UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

| | |
|---|---|
| 1. REPORT NUMBER: AFAL-TR-78-166 | 2. GOVT ACCESSION NO. |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE *(and Subtitle)*: MATE SUPPORT SOFTWARE ANALYSIS | 5. TYPE OF REPORT & PERIOD COVERED: Feb-Aug 78 Final Technical Report |
| 6. PERFORMING ORG. REPORT NUMBER | |
| 7. AUTHOR(s): W. Lynn Trainor | 8. CONTRACT OR GRANT NUMBER(s): F33615-77-D-1042 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Systran Corporation, 4124 Linden Ave., Dayton, OH 45432 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Project 2003 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: USAF AFSC-AFAL/AAA, System Avionics Division, Wright-Patterson AFB, OH 45433 | 12. REPORT DATE: August 1978 |
| | 13. NUMBER OF PAGES: 84 |
| 14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: N.A. | 15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N.A. |

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

N.A.

18. SUPPLEMENTARY NOTES

N.A.

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

| | | |
|---|---|---|
| MATE | DAIS | OPERATING SYSTEM |
| SOFTWARE | EXECUTIVE | CONFIGURATION CONTROL |
| ATE | INTERPRETER | |
| UUT | ASSEMBLER | |
| HOL | COMPILER | |

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report describes a research study, the objective of which was to analyze and document those elements of support software used on the Digital Avionics Information System (DAIS) Program which could satisfy the support software requirements of the Modular Automatic Test Equipment (MATE) Project. Since many of the objectives of the MATE Program are closely aligned with those of DAIS, it was felt that some of the software items and standards developed for DAIS could be applicable to the MATE Program. (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

BLOCK 20 (Continued)

The first sections of this report present an overview of the three areas of ATE software: ATE software of current-day systems, additional ATE software requirements beyond those satisfied with current-day ATE software, and MATE-unique software requirements. Using these as a basis for comparison the DAIS software was analyzed for applicability to the MATE system. The results of this analysis are presented in the final sections.

AFAL-TR-78-166




# MATE SUPPORT SOFTWARE ANALYSIS

SYSTRAN CORPORATION
DAYTON, OHIO

AUGUST 1978



TECHNICAL REPORT AFAL-TR-78-166
FINAL REPORT FOR PERIOD FEBRUARY-AUGUST 1978



AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

J. GREGORY JOLDA, Capt, USAF
Project Engineer

DIANE E. SUMMERS, Acting Chief
Avionic Systems Engineering Branch
System Avionics Division

FOR THE COMMANDER

RAYMOND E. SIFERD, Colonel, USAF
Chief, System Avionics Division
Air Force Avionics Laboratory

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFAL/AAA-2, W-PAFB, OH 45433 to help us maintain a current mailing list."

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

## FOREWORD

This Technical Report covers work conducted by SYSTRAN Corporation under Contract F33615-77-D-1042. This contract was managed by the Air Force Avionics Laboratory, AFAL/AAA, Wright-Patterson AFB, Ohio, and additional contract technical direction was obtained from the Aeronautical Systems Division, MATE Project Office, ASD/AEGS, Wright-Patterson AFB, Ohio. The two Air Force technical monitors were Capt. William V. Green (AFAL/AAA) and Capt. Charles F. Gembrowski, ASD/AEGS.

This report was researched and written primarily by Mr. W. Lynn Trainor, but other SYSTRAN staff members contributed technical expertise and assistance in various speciality areas.

A special note of appreciation is extended to Mr. Gaylen Pederson, Ogden ALC/MACT, Hill AFB, Utah, and his staff members. Mr. Pederson's office was helpful in the early phases of this project by contributing discussions relating to current Automatic Test Equipment (ATE) system utilization and problem areas.

TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

# SECTION I

# INTRODUCTION

## 1. PROJECT OBJECTIVES

This study effort was sponsored jointly by the System Avionics Division of the Air Force Avionics Laboratory and the Modular Automatic Test Equipment (MATE) Project Office of the Aeronautical Systems Division. The objective was to analyze and document those elements of support software used on the Digital Avionics Information System (DAIS) Program which could satisfy the support software requirements for MATE. Since many of the objectives of the MATE Program are closely aligned with those of DAIS, it was felt that some of software items and standards developed for DAIS could be applicable to the MATE Program.

## 2. MATE PROGRAM OVERVIEW

The MATE Program is sponsored and managed by the Aeronautical Systems Division (ASD/AEGS) at Wright-Patterson Air Force Base, Ohio. MATE is targeted to reduce the life cycle cost of Air Force weapon systems by technically addressing increased commonality of test equipments across aircraft subsystems and by addressing cost effective engineering and management tools. In this regard, five objectives have been established:

- Reduce the life cycle costs of weapon system support and automatic test equipment (ATE),
- Reduce the proliferation of ATE,
- Improve the operational utility and test efficiency,

- Improve ATE management,
- And improve ATE procurement practices.

The overall MATE objectives are to be achieved through a series of incremental projects, each addressing one or more of the objectives above. Initial efforts are to focus on the derivation and documentation of MATE "system" concepts, and this initial phase will be followed by a prototype development of a MATE system. In parallel to these "system level" efforts, the MATE Program Office is conducting research and study efforts in such areas as: programming aids for simulation and automatic test program generation, ATE test software verification and validation techniques, and documents and handbooks for enhancing the specification and management of MATE elements.

## 3. DAIS PROGRAM OVERVIEW

The purpose of the Digital Avionics Information System (DAIS) project is to demonstrate a coherent solution to the problem of proliferation and nonstandardization of aircraft avionics, to develop and test the DAIS concept in a "hot bench" configuration (known as the Integrated Test Bed - ITB), and to permit the Air Force to assume the initiative in specifying avionics configurations for future Air Force weapon systems acquisitions. This project is managed by the Air Force Avionics Laboratory.

Historically, Avionics mission information requirements have been established along semi-autonomous subsystem areas, such as

navigation, weapon delivery, stores management, flight controls, communications, etc. Within each of these functional areas the trend has been toward digital systems, each with its own unique processing, transfer and display of information. There has been an integration of requirements between functional areas only as necessary for interaction purposes. The DAIS concept proposes that the processing, multiplex, and display functions be common and service all the previously described areas or subfunctions on an integrated basis. When coupled with other existing programs and facilities, the DAIS hot bench will contain the flexibility to evaluate a spectrum of multiplex, processing and display approaches such that decisions can be made regarding interface standards, processing architectures, display concepts, etc. As technology becomes available and the hot bench is programmed to solve desired aircraft avionic problems, the built-in flexibility will accept adaptation. In this manner, an evolutionary growth will continually update the hot bench configuration whenever the capability or need exists.

The DAIS design approach reflects a total system concept which is functionally oriented rather than hardware oriented. For example, a "navigation subsystem" in DAIS does not refer to a set of black boxes, but it refers to a set of identifiable functions which are performed at various places throughout the DAIS system. Note that the system is not dedicated exclusively

to doing the navigation function alone; it is also used to perform the functions of many other "subsystems". For this system approach to avionics, DAIS will certify ideas and classes of equipment that can satisfy weapon system needs.

Specific objectives of the DAIS program include:

- Develop an AFSC in-house capability to define, demonstrate, test, and evaluate evolutionary changes and requirements in digital avionics.
- Define and design a hot bench configuration for a limited hardware demonstration with growth potential to accommodate a large class of weapon systems.
- Identify and recommend standards, criteria, and specifications which must be instituted to reduce the proliferation and complexity of avionics systems.
- Provide the means for quantitatively evaluating cost (both acquisition and life cycle) aspects and for exploiting potential increases in reliability, maintainability, and versatility of future weapon systems.
- Influence the design and development of sensors via input-output format specifications which will allow the new sensors to be compatible with the DAIS concept and ensure optimal information transfer and management.
- Identifying the many diverse programs, offices, etc., involved in digital avionics with the resulting integration of their requirements and actions into one coherent program.

## 4. REPORT ORGANIZATION

The remaining portions of this report are organized into six major sections. The order of these sections roughly corresponds to the sequence of tasks performed on this contract. Section II discusses the status of current-day ATE systems and

software and also serves to establish the definitions and terminology for the remaining report sections. Since there is such a wide variation in vendors' approaches to ATE system and ATE terminology lacks any significant degree of standardization, this section was needed in order to establish an ATE "baseline" for the latter sections. This section thus briefly defines all the items of ATE software used for current-day systems.

Section III is an ad hoc section which addresses one area of ATE software in addition to those items in Section II. Specifically, the need for UUT Test Software simulation and debugging aids is addressed.

Section IV presents an overview of a "postulated" MATE System hardware architecture which was used as a basis of the subsequent software comparison and analysis tasks, as reported in the latter sections of this report. Also discussed are additional MATE software requirements which were derived from the unique system design aspects of MATE.

Section V presents a short overview of the relevant DAIS hardware and software items. The DAIS airborne avionics and support facility are discussed to the level of detail that those DAIS software items of potential use in MATE can be understood. No attempt was made to include detailed descriptions of all DAIS software items since this detailed information is readily available from the DAIS Project Office's library of specifications.

Using the data obtained and documented in Sections II through V, an analysis was made of the items of DAIS software that are potentially applicable to the MATE system. However, due to the preliminary nature of the MATE system design, no firm MATE "system architecture" was available for use in this analysis. Instead, only the postulated architecture as discussed in Section IV was available. Section VI does, however, discuss several recommendations where DAIS software items would potentially fulfill MATE software requirements. Again, these recommendations are very sensitive to a "yet-to-be-defined" MATE system architecture, and may be negated if this MATE architecture differs significantly from that discussed in Section IV.

Finally, Section VII contains a summary of the report, recommendations and analyses.

# SECTION II

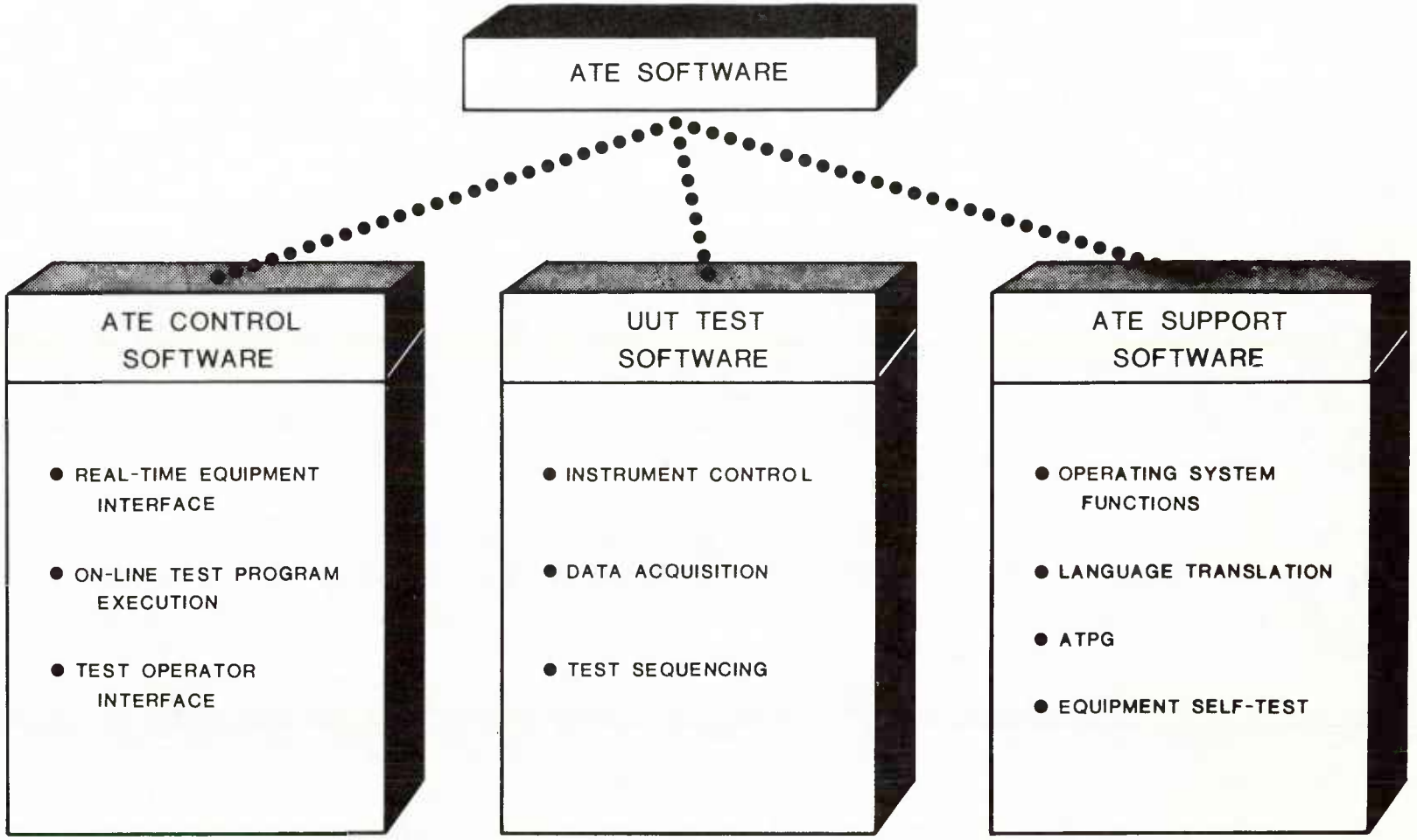
# ATE SOFTWARE TODAY

## 1. SECTION OVERVIEW

In this section an overview is given of each major element of software needed for current day ATE systems. These data represent a compilation of information from many sources, and as such do not represent any one particular vendor's approach to ATE design. By briefly discussing at the beginning of the report each of these software items and then establishing a "baseline" of definitions and terminology, the reader can better understand the latter sections of this report. These latter sections will rely upon this terminology baseline in the comparison analysis with the DAIS software items.

In addition to the variations in terminology used in industry, there are also notable variations in the basic ATE "system concepts" used by some of these vendors. This leads to differences in the procedures for using ATE and in the software items needed to support the ATE equipment.

