHA : T1_T2_MEASUREMENT_EXTRACTION

ALPHA : T3_MEASUREMENT_EXTRACTION

ALPHA : UPDATE_STATE

DATA : GHOST_IMAGE

DATA : LOW_ELEVATION

DATA : RADAR_TYPE

DATA : REDUNDANT_IMAGE

DATA : RETURN_IMAGE_STATUS

DATA : VALID_RETURN

EVENT : ALLOCATE

INPUT_INTERFACE : RADAR_IN

OUTPUT_INTERFACE : DATA_RECORD.

ABBREVIATED BY:

SYNONYM: RESPRAD.

ENABLED BY:

INPUT_INTERFACE: RADAR_IN.

TRACED FROM:

ORIGINATING_REQUIREMENT: DPSPR_3_2_5_C_FUNCTIONAL.

STRUCTURE:

INPUT_INTERFACE : RADAR_IN
 ALPHA : ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE
 CONSIDER DATA : RETURN_IMAGE_STATUS
 DO

(IN_TRACK)

CONSIDER DATA : RADAR_TYPE
 DO

(T3)

ALPHA : T3_MEASUREMENT_EXTRACTION
 OTHERWISE

ALPHA : T1_T2_MEASUREMENT_EXTRACTION
 END
 DO

(VALID_RETURN)

ALPHA : UPDATE_STATE
 DO

OUTPUT_INTERFACE : DATA_RECORD

AND

ALPHA : REDUN_DETERMINATION
 DO

(REDUNDANT_IMAGE)

ALPHA : REDUN_TERMINATION

EVENT : ALLOCATE

OUTPUT_INTERFACE : DATA_RECORD

OTHERWISE

TERMINATE

END

AND

ALPHA : LOW_ELEVATION_DETERMINATION
 DO

(LOW_ELEVATION)

ALPHA : LOW_TERMINATION

EVENT : ALLOCATE

OUTPUT_INTERFACE : DATA_RECORD

OTHERWISE

TERMINATE

END

END

OTHERWISE

ALPHA : GHOST_DETERMINATION
 DO

(GHOST_IMAGE)

ALPHA : GHOST_TERMINATION

EVENT : ALLOCATE

OUTPUT_INTERFACE : DATA_RECORD

OTHERWISE

TERMINATE

END

END

OR

(DROPPED)

TERMINATE

OTHERWISE

ALPHA : RH_ERROR_PROCESSING
 TERMINATE

END

END .

R_NET : SKED_R.

DESCRIPTION:

"SKED_R CONSTRUCTS THE ORDERED FILE OF DATA FOR
A FRAME."

REFERS TO:

ALPHA : INITIALIZE_SKED_R
ALPHA : PICK_CANDIDATES
DATA : LAST_PULSE
DATA : MODE
DATA : TEOF
DATA : TRACK_RATE
ENTITY_TYPE : IMAGE_IN_TRACK
EVENT : XRB
FILE : CANDIDATE
SUBNET : FORM_FRAME.

ENABLED BY:

EVENT: SCHEDULE.

TRACED FROM:

DECISION: RADAR_SCHEDULER_PRIORITIZATION.

STRUCTURE:

CONSIDER DATA : MODE

DO

(ENGAGED)

ALPHA : INITIALIZE_SKED_R
FOR EACH ENTITY_TYPE : IMAGE_IN_TRACK SUCH THAT
(LAST_PULSE+(1.0/TRACK_RATE)<TEOF)

DO ALPHA : PICK_CANDIDATES END

FOR EACH FILE : CANDIDATE

DO SUBNET : FORM_FRAME END

EVENT : XRB

TERMINATE

OTHERWISE

TERMINATE

END

END .

R_NET : XMIT_R.

DESCRIPTION:

"XMIT_R BUILDS AND FORWARDS TO THE OUTPUT_INTERFACE
RADAR_OUT THE COMMANDS OF THE FRAME."

REFERS TO:

ALPHA : FORM_T1_T2
ALPHA : FORM_T3
ALPHA : PICK_COMMAND
DATA : FOUND
DATA : RADAR_TYPE
EVENT : SCHEDULE
EVENT : XRB
OUTPUT_INTERFACE : RADAR_OUT.

ENABLED BY:

EVENT: XRB.

TRACED FROM:

DECISION: RADAR_SCHEDULER_PRIORITIZATION.

STRUCTURE:

ALPHA : PICK_COMMAND
DO
 (FOUND)
 EVENT : XRB
 CONSIDER DATA : RADAR_TYPE
 DO
 (T3)
 ALPHA : FORM_T3
 OTHERWISE
 ALPHA : FORM_T1_T2
 END
 OUTPUT_INTERFACE : RADAR_OUT
OTHERWISE
 EVENT : SCHEDULE
 TERMINATE
END
END .

REFERENCE : CISS_TDP_BASELINE_CONSTRUCT.

DESCRIPTION:

"TRW REPORT NO. 22944_9771_RE_01, VOLUME I, REVISION A."

REFERENCE : DPSPR_SPECIFICATION.

DESCRIPTION:

"TRW REPORT NO. 27332_6921_015, REVISION 1, CDRL A00E,
SECTION 3.2".

REFERENCE : TLS_C2_DPS_INTERFACE_SPECIFICATION.

DESCRIPTION:

"TRW REPORT NO. 27332_6921_013."

REFERENCE : TLS_ENVIRONMENT_AND_THREAT_MODEL.

DESCRIPTION:

"GRC REPORT NO. DRC_72_25499 (SECRET), VERSION I."

REFERENCE : TLS_RADAR_DPS_INTERFACE_SPECIFICATION.

DESCRIPTION:

"TRW REPORT NO. 27332_6921_012."

EXPLAINS:

INPUT_INTERFACE: RADAR_CLOCK_IN

INPUT_INTERFACE: RADAR_IN

MESSAGE: T1_T2_COMMAND

MESSAGE: T1_T2_RETURN

MESSAGE: T3_COMMAND

MESSAGE: T3_RETURN

OUTPUT_INTERFACE: RADAR_OUT.

REFERENCE : TLS_RADAR_PERFORMANCE_SPEC.

DESCRIPTION:

"GRC REPORT NO. CR_3_386 (SECRET)".

REFERENCE : TLS_SYSTEM_REQUIREMENTS.

DESCRIPTION:

"TRW REPORT NO. 27332_6921_015, REVISION 1, CDRL A00E
SECTION 1.1".

SUMNET : FORM_FRAME.

DESCRIPTION:

FORM_FRAME PROVIDES RADAR CONFLICT RESOLUTION.

REFERS TO:

ALPHA : FIND_CONFLICT

ALPHA : MAKE_COMMAND

DATA : DROP_FLAG.

REFERRED BY:

R_NET : SKED_R.

STRUCTURE:

ALPHA : FIND_CONFLICT

OR

(NOT DROP_FLAG)

ALPHA : MAKE_COMMAND

OTHERWISE

END

RETURN

END .

SUBSYSTEM : SSC2.
CONNECTED TO:
INPUT_INTERFACE: CC_IN
OUTPUT_INTERFACE: CC_OUT.

SUBSYSTEM : SSPERMFL.
CONNECTED TO:
OUTPUT_INTERFACE: DATA_RECORD.

SUBSYSTEM : SSRADAR.
CONNECTED TO:
INPUT_INTERFACE: RADAR_CLOCK_IN
INPUT_INTERFACE: RADAR_IN
OUTPUT_INTERFACE: RADAR_OUT.

SYNONYM : CKRADMES.

EQUATES TO:

ALPHA: ACCEPT_AND_CHECK_RADAR_RETURN_MESSAGE.

SYNONYM : CONTRES.

ABBREVIATES:

R_NET: CONTROL_RESOURCES.

SYNONYM : RADIN.

ABBREVIATES:

INPUT_INTERFACE: RADAR_IN.

SYNONYM : RADOUT.

ABBREVIATES:

OUTPUT_INTERFACE: RADAR_OUT.

SYNONYM : RESPRAD.

ABBREVIATES:

R_NET: RESPONSE_TO_RADAR.

SYNONYM : T1T2CMD.

EQUATES TO:

MESSAGE: T1_T2_COMMAND.

SYNONYM : T1T2RTN.

EQUATES TO:

MESSAGE: T1_T2_RETURN.

SYNONYM : T3CMD.

EQUATES TO:

MESSAGE: T3_COMMAND.

SYNONYM : T3RTN.

EQUATES TO:

MESSAGE: T3_RETURN.

VERSION : ORIGINAL_PUBLICATION_DATED_AUGUST_1975.
IMPLEMENTED BY:
INPUT_INTERFACE: RADAR_IN
OUTPUT_INTERFACE: RADAR_OUT.

