NBSIR 76-1094(R)

Standards for Computer Aided Manufacturing

John M. Evans, Jr., Ph.D., Project Manager
Joseph T. O'Neill
John L. Little
George E. Clark, Ph.D.
James S. Albus, Ph.D.
Anthony J. Barbera, Ph.D.
Bradford M. Smith
Dennis W. Fife, Ph.D.
Elizabeth N. Fong
David E. Gilsinn, Ph.D.
Frances E. Holberton
Brian G. Lucas, Ph.D.
Gordon E. Lyon, Ph.D.
Beatrice A. S. Marron
Mable V. Vickers
Justin C. Walker

Office of Developmental Automation and Control Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D. C. 20234

Third Interim Report

January, 1977

Prepared for

Manufacturing Technology Division
Air Force Materials Laboratory
Wright-Patterson Air Force Base, Ohio 45433

NBSIR 76-1094(R)

STANDARDS FOR COMPUTER AIDED MANUFACTURING

John M. Evans, Jr., Ph.D., Project Manager
Joseph T. O'Neill
John L. Little
George E. Clark, Ph.D.
James S. Albus, Ph.D.
Anthony J. Barbera, Ph.D.
Bradford M. Smith
Dennis W. Fife, Ph.D.
Elizabeth N. Fong
David E. Gilsinn, Ph.D.
Frances E. Holberton
Brian G. Lucas, Ph.D.
Gordon E. Lyon, Ph.D.
Beatrice A.S. Marron
Mabel V.Vickers
Justin C. Walker

Office of Developmental Automation and Control Technology
Institute for Computer Science and Technology
National Bureau of Standards
Washington, D. C. 20234

Third Interim Report

January, 1977

Prepared for
Manufacturing Technology Division
Air Force Materials Laboratory
Wright-Patterson Air Force Base, Ohio 45433

U.S. DEPARTMENT OF COMMERCE, Juanita M. Kreps, *Secretary*

Dr. Betsy Ancker-Johnson, *Assistant Secretary for Science and Technology*

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Acting Director*

INTRODUCTION

The Air Force is initiating a major new program to accelerate the establishment of Integrated Computer Aided Manufacturing (ICAM) in discrete part batch manufacturing industries in the United States, especially in the aerospace industry. The National Bureau of Standards is providing support to that program by analyzing existing standards relevant to Integrated Computer Aided Manufacturing.

This document is the third interim report to the Air Force Manufacturing Technology Division of the Air Force Materials Laboratory at Wright-Patterson Air Force Base on the ICAM support project. This report covers task 4 of the 5 tasks of the project:

Task 1 Identify current standards applicable to CAM.
Task 2 Analyze existing formal and de facto standards.
Task 3 Assess the actual usage of standards in industry.
Task 4 Recommend optimal standards for CAM system development.
Task 5 Identify standards organizations and outline a proper Air Force role in standards activities.

The first report identified those existing and potential standards which will be useful to the Air Force in the development and implementation of integrated computer aided manufacturing systems. Such systems, when implemented by the Air Force and by Air Force contractors, will increase productivity in discrete part batch manufacturing by several thousand percent.

The second report provided a comprehensive reference data base on all formal and de facto standards that are considered to be relevant to the Air Force Program. The report took the form of an annotated bibliography with data sheets on each standards activity for ease of reference. This report covered Task 2 and 3.

The third report discusses the utility of these standards to the Air Force Program and in each relevant standards area recommends a best approach to follow either toward adopting existing standards or toward developing needed standards. This discussion is embedded in the context of a hypothetical computer based manufacturing environment where standards are shown to play an essential role in three key areas:

System integration: data and communication interfaces between CAM application programs.

Software portability: interfaces between CAM programs and the host computer system, including languages, operating systems, and data based management system interfaces.

Integration of distributed systems: interfaces between computers in distributed systems.

The work reported here was supported by the Air Force Program for Integrated Computer Aided Manufacturing, Manufacturing Technology Division, Air Force Materials Laboratory, Wright-Patterson Air Force Base under MIPR FY 14577600369, Dennis Wisnosky, Program Manager.

SUMMARY OF RECOMMENDATIONS

This third interim report recommends optimal standards for the Air Force ICAM program. Many formal standards that now exist are recommended as relevant to the ICAM program and these are expected to remain so in the future. Futhermore, trends and developments in standardization process are enumerated with key areas identified for monitoring or development. Finally, several areas are cited where formal standards do not exist but where project standards will be necessary.

This report and the recommendations of this report should be considered as a first step in an interactive effort to define the nature, the detailed structure, and the details of implementation of ICAM projects.

Evaluation and Recommendations on CAM Standards

There are few CAM standards that can be evaluated, a result partly of the newness of the field and partly of the way in which CAM systems have been developed primarily by large user industries for their own internal (and hence proprietary) use. In the area of NC programming languages, standards have developed because of the multi-industry development effort and Air Force contractual requirements. The APT language standard is recommended as a minimum, with extensions to cover adequately the post processor area. If two languages can be allowed, the COMPACT II/ACTION/SPLIT family should also be used since it is more efficient on simple parts and can product compatible CL data as an option.

In the CAD/CAM interface area, the ANSI Y14.26.1 effort, NASA's IPAD, and the CAM-I Geometric Modeling Project are considered in relation to the digital representation of physical object shapes. The ANSI approach is recommended, and the Air Force is advised to monitor the other efforts to insure eventual compatibility with the ICAM system. In addition, the Institute for Printed Circuits standard on printed circuit boards is discussed as a tutorial example and recommended where appropriate.

Evaluation and Recommendations on Computer Standards

Communications

In order to construct distributed fourth generation computer systems expected to be in wide use in the 1980's, adequate communications interface standards are a necessity. A suprising amount has been dome on hardware standards and comminucations protocols. A comprehensive set of standards, some of which are only in the formation stages, is recommended on computer peripherals (ANSI proposed channel level and minicomputer device level interfaces), DTE/DCE interfaces (RS 232, RS XYZ, CCITT X.21), and bit oriented link level and packet network protocols (ANSI ADCCP and CCITT Recommendation X.25). Following these standards, a distributed computer system can be developed, using commercial communication services, that will remain relevant into the 1980's.

Codes

The lowest level of information storage and transmission is the character code level. Serious problems may arise in code conversion and in accessing or merging files with different coding schemes. These problems are discussed and the American Standard Code for Information Interchange (ASCII) code is recommended for data crossing any system interface.

Software: Languages, Data Base Management, and Operating Systems

The Air Force has stated that their objectives for the ICAM system include software portability, integration of software modules and, potentially, distributed data processing. These requirements lead to a consistent set of recommendations for programming languages, data base management, and operating systems.

Standardized programming languages offer the key to portable software. Using adequate language standards and requiring validation of compilers against those standards will be required for Air Force ICAM software to be portable. FORTRAN and COBOL will have to be supported to the near term because of the bulk of application programs written in those languages. Eventual conversion to the use of a more modern programming language should be anticipated. Representative of the "modern" languages is PL/I which is the only one that has been submitted for standardization. However, substantial effort remains before PL/I can be termed suitable for ICAM needs.

From the point of view of integration of applications modules, the most critical element is the data base management system (DBMS). The recommendation here is to prepare functional specifications for the competetive procurement of a commercially available data base software package to support all near term ICAM projects. Emphasis should be placed upon obtaining modular architecture, well defined interfaces, portability of applications programs, integration of ICAM modules, and future adaptability to a computer network system with distributed data bases.

In Operating Systems there are no standards. This is a major problem area from the point of view of software portability, but a standard operating system is not feasible for large scale computers because of the differences in architectures. However, a standard operating system for 16 bit or 32 bit byte oriented machines seems at least technically feasible. It may be necessary to implement project standards on file names and library names to avoid problems in portability due to differences in file management conventions.

Documentation, Validation and Testing, and Software Tools

These are some of the most important tools to insure system integration and software portability and maintainability. Detailed recommendations are not possible until the maintenance of ICAM software is better defined, but general requirements and various approaches are discussed and evaluated, and general guidelines are provided. It is recommended that the Air Force use validation and testing procedures, and that general software development tools be developed, used in ICAM development, and then made a part of the ICAM system.

Media

Magnetic Tape and discs and direct communication links are the primary recommended standards for transmitting ICAM data and software within a given installation and between installations. Where punched cards must be used, standards are available. Paper tape, even for NC, is not recommended; instead direct communications links (DNC) should be used.

Summary Matrix

There is no comprehensive set of present day standards that will solve all of the Air Force's needs. However, where today's standards are inadequate, major trends, developments and needs for project standards have been identified that should provide the Air Force ICAM program with sound initial guidance. A summary of recommendations is given in the matrix of Figure 1.

Standard Cases	Major Standards	Recommendations	Further Efforts Required
CAM STANDARDS			
1. NC Languages	APT COMPACT II	Use both languages, require CLDATA as interface	Standardize postprocessors MonitorX3J5, X3J7 for future developments
2. CAD/CAM	ANSI Physical Object Description (Y14.26.1) NASA IPAD CAM-I IPC Printed Circuit Standard	Use ANSI Y14.26.1, IPC standard where appropriate	Monitor IPAD, CAM-I, resolve any incompatibilities that develop
COMPUTER STANDARDS			
1. Computer and Communications Interface Standards	EIA RS232, RSXYZ (data terminal to analog communications) CCITT X.21 (data terminal to digital communication) Peripheral Interface Standards ANSI ADCCP Link Level Protocol ANSI X.25 Packet Network Protocol	Use RSXYZ or X.21 for systems in 1980's; use peripheral interface standards as adopted; Work with commercial networks for distributed systems	Monitor and evaluate further development of link level and network level protocols Develop host-to-host level protocols for distributed systems
2. Communication Codes	ANSI X3.4 (ASCII) IBM EBCDIC Encryption Algorithm	Use ASCII at all system interfaces and for collating; use encryption algorithm if security required	
3. Programming Languages	FORTRAN COBOL PL/I BASIC	Support for FORTRAN and COBOL must be provided for near term PL/I is best of existing standard languages for ICAM with extensions and subsets (PL/S, PL/C, PL/M)	If PL/I is used, the availability of PL/S must be investigated and standard subsets of PL/I developed. DoD-1 should be investigated for possible use by ICAM Validate compilers and run time support routines to insure portability

4. Operating Systems	No standards	No formal standards are likely	Develop project standards on certain aspects of operating systems, particularly system calls and file names Consider development of standard operating system for 16 bit minicomputers and cross system support for microcomputers
5. Data Base Management Systems	No standards	Use network or relational data structures for 1980's system	A DBMS should be developed for ICAM based on existing software as much as possible; Data integrity and security should be considered in early design stages Develop project standards for DBMS interfaces and use
6. Software Testing and Tools	COBOL, FORTRAN validators	Use compiler Validation Use dynamic and static analyzers	Develop Validation for future languages Develop test for accuracy of mathematical software Develop acceptance tests for each major software package Develop and implement software development tools early in program
7. Documentation	FIPS PUB 30 FIPS PUB 38 CAM-I Standards	Use FIPS PUB 38 and CAM-I to formulate specific ICAM documentation guidelines.	Develop project standards for documentation Appoint a documents administrator to specify and maintain system design, documentation standards, and documented software
8. Media	Magnetic Tape (X3.40, X3.14, X3.22) Paper Tape (X3.29, X3.18) Punched Cards (X3.11, X3.21)	Use magnetic tape as primary media; use ANSI standards If cards are used, use ANSI standards Paper tape not recommended; for NC systems an on line DNC configuration is recommended	Clarify DNC system interface in cooperation with EIA Committee IE-31

STANDARDS FOR COMPUTER AIDED MANUFACTURING

1. NC PART PROGRAMMING LANGUAGES

BACKGROUND

APT STANDARDIZATION

COMPACT II STANDARDIZATION

COMPARISON OF NC LANGUAGES

CRITERIA FOR NC LANGUAGE SELECTION

RECOMMENDATIONS

COMMENTS

REFERENCES

BACKGROUND

An NC machine tool accepts commands from a punched paper tape or from a computer to control the operations of that tool. These control signals are strings of bit patterns that are decoded by the tool into the proper locations, movements, and actions, to produce the desired part. Following a standard code for the different control signals, an operator can punch these values into a paper tape. Simply rerunning the paper tape into the NC tool allows the tool to produce automatically as many parts as desired while the operator is free to do other jobs.

For simple parts, and originally for all parts, the coding of the control tape is carried out directly according to the EIA standard tape formats. (RS-247C with RS-358 character code.)

As the parts to be made become more complicated, the programming becomes much more involved. Higher level NC programming languages have been developed for these more sophisticated cutting operations. These languages are typically made up on a number of English-like commands which are translated by a computer program (processor) into either the proper bit pattern for a particular NC machine tool, or into an intermediate machine-tool-independent data file (Cutter Location Data (CLDATA) file). This CLDATA file will then be fed into another computer program called a postprocessor. It is the function of the postprocessor to translate the cutter location data into the appropriate commands for the selected machine tool necessary to machine the desired part. The postprocessor also checks for various error conditions and produces the printed listing to assist the machine operator.

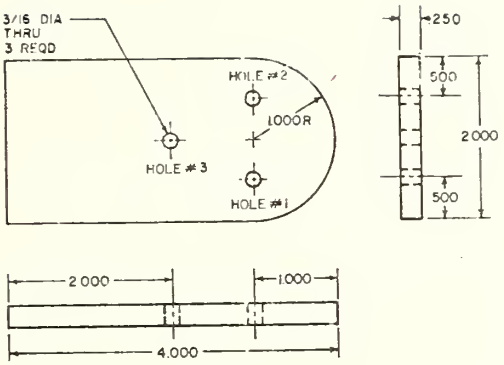
Thus, the CLDATA file is a machine-independent data file that describes in detail the path the cutting tool must follow to make the part. This file is created by a single processor.

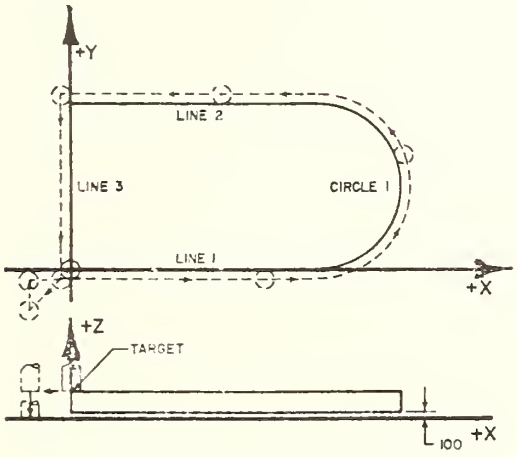
Since the postprocessor is dependent upon both the machine tool and controller, there are as many postprocessors as there are different models of machine tools and controls.

Although there exist over 40 NC programming languages, only two are in widespread, productive use. These are APT (Automatically Programmed Tools), the first of the higher level NC languages, and COMPACT II (COMputer Program for Automatically Controlling Tools). These two languages are representatives of two families of NC language processors. The APT family includes the APT, UNIAPT, and ADAPT processors, while COMPACT represents the COMPACT II, ACTION and SPLIT processors. Both of these language families are proceeding toward formal standardization. An example of each language family is given in the accompanying figures. A simple test part is shown in Figure 1 while the respective part programs are given in Figures 2 and 3.

APT STANDARDIZATION

The revised APT standard (X3.37) (presently undergoing final balloting) is expected in January of 1977. This standard is created and maintained by the American National Standards Institute (ANSI) Committee X3J7 under the Business Equipment Manufacturer's Association (BEMA). The standard is written in a meta-language format which is computer independent. This format gives a complete and vigorous definition of all elements of the language, permissible combinations of these elements, and the meaning of these combinations. While somewhat difficult to read the meta-linguistic format provides a concise and comprehensive technique to itemize all of the combinations and their meanings in a reasonable length document. The new standard will contain the original (X3.37-1974)


3/16 DIA
THRU
3 REQD
HOLE #2
HOLE #3
HOLE #1
1000R
250
500
2000
500
2000
1000
4.000


+Y
LINE 2
LINE 3
CIRCLE 1
LINE 1
+X
+Z
TARGET
100
+X

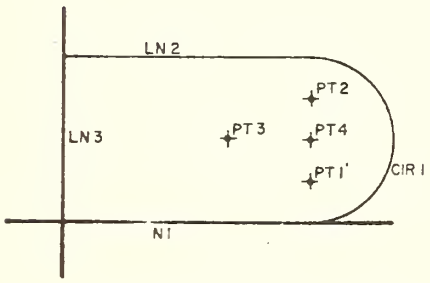

LN 2
PT 2
LN 3
PT 3
PT 4
PT 1'
CIR 1
N 1

Figure 1

```
10 PARTNO TEST1*
15 CLPRNT
20 CUTTER/.25
30 SYN/P,POINT,L,LINE,CI,CIRCLE,GT,GOTO,GF,GOFWD,GD,GODLTA,R,RADIUS,$
40 XA,XAXIS,YA,YAXIS
50 INTOL/.001
60 OUTTOL/.001
70 SP=P/0,0,.35
80 FEDRAT/5
90 L1=L/XA
100 L2=L/PARLEL,L1,YLARGE,2
110 L3=L/YA
120 C1=CI/3,1,1
130 P1=P/3,.5
140 P2=P/3,1.5
150 P3=P/2,1
160 CYCLE/MILL,0,0
170 $$ TARGET AT LEFT BOTTOM CORNER
180 FROM/SP
190 GD/-.5,-.5,0,150
200 GD/0,0,-.35,150
210 GO/TO,L1,150
220 TLRGT,GORGT/L1
230 GF/C1
240 GF/L2
250 GOLFT/L3,PAST,L1
260 $$ CUTTER MOVES TO HOME POSITION AND STOPS. CHANGE TO 3/16 DRILL
270 FEDRAT/.004,IPR
280 CYCLE/DRILL,0,.45
290 GT/P1
300 GT/P2
310 GT/P3
320 CYCLE/OFF,OMIT
330 END
340 FINI
```

Figure 2

```
MACHIN, KTGE7500
IDENT,NCLE TEST #1 COMPACT II
SETUP,EB,INDEX45,-.125LX,.125LY,10LZ,LIMIT(X-10/10,Y-10/10,Z-10/10)
$TARGET AT LEFT BOTTOM CORNER
BASE,XA,YA,ZA
PT1,3XB,.5YB,.35ZB
PT2,3XB,1.5YB,.35ZB
PT3,2XB,1YB,.35ZB
PT4,3XB,1YB,ZB
LN1,YB
LN2,LN1/2YL
LN3,XB
CIR1,PT4,1R
MTCHG,TOOL1,.25TD,5IPM,0GL
MOVE,-.5X,-.5Y
MOVE,-.350Z
MOVE,TOLN1
OCON,CIR1,CCW,S(TANLN1),F(TANLN2)
CUT,PASTLN3
CUT,PASTLN1
HOME,STOP
$CUTTER MOVES TO HOME POSITION AND STOPS. CHANGE TO 3/16 DRILL
MTCHG,TOOL2,(3/16)TD,1800RPM,0GL,.004IPR
DRL,PT1,.25THRU,.1CLEAR
DRL,PT2,.25THRU,.1CLEAR
DRL,PT3,.25THRU,.1CLEAR
HOME,STOP
END
```

Figure 3

standard for the processor, plus updates and corrections in addition to a standard for postprocessor language.