In the paragraphs to follow, a "standard" set of ATE terms and procedures will be defined for use in this report. Figure II-1 shows a hierarchial view of this terminology. The first major category of ATE software is Control Software. This encompasses three functional areas: real-time equipment interface control, on-line control of test software execution, and



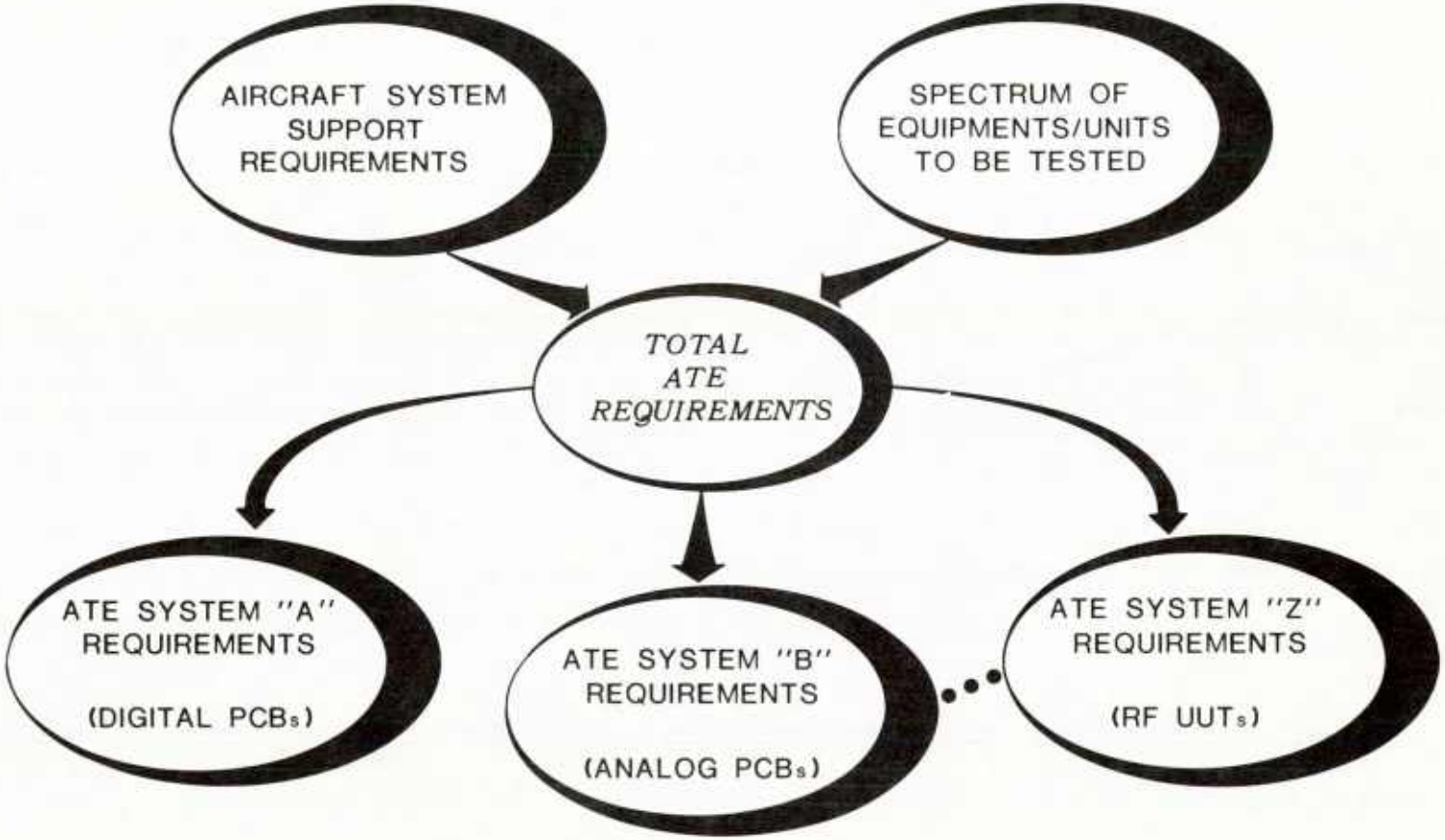

## ATE SOFTWARE PARTITIONING

Figure Number II-1

control of the test operator interface. The second major category is the UUT Test Software which concerns the actual "test applications". This software specifies the detailed test sequencing and flow. The major functions performed are: control of the ATE hardware instruments, acquisition of test data, and sequencing of tests. The third category is the ATE Support Software which contains primarily off-line or non-real-time support items. The four major functions included are: operating system services, language translation facilities, ATPG tools, and self-test of the ATE station equipments.

## 2. THE ATE SYSTEM

The ATE design usually begins early in the life cycle of a weapon system. Based on the testing and operational requirements of the particular aircraft system to be supported by the ATE, and based on the spectrum of stimuli and measurement parameters that are required of all of the various Units Under Test (UUTs) that are to be tested, the total set of ATE requirements are defined as shown in Fig. II-2. This process usually takes place soon after aircraft production is authorized. These studies and analyses are usually then taken a step further to, in turn, define the particular ATE "Systems" that will be needed. These ATE Systems are usually designed or selected to optimize support of generic "families" of UUTs; for example, a digital printed circuit board (PCB) ATE System, an analog PCB ATE System, etc.



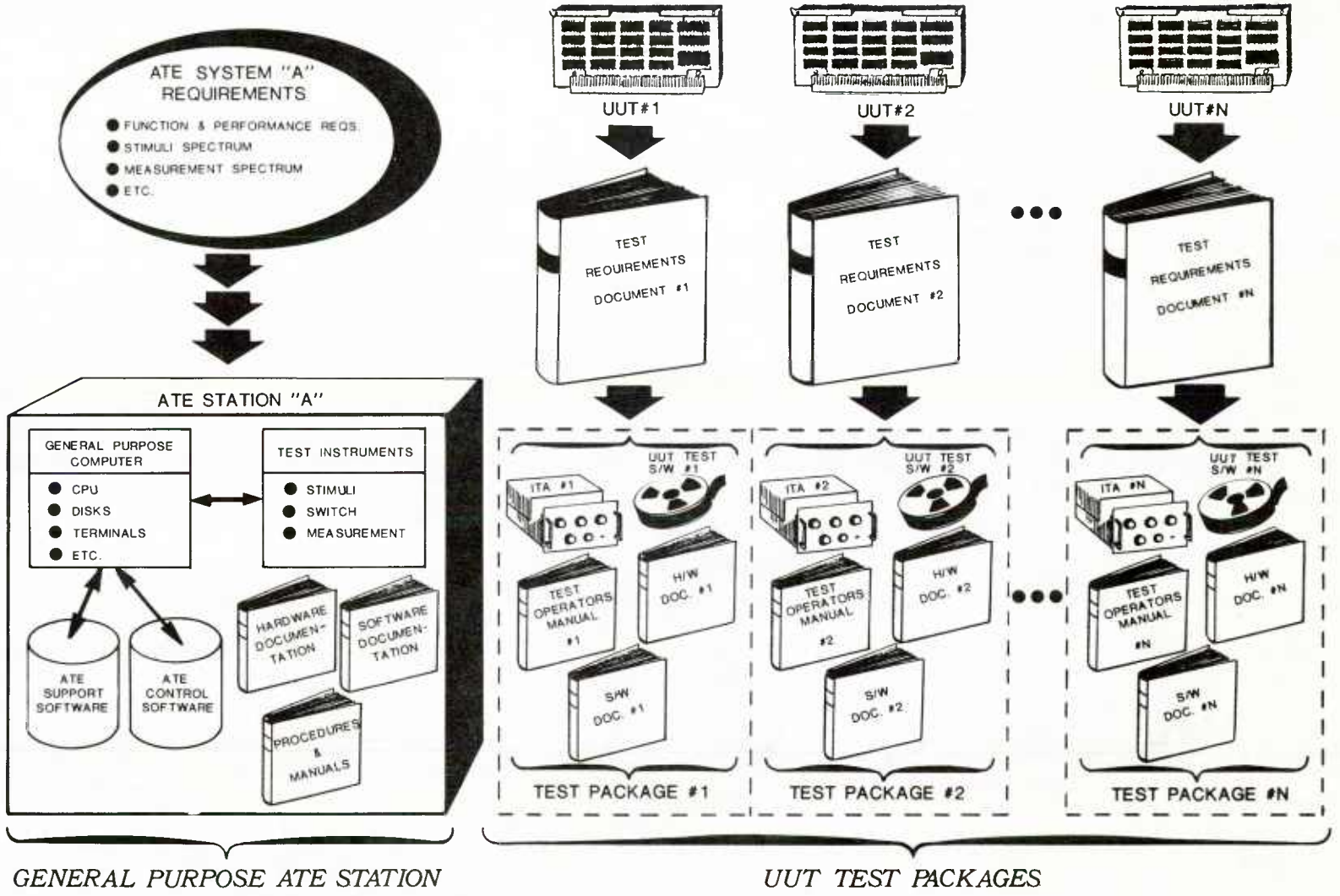

ATE REQUIREMENTS ALLOCATION PROCESS

Figure Number II-2

Most currently available ATE Systems can be represented by two distinct conceptual "parts": first, a generalized ATE Test Station for use in the testing of many (however, similar) UUTs; and second, the individual UUT Test Packages for each UUT supported by that station. This relationship pictorially is shown in Fig. II-3 and is discussed in more detail below.

a. ATE Test Station: The Test Station of today is usually composed of a general purpose minicomputer consisting of a central processor, processor memory, disk and/or tape memory, and terminals for operator interaction. The other major hardware items consist of the various electronic instruments used to generate stimuli for test inputs and to perform measurements of test outputs. Again, the stimuli and measurement devices are normally of a general purpose nature and can be "programmed" or "commanded" to perform in a multitude of modes or signal environments.

Two groups of general purpose software items are also associated with the Test Station. The first, termed ATE Control Software, is executive and on-line software which schedules and controls UUT testing by using the various hardware and software resources available within the Test Station, and usually provides the user with a high-level interface to the various computer resources. The second group of software, termed ATE Support Software, is composed of a general purpose operating system and a collection of various other support programs, such as compilers,




# ATE TEST STATION AND PACKAGE CONCEPT

Figure Number II-3

assemblers, linkers, text editors, etc. These items can be thought of as "tools" used for the development and testing of all software items for the ATE.

The final element of the Test Station is the documentation needed for both the operational and maintenance activities. Included is documentation on the various units of hardware and software which comprise the Test Station and the operational procedures and manuals needed for "general" operation of the Test Station. Also included are procedures and methods for maintaining and modifying the Test Station, when required.

Note that all of the "generalized" hardware and software items and documents are associated with the Test Station, and that none of the "UUT particular" items are associated with it. Since a Test Station is usually designed to support a generic set of UUTs, the Test Station designers must take care not to make a point design of any parts of the Test Station.

b. UUT Test Packages: A UUT Test Package consists of all of the "UUT unique" items needed to adapt the ATE Test Station for testing of a particular UUT. A set of such UUT Test Packages is thus developed, one for each UUT to be tested. Typically, such a package consists of five items:

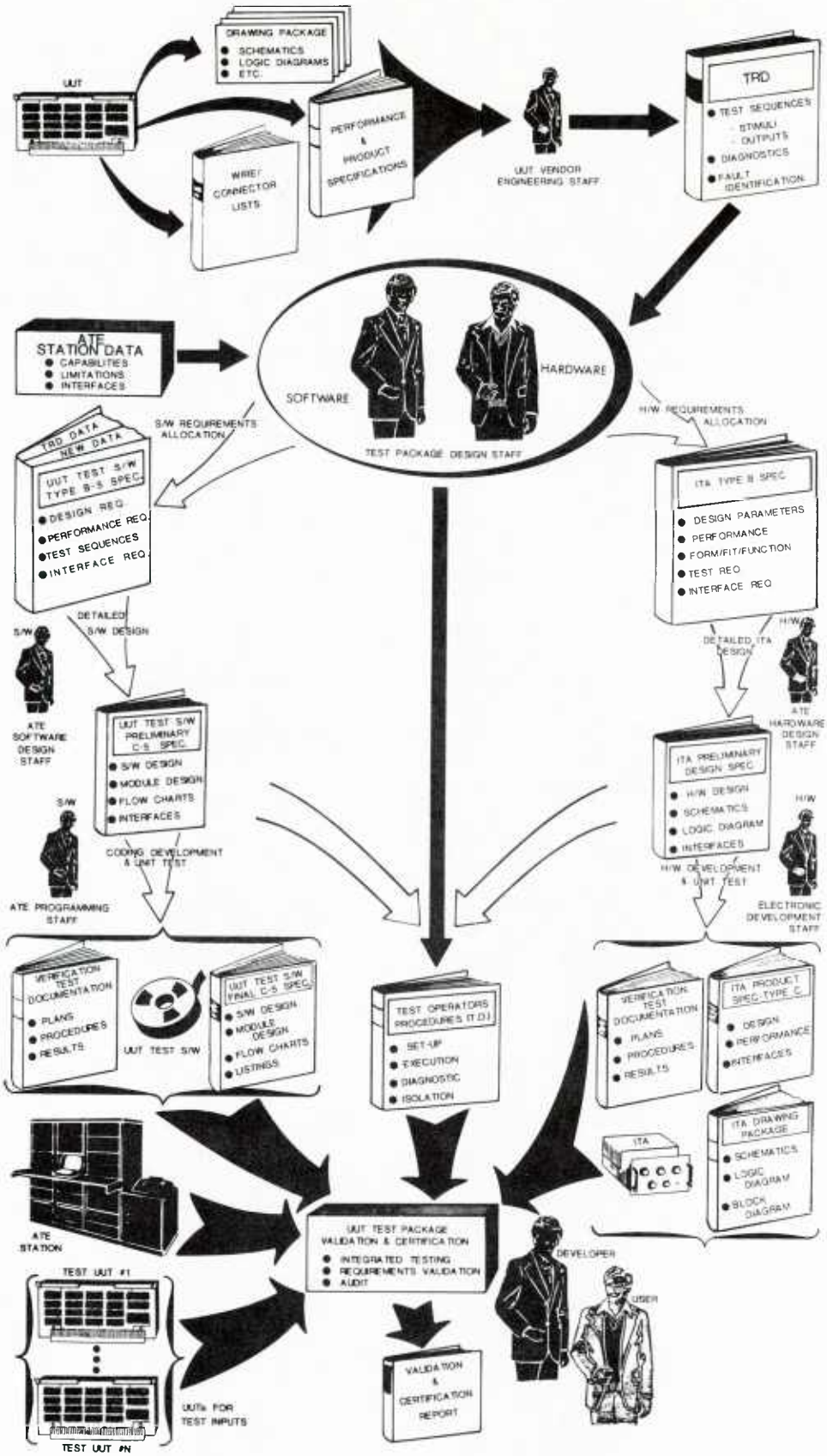
- An Interface Test Adaptor (ITA) hardware unit which connects the particular UUT connector/pins to the generalized Test Station interfaces.
- UUT Test Software (which executes under the control of the ATE Control Software) which contains the UUT Test sequences, diagnostics, etc.

- Test operator procedures or manuals for conducting the UUT tests.
- Hardware documentation (e.g., specifications, drawings, interface descriptions, etc.) on the ITA for the UUT.
- Software documentation (e.g. specifications, flowcharts, etc.) on the Test Software for the UUT.

As shown in Fig. II-3, each Test Package is developed for a particular UUT by utilizing the Test Requirements Document (TRD) as a specification medium. The TRD specifies all of the stimuli and measurements needed to successfully test the UUT, and it also specifies the fault diagnostic and location procedures to be followed. Any number of generic UUTs can be tested on a single station, ranging from one to several hundred.

c. UUT Test Package Development Process: It is important to note some of the detailed steps and processes involved in developing a UUT Test Package, for this is typically one of the most costly areas of ATE systems. For example, a Test Package can cost from $100K to $300K for PCBs and from $200K to $3,000K for Line Replacable Units.(7)

A generalized diagram of the Test Package development process is shown in Fig. II-4. This begins (shown at the upper left corner of this figure) with the UUT itself; in this case a PCB is shown. The "as built" documentation for the UUT (which consists of drawings, wire lists, specifications, etc.) is used to develop the TRD. This function is usually accomplished by the UUT vendor's




UUT TEST PACKAGE DEVELOPMENT PROCESS

Figure No. II-4

engineering staff, typically by the UUT design engineer who is thoroughly familiar with the UUT theory of operation, limitations, etc. The TRD contains a detailed listing of test conditions, sequences of stimuli which should be applied to UUT inputs, and specifies the output parameters to be measured. Particular attention is given to the procedures for use in isolating faults to particular UUT components. The TRD sequences usually take the form of test flow diagrams which detail the exact sequencing of tests and each decision point required.