APPENDIX F
TLS SOURCE SPECIFICATIONS

APPENDIX F-I: TRACK LOOP DPSR

I. INTRODUCTION

1.0 PURPOSE AND SCOPE

This report presents the preliminary Data Processing Subsystem Performance Requirements (DPSPR) for the Track Loop Experiment (X-1). The goals for this experiment are presented in Section 2.0.

The primary intent of this document is to provide the initiating input to the Software Requirements Engineering Methodology which will subsequently produce a Process Performance Requirements (PPR) specification for the Track Loop System Data Processing Subsystem. It is further the intent of this document to present an example of the required contents and level of detail of a DPSPR discussed in Reference 1. As such, this example will provide a more concrete basis for technical exchange between the Software Requirements Engineering, DPSPR and V&V contractors. Such interchange is considered necessary to arrive at a final definition of the required form, contents and format of a DPSPR. Part II of this report constitutes the example DPSPR.

1.1 The Track Loop System

The Track Loop System (TLS) is a subset of a Preliminary Ballistic Missile Defense System which is capable of nearly autonomous execution in response to external stimuli. It is the simplest known subsystem with properties of interest for software definition, and it is one which has been studied extensively, both in the academic literature and in such practical programs as Site Defense. Therefore, it has been selected as the testbed for supporting experimentation in development of the methodology for software requirements. A pictorial representation of the TLS is provided in Figure F-1.

1.1.1 Preliminary Ballistic Missile Defense System

A Preliminary Ballistic Missile Defense System (PBMDS) has been postulated as an environment in which the TLS would execute. It is a generalized representative of the class of systems currently in development, and is particularized for the TLS through representative but non-real specifications where required.

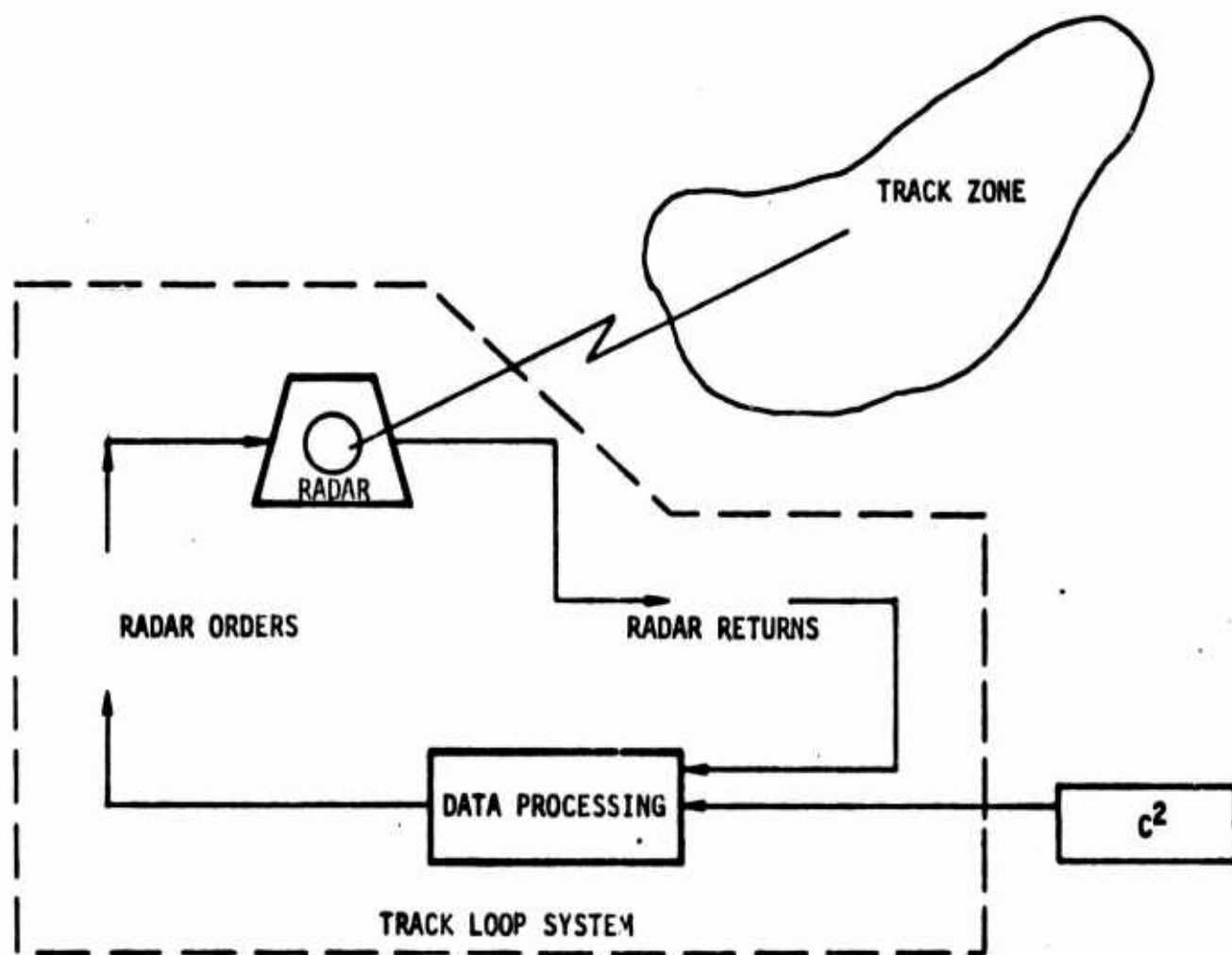


Figure F-1 Track Loop System

The top level flow of the PBMDS is shown in the functional flow block diagram, Figure F-2. In the Conduct Engagement mode, an object entering the search region will be detected and designated, tracked, discriminated, and engaged (as required) in defense of the ground facilities. Those functions are implemented through the Data Processing System (DPS), a radar or other sensor, and a means of neutralizing hostile objects. For the purpose of the TLS, only the radar need be defined in detail; other system elements are identified only to the extent that they impact DPS requirements.

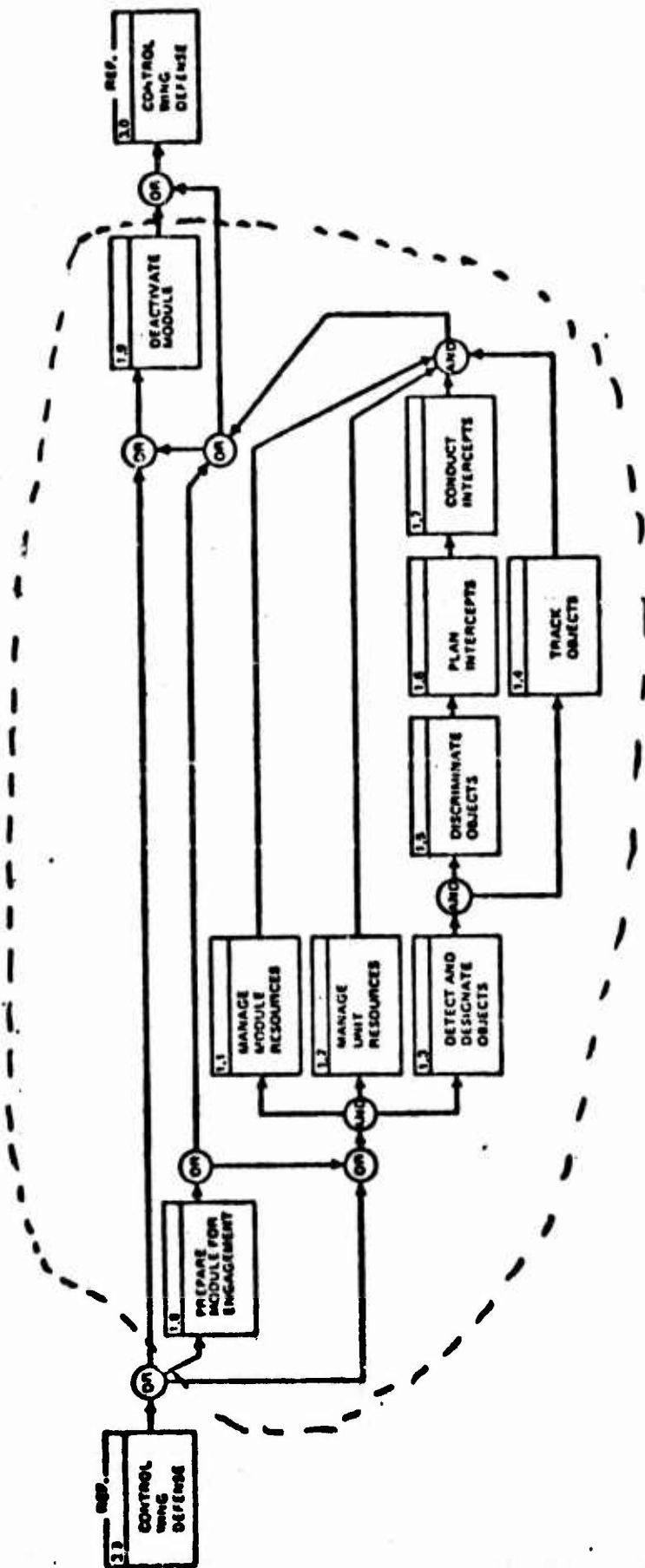
1.1.2 TLS Requirements

Functional Requirements on the TLS would normally be contained in a system specification (Level 1, or A in MIL STD 490 terminology). If software is developed from the requirements provided in Section II of this report, and if that software is to be installed and exercised in the field, then such a system specification may be required. In the interests of both economy and timeliness, only the subsection of the A Specification required for the DPSPR is provided here.