The new standard is unique in its consideration of the postprocessor language. This is the language which enables the control of the non motion functions of a machine tool such as choice of spindle speeds, control of coolant, and selection of cutting tools. Prior to this document guidelines for postprocessor language have been scanty with the result that developers of software programs have had to sometimes choose language syntax themsleves.

As various new hardware or electronic options were developed in the marketplace so were new APT language commands to control them. While the commands for a single function like a tool change are similar among all postprocessors, minor differences exist in each software implementation. These differences have the effect of forcing a part programmer to choose a specific NC machine tool before he starts to develop the APT language to produce the desired item.

The lack of a fully specified APT language tends to defeat the intended universality of the higher level language concept. The design intent of APT was that a part program could be easily processed for any appropriate machine tool through the use of different postprocessors. Increasingly today one finds that not to be the case. Computer runs are aborted for trivial problems such as a command to postprocessor "A" calling for SPINDL/1000, CLW causing an error in postprocessor "B" which requires SPINDL/CLW, RPM, 1000. The revised APT standard is aimed at correcting this problem.

COMPACT II STANDARDIZATION

The COMPACT II/ACTION/SPLIT Standard proposal is currently under consideration by the X3J5 standards committee of CBEMA. SPLIT is the parent language of a group of languages comprising SPLIT, ACTION, COMPACT II in a father/son/grandson relationship. The languages are very similar, but the processors are quite different. It was decided in developing the standard that a standard CL (Cutter Location) output would be optional since this family of languages does not necessarily generate an intermediate data output medium.

The SPLIT processor is machine dependent and does not create an intermediate cutter location (CL) file. The ACTION and COMPACT II processors, however, are machine independent; but they work in conjunction with their respective postprocessors. In this integrated mode, each statement is processed into a CL file statement and then postprocessed by the selected postprocessor into a machine control format before the program moves to the next statement.

ACTION can be run on a minicomputer and in that situation operates in the re-entrant mode - i.e., all the statements in a program are processed and a CL file is generated; then that file is postprocessed to produce the machine control output.

The ANSI committee feels that to make intermediate output (CLDATA file) the mandatory output of the standard would be to deprive the users of many of the inherent economics of the languages. However, the committee is recommending that the intermediate data output be a user or implementor option, and where offered it should conform to the existing CLDATA Standard. The University Computer Corporation (UCC) COMPACT II processor already produces an intermediate data file in accord with the CLDATA requirement of the APT standard.

The long term objectives of the COMPACT II Standards committee are to provide most of the capabilities already present with APT or under research effort. These include working standards for graphic output, for incorporation of machining technology, for programming sculptured surfaces, and for the interface of the NC language to total CAD/CAM systems.

There are presently 1400 installations using the COMPACT II family of languages, representing 6000 NC machine tools. The five year projection (by 1981) estimates 3500 users (20,000 NC machine tools) in the US and 6000 users (40,000 NC machine tools) worldwide.

At present, about 50% of all NC machine tools are being programmed by computer assist. Of these, about 40% are being programmed by COMPACT II family and about 40% by APT with the remainder using the other 40 languages. The five year prediction is for 75% of all NC tools to use computer assisted programming with close to half in COMPACT II and half in APT.

Thus, even though the COMPACT II family is a late entry, (circa 1967 vs. 1950's for APT) it has quickly found widespread acceptance. The main reasons for this are several. The COMPACT II family is less sophisticated than APT and for that reason many users feel that for their more limited requirements that COMPACT is easier to learn. Lathe (2 axis) programming is much more efficient in COMPACT II because of certain language features not available in APT. Lathes represent 40% to 50% of all of the NC tools being shipped. COMPACT II has also been well provided on a time-shared remote service bureau basis by Manufacturing Data Systems Inc. (MDSI) with excellent support.

COMPARISON OF NC LANGUAGES

In March 1974 the Numerical Control Society submitted a final report on the US Army Electronics Command Numerical Control Language Evaluation. This study analyzed seven general purpose NC programming languages and presented data concerning their performance on ten test parts representative of Department of Defense workload. The test parts all of the milling-drilling-boring variety spanned the entire range from 2 axis to 5 axis complexity.

While no definite conclusions were reached in the study, sufficient data is presented and analysis factors explained that a prospective user can perform benefits analysis in the context of his own shop environment.

One fact is clear - that of the general purpose NC language processors now in widespread productive use only two language families are prevalent, APT and COMPACT. It is again only these two language forms that are being considered in government and national standardization activities. As such both merit the attention of the Air Force ICAM Program.

CRITERIA FOR NC LANGUAGE SELECTION

Several technical factors should be considered in choosing a programming language for numerical control:

Language Programming Capability
Processor Availability
Language Documentation
Processor Maintenance
Programming Time
Processing Costs
Proprietary Nature of Language

Study of NC languages must be placed into the perspective of the final goal of the Air Force program, an integrated computer base manufacturing system. It is expected that when this goal is realized, a designer may sit down at a computer terminal with a CRT, design some object, then allow the computerized manufacturing system to manage the supply of raw materials, schedule machines, decide on cutters, manage inventories and produce a final product while providing management and designers with the appropriate feedback information Critical interfaces in this final system should be identified now and carefully standardized so that a workable system can be developed. In the area of the actual machining and forming of parts, the most important interface is between the CAM (computer aided manufacturing) system and the actual production machines. This interface is defined by the CLDATA file. This is the standardized part description data that describes exactly how to make any part. A postprocessor of any machine tool will convert this standardized data to the specific command statements necessary for that particular tool to make the part. The CLDATA file can also be used by graphics devices to display in visual form information concerning the part.

At the present time this CLDATA file is generated by the NC programming language APT, and is being considered as an optional requirement for COMPACT II. It is the standard for the International Standards Organization (ISO). A designer now gives a part programmer either his own drawings or design drawings made with varying degrees of computer assist. The part programmer then generates the necessary code to make the part. This is passed through a processor to create (in APT) a CLDATA file which should be a totally machine-independent representation of the part. This is customised to the requirements of the individual machine tool by putting the CLDATA file through the postprocessor for that tool. Eventually the part programmer should be eleminated with the CAM system providing the CLDATA file from the designer's requirements. Thus, while the NC programming language standard is important, indeed crucial during the interim stage, its importance will decrease as the full CAM system is realized. The CLDATA file, however, will become the link between the CAM system and the real world of production. If this CLDATA file can be properly standardized, it can be the common data base between any CAM system and any set of machine tools or any programming language and any CAM system or machine tool. It would make it easy for any machining facility to produce any parts regardless of their own CAM capability, merely by having access to the CLDATA file. It would allow government, for instance, to make replacement parts or additional units from CLDATA files without having to attempt to access contractor CAM systems that might be proprietary. Again this interface, the CLDATA file, is considered one of the most crucial for a truly flexible computer aided manufacturing system.

RECOMMENDATIONS

With this in mind our recommendations considering NC part programming languages follow.

(1) If a single part programming language is desired to cover all applications then this language should be APT. APT produces the CLDATA file standard. It is the most sophisticated including such unique capabilities as producing part programs for milling a non-formula or sculptured surface (a surface defined by a lattice of coordinate points) such as found commonly on aerospace parts. Thus far it is the only language for which there is a formal draft standard. There are several areas of research and development of advanced capabilities such as process planning, geometric modeling, and sculptured surfaces that will be compatible with existing APT processes.

(2) If more than one language can be considered, then it is recommended that both APT and COMPACT II be used. APT provides the sophistication for complex parts. COMPACT II, however, offers significant advantages in speed and ease of programming of simpler parts and expecially lathe work.,

(3) If COMPACT II is included as a standard language then the capability to produce a standard CLDATA file must be included. This would allow part programming in either APT or COMPACT II with their CLDATA files to be the common interface to the production machines.

(4) While the current standardization activity with the APT postprocessor language is encouraging, it falls short of the capability truly needed by the Air Force in manufacturing. Even the most recent proposed standard for APT allows too much latitude in the choice of language syntax. Anything short of a complete and comprehensive language specification obviates the possibility of being able to rapidly and easily exchange NC workload among functionally equivalent machines. This capability is central to the concept of integrated and flexible manufacturing. The Air Force can and should provide the impetus to a widespread implementation of a complete government standard on postprocessor language and philosophy of post-processing which would bring about this flexibility. Only with this technique can NC data be made transferable among different machines, different shops and different contractors.

(5) Further work on additional language capabilities for both APT and COMPACT II is being carried out by the relevant ANSI committees on NC part programming languages. It is recommended that the Air Force monitor this work and help provide direction for the implementation in CAM systems. Work is progressing in the areas of a) sculptured surfaces, non-analytical sculptured shapes (shapes arrived at by sculpturing processes), unconventional analytical shapes (e.g. parametric surfaces), and any combination of these two; b) bounded geometry, 3-dimensional modeling capability within the computer. Objects would be represented and manipulated as bounded, closed entities rather than as bounded by a set of possibly infinite faces combined in specific ways; c) lathe language - a study of the various capabilities of several languages in their ability to efficiently program lathes which account for close to half of all NC machine tools.

COMMENTS

The emphasis of the proceeding report on NC Programming Language Standards is the important interface between future CAD/CAM systems and the production tools. The CLDATA file appears to be a good starting point for the development of this crucial interface standard. The recommendations above suggest some important additions necessary if real flexibility is to be obtained at this interface.

There are additional considerations which will be mentioned here.

The CLDATA file is not a totally independent description of the necessary commands and cutter path. When the program is written, certain data such as the diameter of the cutting tool, the length of the tool, etc. are included in the program and these affect the cutting path motions. The CLDATA file with postprocessor commands can be used as a description of the machine tool operations only as long as these additional parameters are kept constant. If a shop does not have the

correct size cutter it would be advantageous if the part program could be modified to accomodate the cutter size availabe. For contouring operations, this implies new geometric calculations and the need for source code modifications. It would thus be advantageous to have the NC system on-line in a DNC configuration. This is a reasonable plan for systems for the 1980's. This would require better identification of relevant statements in the CLDATA file, perhaps through flags, comment statements, etc. to allow for possible editing.

REFERENCES

(1) American National Standard APT - Proposed Revision, March 1975, American National Standards Institute, 1430 Broadway, New York City, New York 10018

(2) Numerical Control Language Evaluation, Numerical Control Society, March 1974, 1201 Waukengan Road, Glenview, Illinois 60025

(3) CAM-I Special Projects 1976, PR-75-ASPP-01, Computer Aided Manufacturing-International, Inc., 611 Ryan Plaza Drive, Suite 1107, Arlington, Texas 76012

(4) Proposal for the creation of an X3 Standards Committee covering the COMPACT II, ACTION, and SPLIT family of Numerically Controlled Machine Languages, 1975; Submitted to: American National Standards Committee X3, Secretary X3, CBEMA; By: Manufacturing Data Systems, Inc., 320 North Main Street, Ann Arbor, Michigan 48104; Sundstrand Machine Tool Compary, 3625 Newburg Road, Bervidere, Illinois 61008.

(5) The letter of August 27, 1975 by Elliot Brebner, Chairman of X3J7, in response to the SPARC committee requesting that X3J7 comment on the COMPACT II, ACTION, SPLIT proposal for the formation of a standards committee.

(6) Computer Software for Numerically Controlled Manufacturing - 1973; By: Bradford Smith, Report # NSRDC 4327, Computer Aided Design Division, Naval Ship Research & Development Center, Bethesda, Maryland 20084

STANDARDS FOR COMPUTER AIDED MANUFACTURING

2. CAD/CAM INTERFACE STANDARDS

INTRODUCTION

The CAD/CAM interface is a boundary, as yet ill defined, across which information must be communicated. The flow of information is primarily in the CAD→CAM direction, although ideally there is a reverse flow giving the design or process engineer information concerning tool availability, material inventory, etc. The simplest, historical design/manufacturing interface was the set of engineering drawings describing the part to be manufactured. In a CAM system, the interface is the appropriate data base representing the same data as the part drawings.

APT AS A DE FACTO CAD/CAM INTERFACE

There presently exists no consensus as to what the CAD/CAM interface is or precisely when it should be drawn. For example, are Automatically Programmed Tool (APT) programs part of the design process or the manufacturing process? Many small stand-alone interactive CAD systems produce APT source code, APT CL file data or machine tapes as direct output. This data is then carried to a manufacturing installation where it is put through a processor and/or post processor (if necessary) and used to control NC machine tools. The APT part description is thus a direct CAD/CAM interface for small systems.

In larger installations, where CAD/CAM is more integrated, the data base which describes the physical parameters of the parts to be manufactured is usually considered to be the CAD system output. APT tool programming is treated as one part of Process Planning (part of CAM, not CAD). The CAD/CAM interface is thus considered the drawing or data base describing the part. In Figure 1, if APT programming is considered to be a part of manufacturing, the CAD/CAM interface can be drawn as the dashed line. If, however, APT is included in design, then the interface can be drawn as the dotted line in Figure 1. In either case, it is possible, given the structure shown in Figure 1, to draw the CAD/CAM interface such that it cuts only the outputs of data bases. This would appear to be a useful concept in that it makes for clearly defined interfaces both physically and logically.

STANDARDS ACTIVITY IN DIGITAL REPRESENTATION OF PHYSICAL OBJECT SHAPES

Whether any particular CAD/CAM system adopts the configuration of Figure 1 or some other, it is clear that the data base consisting of a numeric description of physical objects is central to the entire CAD/CAM processes. This data base provides the working input to CAD displays and to CAD analysis programs. Indeed, the entire CAD process does nothing more than generate, analyse, and manipulate this data base. Once finalized, the part descriptor data base provides the primary input to APT tool programs, planning and scheduling programs, and eventually to inspection and quality assurance programs.

Thus, the part description data base is central to the entire CAD/CAM concept and, in large measure, will define the CAD/CAM interface. This implies that efforts to develop standard methods for representing part shapes, dimensions, tolerances, materials surface finishes, etc. are prerequisites to developing standards for CAD/CAM interface. The American National Standards Institute (ANSI) Y14.26 subcommittee on Computer Aided Preparation of Product Definition Data is presently working on a Y14.26.1 standard for the Digital Representation of Physical Object Shapes.

The stated aim of this standard is to facilitate the communication of physical object shape descriptions among CAD/CAM programs and data bases of organizations engaged in interfacing activities such as contracting and subcontracting. The approach is to abstract the spatial property of shape

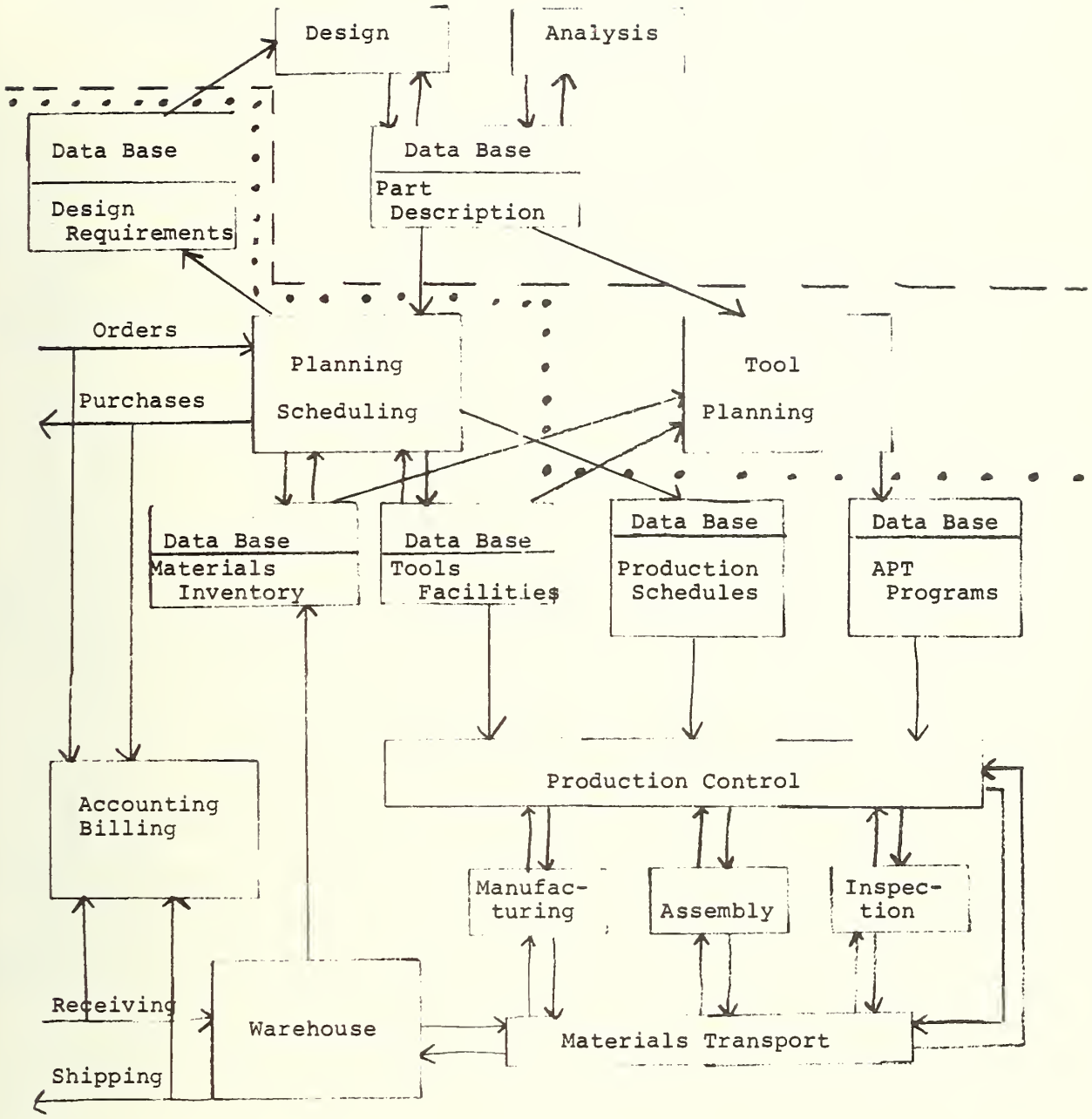


Figure 1

SCHEMATIC LOCATIONS OF THE CAD/CAM INTERFACE

by representing physical objects as geometric solids. The problem then reduces to describing solids.

A solid may be considered to be a geometric structure constructed out of building blocks of simpler geometric entities. The description of that solid is then an information structure constructed out of building blocks of digital data. This leads to a hierarchy of building blocks.

At the lowest level in the geometrical hierarchy is the point. A point moving along a trajectory generates a line, a line moving along a trajectory generates a surface. A surface moving from a start to an end surface generates a solid element. A succession of solid elements can be joined to form a complex solid. An example of this method of generating and describing physical object shapes is shown in Figure 2.