The TRD can be conceptually viewed as the "top most" requirements document for UUT testing, for it is, in turn, used as a basis from which the test and design requirements are allocated to all subsequent UUT test and design documentation. In particular, the next step is to allocate the TRD requirements to either the UUT Test Software (via a type B-5 Design Requirements Specification) or the ITA hardware (via a type B-1, Design Requirements Specification). This function is usually conducted by the ATE vendor using both test hardware and software design specialists. Since there may be several methods of implementing a single TRD test sequence using various combinations of software or ITA hardware, it is important that both hardware and software specialists participate in this process.

Also the designers at this step must take care to consider the capabilities and limitations of the particular Test Station to be used since the Test Station design is typically "frozen" at this point in time.

As a result of this step, the software design will usually be documented and baselined in terms of a Design Requirements Specification. This document will repeat much of the TRD data, particularly the test sequence diagrams since these diagrams will largely be implemented "as is" in the UUT Test Software. However, other data will be included in this specification, such as software performance requirements, software/hardware interface definitions, software validation requirements, etc.

The second document, an analog of the above software specification, is the ITA Hardware Design Requirements Specification. Likewise most of these hardware requirements are allocated from the TRD, with added data regarding specific performance parameters, form/fit/function data, ITA validation, etc. This ITA specification will form the baseline for the subsequent detailed design of the ITA hardware; and since this specification was developed in concert with the software specification, it will insure interface and operational compatibility.

Although not shown in Fig. II-4, it should be noted that if the ITA itself becomes a complex unit, there may be a need to also develop ITA Unit Test Software. Many times when the Test

Station has a limited capability or it is not well suited for testing a particular type of UUT, the ITA will be designed to incorporate active components, UUT stimuli, and measurement circuits of its own. In these cases of complex ITAs, software is needed to perform unit testing on the ITA hardware. This software would operate on the Test Station and could follow the same generic development procedures used for the UUT Test Software.

The next few steps in development of the ITA hardware and UUT Test Software focus on the detailed design, implementations, and checkout of each item. These items are verified as separate units, primarily using three mechanisms:

- The items (UUT Test Software and ITA hardware) are unit tested against the requirements in the respective B-level specifications.
- The items are audited for compliance with the B-level specifications.
- Interfaces between the UUT Test Software, the ITA, and the Test Station are tested to verify compatibility.

The output of these design, development, and verification processes will generally be test documentation (plans, procedures and reports), C-level Product Specifications and drawing packages for the ITA, and the software and hardware items themselves. Together these items and documents form a baseline which will subsequently be used for UUT Test Package validation and certification.

The final step is to perform overall validation and certification of the UUT Test Package as a whole. This is typically performed by a combination of personnel representing both the developer and the end user. At this point, the individual items of hardware and software must be unit tested and the interfaces verified for compatibility. Thus, the tasks remaining focus on three areas:

- Validating that the total UUT Test Package operates properly with the ATE Station and does, indeed, perform the testing as documented in the TRD. Normally a sample of UUTs (with various faults manually inserted) will be used during this process in order to assess the ability of the Test Package to detect and isolate UUT faults. This particular validation procedure is expensive and may indeed be cost prohibitive, but it may be the only method available to validate compliance with the requirement for percent of fault detection (e.g., 95%).

- Audits are also performed of the total UUT Test Package to verify integrity and accuracy of all data.

- Certification is performed by the user's representatives agreeing that the completed Test Package satisfactorily performs in the intended operational environment. This is usually a combination of testing and audit functions similar to those used in the validation process.

## 3. ATE CONTROL SOFTWARE

The ATE Control Software is composed of all the real-time software (except the UUT test software itself) which is required for operating the Test Station. Generally this software can be functionally treated as three groups: the Test Control

Executive, any language interpreters which may be required, and real-time interfaces with the test technician. Each of these groups of software is further discussed below.

a. Test Control Executive: The Test Control Executive software controls all Test Station functions or resources needed to load and execute UUT Test Software. If the UUT Test Software is written in a Higher Order Language (HOL) such as ATLAS or BASIC and an interpreter is to be used for translation of this code, then this HOL "on-line" interpreter will be closely tied to the Test Control Executive. In most instances the functions performed by this Executive are the same on those of any minicomputer real-time operating system; however, several unique models are required. These unique items are briefly discussed below.

- Resource/Path Selection: The Test Control Executive should provide the capabilities for both automatic and manual selection of both equipment resources and data paths for connecting these resources. For example, some ATLAS based test systems allow for the programmer to "manually" specify (in the connection fields of the ATLAS statements) the specific instruments, paths, and buses to be used for a test. Otherwise, the Test Control Executive should automatically select the instruments and paths required to fulfill the UUT Test Software commands.

- ATE Station Protection: In most Test Station configurations there is an ever present danger of damaging the Test Station's resources by software commands misconnecting some of the hardware resources. For example, a manually specified connection might connect power supply outputs to low power input devices. In order to obviate this situation the ATE Station

Protection function would provide a cataloging of both "legal" and "illegal" connections and, in turn, disallow all illegal connections. This function is particularly useful in protecting equipment from damage during UUT Test Software debugging phases.

- Test Instrument Controllers: In order to properly interface with and control each individual instrument associated with the Test Station, a set of Test Hardware Controllers is needed. These Controllers should be modular with one Controller for each item of test hardware used in the station. Generally these Controllers will support three types of station hardware measurement devices, stimulus devices, and power supplies.

b. HOL Interpreter: An Interpreter is a program which does a HOL translation at the time of execution. The important difference between an Interpreter and a Compiler is that with an Interpreter no object code is written in the translation process. An Interpreter reexamines the source program every time it is executed. In fact, every statement is reexamined and reinterpreted every time it is executed during a program. Hence, in a large loop, a statement could be interpreted hundreds or thousands of times.

There are various ways Interpreters implement their function. For example, some convert part of the program to an intermediate language before execution, while others do all the interpretation during execution. Also some Interpreters, known as Direct Interpreters, simply execute the statement without any translation to machine language.

One of the advantages of Interpreters over Compilers is that they may perform better when exceedingly complex or non-sequential programs are input. Also program changes can be made during execution without losing the current values of the program variables. The obvious drawback to Interpreters is their slow execution time.

The two HOLs most commonly used for ATE applications are BASIC and ATLAS. These two languages are each summarized below, but both can (and have) been used with either an Interpreter or Compiler language translator for the ATE Test Station.

- ATLAS: The Abbreviated Test Language for Avionic Systems (ATLAS) was first defined by ARINC in Specification 416-1 in June 1969. ATLAS is a "problem-oriented" language specifically designed for ATE usage. It was orginally designed and developed by a committee of experienced ATE users[9], and as such, has subsequently been adopted by numerous ATE users and vendors. Since it is problem-oriented, the ATLAS language forms resemble a test specification language rather than a traditional computer programming language. This allows for ease of development of the UUT Test Software. The prime weakness of ATLAS is that it allows users to develop "adapted" ATLAS languages, which, if one takes all of the liberties allowed, can result in a new language bearing little resemblance to another "ATLAS language". Hence the buyer of an ATLAS language package must look closely at the particular ATLAS capability offered by a vendor. The potential disasterous effects on language standardization and software portability are obvious.

- BASIC: The Beginner's All Purpose Symbolic Instruction Code (BASIC) language was developed by Dartmouth College in 1965. It was designed to be a very simple language to learn. In fact, the designers intended it to be a stepping-stone for students to, in turn, learn more powerful languages such as ALGOL or FORTRAN. Since BASIC is a

"procedure-oriented" language, it is significantly different from an ATE problem-oriented language such as ATLAS. Thus for use with ATE systems, the BASIC language is usually extended to provide the needed ATE-oriented syntax.

c. User Interface: The test technician must be presented with a simple, user-oriented interface to the Test Station. For current day ATE systems, this interface is typically composed of a CRT terminal for control of the test sequences. In addition, hard copy print devices are usually required to document (or log) some aspects of the test activities.

The critical design requirement of this User Interface Software is that it be designed so as to provide a simplified, "test oriented" interface, and not merely an interface which reflects the minicomputer operating system commands. The ATE user is a test technician and not a computer programmer, thus he should be communicated with in a manner that does not require large amounts of additional computer training.

## 4. ATE SUPPORT SOFTWARE

The ATE Support Software is composed of numerous software "tools" and utility programs to aid the programmer in developing and executing UUT tests. Many of these Support Software items are available as "standard" packages which are sold as a part of virtually any of today's minicomputer systems. Examples are the Operating System, Assembler, Text Editor, etc. The other remaining items of ATE Support Software are "ATE unique" or

"Station unique" items and are developed particularly for ATE Test Station use. Examples are an ATLAS Compiler, ITA Unit Test Software, and ATPG Programs.

An overview of each of the major items of ATE Support Software is given in the following paragraphs.

a. Operating System Executive: The Operating System Executive is a non-real-time executive which is generally furnished as a standard commercial software package with the ATE Test Station minicomputer. Although the particular capabilities of this Executive will vary from one computer vendor to another, the general functions will remain the same. In these general terms, the Executive can be viewed as a resource manager for allocating the four primary computer resources: memory, processors, devices, and information. The functions of the Executive are to assign and control the efficient utilization of these resources, whether in a batch, multi-user, or multi-processor environment.

Table II-1 summarizes the functions of a typical Operating System Executive.

b. Operating System Services: The Operating System Services software can be viewed as a set of adjunct software utilities which are extensions of the basic Operating System Executive. These utilities are generally available as standard items with today's commercial minicomputers used in ATE Test Stations. The major utilities required are briefly discussed below.

| RESOURCE TYPE | EXAMPLE RESOURCES | EXAMPLE FUNCTIONS |
|---|---|---|
| MEMORY | CORE, REGISTERS | MEMORY MANAGEMENT, PAGING |
| PROCESSORS | CPUs, I/O PORTS | SCHEDULING, CONTROLLING |
| DEVICES | TAPES, DISKS, PRINTERS | SPOOLING |
| INFORMATION | USER OR SYSTEM DATA | FILE STRUCTURING |

OPERATING SYSTEM EXECUTIVE FUNCTIONS

TABLE II-1

- Text Editor: A text editor is a program which allows an on-line user to alter his program or other stored data without having to retype the entire program. There are two basic methods of implementing this task. The more elementary method is a line-number based editing system for which each program "card image" is given a line number. The program can then be edited by inserting lines between existing lines, or overwriting existing lines. The second (and more sophisticated) method of text editing uses no line numbers but instead uses a position pointer which can be moved anywhere in the program or data text. Since this is a character-oriented rather than line oriented editor, the user generally has more flexibility and power available in the editing commands.

- Debugging Aids: A debugging aid is either a Operating System option or a statement in the program written especially to check the program operation. Non-Operating System aids are extremely varied depending on the program and application and cannot be covered in an overall summary. There are, however, several typical types of Operating System debugging aids. These usually consist of Dumps, Traces, Subscript Checks and Displays.

  A Dump is simply a record of information (at any given time) of the status of the program. Since it is usually written in machine language, the Dump has limited usefulness.

  The Trace, which comes in three basic forms is a record of the program execution. One type of Trace is a program flow trace, which prints the statement labels as they are passed during execution. The second type is the Variable Trace. Every time a variable changes value the name and new values are printed. In some systems the Variable Trace will have switches available so that only certain variables will be traced. The third type of Trace is the Subroutine Call Trace. When a subroutine is called, the program will output the name of that routine. When control is subsequently returned to the previous program section, a return statement is printed.

  A Subscript Check makes sure that whenever a subscripted variable is used the subscript

is valid. If the subscript is not valid, an error message is printed.

A Display (also called a snapshot) is very similar to a Variable Trace. The major difference is that the variable and value are output at user defined points rather than every time these change value.

- Linker-Loader: The Linker-Loader is a program that takes the relocatable object language from the Compiler or Assembler and aligns it into absolute machine language. The alignment occurs in two separate steps, the first of which is executed by the Linker. The Linker searches through the relocatable program for external (global) references. It then accesses the file, finds the references, and links these to the original program in relocatable machine language. The Loader is the next section of program to operate. It takes the linked program and assigns absolute address locations to all the relocatable addresses.

- Bootstrap Loader: A Bootstrap Loader is a computer initialization program used to start the computer after it has been completely shut down. The user will set certain switches to load the first few steps of the bootstrap program. From here, the Bootstrap Loader will take these instructions and finish loading itself. This process is also called initial program load (IPL) or cold start. The IPL loads in an executive loading program which then loads the compilers, assembler, text editors and other needed programs.

- File Management: The File Management software is usually composed of a general purpose file utility package for both the general user and the system programmer/manager. These software aids normally handle all files with the Operating System's standard data formats. In general, these programs transfer data files from any device in the system to any other device in the system. Also, usually provided is the capability to delete or rename any existing file. Some Operating Systems include special file management operations, such as directory listings, device initialization and formatting, and account creation.

c. Assembler: The Assembler is used to translate "assembly" language (which is usually little more than a one-for-one mnemonic representation of the computer instruction set) into the binary codes or instructions which can be executed by the computer. Since higher order languages are much preferred over assembly languages for coding of applications software, there is usually limited usage of an Assembler for user applications. However, in many of the operating system functions assembly language is prevalent since the higher order languages generally cannot provide enough fidelity in controlling the detail of bit-level computer resources. The Assembler then provides for the translation of assembly language mnemonics into machine code, and is then used in both the development and maintenance of any assembly language programs. For example, the Operating System for the ATE Test Station may be written in assembly language, and for this situation, the Assembler would be used for translation of the Operating System source code.

d. Compilers: The HOL Compiler is a language translator as is the Assembler; both normally producing relocatable object code for a particular computer. However, the Compiler is much more sophisticated than the Assembler due to the increased complexity of the input language. The Compiler accepts a higher order language (such as ATLAS or BASIC) as input.

Although the purpose of a Compiler is the same as that of an

Interpreter, these are two distinctly different software items. The Interpreter is executed as an extension of the real-time ATE Control Software and "interprets" the UUT Test Software as it sequences through each HOL source code statement. Thus Interpreters do not produce a relocatable object code as Compilers do, and as such, are less efficient in run time execution speed. By contrast, the Compiler executes in an off-line (or in non-real-time) mode and completely translates the source program (e.g., ATLAS) into a machine program form suitable for linking and loading. While Interpreters are useful translators for some ATE Systems, the Compiler offers much more power and sophistication to the user and is generally a preferred implementation for ATE applications.

e. Configuration Management Aids: Since a single ATE Test Station will typically support the testing of many UUTs, there will likewise be many packages of UUT Test Software used on the Test Station. Even for a single UUT model type, there may be many different versions of the UUT Test Software required if the UUTs have various engineering changes incorporated. This multitude of software packages can lead to a significant configuration management and status accounting problem.