1.1.2.1 Initialization

The TLS shall accept C^2 messages for initialization with the following properties:

- a) An estimate of state shall be generated external to TLS and forwarded to TLS over the communication channel to initiate tracking.
- b) If an object corresponds to that estimate, the estimation accuracy shall be such that the expected (1-sigma) deviation of the object from a perfect extrapolation of state shall be in accord with Table F.1.
- c) Initialization estimates shall be provided (handed off) at a rate not exceeding 150 per second over any interval greater than 50 milliseconds.
- d) The total number of handoffs shall not exceed 1500.
- e) The total number of real objects shall not exceed 300.
- f) A handoff may be an estimate on a real object or an estimate relating to a state at which no object is located (ghost). Multiple handoffs of a single object may be generated.
- g) Each handoff shall consist of a unique designator, the state vector and its covariance matrix.



NOTES:
 1. 1.1 AND 1.2 PROVIDE DIRECTION TO AND RECEIVE INPUTS FROM 1.3, 1.4, 1.5, 1.6, AND 1.7 THROUGHOUT THE ENGAGEMENT.
 2. FURTHER FEEDBACKS AND INTERCONNECTIONS ARE SHOWN AT LOWER LEVELS WHERE THEY EXIST.
 3. ORDER SHOWN DOES NOT IMPLY THAT FUNCTION ENDS WHEN FOLLOWING FUNCTIONS BEGIN

Figure F-2 Conduct Engagement Function (PBIDS Function 1.0)

Table F.2 Table F.1 Handover Errors in Radar Face Coordinates

<u>COORDINATE</u>	<u>MINIMUM</u>	<u>MAXIMUM</u>	<u>UNITS</u>
R	3	5	M
U	0.4	0.6	msine
V	0.03	0.05	msine
$\dot{R}$	40	55	m/sec
$\dot{U}$	2	4	msine/sec
$\dot{V}$	2	4	msine/sec

NOTES:

1. All data 1- σ
2. Handover altitude ≥ 65 K meters

1.1.2.2 Termination

- a) Redundant images of objects shall be dropped from track in order to conserve radar resources. The probability of dropping track on a non-redundant image shall be considered in determining leakage.
- b) Ghosts shall be dropped from track in order to conserve radar resources. The probability that a non-redundant image is dropped as a ghost shall be considered in determining leakage.
- c) For flight safety, no track pulse shall be commanded with true elevation angle less than 3° .
- d) Track shall be dropped in response to an external command representing handoff to another defense facility or successful intercept. No track pulse shall be transmitted to a designated image more than 100 milliseconds after such a command appears at the TLS port, with probability .3.

1.1.2.3 Tracking

- a) The TLS shall generate state estimates sufficient to support discrimination through beta estimation accuracy in accordance with Figure F-3 and impact point prediction in accordance with Figure F-4.

Beta is required in the system for a variety of purposes at different stages in the engagement of an object. Early in track, it is used for junk rejection; at an intermediate state, it forms a key element of discrimination in elimination of decoys and assessment of the danger imposed by an RV; shortly thereafter, it is essential to intercept planning in estimating the intercept point. The needs overlap in practice, so that a common ordinate for the plot of accuracy requirements is needed. But each of the aspects of use of beta dictates a different ordinate at the system level. The ordinate selected in Figure F-3 is time in track, a parameter known to be useful to the Software Requirements Engineer, and as useful for the systems-level definition as any of the other choices.

Similarly, impact point prediction is used initially to reject cross traffic, then to assess the threat posed by an RV; in some schema, it also assists discrimination. The first might be expressed by the system engineer in terms of accuracy against track energy; the second in terms of time to commit contour (which is in turn a function of RV type, intercept capabilities, and other parameters). Again, time in track was chosen as a convenient ordinate for the error requirements in Figure F-4.

- b) The TLS shall generate state estimates sufficient to support object intercept in accordance with Figure (TBD).
- c) Leakage is here defined to be the probability that all images of a real object entered at an altitude not less than 150K feet are dropped from track for any reason other than the minimum-elevation constraints or external command defined in 1.1.2.2. Leakage in TLS shall not exceed .03.