Generating a trajectory requires a rule (or set of rules, procedures, or equations) which describes the motion of the generatrix (point, line, surface, solid element). Each element in the hierarchy depends on the available set of subelements and generating procedures in the lower levels of the hierarchy. A judicious choice of lower level subelements can produce a very broad variety of complex shapes.

This work is proceeding steadily, although rather slowly. But even when this standard is formalized, it will represent only a first step toward solving the larger problem of completely describing physical objects.

A related effort is currently being funded by Computer Aided Manufacturing-International, Inc. (CAM-I). The CAM-I Geometric Modeling Project is attempting to develop 3-dimensional modeling tools based on digital descriptions of geometric shapes. On August 25, 1976, CAM-I acccepted a bid from Sof-Tech, Inc to develop a Geometric Modeling System (GMS). This will be a generic system capable of incorporating software modules for part description languages, geometric modeling mathematics, display and communication technology, and end use applications. GMS is to "conform to ANSI standards and be as computer-independent as possible."

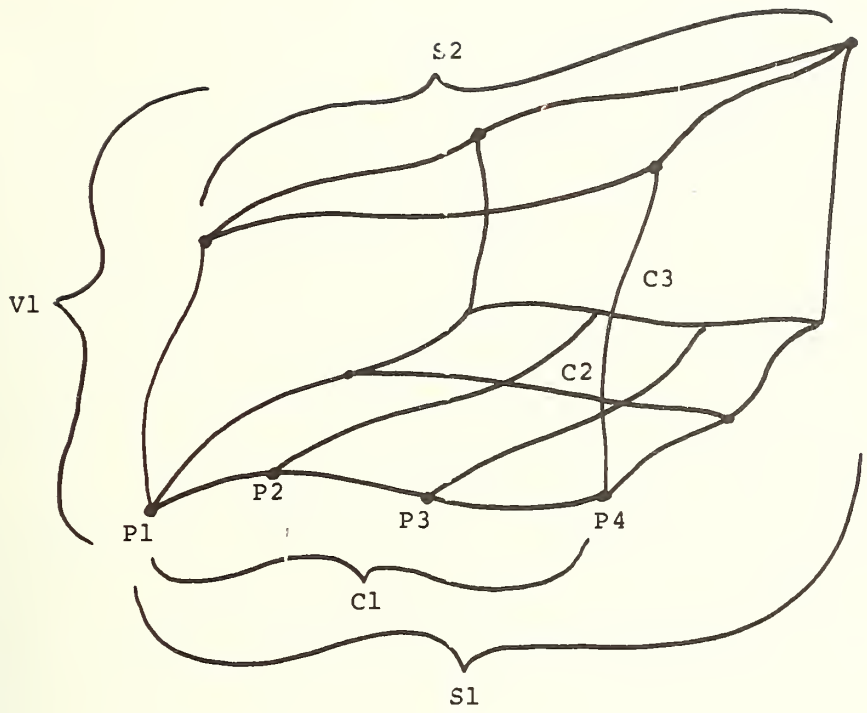
THE CAD/CAM INTERFACE IN PRINTED CIRCUIT BOARD MANUFACTURING

The Institute of Printed Circuits has published a standard entitled "End Product Description in Numeric Form for Printed Wiring Products."

This standard not only defines methods for describing geometric shapes of printed wiring boards but prescribes record formats for describing the end-product in digital form. An example of four records describing four segments of a printed wiring circuit is shown in Figure 3. This digital data, when recorded on punched cards or magnetic tape, contains sufficient information for tooling, manufacturing and continuity testing of printed wiring products. These formats thus may be used for transmitting information between the designer and the manufacturing facility after the design has been completed by a computer-aided process. Such data format standards are particularly useful when the manufacturing process includes numerically controlled machines.

The data records specified in this standard are general, not in any particular machine language, and can be used for both manual and machine interpretation. Thus each facility can produce an end-product from the data by the most efficient method available.

Unfortunately, this standard addresses only a tiny fraction of the set of manufactured products, namely two dimensional printed circuit boards. Nevertheless, it is complete, is presently in use, and does deal with the problem of describing a physical object with sufficient completness to define not only the manufacturing process, but the inspection and acceptance testing process as well.



C1 = G06 (P1, P2, P3, P4)

The curve C1 is generated by the parametric cubic operator G06 operating on the points P1, P2, P3, P4.

S1 = G05 (C1, C2, C3)

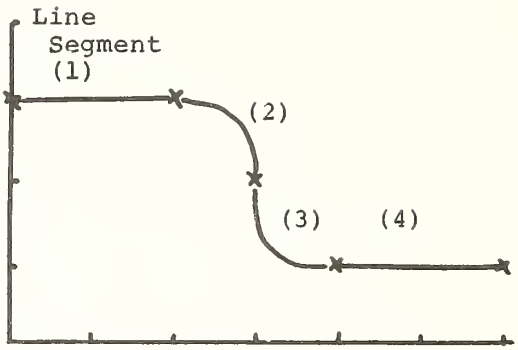
The surface S1 is generated by the operator G05 operating on the curves C1, C2, C3.

V1 - G04 (S1, S2)

The solid V1 is generated by the operator G04 operating on surfaces S1, S2.

Figure 2

ANSI Y14.26 METHOD OF DESCRIBING PHYSICAL OBJECT SHAPES



Op. Code	Data Field						Record #
	Start Point		End Point				
111	X+00	Y+03	X+02	Y+03			(1)
	Circle Center		Start Angle	Finish Angle	Radius	Direc-tion	
021	X+02	Y+02	X+90	Y+00	X+01	Y+01	(2)
021	X+04	Y+02	X+180	Y+270	X+01	Y-01	(3)
011	X+04	Y+01	X+06	Y+01			(4)

Set size of XY fields
1 = linear 2 = circular interpolation
1 = begin line 0 = continue line

Figure 3

EXAMPLE OF IPC STANDARD REPRESENTATION OF PRINTED WIRING CIRCUIT

POTENTIAL IMPACT OF NASA'S IPAD PROJECT

The work on an Integrated Program for Aerospace-Vehicle Design (IPAD) being funded by NASA Langley Research Center is not a standard by the common definition. It is merely one more integrated software system which attempts to computerize, in so far as possible, company-wide design information processing. IPAD will be composed of 1) executive software that will control user-directed processes through interactive interfaces with a large number of terminals in simultaneous use by engineering and management personnel, 2) a large number of utility software packages for information manipulation and display functions, and 3) data management software to store, track, and retrieve large quantities of data in multiple storage devices.

However, IPAD is different from other integrated software design systems in that it is scheduled to be released by NASA to become public domain under NASA's FEDD (For Early Domestic Dissemination) policy. If IPAD is a successful system it will undoubtedly be widely used by many industries, especially those which are too small to afford to develop their own internal CAD systems. The formats used by IPAD for digital description of physical objects shapes, and even for describing end-products, will thus become common usage in many CAD/CAM systems in the future.

The result will be that even though IPAD does not pretend to be a standards setting project, it nevertheless will set precedents which are almost certain to become de facto standards for data base formats, man-machine interfaces, and eventually CAD/CAM interfaces.

There will probably arise many situations where IPAD data bases will not conveniently conform to standards being developed under ANSI Y26.14.1. The temptationwill be to ignore the ANSI standards since they have not yet been formally adopted. Every effort should be made to resolve such conflicts whenever they arise for otherwise the general applicability and usefulness of both IPAD and Y26.14.1 will be reduced. The result will be that future CAD/CAM systems such as the ICAM system of the US Air Force will be adversely impacted.

The Air Force should take every effort to avoid such conflicts, working closely with NASA in the manner outlined in the existing Memorandum of Agreement between NASA and the Air Force.

SUMMARY

To date, all operational CAD/CAM systems have adopted ad hoc techniques custom tailored to specific applications. To some extent this is acceptable as long as a CAD/CAM installation is confined to a single plant or a single company where local custom can serve as an ad hoc standard. It is, however, completely unacceptable in a wider context where many different contractors and subcontractors will be required to use the same numeric descriptors for competitive bidding and for manufacturing operations. For a project such as the Air Force is presently contemplating, it is critical that efforts to achieve systematic set of numerical product descriptors be given top priority. Full cooperation and support should be given to the ANSI Y26.14 subcommittee as well as to the CAM-I Geometric Modeling Project. Close liaison should be maintained with the NASA's IPAD and every effort made to see that conflicting and competing standards do not proliferate.

RECOMMENDATIONS

It is recommended that the Air Force:

1. Maintain close liaison with the ANSI Y14.26 subcommittee.

2. Insist that all of its contractors adhere to the ANSI proposed standards whenever possible.

3. Maintain its close liaison with the IPAD project as outlined in its present Memorandum of Agreement with NASA.

4. Be aware of potential conflicts with IPAD and take whatever steps possible to prevent serious incompatabilities from developing.

5. Monitor the CAM-I Geometric Modeling Project to identify any compatability problems that may develop.

6. Insist on the use of the IPC-D-350A standard in future wedges relating to electronics or systems including printed wiring products.

REFERENCES

(1) Statement of Work - Development of Integrated Program for Aerospace Vehicle Design (IPAD), Ap 15, 1976, Langley Research Center, Langley, VA.

(2) Product Manufacturing Interface, October 1976 D6-IPAD-70011-D, NASA Langley Research Center

(3) End Product Description in Numeric Form for Printed Wiring Products, IPC-D-350A, September 1974, Institute for Printed Circuits

(4) Digital Representation of Physical Object Shapes, ANSI Y14.26.1 Draft Report, June 1976, American National Standards Institute, New York City, 10018

(5) CAM-I Special Projects, 1977, PR-76-ASPP-01, Computer Aided Manufacturing-International, Inc., Arlington Texas, 76012

(6) Minutes of Geometric Modeling Project Meeting, M-76-GM-01 held August 24-26, Rochester, N.Y., CAM-I, Arlington, Texas 76012

1. COMPUTER AND COMMUNICATIONS INTERFACE STANDARDS

INTRODUCTION

For purposes of the following discussion, an interface is defined to be the point of interconnection between two logically and physically separate components to enable the interchange of information. Depending upon the operational capabilities and functional complexities of the components, specification of an interface may require the definition of parameters and performance characteristics at several levels.

At the most basic level, for example, the physical interconnection of two components requires that they be electrically and mechanically compatible at the interface point, i.e., the signalling voltages and currents presented at the interface by each component must be compatible with the impedances and receiving circuit sensitivities of the other and the two interconnection plugs must mate. In addition, also at the basic level of interface definition, it is essential for information interchange that the components be functionally compatible, i.e., every function required by one component must be generated and presented at the interface in proper sequence by the other.

For some kinds of relatively unsophisticated equipment, conformance to the basic electrical, mechanical, and functional interface characteristics is sufficient to ensure operation. Complex systems also require that higher level operational and procedural definitions be provided. At the highest level where the components being interconnected have a range of operating capabilities, the formats and information transfer sequences must be also defined to ensure component interoperability.

There are generally three different kinds of interfaces that have been established for ADP systems that govern the interconnection of these systems with external devices and facilities and which enable the input/output interchange of internally stored information with the data collection, storage, or distribution environment external to the ADP system. The three kinds of interfaces are for:

(1) Computer peripheral devices, such as magnetic tape or disk that may serve both as intermediate or long term storage as well as a means for the direct input and output of data.

(2) Instrumentation and control devices that may be employed in a laboratory experiment or process control environment where the ADP system collects data produced by environmental or positional sensors and as a result of processing this data generates correctional control sequences to operate other machinery or equipment involved with the performance of the process.

(3) Communications, where the ADP system is to be interconnected with analog or digital telecommunication facilities in a teleprocessing environment.

For each of these three different kinds of interfaces, industry or national standards are being developed, or in some cases have already been approved, that specify the interfaces sufficient to ensure that components furnished by different suppliers can be interconnected.

COMPUTER PERIPHERAL DEVICE INTERFACES

Standards for this ADP system interface have proved to be the most difficult to accomplish, not because of their technical complexity but rather due to competitive pressures and fundamental differences in the architectural structure employed by different ADP system manufacturers. However, several draft proposed American National Standards are presently close to completion.

Large Scale Computer System Peripheral Interfaces

The first set of these computer peripheral device interface standards deal with the large scale ADP system and is based upon the IBM 370 type I/O channel-to-peripheral controller interface; the set consists of three kinds of specifications: (1) a document that prescribes the interface electrical, mechanical, and functional characteristics, (2) an interface power control specification, and (3) a series of device-specific (e.g., tape, disk, etc.) operational specifications.

Figure 1 illustrates the architectural structure for a large scale computer system that contains an I/O channel and shows the point in this structure that is defined as the I/O channel-to-controller interface.

Figure 2 provides a listing of functions presented on the two sides of the I/O channel-to-controller interface and indicates the direction of signalling. In general, a command initiating an action (e.g., Select Out) is issued by the channel, while the response, indicating the action has been completed is issued by the controller.

It is anticipated that an I/O channel-to-peripheral interface standard including operational specifications for both magnetic disk and tape devices will be completed and approved by the American National Standards Institute by late 1977.

Minicomputer System Peripheral Interfaces

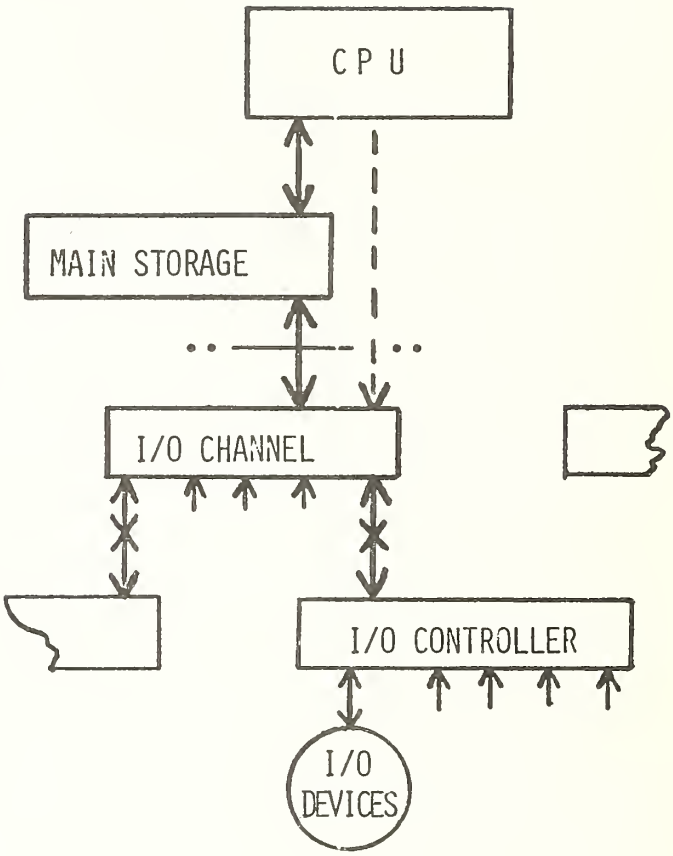
A different kind of device level, but device-specific computer peripheral interface standard is being developed for minicomputer systems. Figure 3 indicates the arrangement of processing logic, control, storage, and I/O components in a typical minicomputer system employing a common bus structure. It also shows the interface point for connecting peripheral devices. In the minicomputer case, a general purpose standard peripheral device interface is being perscribed that contains a total of some 40 functions--all of which would be presented on the CPU side of the interface; devices conforming to this interface, however, will only employ the functions they actually require, elg., a printer cannot perform the function "read media" and thus would not implement this function.

Figure 4 lists the kinds of functions and indicates the signalling directions for this general purpose interface as it would be implemented between a magnetic tape transport and a controller.

It is anticipated that this general purpose device-level minicomputer interface standard will be completed and approved by the American National Standards Institute by the End of 1977. Furthermore, it is planned that in conjunction with the final stages of processing by ANSI these computer peripheral interface standards will also be processed for adoption and implementation as Federal Information Processing Standards.

Recommendation for Computer Peripheral Device Interfaces

The General Services Administration has established a number of Mandatory Requirement Contracts dealing with the procurement of "plug compatible replacement" peripheral devices for the product lines furnished by several of the major manufacturers. These contracts cover magnetic tape and magnetic disk subsystems (including the respective controllers), add-on memory, and input/output punched card facilities. All agencies are obligated to use these GSA Mandatory Requirement Contracts whenever practical to do so. It is recommended, however, that the Air Force



X - I/O Channel to Controller
Interface Points

FIGURE 1: LARGE SCALE COMPUTER SYSTEM ARCHITECTURE

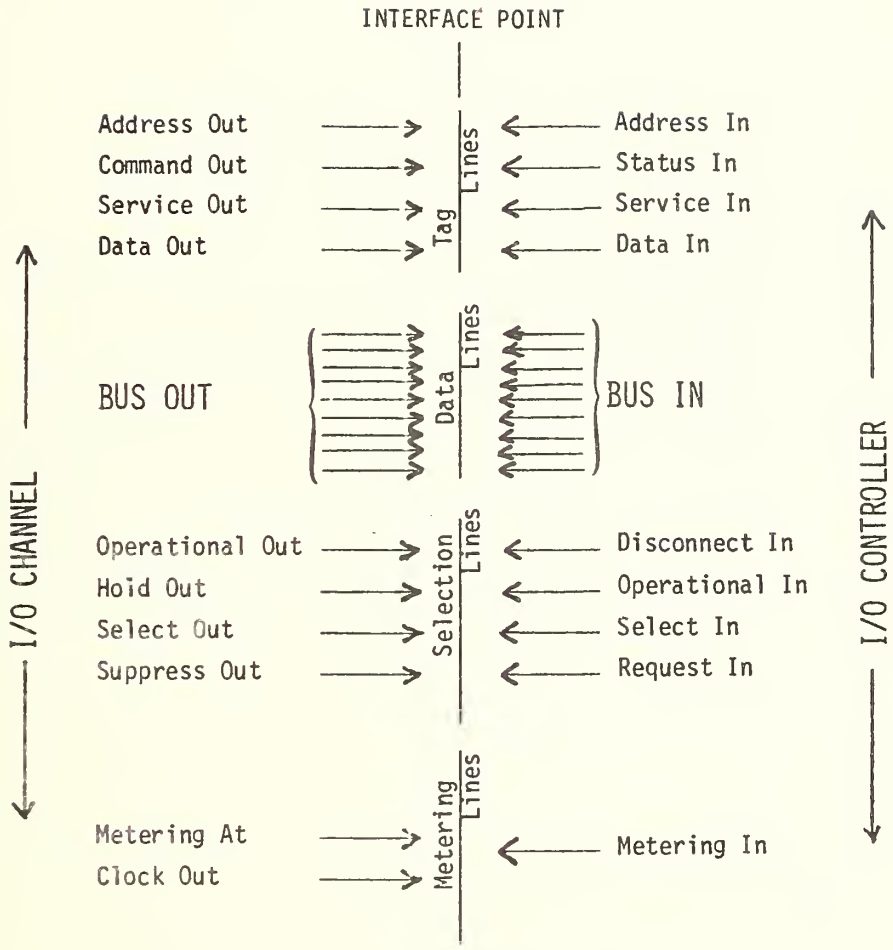
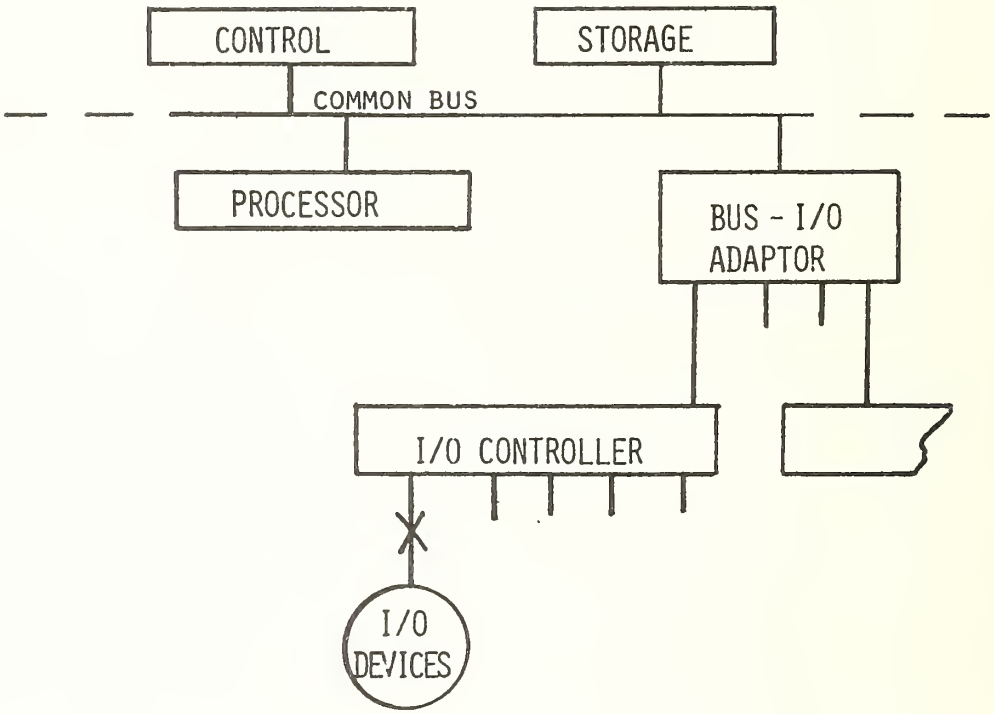