The Test Station provides an excellent mechanism for performing the necessary configuration management of the ATE software. Software Configuration Management Aids can be provided on the Test Station minicomputer which will provide the follow-

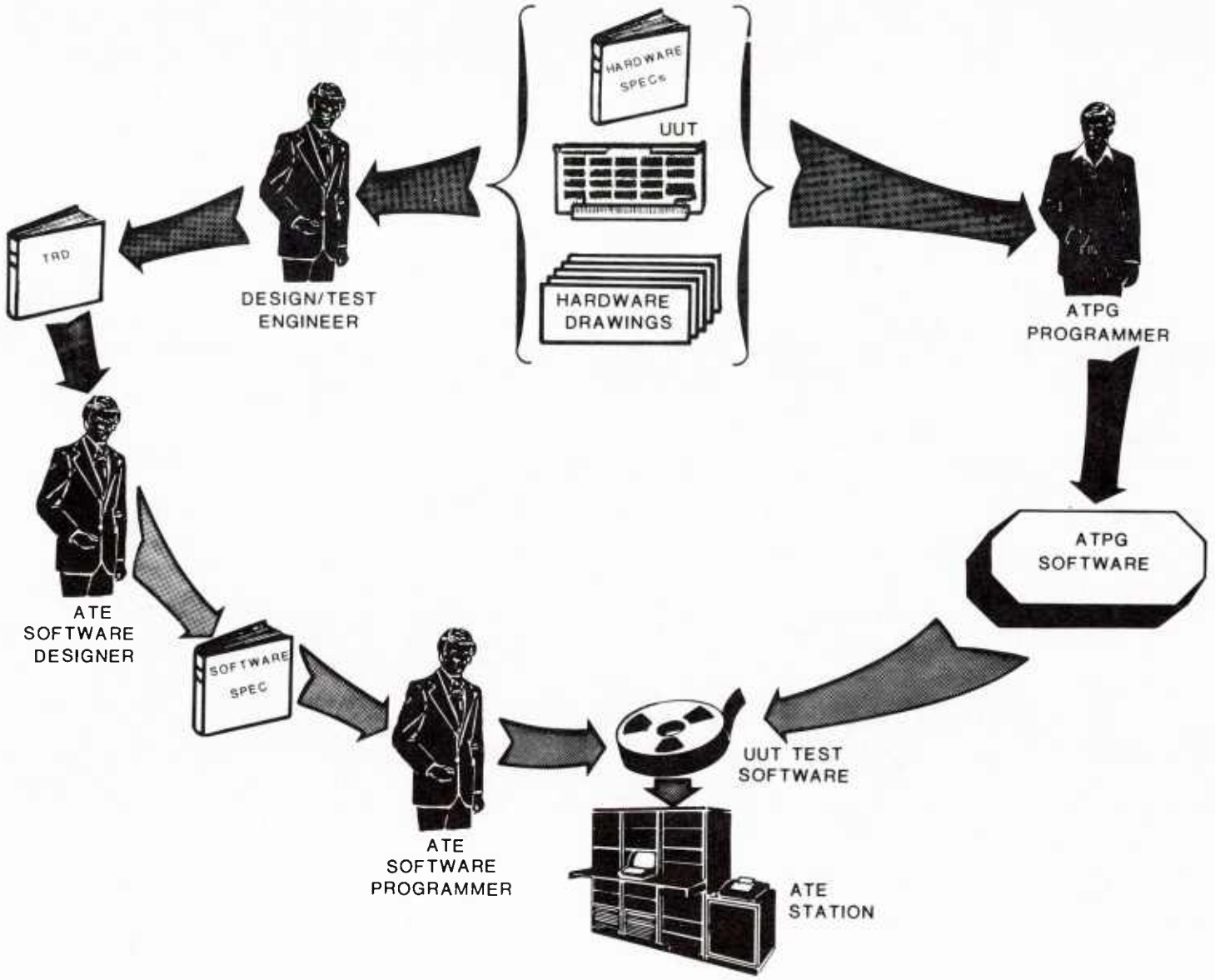
ing types of functions:

- Limiting the number of users which have change control access to the baselined software.
- Providing a library of "sanctified" software modules.
- Providing configuration status accounting information.
- Providing configuration identification of the various hardware items (UUT and Test Station) and software items.
- Providing software file comparison routines which allow for automated comparisons of versions of the same software modules.

f. ATPG Tools: An Automatic Test Program Generation (ATPG) Tool is a software program which accepts as input a circuit description of a UUT and generates as output the required UUT Test Software. Figure II-5 shows the relationship of this "automatic" process to the manual programming process. For those areas where ATPG can be effectively used, the advantages are obvious. ATPG can save both time and manpower required to develop UUT Test Software, and in certain cases, can generate superior test software compared to the manual methods.

The use of ATPG tools and techniques has most effectively been applied to digital card units, mainly because digital functions can more readily and precisely be simulated on a digital computer. This use has significantly increased in recent years, due primarily to two reasons:

- With the increased capabilities available in



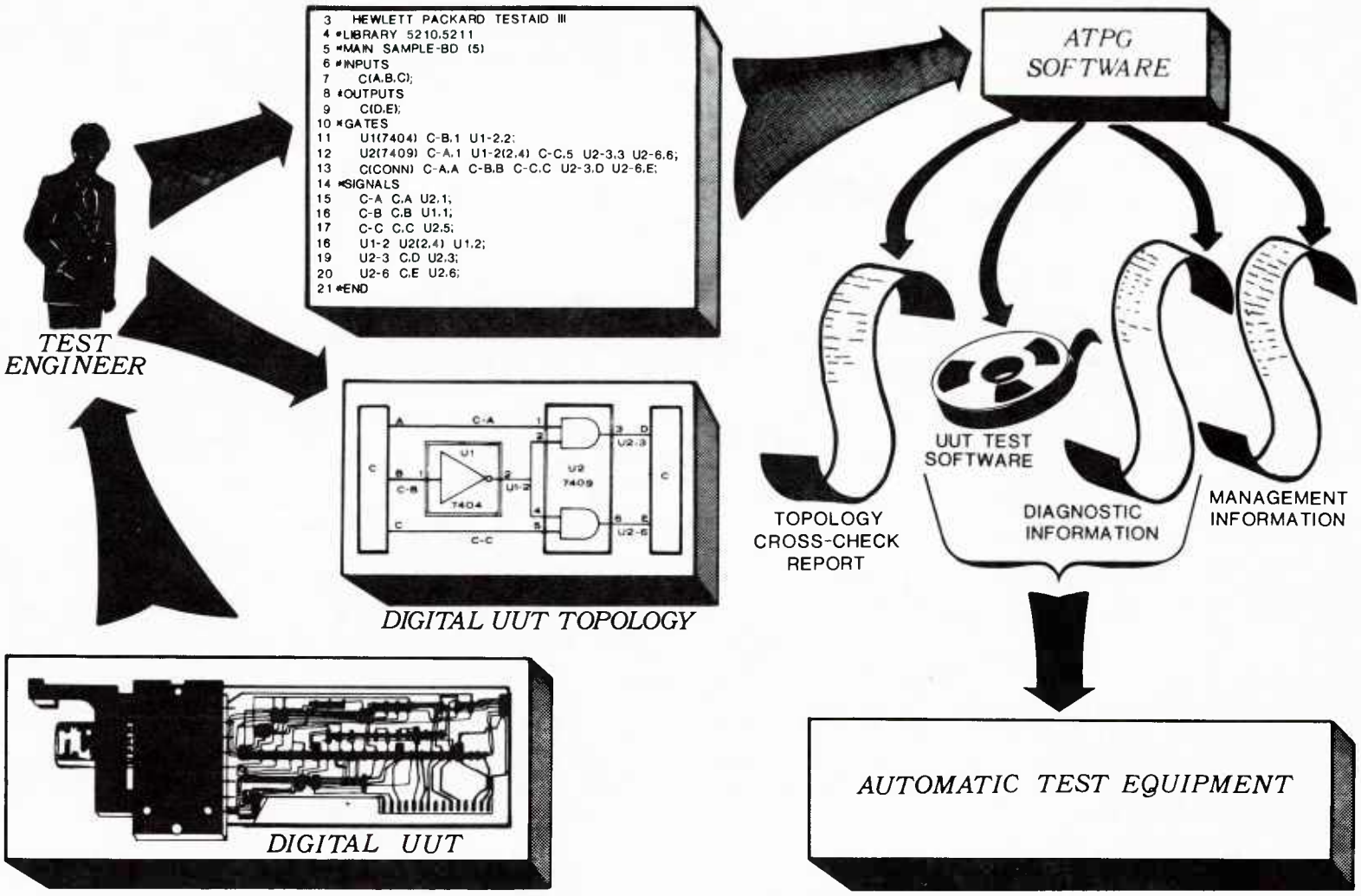

GENERATION OF UUT TEST SOFTWARE

Figure Number II-5

today's minicomputers for ATE Stations, these minicomputers are being used to host some ATPG software packages. This makes the ATPG process much more responsive to the user's need. With the ATPG software Resident on the ATE minicomputer rather than a distant (and often slow turn-around) large-scale computer complex, the test designer can usually operate more effectively. In fact, the writing and debugging of UUT Test Software is today becoming a large factor in total support costs[1], and thus is a promising area for potential savings. Minicomputer ATPG packages are expected to make a significant contribution to this savings through automation of UUT Test Software generation.

- The use of digital ATPG methods by numerous and varied organizations has resulted in a rapid development of this technology. Numerous software packages have been developed, and both the developers and users are striving to continually enhance these ATPG Tools. As a result, ATPG Tools offer: (a) dramatic reduction in UUT checkout and repair time (25:1 in some instances[1]), (b) greatly improved UUT fault detection rates (100% versus 50% for one test case of 100 IC boards[1]), (c) higher product confidence levels, and (d) the option to use lower skilled personnel for testing.

To best illustrate how a digital ATPG Tool is used, an example[1] is outlined below using the Hewlett-Packard TESTAID III System. Also, the overall process of using the digital ATPG Tool is pictorally shown in Fig. II-6. The first step in ATPG is for the test engineer to develop a "topological" description of the UUT. For TESTAID-III, this description can be either "device oriented" (which specifies all input and output signals for each device), or "signal oriented" (which specifies all




# DIGITAL ATPG PROCESS

Figure Number II-6

devices or gates that are connected by each signal line). If both the device and signal descriptions are used as inputs, then TESTAID-III will cross-check these for discrepancies to help ensure correct coding.

Once this topology coding is completed, it is input to the ATPG software for subsequent processing. Within TESTAID-III there are numerous functions performed. The major ones are highlighted below:

- UUT input test patterns are generated if automatic generation is requested. However, a provision is made for the test engineer to input manually generated test patterns if desired.
- UUT "good" output patterns are generated for all input patterns.
- UUT fault signature patterns are generated to identify "failed" devices based on the detection of other than "good" UUT outputs.
- UUT signal mode states are generated for use in diagnostic procedures.
- Summary information is generated to report such items as the test coverage (typically as a percentage) and listings of undetected faults.
- And finally, the source code (e.g., ATLAS or BASIC) is generated. This source code can subsequently be used (via a Compiler or Interpreter) to execute the test patterns derived. Detailed "run-time" procedures are given to the test operator during UUT testing, thus allowing for the use of lower skilled personnel than with less automated methods.

The payoffs associated with the use of ATPG can be grouped into three main categories. These are briefly discussed below:

- The use of ATPG in combination with ATE can result in a significant savings in the time required to test, diagnose, and repair a UUT. Some example data[1] are shown in Table II-2 for cards produced and maintained by Hewlett-Packard. These data indicate that a 10:1 reduction in test and diagnostic times can easily be achieved.

- The skill level required of test conductors can be significantly reduced. Trained and experienced technicians are required for manual or semi-automatic methods, whereas fully automatic methods can be conducted with a minimally trained operator.

- The thoroughness of the automated methods has led to a greatly improved fault detection rate. Again, some Hewlett-Packard data covering some 100 IC boards demonstrated fault detection rate improvements from 50% using old methods to 100% using ATPG/ATE methods.

It should be noted that these payoffs are a combination effect of using both ATPG and ATE. Ideally, a good ATE programmer could generate the same test procedures (and UUT Test Software) manually without the benefit of ATPG. These tests could, in turn, be run on the ATE and the above benefits would again be attained without ATPG usage. However, in the practical world this could seldom be attained. The complexity of the UUTs, the massive amounts of source code UUT Test Software required, and the difficulty in locating and retaining these highly skilled programmers, all are practical limitations to the usefullness of ATE without ATPG.

However, digital ATPG tools are not without their limitations. Most digital ATPG systems are based upon path-sensitizing techniques which trace (sensitize) paths through a simulated digital

# TABLE II-2

## ATPG/ATE TIME SAVINGS

| BOARD DESCRIPTION | TEST & DIAGNOSE TIME | |
|---|---|---|
| | Manual Or Semiauto Methods | Using ATPG/ATE Methods |
| BOARD #1 (COUNTER)<br>● 35 ICs<br>● 48 PINS<br>● 500 (APPROX.) GATE EQ. | 90 MINUTES * | 8 MINUTES * |
| BOARD #2 (CONTROL)<br>● 95 ICs<br>● 120 PINS<br>● 1200 (APPROX.) GATE EQ. | 45 MINUTES | 3 MINUTES |
| BOARD #3 (ROM CONTROL)<br>● 88 ICs<br>● 102 PINS<br>● 864 GATE EQ. | 160 MINUTES | 6 MINUTES |

* includes repair time

network[8], from given input vectors to the resultant output vectors. In general these ATPG systems suffer from several restrictions and may require manual "work-arounds" in order to construct comprehensive tests. Examples of where path-sensitizing ATPG systems cannot be effectively used are: (1) cards that contain curcuits of large memory elements (counters, shift registers, or RAMs), (2) cards with large scale integration (LSI) circuits such as microprocessors, and (3) circuits which use asynchronisms in their operations.

g. ITA Unit Test Software: The Interface Test Adaptors (ITAs) for complex UUTs may themselves become complex electronic units which deserve some level of stand-alone testing ("unit test") prior to operation of the UUT. Also, ITAs can become complex as a result of the ATE Station not having all of the requisite capabilities (i.e., stimuli and measurement devices) which are required to satisfy the TRD requirements specified for a particular UUT.

For these more complex ITAs, ITA Unit Test Software is developed which, when executed, will insure that the ITA is working properly prior to testing the UUT.

h. ATE Station Self Test Software: Since the ATE Test Station itself is a collection of equipments, all of which are subject to malfunctions and maintenance needs, there is a requirement to perform routine testing to ensure proper operation of the Test Station hardware items. This testing is usually divided

into three functional areas, each being implemented to a large extent in software. These three software packages are briefly discussed below.

- Station Confidence Test: The Confidence Tests are designed to exercise all major functions of the ATE Test Station to verify that each Station item is powered, initialized properly, and communicating properly with the Station control elements. Also, the gross functional characteristics of the stimulus and measurement devices would be checked. Since this software may be required to run on a daily basis or at the beginning of a test, it cannot consume an extensive amount of time. Thus, exhaustive testing of the Test Station is prohibited. For example[10], the Confidence Test Software for the F-16 Avionics Intermediate Shop Station is designed to execute in less than 15 minutes and to provide at least 85% assurance of the Station operability. Also, this F-16 software executes without the need for external Test Station connections.

- Station Maintenance Test: When a malfunction is detected in the Test Station operations, the Maintenance Test Software is used to further characterize the failure and subsequently isolate the failure to a particular hardware unit. In contrast to the Confidence Test Software which performs gross functional tests, the Maintenance Software performs exhaustive testing of each hardware item. Also, external connections or manual test procedures are usually used to complement the software controlled testing. This testing includes coverage not only of the test instruments in the Test Station, but also the Test Station computer and its peripherals. Isolation of faults should be provided to either the LRU or SRU level depending on the particular operational environment.

- Station Alignment: This software is designed to automatically update the data base of alignment parameters for all test instruments used

in the Test Station. Using the internal calibration reference standards (voltage, frequency, and resistance), this software measures and records the gains and offsets of the Test Station instruments and signal paths. For example, if a power supply develops an output of 15.5 volts when commanded to supply 15.0 volts, this inaccuracy might significantly affect UUT testing. Thus, the Alignment Software would determine the proper input command offset needed (e.g., 14.5 volts commanded) in order to receive the desired output level of 15.0 volts. This offset (-0.5 volts) would be stored in the Test Station calibration data base and used to offset all subsequent input commands to that particular power supply.