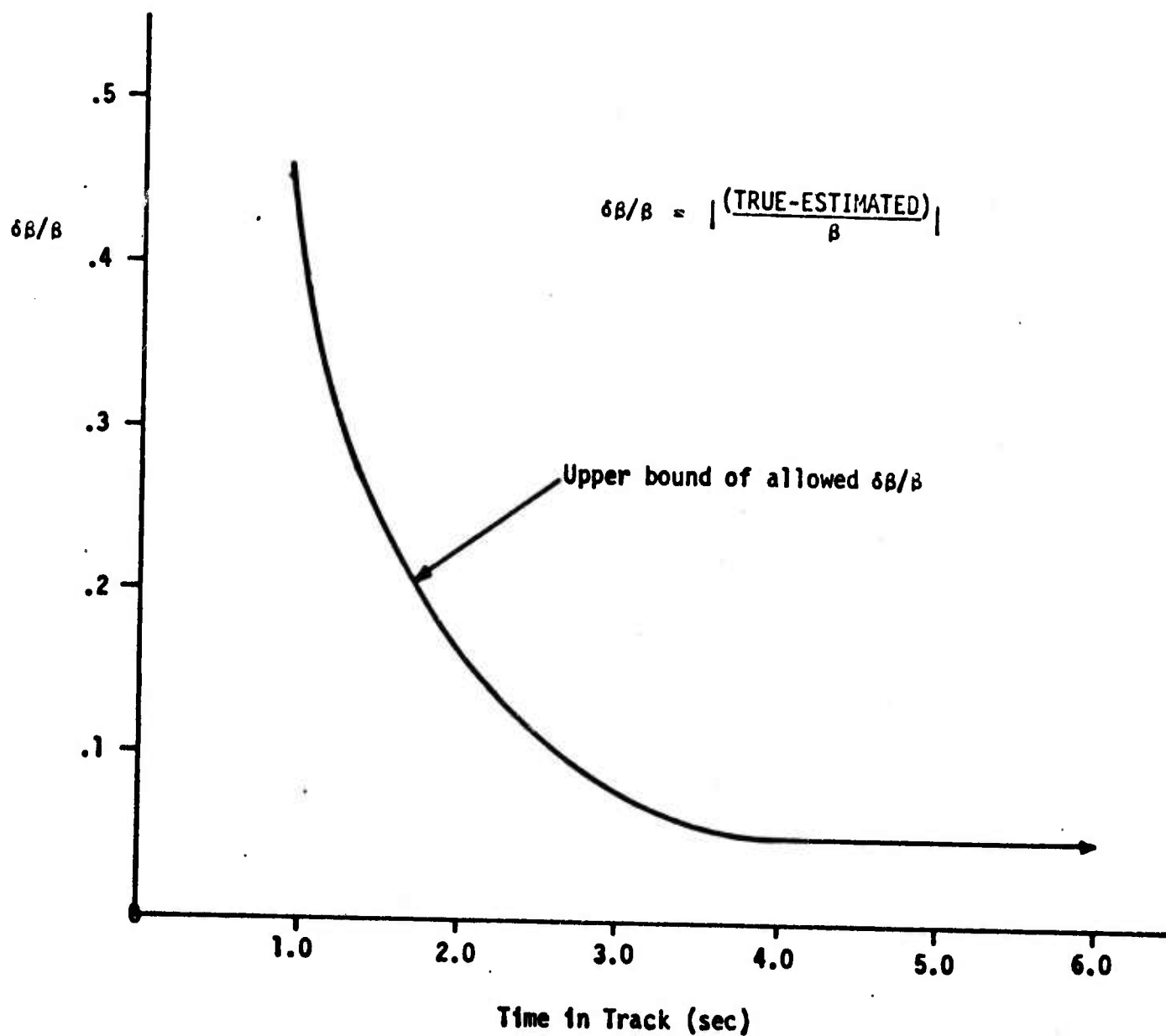


Figure F-3 Beta Error Requirements

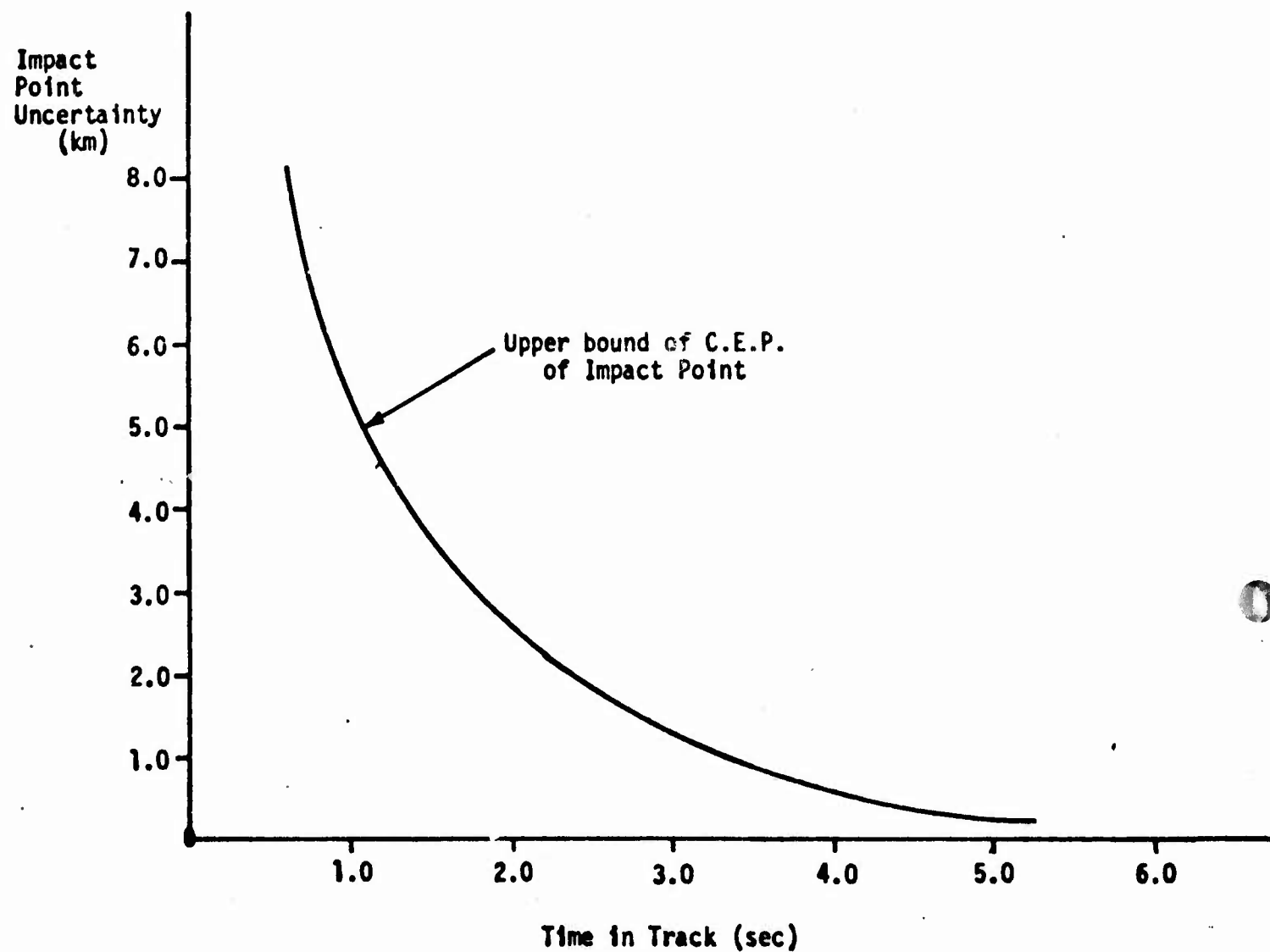


Figure F-4 Impact Point Error Requirements

1.1.2.4 Resource Control

- a) The TLS shall commit to tracking functions not more than 1800 radar pulses per second, using not more than 25 kilowatts ERP.
- b) The TLS shall commit to tracking functions not more than 2500 joules per object.

1.1.2.5 Data Recording

The TLS shall provide records for post test analysis of the following data:

- a) Time of handoff and designator and estimate received.
- b) Time of termination of track on each designation with reason for termination.
- c) At intervals not greater than 100 milliseconds, the estimate of state for each designation received and not yet terminated. That estimate shall consist of the following data: position, velocity, a beta estimation parameter and estimates of the uncertainty of each.
- d) At intervals not greater than 300 milliseconds, the usage of radar resources. That measure shall include: radar energy profile and DPS estimates of expected and maximum energy commanded.

2.0 SREP OBJECTIVES FOR X-1

- a) Demonstrate the scope, contents and level of detail of the DPSPR, described in Reference 1, and transmit the results to the DPSPR contractors for their evaluation.
- b) Demonstrate the scope, contents, format, and level of detail of PPR Volume I specified in Reference 2. The PPR will be written in preliminary RSL (see Reference 4). In particular:
 - 1) Evaluate the format of the PPR described in Reference 2 for completeness and adequacy.
 - 2) Evaluate the adequacy of the preliminary RSL definition for stating requirements.
 - 3) Provide the PPR for evaluation; in particular, the form and content of the performance requirements.
 - 4) Evaluate the extent of the traceability of the PPR to the DPSPR.
- c) Evaluate the steps of SREM (see Reference 3) from the DPSPR to a PPR; in particular, the procedural steps, form, and content of the performance allocation activities.
- d) Exercise the available experimental software to generate portions of the PPR (e.g., the structure segment of RSL).

3.0 LIMITATIONS

Two primary forces limit the application of the DPSPR of Section II. The first is the result of excising the track loop from the total PBMDs. Although the loop is nearly self-contained, it has extensive interfaces both in terms of requirements and in the sense of implementation with other software functions. Consequently, requirements are imposed on TLS which originate or terminate within the DPS in ways and to an extent not believed appropriate for a true, functional system.

The second major area of limitation arises from the novelty of the approach. While the methodology employed is believed valid, it has not yet been tried. (In fact, this experiment is its trial.) Consequently, it is likely that this initial DPSPR will be found to be incomplete in some materials needed for the PPR and for software development. Every effort has been made to anticipate such failings, but we anticipate methodologic bugs in the same way we would expect to find bugs in a new compiler or other major software development.

3.1 System Objectives

Isolation of TLS from PBMDs was artificial, and the objectives of Section 1.1 are correspondingly less than real. The selection of those objectives was based on simplicity of implementation, expected usefulness of the TLS as a testbed, and availability of comparison criteria. Since no DPSPR had ever been prepared before, it was not possible to select criteria to optimize its quality. The objectives provided are believed to be good for generation of the DPSPR, but cannot be shown to be optimal.

3.2 Allocation to DP

Again, the absence of precedent and the artificiality of the TLS make an "optimum" allocation of functions unreasonable. A complete allocation is provided, which is believed to be sufficient for the experiment.

3.3 Level of Detail

The objective of the DPSPR of Part II is to provide the complete specification required for the PPR Feasibility Demonstration. It is likely

that some of the material will prove to be unnecessary; it is certain that not all data presently missing will be required before initiating work on Volume I of the PPR. Therefore, revisions are planned both to complete specification of the requirements and to delete data found to be non-essential.

3.4 Formulation of DP Requirements

The X-1 DPSPR defines some data in a form and to a depth we feel to be undesirable. In particular, efforts are now under way to express requirements on Redundant Track Elimination in terms of total radar energy per object. Converting both redundant tracking and other explicit requirements to a derivable form will be an objective for revisions of the DPSPR.

3.5 Corollary Documents

The DPSPR is one document in a family required to develop the Process Performance Requirements (PPR). For the TLS, the other documents are:

- TLS System Requirements
- TLS Radar/DPS Interface Specification
- TLS C²/DPS Interface Specification
- TLS Radar Performance Specification*
- TLS Environment and Threat Model Definition.*

The system requirements are sketched in Section 1.1 above. The starred (*) documents in this family are not yet prepared.