FIGURE 2: I/O CHANNEL TO CONTROLLER INTERFACE



X - Device Level Interface
Point

FIGURE 3: MINICOMPUTER SYSTEM ARCHITECTURE

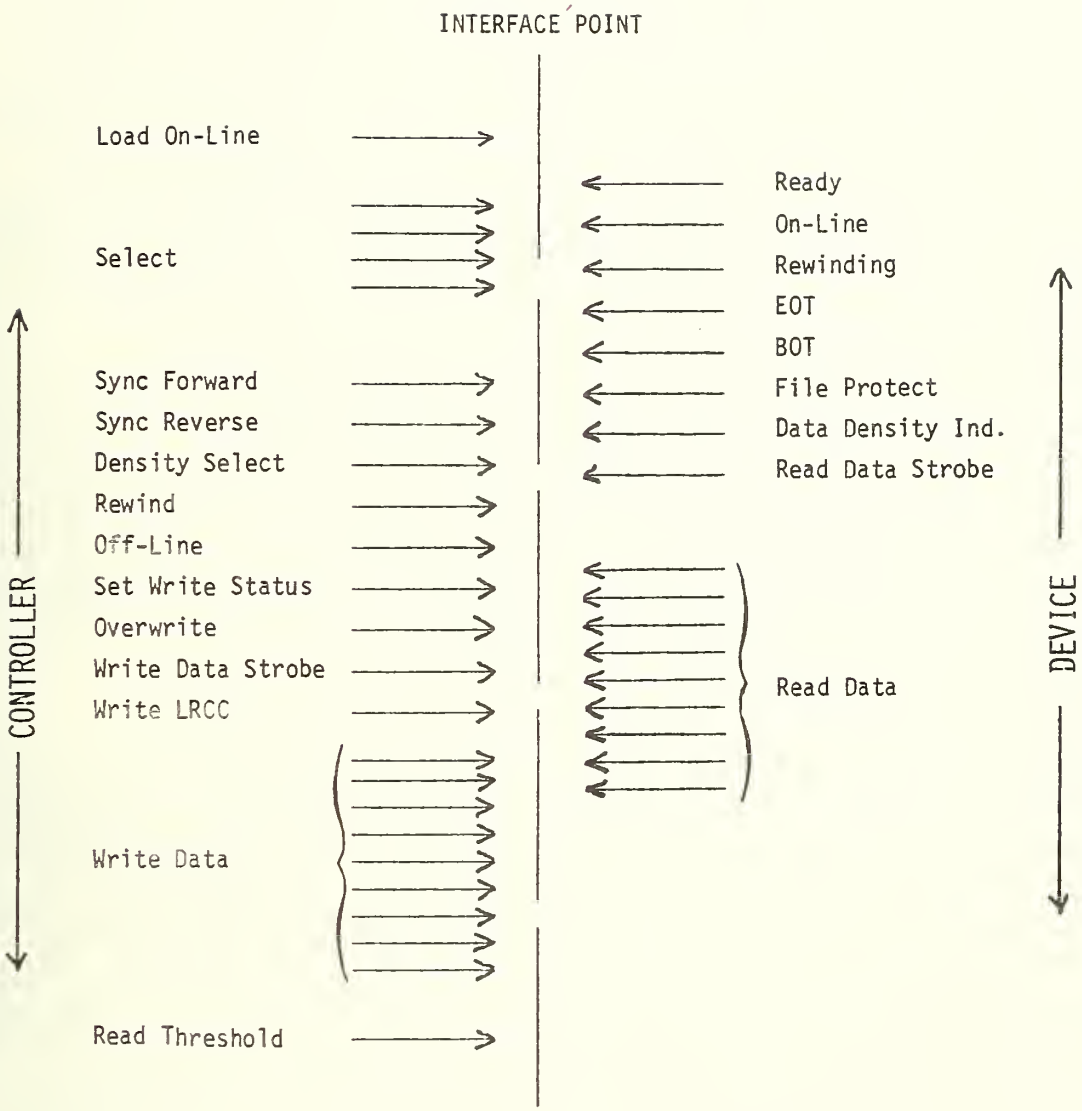


FIGURE 4: I/O CONTROLLER TO MAGNETIC TAPE DEVICE INTERFACE

carefully follow the standards being developed for the computer peripheral device interface and be prepared to implement these in CAM applications as soon as these standards are proposed for Federal adoption.

INSTRUMENTATION INTERFACES

The IEEE has developed and approved (as of July 1974) an industry standard for instrumentation applications entitled "IEEE Standard 488--Digital Interface for Programmable Instrumentation." This standard has also been approved by the American National Standards Institute as ANSI MC 1.1-1975. Although this standard is not limited by its scope, it appears that its principal application is concerned with minicomputers instrumented in close proximity for limited process control functions such as in a laboratory type environment. This standard deals with systems and components that employ byte-serial, bit-parallel data transfer. Figure 5 illustrates the 16 wire bus structure of the IEEE 488 programmable instrumentation interface and indicates some of its characteristics as well as the functional properties (talker, listener, etc.) of some of the components that may be interconnected by it.

Recommendation for Instrumentation Interfaces

It is anticipated that implementation of the IEEE Standard 488 will probably be contrained to minicomputers interconnected in close proximity with digital instruments and devices normally employed in laboratory type experimental situations, e.g., temperature, signals various form sensors, or positional measurements with the processing of these measured data being employed to correct and control their future values. While this interface is not considered to be of general purpose utility for data processing, some of the instruments and devices that are available as "off-the-shelf" items for use in CAM applications are designed to the IEEE Standard 488 interface. For this reason, it is recommended that the Air Force be aware of the existence of IEEE Standard 488.

COMMUNICATIONS INTERFACES

Perhaps the most dynamic areas of computer utilization are currently those concerned with teleprocessing and computer networking that are dependent upon advances in data communication technology. Within the past few years, there have been a number of significant developments in establishing standards for data communications and computer networking. New standards that are in various stages of development include replacements for such widely accepted and implemented standards as RS-232 at the physical interconnection level and binary synchronous (bisync) link control at the link protocol level as well as for higher levels not previously covered, such as for packet switching. These standards are being developed both on a national and international scale by such groups as the American National Standards Institute (ANSI), the International Standards Organization (ISO), and the Consultative Committee on International Telegraph and Telephone (CCITT). Most of these standards eventually will be adopted for mandatory use within the Federal Government by either or both the National Bureau of Standards (NBS) and the National Communications System (NCS). Because most of these standards pertain to the interconnection of computers or data terminal equipment with data communication or telecommunication facilities, they all may be characterized as interface standards dealing with the computer communications interface.

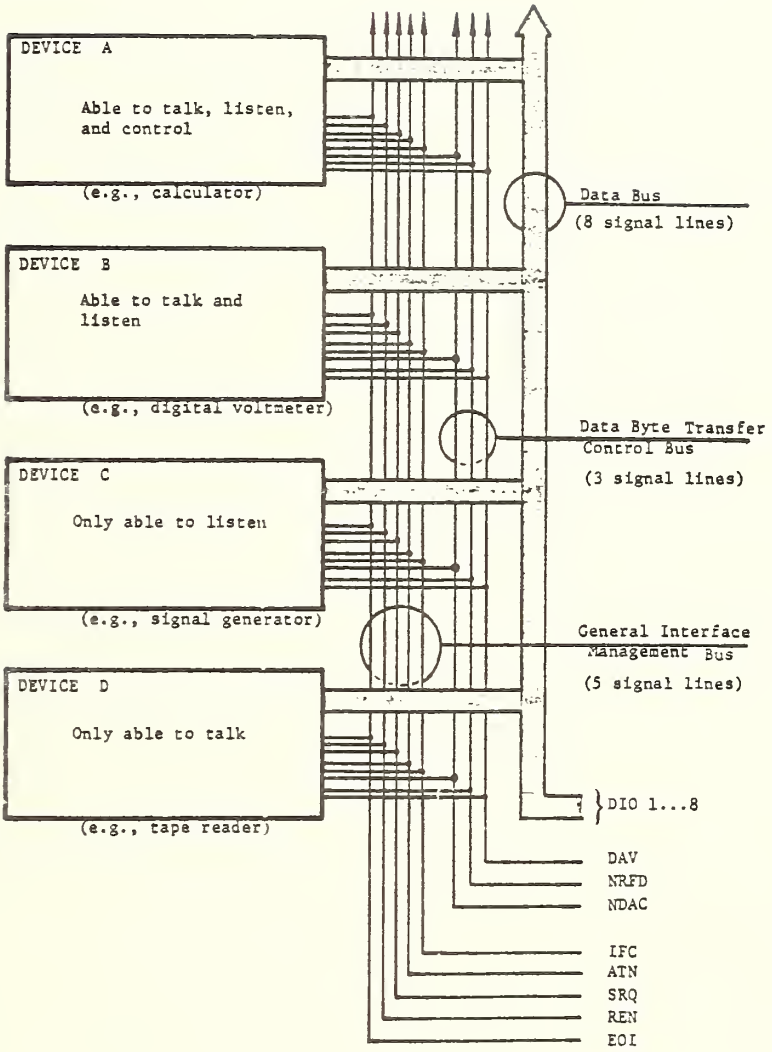


FIGURE 5: THE BUS STRUCTURE FOR THE IEEE STANDARD 488--DIGITAL INTERFACE FOR PROGRAMMABLE INSTRUMENTATION

Hardware Interconnection Level Interfaces

With the data networks of the future expected to be digital from end to end, standards are being developed to interface terminals to such networks. This includes replacement for RS-232, presently being developed by EIA and known as RS-XYZ, as well as the addition of a signalling scheme to initiate and terminate calls (replacing manual dialing). Internationally, CCITT Recommendation X.21 is being proposed for synchronous terminals and provides a means to initiate calls, exchange call progress signals, transmit data, and finally terminate calls on new public data networks. This CCITT Recommendation is also under consideration for adoption as American National and Federal standards.

Figure 6 and 7 show the respective functional properties of the RS-XYZ and X.21 interfaces. Note that while the RS-XYZ interface provides a separate interchange circuit for each function, the X.21 interface accomplishes essentially the same functional interchanges by combinations of signals presented on the circuit pairs TRANSMIT (data) with CONTROL and RECEIVE (data) with INDICATION, i.e., when CONTROL is "on" the information on the TRANSMIT circuit is interpreted as control--otherwise it is data.

Data Link Level Interfaces

At the data link level, ISO has been working for the past few years to complete the details of a new, bit-oriented High Level Data Link Control Procedure (HDLC). The American National Standard version of this procedure is called the Advanced Data Communications Control Procedure (ADCCP). The concept of a data link to which these control procedural standards apply is defined as an assembly of two or more data terminals and the interconnecting line operated according to a particular method or protocol that permits information to be exchanged.

So far, international agreement has been achieved for both the HDLC frame structure and the elements of procedure (definition of the command and response repertoire). International arguments are still going on regarding the way in which these commands and responses are to be used for various applications involving different terminal and link configurations. Consensus is slow to achieve because of the many different interests that must all be satisfied with the proposed standard.

IBM, because of its ability to act unilaterally in product announcements, has announced a product implementing its own Synchronous Data Link Control (SDLC) procedure which is quite similar to HDLC. Some other vendors such as Burroughs have announced products that they claim will be fully compatible with SDLC, ADCCP, and HDLC.

DEC, on the other hand, has continued to pursue its own link control procedure (DDCMP) which is quite different from any of the proposed standards. As strong vendor participation continues in the final development of both HDLC and ADCCP, it seems likely that both an international and compatible national standard will eventually emerge that will be implemented by most of the major vendors.

Network Level Interfaces

One of the most dramatic standards developments in the last few years has been the adoption of Recommendation X.25 by the CCITT at its quadrennial plenary assembly this past September. Recommendation X.25 is a standard for interfacing host computers to public packet switching networks. It includes both X.21 and HDLC in the appropriate portion of the standard, and adds a set of packet formats and commands and responses for setting up "virtual calls" and transferring data through the network.

CIRCUIT MNEMONIC	CIRCUIT NAME	CIRCUIT DIRECTION	CIRCUIT TYPE	
SG	SIGNAL GROUND	–	COMMON	
SC	SEND COMMON	TO DCE		
RC	RECEIVE COMMON	FROM DCE		
IS	TERMINAL IN SERVICE	TO DCE	CONTROL	
IC	INCOMING CALL	FROM DCE		
TR	TERMINAL READY	TO DCE		
DM	DATA MODE	FROM DCE		
SD	SEND DATA	TO DCE	DATA	PRIMARY CHANNEL
RD	RECEIVE DATA	FROM DCE		
TT	TERMINAL TIMING	TO DCE	TIMING	
ST	SEND TIMING	FROM DCE		
RT	RECEIVE TIMING	FROM DCE		
RS	REQUEST TO SEND	TO DCE	CONTROL	
CS	CLEAR TO SEND	FROM DCE		
RR	RECEIVER READY	FROM DCE		
SQ	SIGNAL QUALITY	FROM DCE		
NS	NEW SIGNAL	TO DCE		
SR	SIGNALING RATE	TO DCE		
SSD	SECONDARY SEND DATA	TO DCE	DATA	SECONDARY CHANNEL
SRD	SECONDARY RECEIVE DATA	FROM DCE		
SRS	SECONDARY REQUEST TO SEND	TO DCE	CONTROL	
SCS	SECONDARY CLEAR TO SEND	FROM DCE		
SRR	SECONDARY RECEIVER READY	FROM DCE		
LL	LOCAL LOOPBACK	TO DCE	CONTROL	
RL	REMOTE LOOPBACK	TO DCE		
TM	TEST MODE	FROM DCE		
SS	SELECT STANDBY	TO DCE	CONTROL	
SB	STANDBY INDICATOR	FROM DCE		

FIGURE 6: INTERCHANGE CIRCUITS DEFINED FOR RS-XYZ
THE DTE/DCE INTERFACE FOR ANALOG NETWORKS

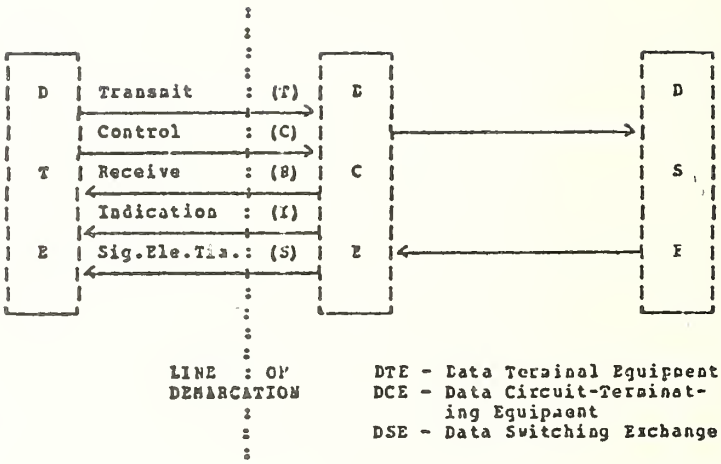


FIGURE 7: INTERCHANGE CIRCUITS DEFINED FOR CCITT RECOMMENDATION X.21 THE DTE/DCE INTERFACE FOR DIGITAL NETWORKS

Figure 8 shows the enveloping format prescribed by X.25 for the interchange of information between a host computer and a packet switched network. A packet consists of data to be transferred between two users. This data is preceded by a packet header that identifies the sender and intended recipient; the network uses information contained in the header for routing, billing, and network control purposes. The packet is then enveloped by an HDLC frame for transmission between the host computer and the network. The HDLC frame provides for link level control and consists of bracketing opening and closing flag octets, a link address octet and a control octet identifying the type of command or response frame; the frame is ended with the two octet Frame Check Sequence provided for error detection just prior to the closing flag octet.

Agreement on such a worldwide standard seemed quite remote only a few years ago, and it was not until the major packet switching carriers around the world got together privately that a consensus emerged. The standard has been criticized by some as lacking in certain features--notably the standard does not presently provide for the "datagram" type of service--but this particular deficiency is already being addressed by proposals to add to the standard.

The key point to note about X.25 is that a workable solution has been adopted which averts the situation of multiple incompatible interfaces being implemented by carriers in each country. Thus, it will be possible for computer manufacturers and software houses to build and support only one interface for packet switching.

Recommendations for Computer Communications Interfaces

The data communications area is a very dynamic one at present, and standards will continue to evolve to keep pace with the state of the art. Significant developments to look for over the next few years are the completion and large scale implementation of work already begun, such as HDLC, additions and modification to recently adopted standards, such as X.25, and the initiation of new work in areas not currently addressed, such as end-to-end protocols between host computers.