## 5. UUT TEST SOFTWARE

The UUT Test Software operates on the Test Station under control the ATE Control Software. It contains the control logic which specifies the detailed sequencing of the UUT testing, and thus, the UTT Test Software packages are unique to each separate UUT.

The major functions implemented by the UTT Test Software are summarized below.

- Definition of input signal parameters and power requirements.
- Specification of test sequencing, including "go" and "no go" conditions, fault isolation algorithms, etc.
- Notification and display of test status and results to the test operator.

As discussed in the previous sections, the trend in ATE systems is toward the use of HOLs as programming media for UUT Test Software implementation. An example section of a UUT Test Software package is shown in Fig. II-7 below using the ATLAS language.

```
 ØØØØØØ BEGIN, ATLAS PROGRAM 'VOLTAGE TUNED OSCILLATOR TEST' $
C $
C   PREAMBLE-DEFINES SIGNAL SOURCES AND POWER SUPPLIES $
C $
     Ø1 DEFINE, 'DC-POWER', SOURCE, DC SIGNAL,
            VOLTAGE 12 V ERRLMT +- .1 V,
            CURRENT MAX 1ØØ MA,
                CNX HI J1-5 LO J1-6$
     Ø2 DEFINE, 'TUNING-VOLTAGE', SOURCE, DC SIGNAL,
            VOLTAGE () RANGE 1 V TO 1Ø V ERRLMT +- 5Ø MV,
                CNX HI J1-8 LO J1-6$
C $
C   MAIN SECTION OF PROGRAM-SETS VOLTAGE, MEASURES FREQUENCY $
C $
 Ø1ØØØØ PRINT, MESSAGE, CONNECT UUT,PRESS RUN BUTTONS
     Ø1 WAIT FOR, MANUAL INTERVENTIONS
     Ø2 APPLY, 'DC-POWER'$
     Ø3 APPLY, 'TUNING-VOLTAGE', 5 V$
     Ø4 VERIFY, (FREQ), AC SIGNAL
            FREQ RANGE 4 MHZ TO 6 MHZ,
            VOLTAGE RANGE 2 V TO 5 V,
            LL 4.98 MHZ UL 5.Ø2 MHZ,
                CNX HI J2-1 LO J2-2$
     Ø5 REMOVE, 'DC-POWER'$
     Ø6 REMOVE, 'TUNING-VOLTAGE', 5 V$
     Ø7 GO TO STEP 3ØØØØ IF NOGO$
     Ø8 PRINT, MESSAGE, VIO FREQUENCY IN SPEC$
     Ø9 GO TO STEP 4ØØØØ$
 3ØØØØ PRINT, MESSAGE, VIO FREQUENCY OUT OF SPEC$
 4ØØØØ FINISH$
 999999 TERMINATE, ATLAS PROGRAMS
```

Example of UUT Test Software Using ATLAS

Figure II-7

## SECTION III

## ADDITIONAL ATE SOFTWARE REQUIREMENTS

### 1. SECTION OVERVIEW

The verification and validation (V&V) of UUT Test Software is today a costly and time consuming process. However, this problem is not unique to ATE software. In general, the testing of any type of software item is the most tiring, expensive, and unpredictable phase of the overall software development process. To aid in these testing efforts, there have been many types of software test tools developed. The idea being, computers should be used not only to execute software, but computers should also aid the programmers in the checkout and test activities.

The V&V tools available today can be effectively partitioned into two general categories. The first category is that of hardware simulators. Hardware simulators are used to emulate the interface signals and functional operations of specific "boxes" of hardware. For example, in the integration of an avionics system it is common practice to simulate such items of hardware as the radar, inertial measurement unit, air data computer, etc. This level of simulation is generally expensive, but is many times required due to the non-availability of the hardware items.

To the contrary, simulation of UUTs for V&V of the UUT Test Software is not a common practice. Due to the high cost of developing a high-fidelity simulator for a UUT, such a simu-

lator is usually not undertaken. Also, for many developments the actual UUT hardware items are readily available for use during the UUT Test Software V&V phase.

The second category of V&V tools is "test aids". These test aids constitute a less well defined category then the hardware simulators above, and are in many respects a nonhomogenous set of items. However, they do form a good category for discussional purposes. Examples of test aids are: dynamic simulators, data base analyzers, automated test generators, automated flowcharters, instruction tracers, test drivers, timing analyzers, etc. Use has been made of test aids in virtually all application areas of software, including ATE systems. One of the areas in which extensive use of test aids has been made is in the V&V of real-time avionic software. However, the use of this test aid "technology" has somewhat lagged in the ATE community, even though it is still a cost-effective undertaking.

Of the two categories above, hardware simulators and test aids, the latter appears to offer the most leverage in the V&V of UUT Test Software. The state-of-the-art does not appear to support hardware simulators as cost-effective test tools in the V&V of today's UUT Test Software. In contrast, many of the V&V test aids do directly support this activity as a relatively low-cost alternative.

In the section below the concept of an ATLAS Statement Level Simulator test aid is discussed as it would apply to V&V for UUT

Test Software. Some aspects of this concept have been implemented in today's ATE systems, however, not in such a coordinated or "integrated" fashion. By providing the UUT test programmer with a comprehensive Statement Level Simulation capability (in addition to the other support software items discussed in Section II), the UUT Test Software V&V process can be not only completed more quickly, but this can lead to more accurate and reliable software.

## 2. ATLAS STATEMENT LEVEL SIMULATOR

The Statement Level Simulator (SLS) for ATLAS discussed below is patterned closely after the JOVIAL-73 language SLS developed for the DAIS Program[12]. This ATLAS SLS would provide the ATE programmer with a collection of ATE V&V tools which would make use of many of the capabilities (e.g., symbolic references) already provided by the ATLAS compiler. Basically, it would provide for the "symbolic" execution of ATLAS programs in a simulated environment which could be easily created by the UUT test programmer. Instead of undertaking an exhaustive simulation of the UUT, the programmer would be supplied with a high-level mechanism (through a scenario generator) which could map UUT inputs into UUT outputs. By circumventing the functions of the UUT itself, the high cost of building an accurate UUT simulation is eliminated. The programmer would supply a list of the UUT responses (either prior to the SLS execution or interactively).

The ATLAS SLS would provide a run-time environment in which the ATLAS UUT Test Software could execute. The test programmer

could force the UUT Test Software to execute certain paths (e.g., GO or NO GO conditions), start and stop the UUT Test Software based on the occurance of specified "events", and symbolically (e.g., ATLAS names) reference data items or locations. The main functional elements required of the ATLAS SLS are discussed below.

a. Test Scenario Generation: In order to provide the test programmer with a mechanism for specifying the test conditions and data recording requirements, a high-level test scenario language would be provided. This language would allow the test programmer to define "events" in the execution of the UUT Test Software, and in turn, the actions to occur based on these events. Examples are:

- Based on the event of executing a named ATLAS source statement, program execution is to hault and await the programmers manual input.
- Based on the event of a new output being applied to a specified pin number, the output and other selected state variables are to be recorded for post-run analysis.
- Based on the event of an input voltage exceeding a predefined safe range, the ATLAS statement label and the output value are to be recorded for post-run analysis.

In addition, the scenario language would allow the programmer to define the types of data to be recorded during SLS execution. For example, selected data items could be traced, instruction flow could be traced, operator actions could be recorded, and

UUT input/output data could be recorded. These data would, in turn, be available for post-run analyses.

An additional feature of the scenario language would be the ability to specify an input/output data list for the UUT in question. In some respects this can be thought of as a simulation of the UUT, but the intent is for a much simpler mechanism than a traditional hardware simulator.

b. Statement Level Executor: The statement level executor would provide for controlled execution of the UUT Test Software per the test scenario provided. As ATLAS statements are executed by the statement level executor, interactions would be made with SLS event and data tables (which communicate with the run-time execution monitor).

c. Run-Time Execution Monitor: This monitor function would provide for "watching" the execution of the UUT Test Software, and the taking of snapshots of the software's status at the occurance of defined events. This event-oriented sampling technique would gather UUT Test Software performance data by interrupting the normal processing to record the status at the programmer selected events. In essence, control is passed to a data logging program when the monitor detects the occurance of such a predefined event. Upon completion of the data logging, the execution of the UUT Test Software is restarted.

At the programmer's option, the occurance of an event could also halt the UUT Test Software execution, pending an input or

action from the programmer's terminal. Inputs could be to force a particular branch of software execution (e.g., GO or NO GO for test completion), or possibly to specify the bit pattern of a UUT output connector. In any circumstance, data logging would still be performed for later post-processing activities.

d. Post-Run Analysis: After completion of execution of the UUT Test Software, the post-run analysis function would provide the programmer with many options (reports) for analyzing the UUT Test Software's execution. Example reports would include: program flow trace at the ATLAS statement level, data trace and usage analysis at the ATLAS symbolic level, variable range analysis as compared to predefined range bounds, logs of operator actions, timing of ATLAS code segment, and ATLAS instruction frequencies of execution.

# SECTION IV

# ADDITIONAL MATE SOFTWARE REQUIREMENTS

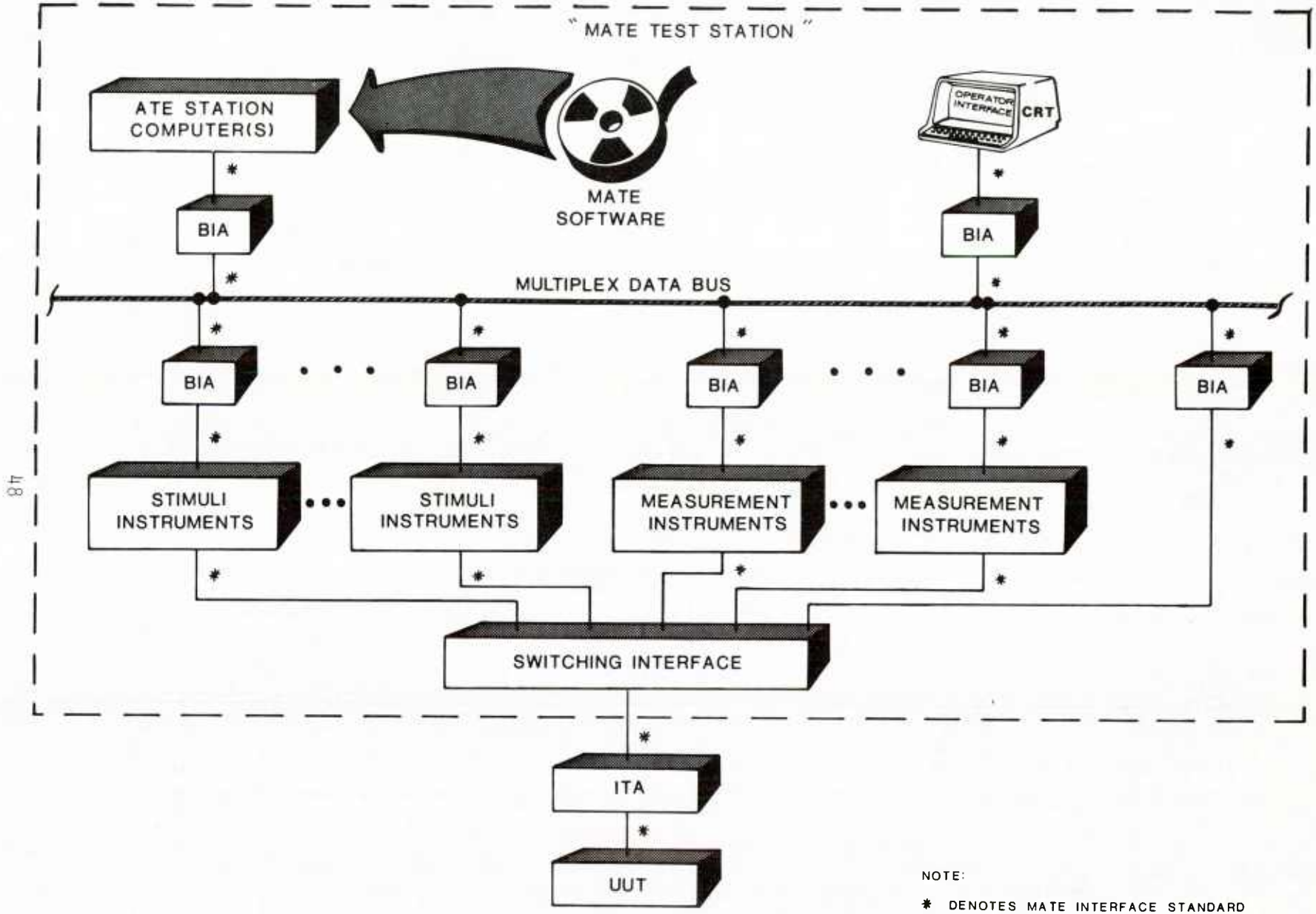
## 1. SECTION OVERVIEW

In addition to the general items and requirements of ATE software discussed previously in Sections II and III, other software requirements are derived from the unique design aspects of MATE. These "MATE-unique" requirements are discussed in paragraph 3 below. Also, paragraph 2 is first included to briefly highlight the MATE system architecture used as a basis for this study.

## 2. OVERVIEW OF MATE ARCHITECTURE

At the date that this report was written, the MATE system architecture was not well defined. Only broad MATE objectives had been established, and system requirements were not yet fully specified. Moreover, a MATE hardware architecture was not fully defined, and thus the software analyses in this report were based on a postulated MATE architecture rather than an actual one. However, it is reasonable to believe that the final MATE architecture will be relatively close to the one described below.

a. MATE Hardware System: Figure IV-1 is a generalized block diagram of the postulated MATE hardware architecture. This architecture is centered on the use of a standard (e.g., IEEE-488) multiplex data bus, standard hardware modules




# POSTULATED MATE HARDWARE ARCHITECTURE

Figure Number IV-1

which can be procured from multiple vendors, and standardized hardware interfaces as shown in this figure. The bus interface adaptors (BIAs) are standard terminals for the data bus and may be provided in various levels of complexity. For example, some units (such as the computers) may require more sophisticated interfaces than others. Also, some BIAs may be required to interface the data bus to multiple hardware devices in cases of lower data rate communications.

The ATE Station Computer is shown as a single unit in this figure; however it could possibly be configured with multiple computers. Several options are available in this area (e.g., single computer, federated computer, multiprocessors, etc.) depending on the processing throughout required for the MATE station. In fact it is likely that the final MATE architecture will, indeed, be able to accommodate from one to several minicomputers depending on a particular station's requirements.

Although the operator interface shown in Figure IV-1 is a CRT terminal, this could also be any one of several standard minicomputer terminal devices. For most MATE configurations a minimum of a CRT and a printer device would be needed.

b. MATE Software: The MATE software requirements will parallel very closely those of any of today's state-of-the-art ATE systems. That is, virtually all of the items discussed in Section II will be needed. More detail concerning these software items is contained in Section VI. In addition to these items, however, the

MATE architecture shown in Figure IV-1 places additional requirements on the software needed. Some aspects of these additional requirements are covered in the paragraphs below.