For the purposes of development of the PPR, the absence of the documents is believed to be less than critical. For the X-1 effort, the contents of the missing specifications are subsets of the corresponding Terminal Defense Program (TDP) documents. Therefore, the required information for the PPR is available from TDP documentation. However, a property of the methodology is its recognition that not all specifications ultimately required to support software design will be available early in that design effort. To the extent that the TLS documentation corresponding to initiation of a PPR is required, the TDP documents are excessive. Thus, there is an effort required as a part of the ongoing work to edit the TDP material to the specific, appropriate contents for the stage of development represented by the DPSPR. That work is under way on the Radar/DPS Interface specification, and will soon be undertaken on the other elements of the DPSPR package.

4.0 REFERENCES

1. TRW Report 27332-6921-003, "Software Performance Requirements - DPSPR Content Requirements", (CDRL A004), Revision 1, December 12, 1975.
2. TRW Report 27332-6921-004, "Software Performance Requirements - PPR Content Requirements", (CDRL A005), Revision 1, December 12, 1975.
3. TRW Report 27332-6921-005, "Software Requirements Engineering Methodology Description Special Report", (CDRL A00C), Revision 1, December 12, 1975.
4. TRW IOC TEB-75-6000.02-135, "Requirements Statement Language Descriptions", 1 May 1975.

II. DATA PROCESSING SUBSYSTEM PERFORMANCE REQUIREMENTS - TRACK LOOP SYSTEM

1.0 SCOPE

This specification establishes the functional, performance, design and test requirements for the Track Loop System (TLS) to the extent necessary to develop the Process Performance Requirements (PPR) for the Data Processing Subsystem (DPS) portion of the system. The TLS is a test configuration of an integral element of a Preliminary Ballistic Missile Defense System (PBMDs).

The primary objective of this specification is to constrain the DPS so that it fulfills the intended obligations to the functions and performance of the overall TLS of which it is a part. To accomplish this, this specification

- identifies the TLS mission and performance goals.
- identifies the various subsystems, their functional capabilities, and the interfaces between the Data Processing Subsystem and each of the others.
- allocates a portion of the system performance to the Data Processing Subsystem.
- describes the system operational design, i.e., how the subsystems are to be used in order to achieve the system performance goals by utilization of the subsystem capabilities.

This specification provides the technical basis for the development of the DPS, but is not a TLS system specification.

2.0 APPLICABLE DOCUMENTS

- 2.1 DPSPR Content Requirements, TRW Report 27332-6921-003, (CDRL A004),
Revision 1, December 12, 1975.
- 2.2 TLS System Requirements, (Part I, Section 1.1 of this report).
- 2.3 TLS Radar/DPS Interface Specification TRW Report No. 27332-6921-012
- 2.4 TLS C²/DPS Interface Specification TRW Report No. 27332-6921-013
- 2.5 TLS Radar Performance Specification
- 2.6 TLS Environment and Threat Model Definition

3.0 DATA PROCESSING SUBSYSTEM

The TLS is defined in [2.2] (Part I, Section 1.1 of this report). Section 1.1.2 provides the system requirements, while Figure F-1 represents the operation of the system. TLS consists of a radar and a data processor, and receives input data from radar echoes and from interface with the C² system. A model of the system environment is to be provided as [2.6]. Within the TLS, the radar-data processor interface specification is to be [2.4], while the C²/DPS interface will be defined in [2.3]. The radar models will be in [2.5].

The TLS DPS has been allocated requirements from those established on the TLS as a whole. The resulting requirements on the DPS are defined in Section 3.2 of this DPSPR. Traceability of the DPSPR requirements to the TLS requirements is shown in Figure F-6. All TLS requirements not allocated to the DPS are satisfied by the radar. Quantitative verification of the sufficiency of the allocation will be undertaken in the preparation of the PPR.

3.1 Traceability

Figure F-6 is the traceability matrix for this DPSPR, and illustrates the relationship between the TLS requirements [2.2], and the DPS requirements of Section 3.2. It also depicts the allocation of portions of system requirements to the radar; not all of that allocation is clearly visible in the interface specification, some of it being located in [2.5].

The function of the traceability matrix is twofold: to locate the subsystem requirements derived from each system requirement, and to facilitate recognition that each DPS requirement originates from an appropriate source. The first function is insurance that the DPSPR (and its counterparts for other subsystems) are sufficient to embody the system requirements, while the second verifies that no gratuitous requirements have been introduced.

To determine the source(s) of a DPS requirement, the appropriate row is located in the left-hand column. Reading across the row, an entry of "C" indicates that virtually complete satisfaction of the system requirement for that column is provided. A "P" indicates that a portion of the system requirement is there satisfied. Scanning the entire chart, one finds that the DPS

PART I 1.1.2 PART II 3.2	.1							.2							.3			.4			.5				
	a	b	c	d	e	f	g	a	b	c	d	e	a	b	c	a	b	a	b	c	1	2	3	4	5
.1	C	C	C	C	C	C	C																		
.2	a							P									P	P							
	b								P								P	P							
	c									C															
	d									P							P	P							
	e										C						P	P							
.3	a												P	P											
	b														P										
	c															P									
	d												P	P											
	e														P										
	f												P	P	P										
	g												P	P	P										
.4	a															P	P								P
	b															P	P								
.5	1																				C				
	2																								
	3																						P		
	4																							P	
RADAR									P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P	P

Figure F-6 Requirements Traceability Matrix

requirement to satisfy the interface specification is not traceable to the system requirements; however, it is clearly required, and is clearly inappropriate for the system specification.

Determining the impact of a system requirement on the subsystems is constructive both in confirming that every requirement is covered and in tracking the consequences of a change to the system specification. Reading a column will show the partial (P) or complete (C) satisfaction of a system requirement in the corresponding DPSR section. In the present case, each system requirement has some DPS impact; that might not be true in general. For example, if a hard stop on elevation angle were implemented in the radar, then the third requirement under 3.2.2 would be deleted, and the implementation of the third requirement of (Part 1) 1.1.2.2 would be virtually completely contained in the radar subsystem.

3.2 Data Processing Subsystem Requirements

Many of the following specific requirements are stated in terms of a value and a probability of its satisfaction. In each such case, the interpretation shall be that at every point in the engagement, the probability of satisfying the inequality shall be at least the stated value. The required confidence in that assertion is established in test planning.

3.2.1 Initialization

- a) The DPS shall accept commands from the C² to initiate track on a designated object. A radar order in response to such a command shall be provided at the radar interface for transmission within 55 milliseconds of the command with probability .9. These commands are defined in Reference 2.4 [TLS C²/DPS Interface Specification].
- b) Capacities. The DPS shall be able to accept handoffs at a rate of 150 per second over any interval greater than 50 milliseconds, and a total of 1500 handoffs per engagement, of which up to 300 will be real images.

3.2.2 Termination

- a) The DPS shall command not more than 15 pulses to a redundant image with probability .7, nor more than 50 with probability .99.

- b) The DPS shall command not more than seven pulses to a ghost with probability .7.
- c) The DPS shall command no track pulse with elevation angle less than 3°.
- d) The DPS shall command no track pulse to an image of an object which has achieved an elevation of less than 5°, or which will attain such an elevation by the time of transmission, with probability .9.
- e) For an image on which a drop-track command is received, the DPS shall command no transmission with execution time more than 100 milliseconds after appearance of the order at the C² interface with probability .7, nor more than 300 milliseconds with probability .99.

3.2.3 Tracking

- a) The DPS shall maintain an estimate of state for each image in track. Defining the true state of an image by Attachment A, estimated state shall deviate from true state by not more than the tolerance of Figure F-7 (TBD) with the probabilities of that Figure, where the assessment is relative to the time of the last processed return.
- b) The probability that all images of an object shall be dropped as redundant shall not exceed .01.
- c) The probability that any image of an object shall be dropped as a ghost shall not exceed .2.
- d) The DPS shall command track pulses at a rate sufficient to keep the propagated error defined by Attachment B less than the tolerances of Figure F-8 (TBD) with probabilities defined in that Figure.
- e) The DPS shall provide orders to the radar in accordance with the interface specification. The relevant fields, timing, etc., are defined in [2.3]. In particular, the DPS shall select waveforms, frequencies, beamwidths and related parameters in accordance with radar constraints in order to satisfy TLS performance requirements.
- f) The DPS shall accept radar returns in accordance with the Interface Specification [2.3].
- g) The DPS shall maintain track on each image until one of the following conditions is satisfied:
 - 1) Drop Track command received,
 - 2) Image determined to be redundant,
 - 3) Image determined to be a ghost, or
 - 4) Estimated elevation of object or of image expected to violate elevation-angle constraint before next assessment.