Designers of networks within the Department of Defense, such as AUTODIN-II and SATIN-IV are generally cognizant of these standards developments and insofar as practical most of these new standards are being implemented as the network design specifications are finalized.

For this reason, it is recommended that the Air Force advise ICAM contractors to confer with commercial data communication carriers concerning alternative network design characteristics and particularly with regard to specific user-to-network interfacing requirements rather than unilaterally prescribing communication interface standards that might subsequently prove incompatible with existing or planned networks.

SUMMARY OF INTERFACE STANDARDS

Figure 9 provides a system level overview of the typical locations of the several standard interfaces that have been described for a system that consists of one large scale computer connected to two remotely located minicomputers via a packet switched public data network. While the interfaces described are not the only interface points in a processing system such as this standardization of these particular interfaces provides the consumer of ADP products and services with a large degree of freedom in the acquisition and interconnection of components furnished by competitive sources. It should be noted, however, that although the various interface standards that have been described do make possible the physical interconnection of independently supplied components as well as

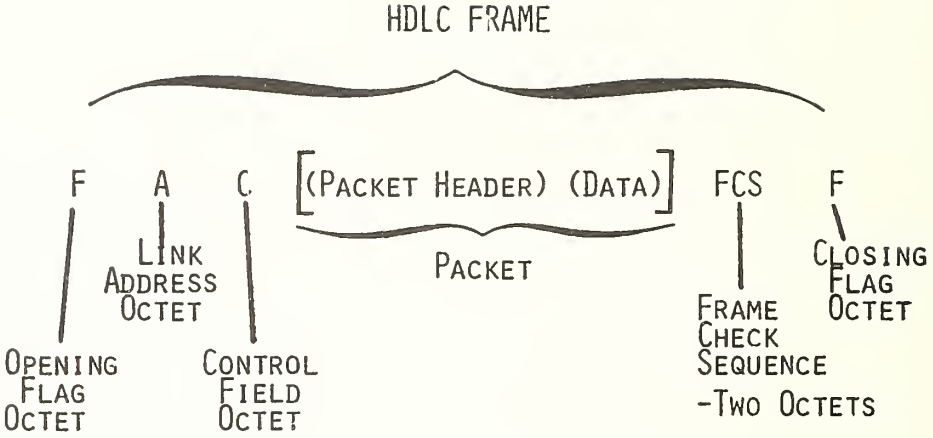
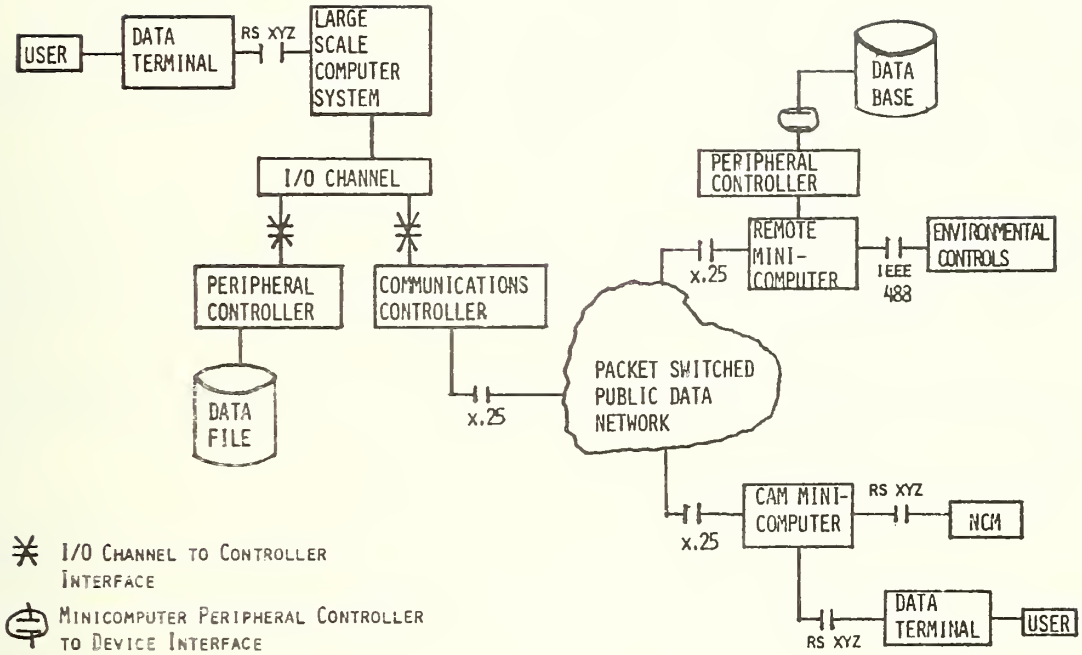


FIGURE 8: THE TRANSMISSION FORMAT PRESCRIBED BY CCITT RECOMMENDATION X.25



NOTE: X.25 PRESCRIBES THREE INTERFACE LEVELS: (1) THE PHYSICAL CIRCUIT LEVEL EMPLOYING X.21, (2) THE LINK CONTROL LEVEL EMPLOYING THE HDLC PROTOCOL, AND (3) THE PACKET LEVEL PROTOCOL INCLUDING SOME 14 PACKET FORMATS.

FIGURE 9: SYSTEM LEVEL PERSPECTIVE OF INTERFACES DESCRIBED

enable the interchange of data among these components it must be emphasized that these standards are not sufficient to ensure that meaningful end-to-end information interchange can occur. End-to-end communication between users in a system such as this also requires that both ends employ a common protocol involving a standard language that is represented with an agreed upon alphabet with characters encoded in a standard manner.

While a number of different multi-computer networking systems have been designed and successfully implemented that are incompatible among themeselves with regard to user protocols, languages, and codes, development of standards for many of these higher level problems has not yet been satisfactorily addressed. Partly, this is because some of these higher level problems are not yet sufficiently well defined that a standard solution can be prescribed--even though the need for standardization is generally recognized; partly, it is because in other cases a number of alternative competing solutions have been proposed, none of which appear optimal.

An an interim alternative to standardization for some of these higher level problems, NBS has designed and implemented a Network Access Machine (see NBS Technical Note 917) that employs a minicomputer to translate from a common user protocol to that required for accessing a variety of services provided by different remote host computer systems.

It is anticipated that these higher level areas of standardization will receive increasingly urgent attention in the near future and it is recommended that the Air Force monitor these activities closely.

SUMMARY OF RECOMMENDATIONS

a) It is recommended that the Air Force carefully follow the standards being developed for the computer peripheral device interface and be prepared to implement these in CAM applications as soon as these standards are proposed for Federal adoption.

b) It is recommended that the Air Force be aware of the existence of the IEEE Standard 488 that prescribes a Digital Interface for Programmable Instrumentation.

c) It is recommended that the Air Force advise ICAM contractors to confer with commercial data communication carriers concerning alternative network design characteristics and particularly with regard to specific user-to-network interfacing requirements rather than unilaterally prescribing communication interface standards that might subsequently prove incompatible with existing or planned networks.

d) It is recommended that the Air Force closely monitor standardization activities in the area of establishing common user, network access, and other higher level standards that will help ensure end-to-end communications in a heterogeneous computer networking environment.

REFERENCES

(1) ANSI X3S34/589 (Fifth Draft), 4/9/76, Advanced Data Communications Control Procedures, American National Standards Institute.

(2) Donnan, R. A., and J. Ray Kersey, "Synchronous Data Link Control: A Perspective", IBM Systems J., 13, 2, 1974.

(3) IBM Corp., "Binary Synchronous Communications", Order No. GA27-3004, IBM Corp., White Plains, N.Y. 10604.

(4) ANSI X3.28-1971, "Procedures for the use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links", American National Standards Institute, Inc., New York, N.Y., 10018

(5) Metcalf, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for local computer networks", CACM, 19, 9, 7/76, pp. 395-403.

(6) Farber, D. J., & K. C. Larson, "The System Architecture of the Distributed Computer System - The Communications System", Presented at the Symposium on Computer Networks, Polytechnic Institute of Brooklyn, 4/72.

(7) C.C.I.T.T., "Recommendation X.25 - Interface between Data Terminal Equipment and Data Circuit-Termination Equipment for Terminals Operating in the Packet Mode on Public Data Networks". See also ANSI documents X3S37-76-]4 and X3S33-76-6.

(8) Pouzin, L., "Virtual Circuits vs. Datagrams - Technical and Political Problems", Proc. NCC, 1976 (V. 45), pp. 483-495.

(9) ANSI X3S37-75-54/4 (Fourth Draft - "ANSI X.21") Proposed American National Standard--"General Purpose Interface Between Data Terminal Equipment and Data Circuit Terminating Equipment for Synchronous Operation on Public Data Networks.

(10) Electronic Industries Association Draft Standard--"Functional and Mechanical Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange"--(Temporarily Labeled RS-XYZ), Twelfth Draft--May 7, 1976, Amended--July 30, 1096, Prepared by EIA Subcommittee RS 30.2.

STANDARDS FOR COMPUTER SYSTEMS

2. COMMUNICATION CODE STANDARDS

CHARACTER CODE SETS

- ASCII
- Hollerith
- EBCDIC
- Numerical Control

CODE CONVERSION PROBLEMS

COLLATING SEQUENCE PROBLEMS

- Relevance to CAM Systems
- Relevance to Software Portability

RECOMMENDATIONS ON CODING

PROTECTION OF CAM DATA BY ENCRYPTION

RECOMMENDATION ON ENCRYPTION

CHARACTER CODE SETS

Successful implementation of modular computer/communications equipment requires well-defined interface specifications to accomplish the successful interchange of control signals and data between the various modules.

For adjacent equipment, the interface may signal each control function on a separate wire, and the data may appear as parallel signalling of bits on many wires. Thus, the interface may contain many wires. Some microcomputer bus-type interfaces employ 100 wires.

Where great distances are involved, the entire interface is reduced to two or four wires or a single microwave beam, and the control and data are accomplished by a stream of bits. Groups of successive bits may represent characters, so that the bit stream groups (usually 5 to 11 bits) represent a character stream. It has been shown that a stream of characters coded according to a recognized standard is the most certain of achieving successful interchange among dissimilar computers.(1)

ASCII Standard Code

In the United States, the standard coded character set is the American Standard Code for Information Interchange (ASCII), ANSI Standard X3.4-1968, also adopted as FIPS PUB 1. Internationally the standard is similar to ASCII and is ISO-646 or CCITT V.3, International Alphabet No. 5. ASCII and its international counterparts are defined as 7-bit codes, having 128 characters. An 8-bit version, having 256 characters, is being developed along the lines described in code extension standards, such as ANSI X3.14-1974, FIPS PUB 35, and ISO 2022 as well as ECMA-35. All of these code extension standards are similar, having been coordinated internationally.

In the numerical control (NC) or computer-aided manufacturing (CAM) areas, the 128 characters of ASCII appear to be adequate. The EIA standard RS-358 is a subset of ASCII for numerical control employing less than half of the ASCII characters. However, many computers represent characters as 8-bit "bytes." In 8-bit environments, ASCII characters should be represented in a standard manner, according to FIPS PUB 35 (ANSI X3.41-1974).

Well-defined ANSI standard interfaces exist for the reading, writing, or representation of ASCII characters on paper tapes, magnetic tapes on reels, cassettes or cartridges, and Hollerith punched cards. In all of these media, the ASCII code should be used as prescribed in these various ANSI standards or pending ANSI standards.

Hollerith Standard Code

The Hollerith Punched Card Code Standard, ANSI X3.26 (FIPS PUB 14) was adopted in 1970 and specifies 256 different hole patterns for twelve row punched cards. Hole patterns include the 128 characters of the ASCII Code, ANSI X3.4-1968 (FIPS PUB 1) plus 128 additional patterns.

EBCDIC Standard Code

ECBDIC is the Extended Binary Coded Decimal Interchange Code defined in IBM Corporate Systems Standard 3-3320-022. The standard specifies the BCD coded representation of up to 256 characters used on IBM 360, 370, System 3 and System 32 computers.

Numerical Control Codes

In the area of character codes for numerical control of machine tools two coding conventions are in popular and widespread use. The older "EIA" code defined by EIA RS-244A of January 1967 is an odd parity code of 52 identifiable characters. This code was that used by Flexowriters in common use in the early days of NC for the preparation of NC control tapes. The newer "ASCII" code is defined by EIA RS-358 of July 1968. It specifies an even parity code for the same character set which is a subset of the full ASCII code.

Originally, both of these standards were recognized and in conflict. More recently the older "EIA" code has been rescinded. Still there exist many numerically controlled machine tools capable of interpreting only the "EIA" code. Newer control units are generally supplied with the ability to read either input coding option.

One slight variation of the "ASCII" coding scheme rapidly gaining acceptance is described in EIA Standards Proposal 1177-A. Recognizing that there is a need for two distinct types of data at the machine tool site, the standards proposal defines a Type 1 and Type 2 data on the input media. Type 1 is the traditional machine program data codes in accordance with EIA RS-358 as above. Type 2 data contains machine set-up instructions, initialization and operational parameter data coded in the full ASCII code. Thus there are three coding schemes prevalent on the input media for numerical machine controllers.

It is expected that future systems operating in a CNC or DNC environment will implement whatever final version of EIA SP1177A is adopted. The Air Force should use this standard for command of any NC tools involved in the ICAM program.

CODE CONVERSION PROBLEMS

Conversion of non-standard codes to and from ASCII is not always a trivial matter. There is supposedly a defined correspondence between the 256 character positions of 8-bit EBCDIC and the 128 character positions of 7-bit ASCII or the 256 character positions in 8-bit ASCII.

The entire basis for correspondence is made possible by the Hollerith Code. That is, both ASCII and EBCDIC representations on a punched card are well defined.

The Hollerith Punched Card Code Standard, ANSI X3.26-1970 (FIPS PUB 14) provides 256 hole patterns mapped into 8-bit ASCII in Table 1 of the Hollerith Standard. These same 256 hole patterns are shown mapped into EBCDIC in Appendix B, which is not part of the Hollerith Standards, but is included there for information. There is thus established a 1 to 1 to 1 correspondence between 256 card hole patterns, 256 ASCII bit patterns, and 256 EBCDIC bit patterns. However, IBM practice does not adhere fully to this correspondence somewhat spoiling the 1 to 1 mapping between EBCDIC and ASCII.

Some EBCDIC control and graphic characters are not contained in ASCII. However, IBM chooses to map these, via the Hollerith Punched Card Code, into ASCII character positions, rather than into the 128 available non-ASCII character positions. EBCDIC equivalent characters to those displaced are mapped elsewhere, and the correspondence is thus spoiled. Selected examples are shown in Figure 1.

There are four interchange separators in EBCDIC which correspond to the four information separators of ASCII. Only IFS and IRS are shown because of the possible confusion of EBCDIC FS (Field Separator) with ASCII FS (File Separator), and of EBCDIC RS (Reader Stop) with ASCII RS (Record Separator).

The EBCDIC square brackets are not shown in the principal defining table (Table IV of the CSS) but are shown as publishing and printing graphic options (in Table VII). There is no Cent Sign in ASCII. IBM has chosen to displace the ASCII Opening Bracket by the EBCDIC Cent Sign. However, in representing the ISO 7-Bit Code of ISO 646-1973, IBM drops the Cent Sign and uses the Hollerith hole pattern 12-8-2 to represent the ISO Left Square Bracket, as shown in Table X of the IBM CSS, as a displacement of a national use symbol.

The substitutions in ASCII of the symbols for Logical Or and Logical Not are permitted in the ASCII standard (FIPS 1/ANSI X3.4-1968). This was done by IBM in order to get all 60 of the PL/I language symbols into the 64 character subset of FIPS PUB 15. The ASCII Exclamation Point is displaced. The EBCDIC Exclamation Point then displaces an ASCII Bracket, and a ripple of confusion follows, as was shown in Figure 1.

It is important to note that these problems in code conversion only occur whenever characters are required to cross an interface. In these cases the coding of characters should adhere strictly to the ASCII Standard. This is true regardless of when characters are enveloped in a "code independent frame" or are represented in serial-by-bit form or in parallel-by-bit form. Internal computer codes, if different than ASCII, such as EBCDIC, should not be allowed to cross such interfaces. In this way the Air Force should not encounter problems with character coding in the multi vender, distributed, integrated computer system that is envisioned for the 1980's.

COLLATING SEQUENCE PROBLEMS

An even more serious problem than code conversion arises from differences in the collating sequence embedded in various coded character sets. The collating sequence in a computer determines

(1) the order of the records in a data file according to the relative binary values of the entries in a "sort key" (does W32 come before or after 37N?)

(2) the results of inequality comparison operations (is ZaQ3 smaller or larger than Z24J?)

As long as one keeps to a single computer system or a network of similar equipment no problems are caused by collating sequence. However, the advent of distributed manufacturing systems opens the prospects of a variety of computer hardware being linked together, of data files on one system being queried by another, and of data files and programs being freely transported between different sites. In this type of environment collating sequence can lead to differing results obtained from identical programs operating on identical data files.

The ASCII standard, ANSI X3.4-1968 (FIPS 1) section 6.3 states; "The relative sequence of any two characters, when used as a basis for collation, is defined by their binary values." The IBM Corporate Systems Standard [illegible]-3220-002 for EBCDIC states in Section 1.1 that it defines a collating [illegible]quence. ANSI Standard X3.27-1969, Magnetic Tape Labels for Information [illegible]terchange also provides guidance on structuring data files.

CONTROL AND GRAPHIC CHARACTERS OF IBM EBCDIC WHICH MAP VIA HOLLERITH HOLE PATTERNS INTO 8-BIT ASCII IN POSSIBLY CONFUSING WAYS

IBM Name of Character	IBM EBCDIC Character	IBM EBCDIC Position Hex. Col. Row	Standard Hollerith Hole Pattern	Corresponding ASCII Position Column/Row	ASCII Symbol	ASCII Name
Interchange File Separator	IFS	1C	11-9-8-4	01/12	FS	File Separator
Field Separator	FS	22	0-9-2	08/2	None	None
Interchange Record Separator	IRS	1E	11-9-8-6	01/14	RS	Record Separator
Reader Stop	RS	35	9-5	09/5	None	None
Tape Mark	TM	13	11-9-3	01/3	DC3	Device Control 3
Cent Sign	¢	4A	12-8-2	05/11	[	Opening Bracket
Open Square Bracket	[	AD	11-0-8-5	13/5	None	None
Close Square Bracket	]	BD	12-11-0-8-5	14/5	None	None
Exclamation Point	!	5A	12-8-2	05/13	]	Closing Bracket
Logical Or	|	4F	12-8-7	02/1	!	Exclamation Point
Logical Not	¬	5F	11-8-7	05/14	^	Circumflex

Figure 1

ASCII and EBCDIC define different collating sequences. The ASCII collating sequence, in general terms is "Space," Special Symbols, Numbers, Capital Letters, Small Letters. The EBCDIC collating sequence in general terms is "Space," Special Symbols, Small Letters, Capital Letters, Numbers.