### 3. UNIQUE MATE SOFTWARE

The only major new software requirement of MATE that has not been previously discussed in Section II or III concerns the MATE data bus architecture. Since this data bus is central to the MATE hardware structure and is the backbone of communications between all major hardware elements, it must be a carefully considered and allocated resource. In fact, much of the inherent flexibility of the MATE architecture shown in Figure IV-1 is directly derived from the data bus and its standard interface connections. This hardware system flexibility will allow for ease of system reconfiguration by interchanging hardware boxes, deleting or adding new hardware boxes, and adding redundant or improved hardware where required.

Although the hardware flexibility can be easily achieved with the use of appropriate interface standards, the total MATE systems flexibility will also depend directly on the software flexibility. In other words, the MATE executive software must be designed to be easily adapted to accomodate the above types of hardware changes. This, in turn, implies that the executive software be highly modular in the input/output areas, and that a generalized (e.g., data table driven) executive approach be used. In conjunction with such an executive, there may be a need to develop some support software tools to assist the users in tailoring such a generalized

executive. These considerations are more thoroughly discussed in Section VI.

## SECTION V

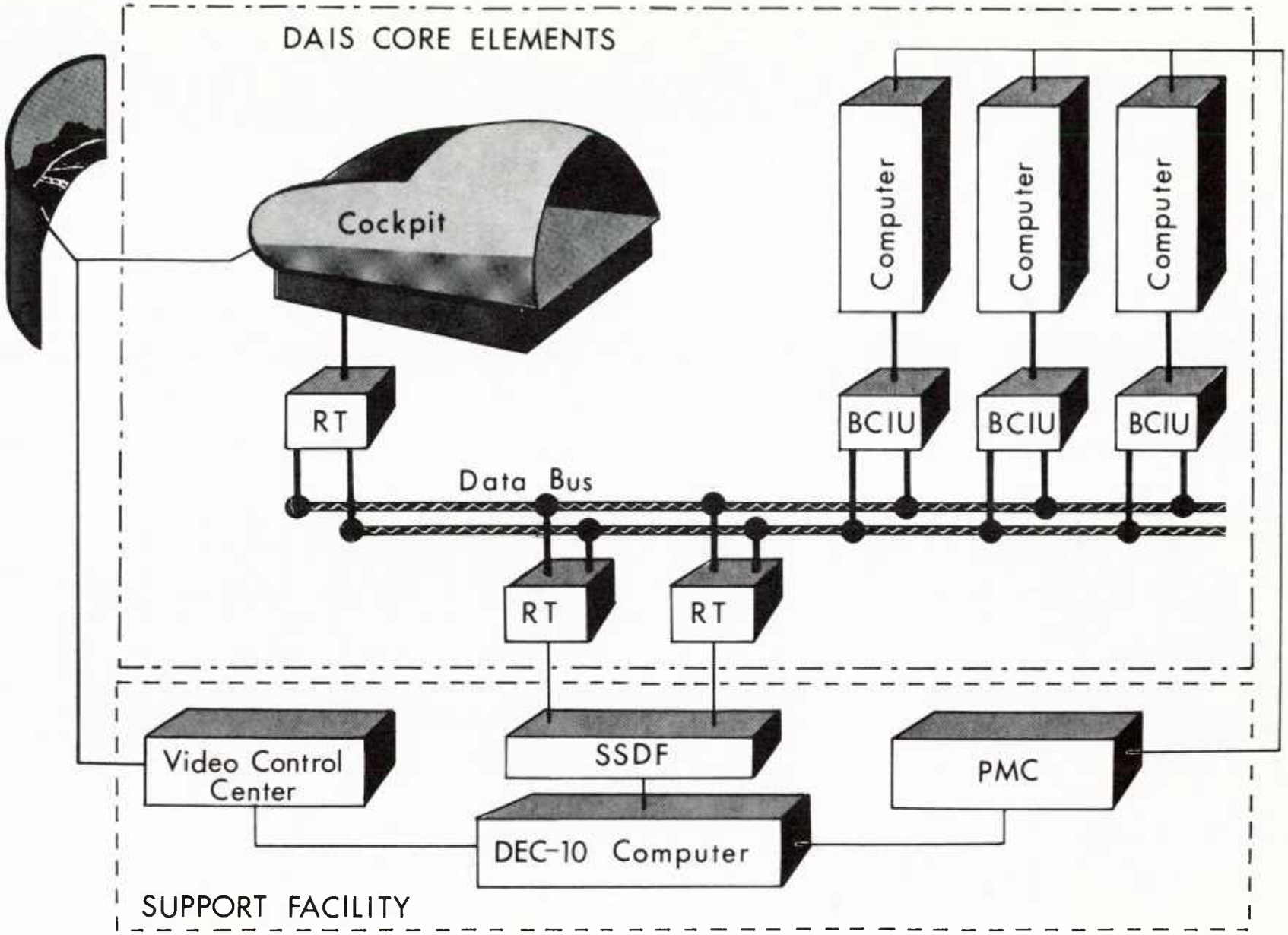
## DAIS SOFTWARE

### 1. OVERVIEW OF REQUIREMENTS AND SOFTWARE

The DAIS "hot bench" or Integrated Test Bed (ITB) is: (1) an information management system consisting of a set of federated airborne computers interfaced to each other, to avionic sensors, and to cockpit controls and displays by a MIL-STD-1553A multiplex data bus; and (2) a support facility to perform the information management system monitor and control functions. Figure V-1 is a simplified block diagram of the DAIS Integrated Test Bed[12]. Control of information management system functions is performed by the DAIS Mission Software which is partitioned among the DAIS airborne computers.

The following paragraphs highlight the functions of the various Integrated Test Bed components.

a. DAIS Information Management System: The DAIS hardware "core elements" shown in Figure V-1 are based on a federated computer architecture. Each DAIS computer is connected to a Bus Control Interface Unit (BCIU) which initiates data transmission over a redundant multiplex bus system between the processors and remote multiplex terminals (RTs), the latter being the interface between the data bus and the simulated avionic equipment. Each BCIU is actually an intelligent input/output (I/O)




# DAIS INTEGRATED TEST BED

FIGURE V-1

channel which executes I/O commands stored in the DAIS computer's memory. Centralized single point data bus protocol is performed by a processor resident software executive and a selected master BCIU.

The remote terminals provide an interface between the bus and aircraft equipments. Conceptually, it functions in a manner similar to a BCIU by transferring data to or from the equipment to which it interfaces. The RT contains interface modules which can be interchanged to provide the correct electrical interfaces for different equipment. It can also be programmed to define the mapping of data between the bus and the aircraft equipments.

The DAIS flight software (Mission Software) is distributed among the set of computers in the system. It consists of Application Software, which performs the processing required for a specific aircraft/mission application, and the Executive Software, which performs system control and provides services to the Application Software.

The Executive Software is further divided into the Master Executive and the Local Executive. The Master Executive, which is responsible for system control, resides in one computer designated as the master computer. A copy of the Local Executive is located in each computer and provides real-time services, including data read and write, task control, etc., to the Application Software.

The Mission Software is implemented in the JOVIAL J73/I

higher order language utilizing structured programming techniques and a modular architecture approach.

b. DAIS Support Facility: The Support Facility provides the necessary interfaces to set-up, provide real-time control and monitoring, and collect data for post analysis for all DAIS testing activities.

A DEC System-10 (DEC-10) computer is used to execute real-time aircraft and environment models, compile the DAIS Executive and Applications Software, generate simulated mission scenarios, perform post run analyses, and maintain all the above files and simulation data under a configuration management system.

The Performance Monitoring and Control (PMC) computer in Figure V-1 is a PDP 11/40 interfaced with the DEC-10 via a Direct Memory Access (DMA) window. This computer is used to load the Mission Software from DEC-10 storage onto the DAIS computers. Operation of each airborne computer is controlled by the PMC which monitors the computer's memory buses and performs such functions as monitoring specified memory addresses, tracing branch instructions, breakpointing based on events, etc. The PMC computer interacts with the user to set up monitoring parameters and can also use predefined scenarios stored on the DEC-10 to set up the PMC. Real-time display of system performance is available on a local CRT.

c. Software Overview: The Software for DAIS is grouped into

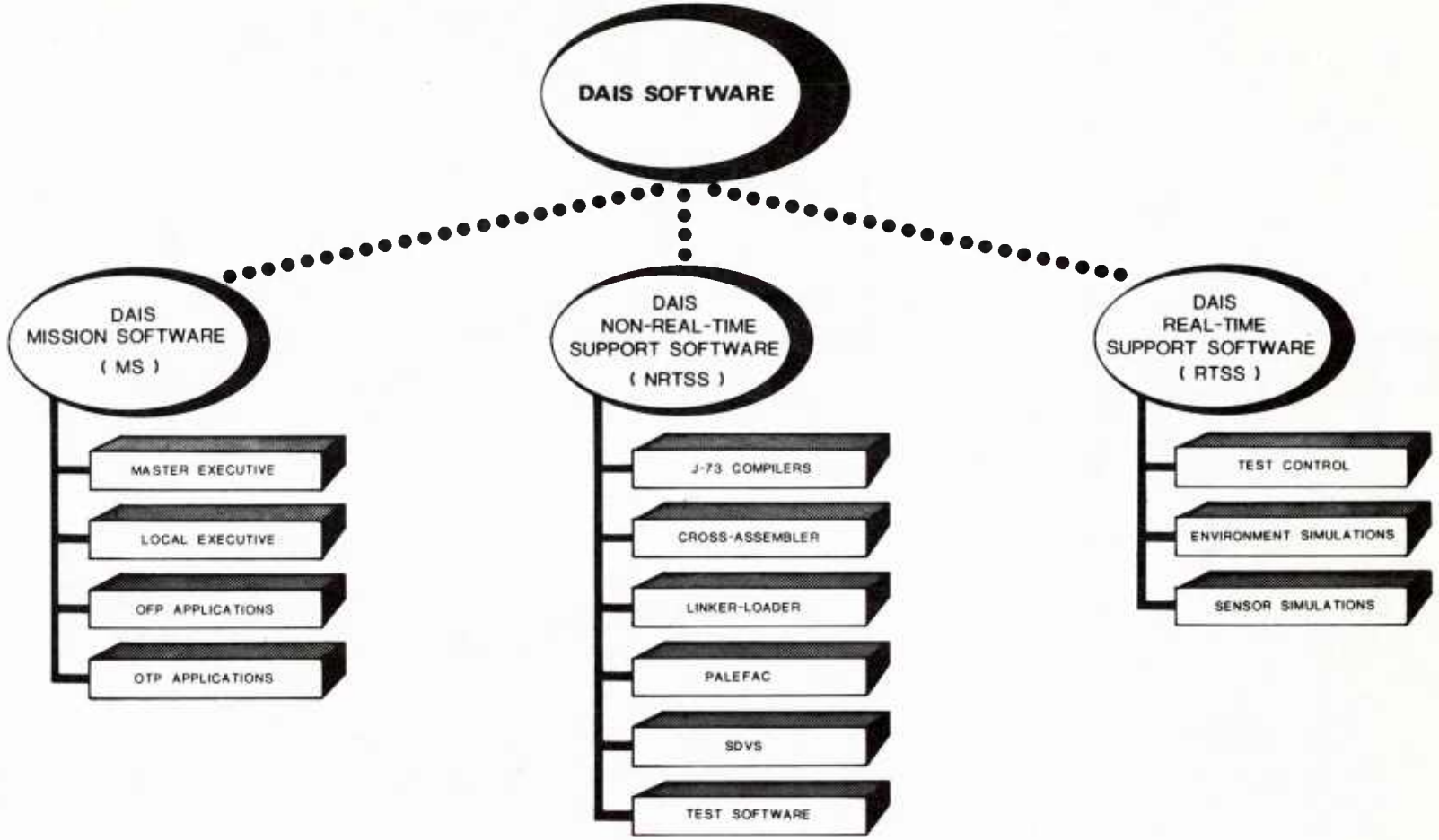
three functional areas as shown in Fig. V-2 The Mission Software performs the traditional airborne functions of the Operational Flight Program (OFP) and the Operational Test Program (OTP). This software resides in the DAIS "hot bench" (airborne) computers. The second category, the Non-Real-Time Support Software (NRTSS), resides on the DEC-10 support facility computer and consists of off-line support software tools used to assist in the design, development, and testing of the Mission Software. The third category is the Real-Time Support Software (RTSS). This software resides in both the DEC-10 and PDP-11 support facility computers and provides for real-time simulation and test control of the DAIS hot bench.

Each of these three major groups of DAIS software are discussed in more detail in the following paragraphs.

2. MISSION SOFTWARE

As noted earlier, the Mission Software is composed of two functional groups (13); the OFP and OTP. Since the DAIS executive software requirements for both the OFP and OTP can be fulfilled with a single executive architecture, only one Mission Software Executive exists. This Executive is then married with either the OFP Applications Software or the OTP Applications Software to form a fully functioning OFP or OTP.

a. Executive Software: The purpose of the DAIS Executive (14) is to isolate the physical aspects of the DAIS federated system




DAIS SOFTWARE

Figure Number V-2

from the Application Software. The Executive allows the Application Software to reference time, Remote Terminals and information in other processors on a logical level. It masks the federated nature of the DAIS computer system so that Application Software can be written as if it were to execute in a single, virtual machine. Finally, the DAIS Executive controls and optimizes the use of system-wide resources, such as the Data Bus and Mass Memory, and provides mechanisms for error recovery.

The DAIS Executive Software consists of two parts: a Local Executive and a Master Executive. Every computer in the DAIS federated system contains a Local Executive, but only one Master Executive exists and is in operation at any given time. The Local Executive controls operations peculiar to a computer, including control of the Application Software within the computer and local participation in the I/O processes. The Master Executive controls system-wide operations, including control of the Data Bus, of Mass Memory, and system-wide initialization and error recovery.

DAIS is a real-time system in which the activities of the Applications Software are coordinated with the passage of real time in the outer world. The minimum granularity of time to which coordination occurs is known as the Minor Cycle. It is possible to specify or determine the time of an action within one Minor Cycle, but not to a fraction of a Minor Cycle. Thus,

the I/O interactions, interprocessor interactions, and task interactions may occur, may be known, and may be controlled within the framework of the Minor Cycle time granularity. This timing is a requirement for I/O control, interprocessor coordination and synchronization, and the Local Executive process handling.

Because of the multi-processing nature of the DAIS system, a designated active Master Executive within one computer controls the federated computer configuration. It responds to data bus transmission errors, and controls communication between data bus terminal units. The Local Executive provides the interface between the application functions and the Master Executive (Bus Controller) functions. In addition, there is an interface between the Master Executive and the application functions with respect to system configuration, initialization, and recovery. The Applications Software controls the execution of software functions by invoking the Executive to schedule and/or activate processes, events, and I/O.

A summary of the Master and Local Executive functions are given below.

MASTER EXECUTIVE

- Bus control - allocates time segments on data bus for synchronous communication and for asynchronous messages.
- Systems error management - monitors and analyzes errors relative to the operation of the processors and data bus communications, and provides control for error recovery.

- Configuration management - initializes multiple computer system at startup and after severe system errors.

- Mass Memory management - provide for the retrieval of information from mass memory.

  Monitor Management - provides for monitoring of the master processor by the monitor processor.

LOCAL EXECUTIVE

- Task state control - uses a task table to activate and deactivate periodic or non-periodic tasks when appropriate conditions have been met. These conditions are based on a logical setting of real time events.

- Event control - uses a table of real time events to communicate conditions signalled between processes whether in the same or different processors.

- Data control - guarantees interlocks between shared data, provides mechanism for transmission and reception, of data over the multiplex data bus.