3.2.4 Resource Control

- a) The DPS shall maintain an estimate of radar energy scheduled by track functions which shall be within three percent of the energy nominally expended, and an upper-bound estimate such that the actual ERP and total radiated energy do not exceed the bound with probability (TBD).
- b) The DPS shall allocate radar commands so that not more than (TBD) joules are commanded per image, nor more than (TBD) kilowatts or (TBD) pulses/second for all images in track.

3.2.5 Data Recording

The DPS shall output to permanent file the following data:

- a) Time of handoff and designation and estimate provided.
- b) Time of termination of track on each designation with reason for termination. The reason shall be one of the following.
 - 1) Drop Track Command
 - 2) Minimum Elevation
 - 3) Image Declared Redundant
 - 4) Image Declared Ghost.
- c) Subsequent to each state update, the resulting estimate of state on that image. Contents of that estimate are TBD.
- d) At intervals not greater than 300 milliseconds the usage of radar resources. That estimate shall include the nominal and upper bounds defined in 3.2.4, and TBD additional data.

4.0 GLOSSARY

State Vector: A set of data pertaining to a specified image defining its location in space and permitting extrapolation of its location to other times. The contents of the state vector may vary with different applications; in particular, the coordinate system employed is dependent on the use to which it will be put. A conventional state vector in RFCC is: $R, U, V, \dot{R}, \dot{U}, \dot{V}, \beta^{-1}$, and the time to which all are referenced.

Object: A physical entity external to the DPS with radar reflection properties corresponding to a reentry vehicle of the defined threat.

Image: A target defined to the TLS DPS for tracking and categorization as the image to be tracked, a redundant image, or a ghost.

Ghost: An image with which no object is correlated.

Redundant Image: An image which correlates with both an object and another image. Of the set of redundant images of an object, one is intended to be retained in track while the others are categorized as redundant and dropped.

Handoff: The receipt by the TLS DPS of a valid order to initiate track on an image.

Track Termination: The receipt by the TLS DPS of a valid order to Drop Track, or the determination by the DPS that tracking should be terminated.

ATTACHMENT A

STATE VECTOR PREDICTION

The filter maintains the state vector in the RFCC system at all times. Specific details of the equations of motion and the method of trajectory integration employed for state vector prediction in the RFCC system are presented in this section. Figure A-1 defines the RFCC system.

1.1 Equations of Motion

In the general formulation of the equations of motion, it is assumed that all vector variables appearing in the differential equations are appropriately referenced to the RFCC system associated with the face of a given radar under consideration. Explicit expressions for the transformed variables are given whenever they first appear.

Let the RFCC position vector to a target, denoted $\bar{r}_f$, have Cartesian components X_f , Y_f and Z_f . The differential equations describing the motion of a target in the rotating RFCC system may, in general, be written as

$$\ddot{\bar{r}}_f = -\frac{\mu \bar{R}_f}{|\bar{R}_f|^3} - \frac{\rho g \lambda}{2} |\dot{\bar{r}}_f|^2 \frac{\dot{\bar{r}}_f}{|\dot{\bar{r}}_f|} - 2 \bar{\omega}_f \times \dot{\bar{r}}_f - \dot{\bar{\omega}}_f \times (\bar{\omega}_f \times \bar{R}_f) \quad (A.1)$$

and assuming constant λ model

$$\dot{\lambda} = 0$$

where

$\mu = GM$

$G =$ universal gravitational constant

$M =$ mass of the earth

$\bar{R}_f$ is the vector from the geocenter to the target

ρ is the atmospheric density which is a function of the altitude h

$g =$ earth's sea level gravity

λ is the inverse of the ballistic coefficient

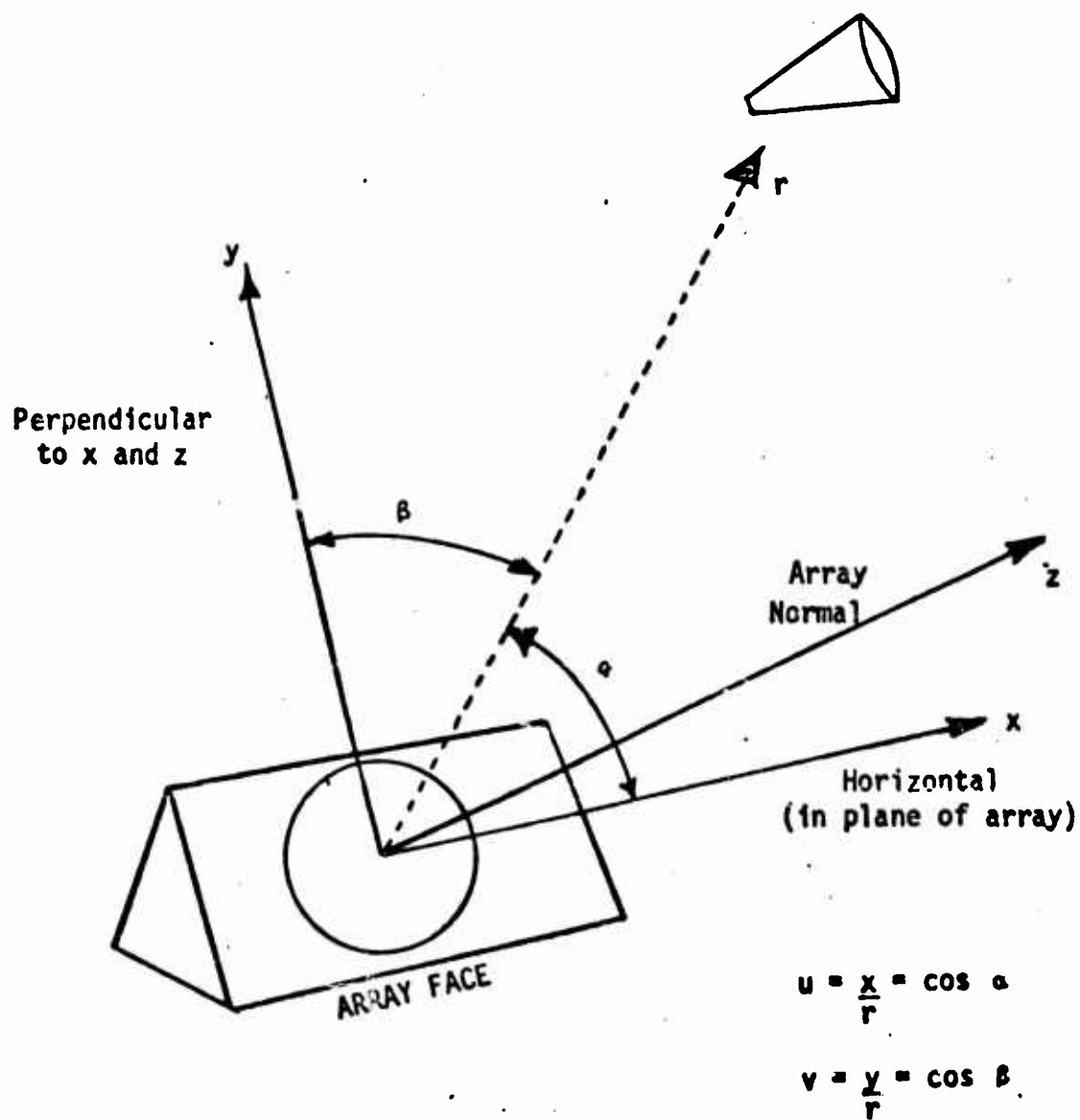


Figure A-1 Radar Face Centered Coordinates

$\dot{\bar{r}}_f$ is the velocity vector of the target
 $\times$ is the vector cross-product
 $\bar{\omega}_f$ earth's angular rotation rate, a vector quantity resolved along an earth-centered Cartesian coordinate system aligned with the RFCC system.

Define $\bar{R}_{sf}$ as the vector from the geocenter to the radar site expressed in the RFCC system. Then

$$\bar{R}_f = \bar{R}_{sf} + \bar{r}_f \quad (A.2)$$

where the vector $\bar{R}_{sf}$ is given by

$$\bar{R}_{sf} = \begin{bmatrix} 0 \\ R_e \cos E \\ R_e \sin E \end{bmatrix} \quad (A.3)$$

and E is the elevation angle of the radar boresight with respect to the local horizon plane.