ASCII collating sequence is defined in FIPS PUB 1 and 7, according to the ASCII standard, ANSI X3.4-1968. EBCDIC is defined only in IBM Corporate Systems Standard 3-3220-002. A draft revision to FIPS PUB 1 and 7, published in the Federal Register on December 29, 1975, pages 59607-08, "Revised Instructions for Implementing Standard Character Codes and Collating Sequence," strengthens the requirements for the use of ASCII collating sequence.

ASCII is the standard collating sequence in most minicomputers and microprocessors. EBCDIC is the de facto collating sequence in IBM 360, 370, System 3, System 32, and directly compatible computers. ASCII is the de facto collating sequence in all DEC computers, most NCR computers and some UNIVAC and Honeywell computers.

A data file in a computer system usually encompasses a well-defined area of interest, such as a "Payroll File," an "Inventory File," and the like. A "file" contains many"records." In a payroll file, there is a record for each item kept in inventory. The records in a file are kept in a specified sequence, usually determined by a "sort key." The various records are arranged according to the sort key, usually in ascending numerical order or in 26-letter alphabetical order. For simple sort keys, the order is the same no matter what kind of computer is used. Some sort keys may be pure binary numbers having any number of bits. Other sort keys can contain more complex arrays of characters, such as mixed upper and lower case letters, punctuation marks, special symbols as well as decimal digits. For complex sort keys, the order of the records is usually a "default" sequence determined by the native character code of the computer.

Two principal character codes are presently used in computers. One is ASCII (American Standard Code for Information Interchange) as specified by FIPS PUB 1, ANSI X3.4-1968, or ISO 646-1973. The other code is EBCDIC (Extended Binary Coded Decimal Interchange Code) as specified in IBM Corporate Systems Standard 3-3220-002 or variations by other mainframe vendors. The collating sequences of ASCII and EBCDIC are the same for simple sort keys, such as numerics or the 26 capital letters. But for more complex sort keys, the collating sequences are radically different. Computer control function characters are, of course, not used in sort keys. For the graphic characters, the collating sequence of ASCII is from low to high value as follows: "Space," punctuation and special symbols, numbers, capital letters, lower case letters, with some special symbols between these major groups. In EBCDIC, the collating sequence of the graphic characters is: "Space," punctuation and special symbols, lower case letters, capital letters, and numbers, with some special symbols between these major groups.

For the same data file, a sort key using most of the graphic characters of ASCII or EBCDIC would produce a record sequence based upon the collating sequence of ASCII or EBCDIC, unless otherwise specified. These two record sequences would be considerably different. A clerk could learn to use either record sequence as an index, but would have great difficulty transferring from one sequence to the other. This has occurred, for example, in the case of large catalogs arranged by Federal Stock Number (FSN, alphanumeric) ordered according to two different collating sequences. Introducing new Federal Stock Number items into the two "master" files would require that the new records be sorted by FSN according to the collating sequence of each file, and then merged into the master file. Data transferred from one "master" file to the other would require a re-sort of the selected records into the sort sequence of the other before the data merge could occur efficiently.

Relevance to CAM Systems

In the CAM arena, it is not apparent whether there are any difficulties that might result from the use of computers having two different collating sequences. It is possible to postulate some. For example, suppose it were desired to generate an index list of all APT part programs. Should "CAP∆37" come before or after "CAP∆FOR∆COVER" where "∆" represents the character "Space"? Since "CAP∆" is the same for both titles, the sequence would be resolved by whether "3" is smaller (lower in the collating sequence) or larger (higher in the collating sequence) than "F". In ASCII collating sequence, numbers are lower than letters, so that "CAP∆37" would precede "CAP∆FOR∆COVER." In EBCDIC collating sequence, numbers are higher than letters and hence "CAP∆37" would follow "CAP∆FOR∆COVER." The point is that, in a large index, a programmer might miss the existence of a desired program if the index had been collated on one machine and was being searched on another.

A standard collating sequence in the CAM area would be preferable to a mixture, sometimes ASCII and sometimes EBCDIC.

The following simple example illustrates the differences between ASCII and EBCDIC collating sequence. A sort key contains only two character positions and the complete character set is comparised of the four characters 1, 9, A, Z. The complete collating sequences are:

Sequence Number	ASCII	EBCDIC
1	11	AA
2	19	AZ
3	1A	A1
4	1Z	A9
5	91	ZA
6	99	ZZ
7	9A	Z1
8	9Z	Z9
9	A1	1A
10	A9	1Z
11	AA	11
12	AZ	19
13	Z1	9A
14	Z9	9Z
15	ZA	91
16	ZZ	99

It can be seen that in a large file index, a clerk would have difficulty locating a particular item without knowledge of the collating sequence.

If both capital letters and small letters are allowed in a sort key, then the confusion would be even greater, since in ASCII capital "Z" collates ahead of small "a," while in EBCDIC small "z" collates ahead of capital "A".

In an alphanumeric sort key, if certain positions are always numeric and other positions are always alphabetic (capital letters or small letters but not both), then the collating sequence will be the same in ASCII or EBCDIC. Thus in the example above, if the first position is always numeric and the second position always alphabetic, the complete sequence will be:

Sequence Number	ASCII or EBCDIC
1	1A
2	1Z
3	9A
4	9Z

It can be seen from the 16-sequence table that the sequence of four does appear in the same sequence in the ASCII or the EBCDIC column. This uniformity in collating sequence is achieved by a constraint on the sort key which greatly reduces the number of keys (records) that can be represented by a given number of positions in the sort key. Consistent use of the ASCII collating sequence will remove the need for such simplifying constraints on sort keys, and will eliminate variations in the sequencing of complex sort keys, and will also give consistent results for computer program comparison operations.

Revelance to Software Portability

Comparison operations in computer programs generally compare one group of characters with another group of characters. If the groups of characters are "simple," such as numerics or 26 letters, then the results of the comparisons will be the same whether the character coding is in ASCII or in EBCDIC. However, if the character groups to be compared are more complex, then the inequality of the two groups can indicate that the former is "larger" in ASCII but "smaller" in EBCDIC. Computer programs in high-level languages, employing such comparisons, can thus give different results in ASCII or EBCDIC, because of the difference in the collating sequences of ASCII or EBCDIC. A standard collating sequence would eliminate this complication along with the sort key sequencing inconsistencies.

In the original COBOL programming language standard, the collating sequence was indicated to be whatever the computer vendor specified. As a consequence, some COBOL programs could, and did, give different results on different computer systems. This had the effect of spoiling the transferability of COBOL programs among various computers, although such transferability was claimed to be one of the advantages of using high-level programming languages. To overcome this disadvantage, the COBOL standard (FIPS PUB 21-1, ANSI X3.23-1974) has been modified to allow the programmer to specify the collating sequence.

SUMMARY OF RECOMMENDATIONS ON CODING

a) It is recommended that the USAF use the FIPS 1 ASCII coding of character set data wherever information crosses an interface between a CAM module and any other CAM, computer or communications module or device.

b) It is recommended that the USAF use the ASCII subset of EIA Standard RS-358 for Numerical Control applications and adopt the "type 1"/"type"2 data conventions of SP177A before it becomes a standard.

c) It is recommended that the USAF use the recognized FIPS/ANSI standard representations of ASCII in media, such as paper tapes, magnetic tapes, punched cards, cassettes, and cartridges.

d) It is recommended that the USAF represent 7-bit ASCII in a standard manner in 8-bit environments, according to FIPS PUB 35/ANSI X3.41-1974.

e) It is recommended that the USAF represent any extensions of ASCII in a standard manner in accordance with FIPS PUB 35/ANSI X3.41-1974.

f) It is recommended that the USAF use the ASCII collating sequence for sequencing file records according to sort keys.

g) It is recommended that the USAF use the ASCII collating sequence for determining the results of comparison operations in computer programs.

PROTECTION OF CAM DATA BY ENCRYPTION

In most CAM applications, no special protection of the data will be required and none should be used. In some cases, protection may be deemed important. If CAM data is to be transmitted by military communications, then military data encryption techniques should suffice. If CAM data protection is desired but military communications are not involved, then such protection can, and should be, accomplished by means of the NBS Data Encryption Standard.

The NBS Data Encryption Standards (DES) algorithm specifies the encryption of 64 bits of data into a 64 bit cipher based on a 64 bit key and the decryption of a 64 bit cipher block into a 64 bit data block based on a 64 bit key. The steps and the tables of the algorithm are completely specified and no options are left in the algorithm itself. Variations in implementing and using the algorithm provide flexibility as to the application of the algorithm in various places in a computer system or network, how the input if formatted, whether the data itself or some other source of input is used for the algorithm, how the key is generated and distributed, how often the key is changed, etc. These issues are covered in a separate NBS guideline.

Basic implementation of the algorithm is most easily done in special purpose electronic devices. Overall security is based on two primary requirements when using the DES algorithm: secrecy of the encryption key and reliable functioning of the algorithm. Implementation of the algorithm in dedicated electronic devices provides the following economic and security benefits:

1) Efficiency of algorithm operation is much higher in specialized electronic devices.

2) Basic implementation of the algorithm in specialized LSI electronic devices which can be used in many applications and environments will result in cost savings through high volume production.

3) Functional operation of the device may be tested and validated independent of the environment.

4) The encryption key may be entered (or entered and decrypted) into the device and stored there and hence never need appear elsewhere in the computer system.

5) The paths of data to and from the device may be controlled and monitored.

6) Unauthorized modification of the algorithm is very difficult in such a device.

7) Redundant devices may simultaneously perform the algorithm independently and the output may be tested before cipher is transmitted.

8) The device can be controlled externally in accordance with the requirements and environment of the application.

9) Implementation in special purpose devices (electronic devices or dedicated micro processing computers) will satisfy Government requirements for compliance with the standard.

RECOMMENDATION ON ENCRYPTION

Wherever security is needed in interchange of CAM information, the NBS Data Encryption Standard algorithm should be applied, unless its use is superseded by military communications requirements.

REFERENCES

(1) "Information Interchange Between Dissimilar Systems" by H.S. Meltzer and H.F. Ickes, Modern Data, Vol. 4, No. 4, April 1971, pp. 56-67.

(2) Hollerith Punched Card Code Federal Informat Processing Standards FIPS PUB 14 1970 U.S. Department of Commerce, National Bureau of Standards.

(3) Extended Binary Coded Decimal Interchange Code IBM Corporate Systems Standard CSS 3-3220-002, November 1970, IBM Systems Standards Department, Poughkeepsie, New York.

(4) Proposed Federal Information Processing Data Encryption Standard 1 Aug. 1975 Federal Register.

(5) Guidelines for Implementing and Using the NBS Data Encryption Standard, Draft Document, 10 Nov. 1975 Institute for Computer Sciences and Technology, National Bureau of Standards.

STANDARDS FOR COMPUTING SYSTEMS

3. PROGRAMMING LANGUAGE STANDARDS

INTRODUCTION

Programming languages serve the same purposes for computing as spoken languages do for human communications. They are the principal mechanisms by which ideas (algorithms), data, commands, response requirements, etc. are communicated from man to machine.

Like spoken languages, they have a tendency to diverge into "dialects," in which case users of different forms of the language find it difficult or impossible to continue communicating with each other. A cooperative standardization effort is frequently required in order to get the various dialects to converge acceptably, since the language compilers can not adapt to slight variations in use as can humans.

Programming language variations are inevitable and in many instances they are desirable, because through them better or entirely new forms of useful expression arise. The "better" forms are perhaps the more dangerous from a communication point of view because, if adopted, they must either supersede the older forms or introduce a redundancy into the language; in either case, considerable attention must be accorded these types of changes, as they constitute deviations from the approved language definition and threaten software portability.

New forms, or "unilateral extensions," are usually outside of the previously defined scope of the language and require some time to be defined, implemented, tested, understood by others and accepted into the language. As a result, they do not pose as immediate a threat to program portability as do the "better" forms. However, consideration must be given to the manner in which new forms are defined and employed in the building of application programs, so that users will be aware that the use of these new forms prevents them from creating portable code, at least until such time as the new form is accepted into the language definition.

STANDARDS ON EXISTING LANGUAGES

There are currently standards in existence or in the process of approval for four general purpose programming languages: FORTRAN, COBOL, PL/I, and BASIC. Of these languages, only PL/I is a "modern" language that has the potential for satisfying the requirements of the Air Force for a general purpose programming language for the CAM program. The choice of a general purpose programming language is not clear as will be shown. In fact, the language chosen by the Air Force for the ICAM program may not be any of these four discussed, although support for at least COBOL and FORTRAN is mandatory for the near future because of the body of existing programs in these languages.

Although ALGOL is mentioned several times in the following text, there is no formal standard and or standards committee for ALGOL and it is considered in terms of historical interest and the heritage it has brought to other languages. Current use of ALGOL is sufficiently limited that it is not considered comparable, even as a de facto standard, to the other languages discussed here.

Independent of which language standard is selected the Air Force must realize that the simple specification of a standard language in a procurement action will not be sufficient. Indeed an entire set of software development and documentation guidelines and validation and testing tools are mandatory to meet Air Force goals, as will be discussed below.

Only general purpose programming languages are addressed here. For specific

problem oriented needs, such as simulation or artificial intelligence, there are languages but no standards or defacto standards. It is believed that existing languages and compilers can be selected to satisfy ICAM project requirements when they are set.

FORTRAN

Originally designed in the early 1950's as a replacement for assembly code, FORTRAN is a simple higher level language that is easy to compile into machine code. However, the requirements of this efficiency have extracted a price which is paid for in annoying restrictions which crop up in use of the language. FORTRAN statements tend to reflect the hardware characteristics of the first machine to support FORTRAN. The memorable fact that every DO is always done at least once is an example. This is due to the fact that the original machine for the language has a test-and-jump instruction which worked by testing at the end of loops, rather that at their entry points.

FORTRAN lacks many features often expected of general purpose languages. Part of the omission is simply because the language is so old, about twentyfive years. Nonetheless, a user of ANS (or Standard) FORTRAN can not expect the following: good string handling; block structure; run-time allocation of space. FORTRAN's virtue is that it is simple and effective, and much preferable to assembly code; this point is important, because for many uses the competition is not other modern languages such as PL/I and PASCAL, but rather, machine code. FORTRAN in conjunction with an optimizing compiler can be very fast.

Elaborate libraries exist of FORTRAN engineering and scientific routines. In addition, techniques are available [Larmouth, 1976] which can stretch the design features a bit to circumvent the more annoying restrictions such as space allocation of arrays. Unlike BASIC, FORTRAN is sufficiently rich in coherent control structures that it can be sensibly used for large-scale implementations. The language is well suited to industrial applications which involve complex numerical calculations such as table interpolations, function integrations, or measurement smoothings and averaging. This is in contrast to COBOL which is rather inefficient and clumsy to use for scientific or engineering evaluations. (FORTRAN is not suited, on the other hand, for generating in a straightforward manner nicely formatted reports of a commercial nature.) FORTRAN input/output is both limited and slow. But as an interim scientific and engineering language for CAM, FORTRAN could serve nicely.

The new FORTRAN Standard, long in gestation, was released in draft form in early 1976. There was an avalanche of criticism--mostly that it should contain each critic's favorite structure--but it appears that debate will be cut off with the addition of IF-ELSE IF--END IF and perhaps STREAM input/output. Committee members hope to have solidified a new Standard by March 1977.

COBOL

COBOL was originally conceived as a business language for commercial data processing. It is an effective means for programming applications that are characterized by the requirement to manipulate characters, records, files and input/output (as contrasted with those concerned primarily with computational problem solving). Quoting Pratt, "COBOL is perhaps the most widely implemented of the languages...[See Pratt's book for the context of this.]... but few of its design concepts have had a significant influence on later languages, with the exception of PL/I. Both of these facts may be partially

attributed to its orientation toward business data processing, a major area of computer application, but one in which the problems are of a somewhat unique character: relatively simple algorithms coupled with high-volume input-output..."[Pratt, 1975,p. 359]

Like some other language of the same period, COBOL was developed and has been maintained by voluntary efforts of implementors and users. The COBOL standard, as is the case with any standard, does not in itself cure all problems associated with computer systems. As the language is used, its flaws and inadequacies become more apparent; action must be taken to correct, adjust and extend the standard definition.

There exists a rather elaborate mechanism dedicated to the continuing process of making COBOL evolve in response to user requirements. In addition, to enhance the viability of Standard COBOL as a tool, ancillary activities have been initiated to provide for testing of compilers for conformance to the standard, for interpretation of the language specification when questions of meaning arise and for development and establishment of policies relative to procurement and testing of COBOL compilers.

In a recent survey (NBSIR 76-1100), of the 132 Federal government computer installations responding to the survey question concerning usage of COBOL, 86.4% indicated that COBOL was available and 94.7% of those who had access to the language actually used COBOL to some extent. (Also see Phillippakis [1973].) A few examples of COBOL applications illustrate potential uses of COBOL within a CAM system. For example:

a. The National Weather Service, an agency of the Department of Commerce, has an operational on-line system providing weather forecast information. Approximately 30 terminals throughout the nation receive and send weather forecast information. Among the users are civilian and military agencies and radio and TV stations. The system is written in various languages; however, three to four dozen COBOL programs accomplish an important function in the system. These COBOL programs perform editing of input data for errors and formatting the data for its presentation over the network. COBOL was selected for use in implementing these programs because of its ability to handle editing and character manipulation.

b. The Defense Supply Agency (DSA), an agency of the Department of Defense, performs central supply service to all Defense agencies. It provides support materiel such as food, medical supplies, clothing and construction material. DSA has a very large logistics system called SAMMS (Standard Automated Materiel Management System) written in COBOL. SAMMS provides the following daily functions for DSA: distribution, requirements forecasting, financial management, procurement, and cataloging. This system is used in each of DSA's five major centers: Richmond, Virginia; Columbus, Ohio; Dayton, Ohio and two in Philadelphia, Pennsylvania. There are 400 to 500 individual reports produced by SAMMS. Examples of some of the reports are management reports, statistical reports, rejection reports, exception reports, and turn-around (time requirement) reports. The system requires about 1000 changes per year, mostly enchancements, because of changing requirements. The number of records in the system varies in each center from 800,000 to 1,500,000; approximately 12,000 records are updated per hour. With some 800 to 1000 COBOL programs, SAMMS is the largest logistics data system in the Federal government and is integral to DSA's daily operations. COBOL was chosen for implementing this system to enhance the portability of the programs and because of the attributes of COBOL for handling character data.

It is evident from the efforts pursuing development and standardization of COBOL and from the examples of how COBOL can be used effectively in its

typical application areas, that COBOL should be given serious consideration for use whenever a problem requires straightforward and few computations on numbers of limited range, along with a heavy volume of such numbers with special commercial formatting requirements (such as flush dollar signs).