- CPU fresh start, restart - used to initialize CPU, to recover from transient failures, and to perform self-test.

b. OFP Applications Software: The OFP Applications Software executes in the DAIS airborne computer under control of the Executive. It provides for direct implementation of many of the functions of the DAIS flight mission, such as navigation, weapon delivery, equipment interfacing, steering, stores management, and mission sequencing. Two of the major functional parts of this software are briefly summarized in the paragraphs below.

First, the OFP Applications Software provides several selectable navigation modes[17]. These can be manually selected by the pilot, or automatically sequenced based on the best available backup modes when the primary equipments fail. The major modes provided are: inertial, air data/heading and attitude reference system (HARS), air data/magnetic compass, and area navigation using TACAN

Second, the OFP Applications Software provides several manually selectable weapon delivery modes. These are: continuously computed impact point (CCIP), angle rate bombing system (ARBs), navigation bombing, radar bombing, manual bombing, and air-to-air gunnery.

c. OTP Applications Software: The OTP Applications Software has not yet been designed or implemented for DAIS, but it is envisioned to operate in a fashion similar to OTPs for other aircraft systems. It will execute in the DAIS airborne computers under control of the Executive, much the same as the OFP Applications Software. The primary function of the OTP is the detection and isolation of avionic system failures, and it is to be controlled by a test technician via the cockpit controls and displays

The baseline design requirements for the OTP includes[18] the following:

- To be organized into pre-flight, inflight, and post-flight (maintenance) testing.
- To provide a maintenance test capability with the ability to isolate failures to the LRU level.

- To collect subsystem status information from subsystem self-tests (e.g., built-in test equipment) and display this information to the test technician.

## 3. NON-REAL-TIME SUPPORT SOFTWARE

The NRTSS items for DAIS consist of a collection of off-line support aids used for the design, development, and integration of the DAIS Mission Software. NRTSS consists of language translators, simulators, management aids, and other such tools. These items are briefly discussed in the following paragraphs.

a. JOVIAL-73 Compiler: The JOVIAL-73 language was developed in the early 1970s, and became an Air Force wide standard in 1976 with the publication of DoD Instruction 5000.31 and MIL-STD-1589. The prime motivation for the development of JOVIAL-73 was the Air Force's desire to have a common, powerful, and easily understandable HOL suitable for a wide-range of Air Force applications[15]. It was designed as a procedure-oriented language, to be relatively computer independent, and with the power of expression in logical operations and symbol manipulation, as well as numerical computation.

The language was designed by a committee of both industry and Air Force representatives, and was primarily targeted to the replacement of the previous JOVIAL-3 language standard (AFM 100-24). The specific application areas targeted by JOVIAL-73 were:

- Avionics Systems
- Executive Writing
- Scientific Programming
- Tactical Systems
- Compiler Writing

- Data Management Systems
- Command and Control
  Real-Time Control

For the DAIS Project the JOVIAL-73 standard was adopted and successfully used in many software applications. The major applications were:

- The Mission Software Executive and Applications.
- The SDVS system, exclusive of existing software modules that were included.
- The JOVIAL-73 compiler itself.
- The PALEFAC support software.

The compiler developed for DAIS was the first "production quality" JOVIAL-73 compiler, and it was hosted on the DEC-10 computer. It was designed to produce object code for two computers: the DEC-10 and the DAIS airborne computer (AN/AYK-15). This compiler produces highly efficient code, is modular in construction, and can be retargeted or rehosted for other applications.

b. Cross-Assembler: For use with the DAIS AN/AYK-15 (hot bench) computer, a Cross-Assembler was developed on the DEC-10 support computer. This Cross-Assembler accepts assembly language statements and produces relocatable object code for the AN/AYK-15 computer. The Cross-Assembler is written in a standard FORTRAN-IV language, and it is designed to be easily retargetable to other target computers.

c. Linker-Loader: A Linker-Loader is also provided on the DEC-10 for the AN/AYK-15 computer. This program accepts relocatable object files (from either the JOVIAL-73 Compiler or the Cross-Assembler) and produces load tapes for the AN/AYK-15 airborne

computers. This program is also written in the FORTRAN-IV language.

d. PALEFAC: Due to the federated computer approach and multiplex bus structure used in the DAIS architecture, the DAIS Mission Software Executive contains much of the inherent design flexibility of the DAIS avionics configuration. As such, the Executive was designed so that this system flexibility could be easily exploited. This Executive is a powerful, general purpose "tool" for use by the applications programmers and can be readily tailored by changing the data table structures which are designed into the executive algorithms. These tables control all data bus traffic, the scheduling and execution of tasks within all airborne computers, and the sequencing of events when anomalous conditions occur.

Due to the large number and critical nature of the interrelationships of these executive data tables, the PALEFAC software was developed to automatically generate these tables. In essence, the programmer is relieved of this laborious and error-prone data generation task.

e. SDVS: The Software Design & Verification System (SDVS) is an integrated set of software tools specifically designed to aid in the development, coding, and testing of the DAIS Mission Software. The SDVS software was developed on the DEC-10 computer and is largely programmed in JOVIAL-73. Through SDVS, an applications programmer has available a powerful support software system.

which provides the following major capabilities:

- The programmers and software project managers are provided with automated tools to control and monitor the application software development, and configuration management aids to control software versions and changes.

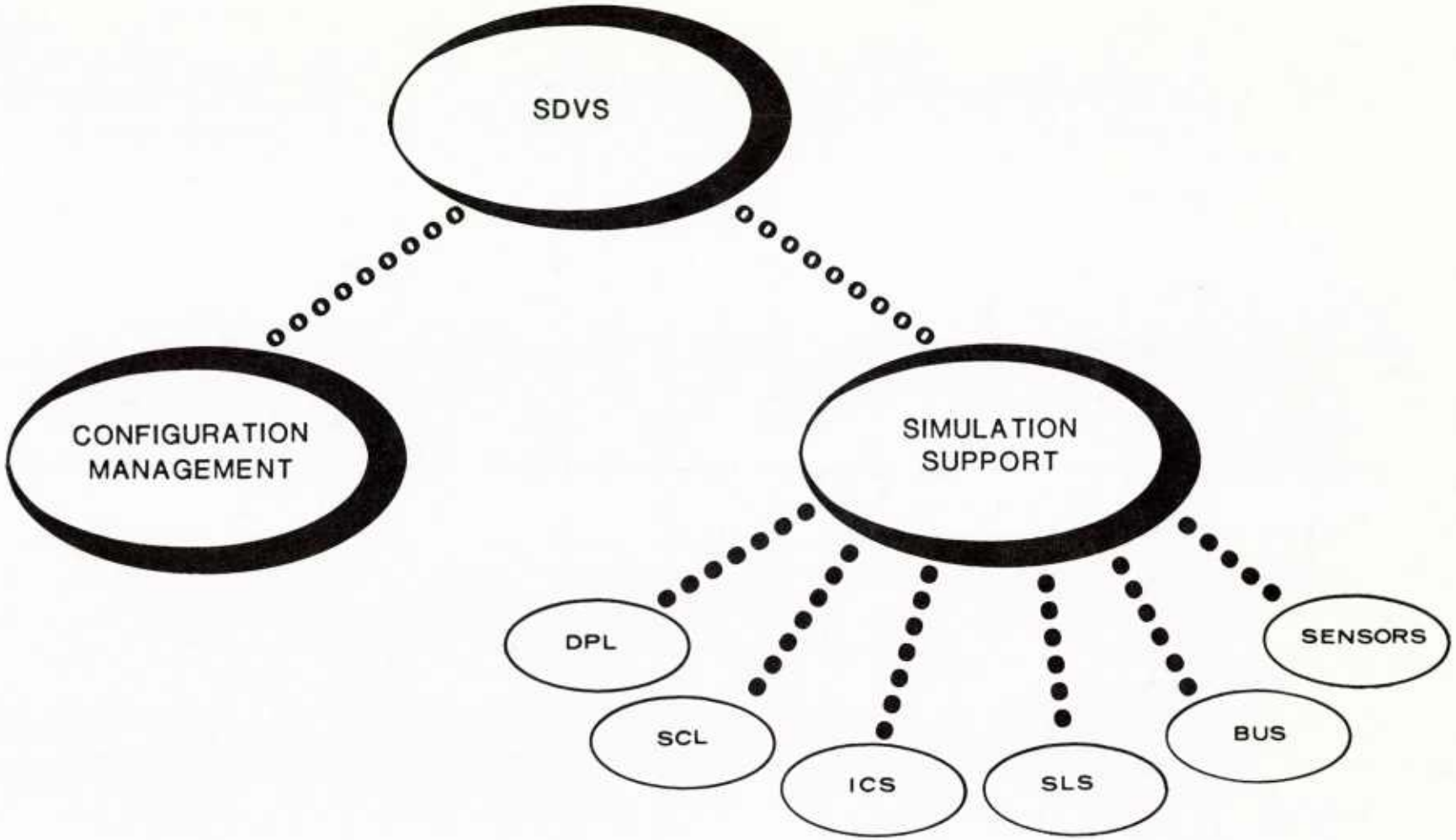
- The programmer is provided with an extensive simulation capability, including airborne computer simulators, a simulation control language (SCL), a simulation data processing language (DPL), a simulation of the airborne MIL-STD-1553 data bus and associated multiplex terminals, and simulations of aircraft sensors connected to the DAIS data bus. The airborne computers are simulated at both a functional level (SLS) and a register level (ICS).

Figure V-3 shows the relationship of these SDVS capabilities. Although SDVS was designed specifically to support the DAIS ITB hardware and software, it was designed such that other system architectures and designs could also be supported. The nature of this adaptability is further addressed in Section VI.

4. REAL-TIME SUPPORT SOFTWARE

The RTSS items consist of those real-time support software tools needed to assist in the integration and operation of the DAIS ITB. These items are centered around two functional requirements: test control and real-time simulation (environment simulation and sensor simulation). These are briefly discussed below.

a. Test Control Software: This software executes on the DAIS Support Facility computers (DEC-10 and PDP-11s) and provides the test set-up and control required for operation of the DAIS ITB. This software provides the following major functional capabilities:




SDVS FUNCTIONAL CAPABILITIES

Figure Number V-3

Test operator interfaces

Test run initialization and initiation

Test monitor and control

Data recording and analysis

b. Environment Simulations: In order to provide a simulated flight environment in which the DAIS hot bench can "fly", the aircraft flight environment must be simulated. The Environment Simulations provide this by modeling (in real-time) the aircraft flight dynamics, the earth and atmospheric effects, and selected battle engagement conditions (e.g., targets, threats, etc.).

c. Sensor Simulations: For the ITB constructed in AFAL, many of the aircraft sensors are not actually included as hardware items. Instead, these sensors are simulated by real-time models to provide complete functional replacements as well as bit-level interface compatibility. Currently, the major sensor simulations provided are: inertial navigation unit, laser ranger, instrument landing system, radar altimeter, and TACAN.

## SECTION VI

## IDENTIFICATION OF DAIS SOFTWARE ITEMS APPLICABLE TO MATE

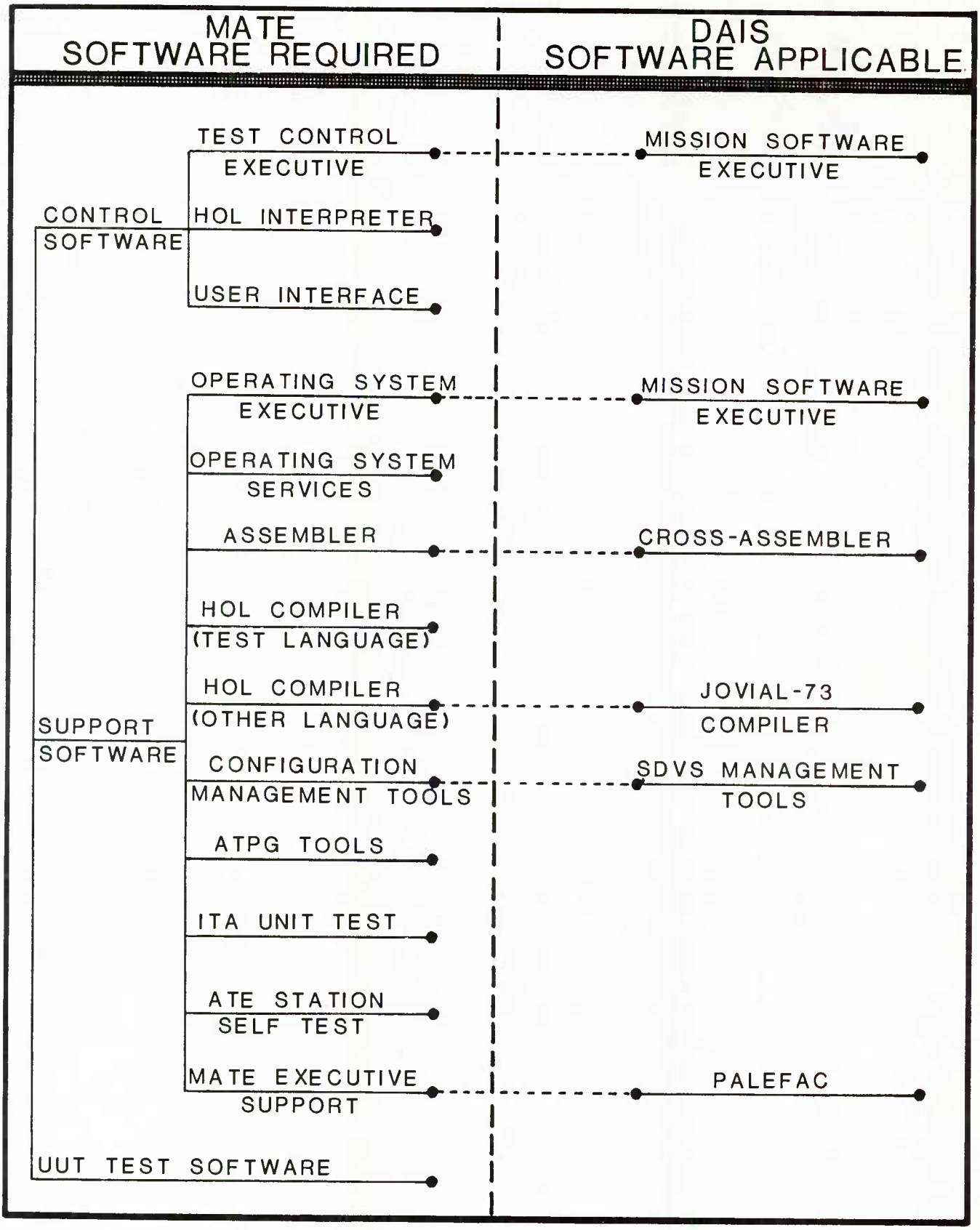
### 1. SECTION OVERVIEW

From the analysis done of the MATE software requirements, there are several areas (see Fig. VI-1) in which the DAIS software items can indeed satisfy these MATE requirements. These are identified and discussed further in this section. However, it should be noted that some of these recommendations are very sensitive to the final system architecture chosen for MATE. For example, if the final MATE system architecture does not use a data bus structure, then some of the DAIS software items would not be applicable. Also should MATE not choose to adopt the JOVIAL-73 language standard, then other DAIS software items might not be applicable.