The earth rotates about the polar axis with a constant angular velocity ω_e . The components of the earth's angular velocity vector in the RFCC system is given by

$$\bar{\omega}_f = \begin{bmatrix} \omega_{X_f} \\ \omega_{Y_f} \\ \omega_{Z_f} \end{bmatrix} = \begin{bmatrix} \omega_e \sin \Lambda \cos \phi \\ \omega_e (\cos E \sin \phi - \sin E \cos \Lambda \cos \phi) \\ \omega_e (\cos \Lambda \cos E \cos \phi + \sin E \sin \phi) \end{bmatrix} \quad (A.4)$$

where Λ is the azimuth of the radar boresight with respect to the radar centered horizon coordinate system and ϕ is the geocentric latitude of the radar site.

1.2 Trajectory Integration

The trajectory integration involved in the prediction of the state vector will be performed by a second-order Taylor's series expansion in $\Delta t_n = t_{n+1} - t_n$ for the position vector $\bar{r}_f$ which gives

$$\bar{r}_f(t_{n+1}) = \bar{r}_f(t_n) + \dot{\bar{r}}_f(t_n)\Delta t_n + 1/2 \ddot{\bar{r}}_f(t_n)\Delta t_n^2 \quad (A.5)$$

and consequently, the velocity vector is given by

$$\dot{\bar{r}}_f(t_{n+1}) = \dot{\bar{r}}_f(t_n) + \ddot{\bar{r}}_f(t_n)\Delta t_n \quad (A.6)$$

and

$$\lambda(t_{n+1}) = \lambda(t_n) \quad (A.7)$$

where $\ddot{\bar{r}}_f(t_n)$ is readily evaluated from Eq. (1.1) by using the current estimate of the state vector at t_n .

ATTACHMENT B

PROPAGATION OF ERRORS

Ignoring the process noise, the propagation of the state error covariance matrix from cycle to cycle is computed by

$$P(n+1/n) = \phi(n+1,n) P(n) \phi^T(n+1,n) \quad (B.1)$$

in which $\phi(n+1,n)$ is the state transition matrix expressed in terms of the appropriate filter coordinate system (RVCC).

The derivation of the ϕ matrix starts with the first order Taylor's expansion of a set of nonlinear differential equations describing the target motion about some nominal solution of the state. For this specific development, the target motion Eq. (A.1) is approximated by

$$\ddot{\vec{r}}_f = -\frac{g\lambda}{2} |\dot{\vec{r}}_f| \dot{\vec{r}}_f \quad (B.2)$$

that is, all other accelerating forces except that due to the atmospheric drag are ignored.

The decoupled in-plane and out-of-plane transition matrices ϕ_p and ϕ_n in the RVCC system are given by

$$\phi_p(n+1,n) = \begin{bmatrix} 1 & 0 & \Delta_n & 0 & 0 \\ 0 & 1 & 0 & \Delta_n & 0 \\ 0 & 0 & 1 & 0 & -D\dot{Z}_1\Delta_n \\ 0 & 0 & 0 & 1 & -D\dot{Z}_2\Delta_n \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (B.3)$$

$$\phi_n(n+1,n) = \begin{bmatrix} 1 & \Delta_n \\ 0 & 1 \end{bmatrix} \quad (B.4)$$

APPENDIX F-II: RADAR/DPS INTERFACE SPECIFICATION

1.0 SCOPE

This specification defines the physical and functional interface between the Radar and the Data Processing Subsystem (DPS) of the Track Loop System (TLS). The TLS itself is a testbed derived from a Preliminary Ballistic Missile Defense System (PBMDS), which is in turn a representative but unreal environment for these studies. TLS is intended to have development and test capability, although realization of that capacity in actual testing is not now contemplated.

This specification corresponds to one which would be available prior to preparation of a PPR. To that end, its contents have been extracted from TDP documentation. Reference is made to that material as the source from which data here labelled "TBS" and "TBD" will be derived. In this publication "TBS" is used to identify material expected to be required during early stages of PPR preparation, while "TBD" denotes material which may be supplied later in the specification process. Material presented in italics, such as this paragraph, is illustrative or informative in nature, and would not normally appear in such a specification.

*Some of the paragraphs in this Interface Specification place constraints or requirements on the DPS; these have been identified by a * in front of that paragraph.*

2.0 APPLICABLE DOCUMENTS

- 2.1 TLS SYSTEM REQUIREMENTS 27332-6921-011, PART I, SECTION 1.1
- 2.2 TLS DATA PROCESSING SUBSYSTEM PERFORMANCE REQUIREMENTS (DPSPR) 27332-6921-011, PART II
- 2.3 TLS RADAR PERFORMANCE SPECIFICATION
- 2.4 TLS ENVIRONMENT AND THREAT MODEL DEFINITION

3.0 INTERFACE REQUIREMENTS

3.1 PHYSICAL INTERFACE

TBD

The physical interface is highly dependent on hardware selection for both the DPS and the Radar; in general, it will be irrelevant to the specification process through the Preliminary Design Review. Some portions of the physical interface may be specified during PPR, notably those relating to timing of signals and clock protocol.

3.2 FUNCTIONAL INTERFACE

The functional interface between the DP and the Radar shall consist of:

- Commands issued by the DP to the Radar
- Returns issued by the Radar to the DP
- Engagement Clock issued by the Radar to the DP

3.2.1 Radar Command Generation

- a. The Radar will execute only the commands issued by the DP.
- * b. Each command shall contain transmit, receive and synchronization data as described herein.
- c. The radar shall transmit one pulse for each command which satisfies the radar and interface constraints. *Within PBMDs, but external to TLS, there are exceptions for pulse pairs and for verify pulses.*
- * d. The DPS shall command a single receive window for each pulse. Within each receive window, the DPS shall command at least one receive gate.

3.2.2 Command Ordering

- a. Radar shall execute commands in the order received.
- * b. The DPS shall provide End of Transmission at least 1 msec before the scheduled execution time of the first command in the message.

3.2.3 Command Unpacking and Decoding

- * The DPS shall issue commands in accordance with the format of TBD.

3.2.4 Timing Control

- * a. The commanded times for Radar actions or for returns shall be in absolute time measured from a clock with nominal 1.68 second rollover. The least significant time bit shall be 6.25 nsec.
- b. Radar shall determine all intermediate times needed to comply with command data for transmission and reception.
- * c. The DPS shall not command receive windows which overlap. The receive window duration in each case shall be at least the uncompressed pulse length plus the desired range coverage.
- * d. The DPS shall not command conflicting transmissions. Consecutive commands shall be separated in execution time by at least the uncompressed pulse length of the earlier plus TBS.

3.2.5 Command Contents

- * a. The DPS shall provide a waveform identifier corresponding to one of the waveforms of Table F.2 (TBD) in each command.
- * b. The DPS shall provide in each command both transmit and receive codes corresponding to direction cosine phase tapers.
- * c. The DPS shall provide a receiver gain setting in each receive window.
- * d. The DPS shall provide a signal processing mode code in each receive gate.
- * e. The DPS shall provide a fixed signal acceptance threshold and threshold type selection for each receive gate.
- * f. The DPS shall provide the range gate mark generation technique for each receive gate.
- * g. The DPS shall provide the transmit power level for each command.

3.2.6 Return Contents

- a. Radar shall return identifier data provided in the command.
- b. Radar shall return actual range mark times and the signal amplitude at each range mark.
- c. Radar shall return video signal amplitudes at commanded points. Amplitude shall be corrected by the Radar for stored instrumental errors.

d. For appropriate commanded signal-processing modes and waveforms, the Radar shall return direction cosines of the echo. Directional data shall be corrected by the Radar for stored instrumental errors.

e. For appropriate commands, the Radar shall return TBS wake array data.

f. Radar shall return noise level relevant to each amplitude in a return.

3.2.7 Error Handling

a. Radar shall return an error message for each discernible command not implemented. That message shall include a code corresponding to the reason for the failure of transmission. Among the reasons may be preemption, receive or transmit window overlap, insufficient time for transmission, and faulty command (internal inconsistency).

* b. The DPS shall initiate a record on permanent file of each error return.

* c. The DPS shall determine whether the fault is persistent or unique; if persistent, whether it is safety-related. (Definitions TBS). A persistent, safety-related fault shall cause test termination to be commanded by the DPS within TBS milliseconds; in particular, no command shall be issued for transmission by the Radar more than TBD milliseconds after a persistent, safety-related fault is detectable from returns.

3.2.8 Mode Change

* The DPS shall control all Radar mode changes through issuance of appropriate commands. The changes shall include startup and shutdown. Timeline constraints on mode changes and on preparation time for transmission are TBS.

3.2.9 Timing

a. Radar shall provide a master timing reference to DPS via TBS interface.

b. The clock shall provide 28 bits of data with a least significant bit of 6.25 nsec.

c. The resetting of the clock shall be entirely under Radar control. In consequence, its absolute value shall be regarded by the DPS as arbitrary. The Radar shall reset the clock within TBS milliseconds of startup, and shall not again reset it during the engagement.