BASIC

BASIC (Beginners All-Purpose Symbolic Instruction Code) is a computer programming language developed in the mid 1960's by Kemeny and Kurtz at Dartmouth College. The primary motivation behind the design of the language was the desire to educate large numbers of undergraduates in the use of a remote-console, time-shared computer to solve simple problems. Limitations on names to letter+numeral have been retained, even though the original purpose of this--to simplify symbol t ble maintenance--is no longer necessary. The language was designed to be learned and used easily, especially for simple problems. In this respect, the design criterion for BASIC differed from other languages such as FORTRAN (execution speed) or Algol 60 (expressing algorithms) which aimed at professional programmers.

Since its original design, more advanced features have been added to BASIC, both at Dartmouth and at other installations, so that BASIC now is often used as an alternative to FORTRAN or Algol 60. The divergence in the design of advanced features, in addition to divergence even in the features of original BASIC, has been a concern among suppliers and users of BASIC. In response to this concern, ANSI established an ad hoc committee "to investigate the computer programming languages generally known as BASIC, and determine the existence of a viable nucleus language suitable for standardization."

This committee recommended that ANSI create a technical committee charged with developing a standard for BASIC. This recommendation was approved by the committee at its January 17, 1973 meeting.

In addition to identifying and standardizing a nucleus for the BASIC language, the ANSI standards committee X3J2 is investigating advanced features in various implementations and is standardizing other modules as it sees fit. The standards committee felt it preferable to standardize more than just a BASIC nucleus, since the greatest divergence in BASIC implementations occurred in the treatment of features that would most likely not be in the nucleus.

The now proposed standard for MINIMAL BASIC reflects the original design of the language for use in a remote-console, time-shared system, though of course the standard does not preclude the use of BASIC in other modes of operation. In practical terms, the standard does assume that BASIC will be implemented in an environment which provides minor editing services and which emphasizes single-pass, fast compilation and execution rather than compilation of optimized code. Educational uses of programming languages require just such an emphasis.

The proposed American National Standard for Minimal BASIC was approved by X3J2 in January, 1976 and was forwarded to X3 for action. X3 has given the proposed standard the reference BSR X3.60 and has submitted the proposed Minimal Standard for public review. Comments were due by the end of September 1976. This nucleus standard contains those portions of the planned language not specifically contained in planned enhancement modules. Standards for enhancement modules concerning files, strings, matrices, subprograms and chaining, and formatted input/output are under development at present.

From the past experience of the development of the Minimal Standard the completion of the enhancements standardization may take on the order of 2 to 3 additional years. Therefore, a full BASIC standard should be available by 1980 provided the committee does not face any voting deadlocks. These estimates assume the regular committee meeting schedule of 4 meetings a year in the last week of each of the months of January, April, July, and October.

For any large scale effort in CAM systems being able to store, retrieve, and manipulate large sets of data is important. Minimal BASIC is not designed for this although future enhancements will allow file and formatted I/O capabilities. Large scale data handling can be accomplished in a more prominent language such as COBOL or PL/I, and with more probable efficiency.

There was an attempt to specify some minimal working precision for numerical constants and variables of at least 6 digits. However, no accuracy specification is imposed on arithmetic expressions or intrinsic functions. This limitation on precision makes BASIC unusable for some engineering calculations.

From the engineering design point of view many good algorithms for matrix manipulation, solving differential equations, etc. have already been coded and tested and installed through library packages such as IMSL (International Mathematical and Statistical Libraries, Inc.) or the Association for Computing's Collected Algorithms. FORTRAN and Algol 60 are the principal languages used for these existing collections.

The standard allows minimal string capability. No comparison except equal or not equal is allowed between strings. This is another disadvantage to the BASIC standard in its present form.

CAM should rely in the short run at least on languages and design support libraries that have the most wide spread use. Although BASIC has recently become popular the original intent of the language was for the learner to step on to another language; BASIC was not intended as a large scale production language. (The reader may want to reference Pratt, pp.475-476 for a lucid discussion of the demands of an interactive language, in his case APL, and possible detriments to doing large production programming.) Because of this fact, and because of the limitation in the BASIC standard, BASIC is not recommended for use in the ICAM program.

PL/I

PL/I is an extensive, general purpose language which was designed originally to enhance the IBM model 360 series of machines. Statements in the language are FORTRAN-like. In fact, the similarity is close enough that long time FORTRAN programmers are prone to lapse back into FORTRAN when writing some PL/I statements, such as the DO. (The meaning of a FORTRAN DO is not the same in PL/I!)

PL/I program structures are borrowed from ALGOL (e.g., BEGIN-END). Data types in the language show a clear COBOL influence. The language was meant to be a "universal" or omnibus for scientific, commercial and diversified general users. Whether it has met this intent is somewhat open to question. Use of PL/I has not grown nearly as fast as had FORTRAN or COBOL in an earlier period. Because of the very size of the language, the initial

compilers were difficult to write and slow in execution. This problem has been remedied somewhat with time.

The draft standard represents an attempt to simplify the formidable, otherwise very large language. Because of the tentative nature of the draft, it is unlikely that any PL/I processor chosen today would conform to the letter of the new standard. Several PL/I dialects exist, including PL/C [Conway 1973], a student subset with fast compiling, and a systems support version PL/S. PL/I is not limited to IBM implementation even for system work. For example, 95% of Honeywell MULTICS is written in PL/I.

PL/I was accompanied soon after its introduction by a formidable formal model-- the Vienna Definition Language. This meta language, known also as VDL, has been retained in the draft of the standard. The modeling language is not very easy to read, and it remains to be seen whether use of it has removed the threat of ambiguities or omissions in the standard.

Besides the difficult VDL formalism, the PL/I standard has another drawback of not defining allowed subsets of the language. Implementation of the full capabilities of the language therefore requires a compiler that can only be run on large scale computers. Subsets of PL/I have been implemented for developing cross software for microcomputers (PL/M, PL/M6800). More extensive standard subsets could be defined for minicomputers and medium scale computers.

If PL/I is used, the Air Force should specify standard subsets of PL/I for various applications within the context of the ICAM program.

Among languages mentioned in this report, PL/I is one that has potential in the long run as a good growth language for both systems and applications. The dialect PL/S[see below] is used by IBM on some of their systems work; student dialects have been mentioned above. Because it borrows from Algol for block structures, it is fairly easy to write "structured programs" in PL/I; in addition, the COBOL heritage provides a more definite input/output capability than that of, say FORTRAN, or (worst) Algol (where i/o is left undefined). Consideration of PL/I, along with other modern languages such as PASCAL and its extenions (e.g. EUCLID), should be made for longer-term planning in the CAM project. The ANS PL/I with specified subsets and with features of PL/S might be, for example, a good vehicle to write most of the CAM systems software.

SYSTEMS IMPLEMENTATION LANGUAGES

The development of large system software projects, e.g. operating systems, compilers, and data management systems, has been, and still is, hampered by the lack of adequate tools. The most important of these tools is a good high level systems implementation language (SIL). (The term systems programming language can be used interchangably.)

Despite the lack of a SIL that can be considered to be really good, the use of existing SILS is preferable to the implementation of system software in assembly language or macro-assembly language. If the resulting compiled code fails to meet execution time constraints, critical inner loops can be recoded in assembly language. If practical, they should not be placed in-line, but rather grouped together in a seperate module (or modules) and referenced

through procedure calls. This will isolate machine dependent code to enhance portability.

Of the existing SILS, there does not currently exist one that possesses a clear advantage over all others. Some notable attempts have been made in SIL design and implementation, but the resulting languages have nearly always been targeted to a single vendor's machine architecture or have not achieved widespead use. As mentioned above, a dialect of PL/I known as PL/S has been used internally by IBM to implement much of their system software. A dialect of Algol has been used by Burroughs in the same manner. Many other systems implementation languages have been developed but have not seen widespread use because of machine architecture dependencies. Several SILs have been designed specifically for microprocessors. There is obviously little incentive for a vendor to develop a systems implentation language that could be readily used to implement systems for another vendor's machines. Thus, if there is to be any movement to more machine independent systems implementation languages, that movement must come from without the mainframe vendors. The Air Force could provide that impetus.

It may be possible to avoid developing a special SIL. For example, 95% of Honeywell's MULTICS is reported to be written in PL/I, the rest in assembly language. The key features of a SIL are the ability to manipulate data at the physical level, rather than at the logical level, and to execute privaleged calls to the hardware. Implementation of a good data base management system and adequately standardized general purpose programming languages may be sufficient for Air Force needs in providing data manipulation and portable software.

The specification and initial design of a machine independent systems implementation language (DOD-1) is currently underway in the Department of Defense for use in system programming of weapons systems [Fisher, 1976]. Although its use is not mandated for general purpose, commercial computer systems, it may prove to be a good choice [DOD, 1976].

In summary, the Air Force must have a SIL. The choice is to pick one or develop one. There is no clear choice between the SILS that exist and have been implemented. With the possible exception of PL/S, a proprietary dialect of PL/I, the potential availability of PL/S should be investigated by the Air Force. Development of a adequate SIL may be the only choice; in this regard, the DOD-1 language effort should be carefully evaluated before an independent development effort is begun under the ICAM program.

FUTURE NEEDS IN PROGRAMMING LANGUAGES

General observations

C.A.R. Hoare[1973] has stated that a programming language should aid in program design, program documentation, and program debugging. He goes on to stress language simplicity, security, fast translation, efficient object code, and readability. (His paper also includes a very interesting annotated bibliography on some common languages, including FORTRAN, ALGOL 60, and COBOL.)

Documentation can be helped by syntactic forms in a programming language, or equally, hindered. Indeed, something as simple as a comment can be more (or less) useful in encouraging clear programs. Scowen and Wichmann[1974] review a number of comment conventions, including those in PL/I, ALGOL 60, FORTRAN, BASIC, and COBOL. They provide six design criteria for comments.

Program debugging occupies a sizeable portion of a programmmer's time and language features can be important. For example, data types in a language can help prevent improper transformations between disparate entities. However, a data-typing feature is defeated by an automatic , transparent type conversion (a la older PL/I), which may then require extremely tedious examinations of identifiers for improper type. Unchecked array bounds provide another very common source of error that can be difficult to catch without help from a compiler.

Although structured programming and related methods have met resistance in the programming community, the ideas are nonetheless attractive [Lucas, 1976; Yourdon, 1976-Chap.4]. Perhaps the situation would be different if programs were physical things which could be viewed for balance and workmanship [Cheatham, 1971]. Programmers may argue for complete latitude in connecting pieces of programs together; however imagine a carpenter who set wall studs sometimes at 16" apart, sometimes 14", and if his lumber was warped, at varying distances. He could argue that his buildings were no weaker than anyone else's, but the insulation workers would rate his handiwork less favorably, since standard batts are 16" wide.

The possibilities for connecting N points of a program are of order N*N. If nothing else the various structuring and programming refinement disciplines seek to introduce some constraint upon this potentially huge N*N. The most notorious restriction has been, of course, E. Dijkstra's condemnation of GOTO's [Dijkstra, 1968]. His point --quite valid--was that GOTOs represented a way of thinking about programming, that many GOTOs indicated shoddy program organization--a "Rube Goldberg" programmer in action. It was not enough that a program worked--so did most of Goldberg's bizarre inventions. The programming task should be thought through as one might organize an essay.

Yet even after the organization of a program has been expressed, it must be written in terms of some programming language. While an organization may reflect the virtues of modular pieces and good, tree-like dependencies among modules, it is equally clear that some languages will not allow one to ban GOTOs easily. COBOL and FORTRAN have control statements dependent upon GOTOs; for example, in COBOL the EXIT statement is, effectively, only an exit label at the end of the scope of a PERFORM; interior "exits" must GO TO this one valid point of egress. Any interior EXITS are treated as no-ops, and do not affect the PERFORM. And since FORTRAN has no compound statements, GOTOs are often introduced to produce the effect. More modern programming languages often include compound statements, conditionals, a DO or FOR statment, WHILEs, the CASE statement, and naturally, procedures including recursive ones.

A second place for a program to become unbuttoned is in its data; Hoare[1973] observes that untyped pointers allow as much arbitrary hazard in the data space of a program as GOTOs pose in the program (or control) space. A pointer can jump around, and if assigned an improper value, jump around into the wrong data locations. On an even simplier vein, it is possible to replace GOTOs by flags, only to find that the flags are so poorly designed that their meaning is dependent upon points of control in the program.

The moral is, if a programmer is messy, nothing will help.

Standards and Limitations

Any discussion on standardized languages and their status could be deceptive if unaccompanied by a caveat on the limitations of the language standards themselves, for in fact there are many system influences on language use, and in the wordings of the standards.

Larmouth[1976] provides details of many loose ends in FORTRAN. For example, local variables in a subroutine can become undefined (of indeterminate value) upon exit from the subroutine, even though most systems preserve local variables, treating them as Algol OWN values or PL/I STATIC. The reason for the Standard's hedging is that on stack-machines, such as Burroughs, the subroutine exit pops the storage stack. Local values are truly lost. On another plane, the recent problem with COMPUTE in COBOL was caused by a failure in the standard to define intermediate results for arithmetic expression evaluation. Some manufacturers used their machines' double precision floating point for the intermediate results, while others incorporated the various numbers of fixed point digits. It is impossible to state concisely all of the problems that one might encounter in a particular standard. The best advice would seem to be to refer the reader to the Larmouth article and indicate that the FORTRAN standard that generated all that discussion is about a tenth that of, say COBOL or PL/I Standards. Caveat emptor.

Files and the handling of system secondary storage exemplify the importance of uniform, simple conventions, especially among programs written in different languages. The dictum of "delayed binding", i.e. late fastening of attributes, implies that files should have no specific characteristics other than those absolutely necessary. This allows flexibility in rerouting inputs and outputs, typical requests for contemporary users. Usually there will be loadable files and text. Nothing else. Text, if sent to the printer process, generates--on paper-- a user's print file. It is not difficult to cite systems in which there are user card files, printer files, data files, and program source code files. For example, on the UNIVAC 1108 under EXEC II, it is quite easy to find that one has a COBOL preprocessor, written in COBOL to convert other COBOL decks, whose output is unreadable by the COBOL compiler! Gerhard Goos [1974] has remarked that:

"The most serious problem of today's system programming languages is the non-existence of a basic model for file-handling and I/O. All models either are developed with a certain operating system in mind and are difficult to adapt to other operating systems. Or they are too simple, allowing for sequential files only while random-devices are modeled by unstructured linear address spaces."

Much as one would like programs written in various languages to share files, one would also like to share library routines. K.W. Morton [1974] discusses the NAG library and practical limits in current operating systems; e.g. to serve both FORTRAN and Algol users some routines have to be coded twice. Hoare [1973] also reflects on the point briefly, and is not generally in favor of shared routines.

In any event, while specification of a standard in a language will improve compatibilities, such standards may require additional constraints to be really useful. This is especially true if distinct programming languages are to share the processing of file information on the system.

RECOMMENDATIONS

1. CAM systems and application software packages should be developed only with high level programming languages, except for the very few instances where acceptable performance can only be achieved by resorting to assembly language for coding of critical algorithms. These cases should be carefully controlled and documented.

2. The Air Force should encourage the use of standardized programming languages. NBS believes their effective use to be the key to software portability.

3. ICAM may not wish to prohibit the use of nonstandard programming languages where the reasons for their selection by a contractor are fully documented and supported. In those cases where the Air Force allows the use of a nonstandard language, it should at the same time initiate a standardization effort to formalize the product definition, through a consensus opinion of users and suppliers, so that compilers can be implemented on other computers to effect portability.

4. Because of the bulk of existing application programs are written in FORTRAN and COBOL, these two languages must be supported for the near term future in the Air Force ICAM program. Eventual conversion of existing programs to a modern language should be planned for under the ICAM program. At the present time FORTRAN and COBOL are the only two general purpose programming languages that are considered to be immediately useful to the Air Force.

5. The Air Force should support the establishment of a Federal FORTRAN standard based upon revision of the ANSI standard, now in progress. Should ANSI fail to approve a revised standard in 1977, the Air Force should support in writing the NBS goal of adopting the next ANSI committee proposal as a Federal standard.

6. Of all the general purpose programming languages submitted for standardization, PL/I is the only one that can be considered a "modern" language suited for Air Force ICAM applications. However, PL/I compilers can produce inefficient code and tend to require a large run-time support system. Furthermore, not all of the major computer manufacturers offer PL/I. Hence, it cannot yet be considered a "standard" language suitable for Air Force use. If it is desired to use PL/I, substantial effort in standardization will be required and particular attention should be given to the definition of subsets to run on smaller computers and to the development of extensions for systems work.

7. The Air Force CAM authorities should monitor the DOD-1 project because it appears to have the broad base of support that could produce a standardized language suitable for CAM needs in the 1980's. Among the candidates being considered in the DOD-1 effort that are particularly relevant to CAM projects are PASCAL and PL/1.

REFERENCES

Andreas S, Phillippakis. "Programming language usage."
ANSI BSR X3.53 Basis/1-12 1975(Feb.) Draft Proposed Standard Programming Language.

BSR X3.53 Chap 1 revised 1976(Feb).

BSR X3.53 Errata Sheet 1976(Jan.)

BSR X3.60

M. Beckmann, et. al. ADVANCED COURSE IN SOFTWARE ENGINEERING. Springer-Verlag (Berlin, 1974).

Bennet P. Leintz, "A Comparative Evaluation of Versions of BASIC," Comm. of the ACM, April 1976, Vol. 19, No. 4, pp. 175-188.

T.E. Cheatham. "The recent evolution of programming languages," Proceedings, IFIP Congress 71, Ljubljana, Yugoslavia, August 1971, pp I-118 -- I-134.

R. Conway and T . Wilcox. "Design and implementation of a diagnostic compiler for PL/I." Comm. ACM. 16, 3(March 1973), 169-179.

Donald R. Deutsch. Appraisal of Federal Government Cobol Standards and Software Management: Survey Results. NBSIR 76-1100, Final Report August 1976, U.S. Dept. of Commerce, National Bureau of Standards. DATAMATION, October 1973, 109-111.

E. Dijkstra. "GO TO statement considered harmful," Letter to editor of COMM. ACM 11, 3(March 1968), pp 147-148.

DoD Directive 5000.29, Management of Computer Resources in Major Defense Systems, 1976(April).

D.A. Fisher, A Common Programming Language for the Department of Defense -- Background and Technical Requirements, 1976(June).

M. Griffiths. "Relationship between definition of implementation of a language," Lecture Notes, op cit.

G. Goos. "Programming languages as tool in writing system software," in Beckmann, et.al, op.cit.

C.A.R. Hoare. "Hints on prgramming language design." Computer Science Department, Stanford University, Dec 1973, STAN-CS-73-403, 29 pp.

D.E. Knuth. "Structured programming with GO TO statments," COMPUTING SURVEYS 6, 4(December 1974), 263-301. (Special issue on programming.)

J. Larmouth. "Serious FORTRAN--The Rules of the Game." Chapter to appear as an NBS Tech Note for the NBS/NSF SOFTWARE ENGINEERING HANDBOOK, July 1976, 20pp. (An earlier version appeared in SOFTWARE--PRACTICE & EXPERIENCE)

Larmouth, J. "Serious FORTRAN," and "Serious FORTRAN-- Part Two." SOFTWARE: Practice & Experience 3, 2-3(1973), pp. 87-108, 197-225. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975.