### 2. MISSION SOFTWARE EXECUTIVE

There are two areas in which the DAIS Mission Software Executive would fulfill MATE requirements. These are for the Test Control Executive and the Operating System Executive functions. As discussed in Section V, the DAIS Executive structure is very flexible and general purpose in nature and was designed to fulfill the requirements of a real-time operating system. It has features included that allow for both synchronous task execution as well as data bus communications. It also supports a multi-computer processing architecture that can be

Figure Number VI-1




DAIS SOFTWARE APPLICABILITY

from one to several minicomputers depending on the processing system throughput required.

a. Test Control Executive: Due to the real-time nature of this ATE software, the DAIS Mission Software Executive is particularly well suited for this application. The major areas of commonality are summarized below:

- The MATE Test Executive must accomodate a distributed data bus architecture which will interface to a multitude of various test equipments. Thus, the Test Executive must be highly modular to allow for easily adding new I/O interface modules. Also, the Test Executive must support a variety of I/O timing requirements, some of which will be synchronous (i.e., cyclic) and some of which will be asynchronous (i.e., interrupt driven). The DAIS Executive was designed to support a distributed data bus communications architecture using MIL-STD-1553. The I/O interfaces to the Executive are highly modular, and new software I/O modules can be easily added for interfacing new units to the data bus. The DAIS Executive also provides for various methods of both synchronous and asynchronous communications with devices connected to the data bus.

- This MATE Test Executive should support both real-time (foreground tasks) and non-real-time (background) tasks for optimal use of the hardware resources. For example, the test operator must be able to communicate with the ATE station to control and monitor testing (typically a background task) while the UUT Test Software is executing (a foreground or real-time task). The DAIS Executive provides the needed mechanisms for satisfying these requirements by allowing for multiple levels of task execution priorities. Real-time tasks can be given the highest priorities, thus guaranteeing their execution. Non-real-time tasks can be allocated to the lower priority levels and thus executed only after other tasks have been completed.

- In order to meet the ATE needs of numerous weapon system applications, the MATE System must "expand" or "contract" by adding or deleting hardware components. This also requires that various levels of computing capacity be available in MATE, probably through the use of either a computer "family" approach or through the use of multiple copies of a single type of minicomputer. The MATE Test Executive must then be capable of supporting such a multiple computer configuration. The current DAIS Executive was designed to support a configuration of multiple, federated computers communicating over a data bus. Thus for avionics applications requiring only minimal amounts of computation, the DAIS Executive and Applications software is used with a single airborne computer. For applications requiring increased computing capacity, the DAIS Executive can be reconfigured to control a federated set of multiple computers, each sharing the total computation load.

- Since MATE will probably utilize hardware from multiple computer vendors, the MATE software should be relatively independent of the computer hardware in which it operates. This requirement for "portability" can best be attained through two design mechanisms: (1) the use of a HOL for programming, and (2) use of software design techniques that do not "lock in" the software to one particular vendor's hardware. The DAIS Executive has been successful in meeting both of these design goals, and thus it could be tailored to execute in a variety of vendors' computers. This Executive is coded in the JOVIAL-73 HOL and could be hosted on any computer for which a JOVIAL-73 compiler exists. However, this approach does require the development of a JOVIAL-73 compiler for each new vendor's computer.

- The MATE System will undoubtably be designed to include hardware redundancy in the more critical areas. Such an approach serves to increase the total ATE System availability. The MATE Executive must, in turn, include redundancy management techniques and graceful

degradation concepts to make optimal utilization of the redundant hardware items. In its current design, the DAIS Executive accomodates the following redundant hardware items; data buses, I/O devices, and computers. Various failure modes can be completely recovered from and others can be managed in gracefully degrading modes.

b. Operating System Executive: The MATE Operating System Executive should support all of the normal batch and time share services of a modern day minicomputer operating system. Although the DAIS Executive does not currently provide these broad capabilities, it could be expanded to fully support this area. The fundamental structure of the current DAIS Executive provides a good basic framework to which the required additional features could be added.

The use of the DAIS Executive is particularly attractive for the MATE Operating System if it is also used for the Test Control Executive. This would allow for a single and common executive to be used for both executive areas and would reduce software maintenance costs over the use of two separate executives.

3. CROSS-ASSEMBLER

Most minicomputers considered for use in MATE would probably be supplied with a standard assembly language and Assembler software package. However, in some instances (e.g., where the vendor's Assembler is severely limited in capability or difficult to use) there may be a need to develop an alternate Assembler for use in MATE. For these instances the DAIS Cross-Assembler deserves close consideration as a candidate. It is a modular, table-driven assembler which was specifically designed to be easily

retargeted or rehosted for other computers. It is coded in the FORTRAN-IV language, and it can be tailored to produce object code for the computer on which it executes or for another computer (i.e., as a cross-assembler).

4. JOVIAL-73 COMPILER

For supporting the coding and development of all of the MATE Software (except the UUT Test Software), a HOL will be needed. A prime candidate for this HOL is JOVIAL-73 which has been adopted as an Air Force-wide standard. The JOVIAL-73 language could support virtually all of the MATE software including the executive programs.

The DAIS Program was one of the first major applications of JOVIAL-73 by the Air Force, and it has proven to be a highly successful application. Some of the more notable experiences of this application in DAIS were:

- JOVIAL-73 was used as a "standard" language within the DAIS Program and was applied to both support software items and real-time flight software items.
- JOVIAL-73 proved to be an efficient language that could cost effectively compete with assembly language implementations.
- JOVIAL-73 was successfully used to code executive software programs. For example, virtually all of the DAIS Mission Software Executive was coded in the JOVIAL-73 language.

If the JOVIAL-73 language is also adopted as a MATE language, the DAIS JOVIAL-73 compiler could be tailored for use on MATE. The DAIS Compiler was constructed such that it could be easily

rehosted and retargeted, and this retargeting has been successfully demonstrated for several other projects. The compiler, itself, is coded in the JOVIAL-73 language and can be bootstrapped to other host computers. Also, the code generator sections of the compiler have been well isolated from the other sections, thus simplifying the compiler retargeting tasks.

## 5. SDVS MANAGEMENT TOOLS

As described in Section V, the SDVS Software provides the DAIS Mission Software developers with two major functional capabilities. The first capability provides non-real-time simulation tools for the purpose of testing the Mission Software. This set of tools is tied very closely to the DAIS Mission Software environment, to the DAIS hardware architecture, to the use of a large scale computer system, and to the JOVIAL-73 HOL. Thus for a MATE System the simulation capabilities of SDVS would be of marginal, if any, help.

The second major SDVS capability is centered on management support tools. In contrast, these tools are not closely tied to the DAIS architecture or to the DAIS Mission Software environment. These tools are general management aids which can be used to support the configuration management of software modules (e.g., MATE UUT Test Software modules).

Currently SDVS supports configuration management by providing control of all files associated with the development, test, and

verification of DAIS Mission Software. An extensive cataloging and security system is provided for a number of different types of software maintained by SDVS including: DAIS Mission Software, SDVS test case files (defining simulation scenarios and data collection requirements), environment and aircraft models, and post simulation data reduction and analysis programs. Each file type is cataloged by SDVS on a version/revision basis. When a Mission Software file is created, it is cataloged as version I, revision 0 and stored in a "baseline file". As the user edits the file he creates a number of revisions which are cataloged in a "difference file". At any point in time, he can combine all the revisions associated with a particular version and make a new version. Once a user creates a source file, he can compile or assemble that file to generate executable code. This function automatically results in a link being established between the new object code and the original source file. For Mission Software, the JOVIAL-73 Compiler generates code for both the DAIS processor and the DEC-10 computer, and SDVS catalogs and links both object types to the same source file. The definition of the file catalog information and linking structure is known as the data base schema, which provides catalog links necessary for configuration control of the various SDVS files. The automated linking system, in conjunction with the file catalogs, provides configuration control of all versions, revisions, and translated object code for all software files maintained in SDVS.

In addition to maintaining the file cataloging system, SDVS also provides file protection features that prevent unauthorized access to files. A special SUPERVISOR mode is provided in which the file security requirements can be defined only by a special SDVS data base supervisor. In the SUPERVISOR mode, the data base supervisor performs a function to define the files that can be created by the user and therefore cataloged by the Configuration Management function. An SDVS user will be prohibited from creating a file for which specifications have not been provided. For each file, the supervisor also enters a list of SDVS users who will be authorized to have read only, or read and write privileges. The supervisor can change a user's access and add or delete user authorization for a file at any time. This capability enables a data base administrator to distinguish "controlled" software from "developmental" software and control the transition from the latter to the former. Access to controlled software can be limited such that programmers desiring to make a "trivial" change (that they are "positive will word") will be unable to, until and unless they have proper authority. In summary, the SDVS configuration management function provides a supervisory control and file security for all software files in SDVS.

These SDVS configuration management tools could be applied to MATE to satisfy some of the ATE Software requirements. However, the exact applicability can only be determined after the

ATE System architecture is chosen and after software management procedures are selected for use on MATE. Some of the major items to be considered at that time are: (1) the portions of SDVS required by MATE would have to be restructured to execute in a minicomputer environment, (2) the current DEC-10 version of SDVS depends heavily on a DEC-10 resident data base management system and this would not be available on another computer, (3) SDVS is coded in the JOVIAL-73 language, and the MATE minicomputer would be required to have a JOVIAL-73 compiler, and (4) SDVS would have to be modified to use other compilers (e.g., ATLAS).

6. PALEFAC

As discussed previously, the DAIS Mission Software Executive is a modular, table-driven executive that supports a MIL-STD-1553 data bus. It is designed to be easily reconfigured to adapt to changes in hardware units connected to this data bus. For this, executive PALEFAC functions as an integral part of the DAIS Executive in a non-real-time mode. It acts as a "table builder" which generates both bus I/O lists and task execution tables.

PALEFAC will be applicable (and required) for MATE if the DAIS Mission Software Executive is used. It is currently coded in the JOVIAL-73 language and executes on the DEC-10 computer, but it could be easily rehosted to another computer which supports a JOVIAL-73 compiler.

# SECTION VII

## SUMMARY AND RECOMMENDATIONS

This report presents an overview of the three areas of ATE software; ATE software of current-day systems, additional ATE software requirements beyond those satisfied with current-day ATE software, and MATE-unique software requirements. Using these as a basis for comparison, the DAIS software was analyzed for applicability to the MATE system. The results of this analysis was presented in Section VI.

As noted earlier in this report, the recommendations contained in Section VI are very sensitive to the final MATE system architecture, which was not defined at the time of the writing of this report. Thus, a "postulated" MATE architecture (Section IV) was developed and used throughout the analysis effort. Should the final MATE system design differ significantly from that discussed in Section IV, then the validity of the recommendations could be severely affected.

In summary, the DAIS and MATE projects do have much in common at the level of the programs' objectives. If MATE does, indeed, evolve into a system architecture similar to that of DAIS, then there will be many areas of potential commonality in both hardware and software.

# APPENDIX A

## GLOSSARY OF SELECTED TERMS

This appendix contains a glossary of selected terms and acronyms used in this report. These items are included to provide the reader with a single reference point for the most frequently used and most important items. Many of these items are defined as they appear in MIL-STD-1309B,[5] however a few definitions have been expanded for clarity and/or qualification.

Assembly Language: A computer programming language in which computer operations and memory locations are represented by mnemonic symbols.

ATE (Automatic Test Equipment): Equipment that is designed to conduct analysis of functional or static parameters to evaluate the degree of performance degredation and may be designed to perform fault isolation of unit malfunctions. The decision making, control, or evaluation functions are conducted with minimum reliance on human intervention.

ATE Control Software: Software used during the execution of a test program which controls the nontesting operations of the ATE. This software is used to execute the test procedures but does not contain any of the stimuli or measurement parameters used in testing the Unit Under Test (UUT).

ATE Support Software: Computer programs which aid in preparing, analyzing, and maintaining ATE software. Examples are; ATE compilers, assemblers, debugging aids, and text editors.

ATPG (Automatic Test Program Generation): The process of using software to automatically generate UUT Test Software from formal descriptions of the UUT circuitry.

Compiler: A computer program used as an automatic means of translating High Order Language (HOL) statements into computer hardware instructions.

DAIS (Digital Avionics Information System): A project sponsored by the Air Force Avionics Laboratory and targeted to demonstrate a coherent solution to the problem of proliferation and non-standarization of aircraft avionics.

HOL (Higher Order Language): A computer programming language which provides the programmer with a more natural (either problem oriented or English like) medium of expression than is available in computer assembly languages.

Interface Test Adaptor (ITA): A device or series of devices designed to provide a compatible connection between the Unit Under Test and the ATE Test Station.

LRU (Line Replacable Unit): A unit which is designated by the plan for maintenance to be removed upon failure from a larger entity (equipment or system) in the latter's operational environment.

MATE (Modular Automatic Test Equipment): A project sponsored by the Air Force Aeronautical Systems Division and targeted to increase commonality of test equipments across aircraft system.

PCB (Printed Circuit Board): A board for mounting electronic componenets on which most connections are made by printed circuitry.

TRD (Test Requirements Document): The document that specifies the test sequences and test conditions required to test and fault isolate a Unit Under Test (UUT).

UUT Test Package: A UUT Test Pakcage is a collection of unique items for each UUT and consists of three major items; the UUT Test Software, and ITA, and the associated documentation.

UUT Test Software: Software which specifies the sequence of operations for the testing of a particular UUT.

UUT (Unit Under Test): Any system, set, subsystem, assembly, subassembly, and so forth, undergoing testing.

## REFERENCES


1. D. Kline, Software Automation - The Next Step in Weapons Support, Hewlett-Packard technical paper, 1976.

2. J. F. Stay, HIPO and Interactive Program Design, IBM Systems Journal.

3. F. Liguori, Automatic Test Equipment Systems, Programming, and Management, 1973 partial manuscript of unpublished book.

4. A. Greenspan, What is Third Generation ATE?, Notes from course entitled "Commercial Automatic Testing Systems", 1977.

5. Definitions of Terms for Test, Measurement and Diagnostic Equipment, MIL-STD-1309B, 30 May 1975.

6. R. M. Earnest and R. Blowers, Configuration Management of ATE Test Software, NAECON Proceedings, May 1978.

7. D. Day, The Automatic Test Equipment Test Package Development Process, NAECON Proceedings, May 1978.

8. Simulator-Based Interactive Software Aids to Test Generation, General Radio, System Test Division Application Note 3.

9. Automatic Test Equipment: Hardware, Software, and Management, IEEE Press, 1974.

10. F-16 Multinational Computer Resources Integrated Support Plan (CRISP), F-16-1001, Vol II, Rev. O, April 1978.

11. J. J. Donovan, Systems Programming, McGraw-Hill, 1972.

12. M. Hollowich & M. McClimens, Software Design & Verification System, AFAL-TR-76-200, May 1977.

13. Technical Description of the Digital Avionics Information System (DAIS), DAIS Document No. PA100101, Feb. 1978.

14. Computer Program Development Specification for Operational Flight Program Applications Software, DAIS Mission A, DAIS Document No. SA201303 Part I, July 1976.

## REFERENCES (cont'd)


15. L. Trainor and M. Grove, Higher Order Language Standardization for Avionics, NAECON Proceedings, May 1977.

16. M. Hollowich and F. Borasz, The Software Design & Verification System: An Integrated Set of Software Development and Management Tools, NAECON Proceedings, May 1976.

17. System Segment Specification for the Digital Avionics Information System, DAIS Document No. SA100102, July 1977.

18. System Segment Specification for the DAIS Air-to-Ground Missions, DAIS Document No. SA100101, Oct. 1975.




☆U.S.Government Printing Office: 1979 — 657-002/631