3.2.10 Miscellaneous Requirements

- a. Negative numbers crossing the interface shall be represented in two's complement form.
- b. Radar coordinates shall be defined in accordance with Figure F-9.

3.3 DATA REQUIREMENTS AND FORMATS

TBS

Data requirements and formats have been defined for TDP, and that definition might be carried over to the present document. However, it is not characteristic of systems such as TLS that details of bit positions, message formats, etc., would be known at this stage of development. However, the dynamic range, units, and least significant bit information is necessary in order to write performance requirements on radar command generation. The formats identification can be postponed until process design time.

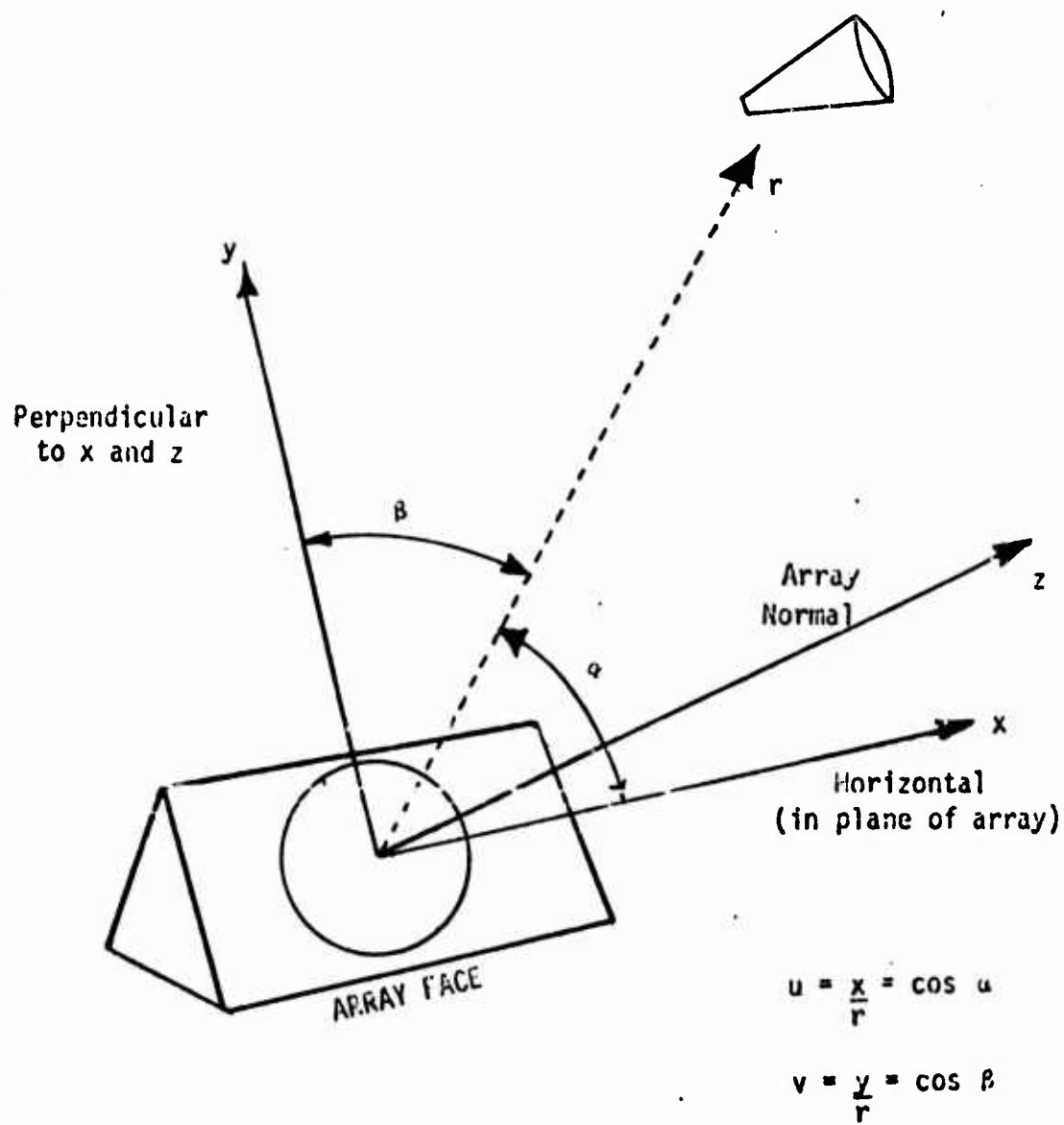


Figure F-9 Radar Coordinate System

APPENDIX F-III: C²/DPS INTERFACE SPECIFICATION

1.0 SCOPE

This specification defines the physical and functional interface between the Command and Communications (C²) System and the Data Processing Subsystem (DPS) of the Track Loop System (TLS). The TLS itself is a testbed derived from a Preliminary Ballistic Missile Defense System (PBMUS), which is in turn a representative but unreal environment for these studies. TLS is intended to have development and test capability, although realization of that capacity in actual testing is not now contemplated.

This specification corresponds to one which would be available prior to preparation of a PPR. To that end, its contents have been extracted from TDP documentation. Reference is made to that material as the source from which data here labeled "TBS" and "TBD" will be derived. In this publication, "TBS" is used to identify material expected to be required during early stages of PPR preparation, while "TBD" denotes material which may be supplied later in the specification process. Material presented in italics, such as this paragraph, is illustrative or informative in nature, and would not normally appear in such a specification.

*Some of the paragraphs in this Interface Specification place constraints or requirements on the DPS; these have been identified by a * in front of that paragraph.*

2.0 APPLICABLE DOCUMENTS

- 2.1 TLS SYSTEM REQUIREMENTS 27332-6921-011, PART I, SECTION 1.1
- 2.2 TLS DATA PROCESSING SUBSYSTEM PERFORMANCE REQUIREMENTS (DPSPR)
27332-6921-011, PART II
- 2.3 TLS RADAR/DPS INTERFACE SPECIFICATION
- 2.4 TLS ENVIRONMENT AND THREAT MODEL DEFINITION

3.0 INTERFACE REQUIREMENTS

3.1 PHYSICAL INTERFACE

TBD

The physical interface between the C² and the DPS is entirely dependent on hardware selection. It will define polarity conventions, signal levels, and related parameters of interest only following PDR, and accountable only from the time of hardware integration. Although this section is required in such an interface specification, it will normally remain undetermined throughout the early stages of software design.

3.2 FUNCTIONAL INTERFACE

The functional interface between the C² and the DPS shall consist of messages of four types transmitted from the C² to the DPS.

- Initiate Engagement Mode
- Terminate Engagement Mode
- Handover Image
- Drop Image Track

3.2.1 Initiate Engagement Mode

- * a. The DPS shall accept an Initiate Engagement Mode message from any of the following prior modes of the DPS: TBD.
- * b. The DPS shall be prepared to accept a Handover Image message within TBS seconds of appearance of the Initiate Engagement Mode message at the interface.

3.2.2 Terminate Engagement Mode

- * a. The DPS shall accept a Terminate Engagement Mode message at any time when it is in Engagement Mode. The DPS shall transition to TBD mode in response to a Terminate Engagement Mode message.
- * b. The DPS shall command no Radar transmission with an execution time more than TBS milliseconds following appearance of a Terminate Engagement Mode message at the interface.

3.2.3 Handover Image

- a. The contents of the Handover Image message shall be

Image designation

Image estimated state

TBS

- b. The C^2 shall transmit no Handover Image message except when the DPS has been placed in the Engagement Mode by transmission of an Initiate Engagement Mode message at least TBS milliseconds earlier, and since the last Terminate Engagement Mode message.

3.2.4 Drop Image Track

- a. The contents of the Drop Image Track message shall be the image designator.

- b. The C^2 shall transmit no Drop Image Track message for an image designator unless that designator was previously included in a Handover Image message.

3.2.5 Message Acknowledgement

TBS

3.2.6 Error Handling

TBS

3.3 DATA REQUIREMENTS AND FORMATS

TBD

Data requirements and formats have been defined for TDP, and that definition might be carried over to the present document. It is not characteristic of systems such as TLS that details of bit positions, message formats, etc., would be known at this stage of development. However, the dynamic range, units, and least significant bit information is necessary in order to write performance requirements on the radar command generation. The formats identification can be postponed until process design time.

REFERENCES

1. "Requirements Engineering and Validation System Users Manual", TRW Report No. 27332-6921-022, 15 July 1976 (Draft).
2. "Software Requirements Engineering Methodology," Final Report, TRW Report No. 27332-6921-019, 12 December 1975.