H. Lucas Jr. and R.B. Kaplan. "Structured programming experiment," COMPUTER JOURNAL 19, 2(1976), pp.136-138.

K.W. Morton. "What the software engineer can do for the computer user," in Beckmann, et.al., op. cit.

Pratt, T.W. Programming Languages: Design and Implementation.

R.S. Scowen and B.A. Wichmann. "The definition of comments in programming languages," SOFTWARE--PRACTICE & Also see J.G.P. Barnes, op.cit.,pp.401-408.

N. Wirth. "On the composition of well-structured programs," COMPUTING SURVEYS 6, 4(December 1974),pp.247-262. (Special issue on programming.)

D.B. Wortman, et.al. "Six PL/I Compilers," SOFTWARE: PRACTICE & EXPERIENCE 6, (1976), pp. 411-422.

E. Yourdon. TECHNIQUES OF PROGRAM STRUCTURE AND DESIGN. Prentice-Hall, Inc. (Englewood Cliffs, N.J., 1976), 364pp.

STANDARDS FOR COMPUTER SYSTEMS

4. OPERATING SYSTEMS

INTRODUCTION

Operating systems can be thought of as the system managers. In response to demands of a user's program, the operating system manages the allocation and use of the central processor unit, main and mass memories, and input and output resources.

The lack of standards and quality in existing operating systems is the major problem in transporting software from one computer installation to another, even with only a single make and model of computer.

Operating systems are at once the best and worst place to consider standardization. Ideally, if one had a standard operating system, then one could imagine true software portability, since all machines would appear identical. From a practical point of view, a standard operating system for a large computer is neither practical nor desirable.

Operating systems for large computers are huge collections of software programs intimately related to the particular hardware architecture for which they were designed. For this reason, those features that are common among large computers, and could be the basis for standardization, are generally a very small subset of the total features implemented in a modern operating system. This lowest common denominator approach would deny the user the best features of the large computers in use today. Further, the mainframe manufacturers have a market incentive to keep operating systems both unique and proprietary.

The second problem for the Air Force in considering operating systems is their size and complexity: the cost of developing a new operating system for a large machine would probably exceed the total resources of the ICAM program. Worse, advances by the industry in hardware and system designs would soon obsolete whatever system was developed.

Incompatible features of operating systems will undoubtedly cause the Air Force serious problems in creating complex integrated systems software that is sufficently independent of the host computer to be portable. However, overall operating system standardization does not seem to be a viable answer. There are several areas in which limited standards can and should be implemented for the ICAM program which will be discussed below.

The situation is somewhat different for mini and microcomputers. The 16 bit minicomputers are sufficiently similar in their hardware characteristics and system architectures that the idea of a standard operating system is feasible. For a distributed, integrated system based on 16 bit minicomputers, the development of a communications oriented standard operating system is probably within the resources of the ICAM program. The 32 bit machines which are byte oriented (in handling internal data communications) are generally extensions of comparable 16 bit machines and could also be considered in developing a standard operating system.

Microcomputers are too small to have much of an operating system. Simple terminal monitors or switch monitors are supplied on ROMS in microcomputer kits to allow the user to load programs, but that plus some simple debugging routines is the extent of the system software. There is an opportunity to facilitate the use of microprocessors in CAM systems through the development of a cross software system based on PL/M or some other subset of a high level language that would run on higher level computers. Such a system would be essentially independent of the rapid hardware innovations at the microprocessor level and could provide full system support capabilities.

OPERATING SYSTEM FUNCTIONS

Historically, operating systems first arose as a matter of convenience rather than necessity. In the early 1950's, each programmer actually operated the machine and debugged his program on-line, controlling card input formats and line printer formats with patch panels inserted in the periphals. Batch processing programs were developed in the late 50's to expedite this situation by automatically loading another program as one was completed.

Executive systems were developed in the early 1960's that provided users with common access to complex programs developed for handling input and output. At this time computers were basically constrained to a single user and each job was completed before the next one began.

Because input and output functions depend on external periphal devices generally much slower than the CPU, single user systems are very inefficient. For this reason, multiprogramming batch systems were developed that allowed more than one job to be executed at once.

The development of time sharing systems, on line file management, real time operating systems, and virtual storage and virtual machine concepts has led to the operating systems of the 1970's, in which multiple users can simultaneously have access to the resources of the computer. The operating system is required to schedule the computer resources while preventing unwanted interaction between unrelated processes and to enforce access restrictions to data.

The primary functions of modern operating systems can roughly be divided into 4 classes:

1. Job control
   - job scheduling
   - process scheduling
   - control of information flow
   - start/stop processes

2. Main Storage management
   - allocate memory (including partitioning and/or paging)
   - access control

3. Device management
   - schedule I/O devices
   - control data flow to I/O devices
   - monitor interrupts on I/O devices

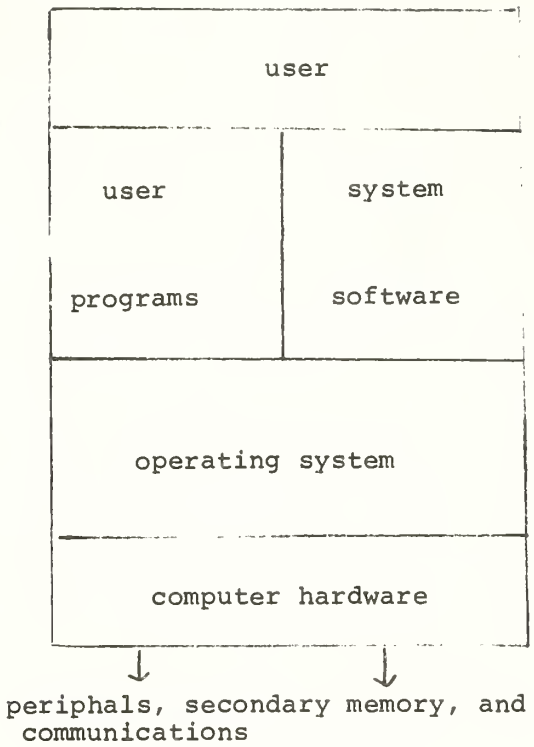
4. File system management
   - create/destroy file
   - open/close file
   - read/write file

It is in this last area of file management that many of the worst problems of software compatibility and portability arise, as we will discuss below.

COMMUNICATION WITH AN OPERATING SYSTEM

The user communicates with an operating system by two methods: system calls and an operating system command language (OSCL).

System calls can be thought of as procedure calls to special operating system procedures. They are used in programs to request services of the operating system. For example, READ and WRITE statements are supervisory



Functions

- specifies desired operation of computer
- user specified applications programs
- compilers, assemblers, loaders
- debugging, text editing
- libraries
- job scheduling and control
- storage management
- device management
- file system management
- main memory
- electronic data processing
- interaction of CPU with outside world

Figure 1

OPERATING SYSTEM FUNCTIONS

calls. The system calls represent the "primitive actions" that an operating system can perform for an executing process.

These primitives vary greatly between operating systems since they represent basic design decisions and implementation realizations. Standardization at the system call level is not practical nor advisable since it might stifle new innovation.

However, it is possible to present a more uniform view of the system call interface to a process by layering it with routines which map user intentions into system calls. This is, in fact, exactly what is done by the I/O runtime support routines for a programming language.

Figure 2 shows schematically how a user program interfaces to an operating system through a runtime support routine. These routines are necessary to translate the varying system calls in different languages to a form understood by the operating system. For example, OUTPUT FILEZ in BASIC and WRITE 600, FILEZ in FORTRAN may be translated into the same system call to initiate an I/O action.

It is at this level that the direct interaction takes place between a user program and an operating system for I/O.

An extension of this approach to other system feature calls may yield improved benefits and warrants investigation. However, any such approach is still limited by the basic primitives that the operating system designers implemented.

An operating system command language (such as JCL) is a self-contained but often rudimentary language for direct communication between a user and the operating system. The command language is used to schedule jobs, assign files, etc. and otherwise direct the execution of programs on the behalf of the user. The design of a command language is greatly influenced by the primary intended mode of operation of the operating system: batch or interactive. Unfortunately, there exist systems orginally designed for batch operation to which an interactive mode was later added. The resultant command languages are often ill-suited for interactive use.

Some attempts have been directed towards the development of a system-independent command lanugage. They have received very little, if any, vendor support and probably for that reason have had no success. However, on some of the more well-designed operating systems, the command language exists as a separable part of the system, and thus can be easily changed. If fact, some of these systems can support more than one command language.

Each vendor of operating systems has a unique appraoch to the implementation of the user-system interface from one generation to another. No operating system in widespread use can be said to possess sufficient redeeming qualities in its user-system interface that acceptance of it as even an ad hoc standard can be advocated.

VIRTUAL SYSTEMS

There are several concepts that can be considered under the general title of virtual systems. These include virtual memory, virtual devices, and virtual machines. All are intended to make a physical characteristic of the computer appear to be more than it actually is in order to help the user and improve the efficiency of utilization of the computer itself.

Virtual Memory: this is by now the well known concept of placing only parts (pages) of a users program or data files in the high speed main memory at any one time. By managing the partitioning of the main memory and by swapping appropriate pages to and from low speed, low cost,

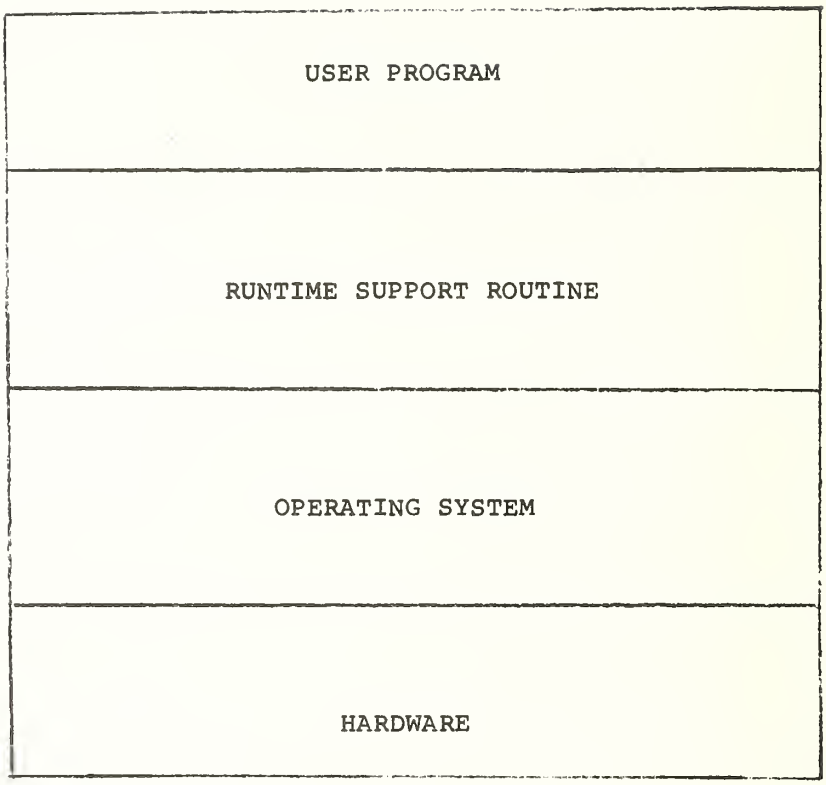


Figure 2

RUNTIME SUPPORT ROUTINE

high volume secondary disc storage, the user program sees a memory that appears to have the capacity of a disc with the speed of the main memory.

Virtual Devices: with multiprogramming systems, the limitations of communication to and from I/O devices can cause the system to bog down. This can be circumvented by creating duplicate, virtual devices. A program, then, will output to a virtual device. After a program is completed, the data file can be scheduled for output on the physical output device. Several different programs may be simultaneously performing I/O operations to the same (virtual) device.

Virtual Machines: the same essential problem exists with process management as with device management. By creating multiple, virtual versions of the operating system hardware interface, several operating systems can (seemingly) simultaneously execute privileged system calls at the hardware level. The virtual monitor, or hypervisor, is shown in Figure 3. The hypervisor operates on an interrupt basis in response to privileged instructions for the operating system. A file is set up of these instructions for execution when the hardware is actually available, and control is returned to each operating system in such a way that it thinks the instruction was executed. This can make one computer look like several computers with different operating systems.

The possibility of extending the virtual machine concept to gain hardware independence for Air Force software has been considered and discarded. The same arguments that were given at the first of this section still hold true:

1. The basic limits are the hardware features of the machine. Using only those features that are common to all large machines is inefficient and too severe a restriction.

2. Adding another layer of interpretation is inefficient.

3. The potential cost of operating systems development is huge and will be quickly rendered obsolete.

For these reasons, extensions of the virtual machine concept are not recommended.

FILE MANAGEMENT PROBLEMS

It is in this area that the Air Force can expect to encounter serious problems unless adequate care is taken in the early design stages. Different computers have different file management schemes which may cause problems in an integrated, distributed environment such as that envisionaged by the Air Force for ICAM.

File management can even be a problem in a single computer environment. For example, a file written by a FORTRAN program may be unreadable by a COBOL program because of the formatting and the addition of "invisible" bits such as file designations and check sums.

These problems can be solved by careful consideration and standardization of the file management system calls made by the runtime support routines for each of the languages to be allowed in the ICAM program. Changes can be made to these routines, if necessary, at low cost.

Standardization of file formats and naming conventions can and should be done for the ICAM program as special project standards. This will simplify the file compatibility problem and will help insure portability. This can be carried out in conjunction with development of the data base management system for the program.

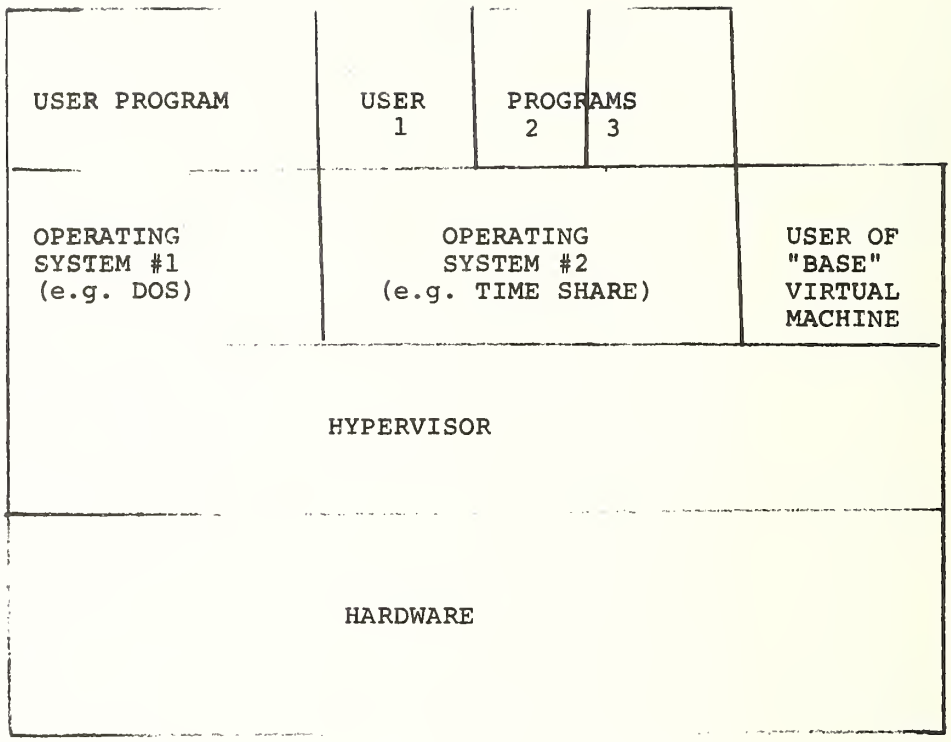


Figure 3

HYPERVISOR CONCEPT

Another problem is in the creation of files when reading from a magnetic tape. For example, the same problem of "invisible" bits mentioned above can occur here. As another example, if a 7 bit ASCII code is used on the tape, a 36 bit machine operating system may pack 5 1/7 characters into one word. This will produce an unreadable file.

Again, using an example from the field of automatic image pattern recognition, in loading digitized image data into a 60 bit word computer, the file management software may pack 7 1/2 8 bit bytes into each word. Since one 8 bit byte is a discrete information element (pixel), further processing of the data may be difficult.

System programmers are used to dealing with these problems. Still, several man-days may be spend in modifying software to read a tape into a computer. These problems can and should be avoided in the Air Force ICAM program through the use of proper specifications and standards for file management.

SUMMARY

In summary, the lack of standardization and quality in available operating system software is a major contributor to the difficulties and costs experienced in transporting program systems to different computer installations. The difficulties may be significant even when the computer hardware configuration is nearly identical between the source and the target installations. The costs due to operating system problems may now exceed the costs resulting from minor discrepancies in the programming language compilers involved. Thus some consideration of operating system standardization is essential to the future success of the Air Force CAM projects. It would not be feasible to seek industry or national standardization of this software in the near future; the extent of previous efforts to do so have never progressed past a study stage. It would not be economical either to consider developing a standard operating system or modifying an existing one for Air Force purposes.

However, several areas of interaction between user and the operating system have been identified in the discussion above where attention will be needed to maximize portability:

1. Runtime support routines between user program and operating system.

2. Operating system control language.

3. File management and data base management system interfaces.

4. Input/output software to read files to and from tapes or other media for transporting software and data.

RECOMMENDATIONS ON OPERATING SYSTEMS

1. The Air Force should not undertake to develop a new operating system or modify existing systems for large machines.

2. Standards on programming languages and data base management systems are the best approach to software portability and integratability. In other words, the operating system area should be avoided and system functions implemented using the general purpose programming languages, if at all possible.

3. Limitation of the number of operating systems for the ICAM system may ultimately be necessary. In any case, identification and isolation of all systems dependent code in ICAM software will expedite transitions to other computer systems.

4. No current operating system command language has such features as to recommend it over others. However, this is one area in which standardization is at least technically feasible and should be considered. Federal standardization is already underway in a limited way, addressing the user access protocol to computer networks and services. This effort in FIPS Task Group 20 could be expanded to consider the full range of command language functions. The Air Force should request the Associate Director for ADP Standards, NBS, to determine the feasibility of expanding the scope of work of TG 20 to address Air Force requirements for its CAM program.

5. A standard operating system could be developed for many of the 16 bit (and 32 bit) minicomputers in use today. For a distributed computer system based on 16 bit or 32 bit minicomputers, this approach is attractive and should be examined in detail.

6. File management standards, such as naming conventions for data files and library software, should be enforced for all ICAM development projects to maximize portability of CAM software products. Many potential problems in file management may be avoided through the use of an adequate data base management system (see next section).

REFERENCES

(1) Madnick, S.E. and Donovan, J.J., Operating Systems, New York, McGraw Hill, 1974.

(2) Organick, E.I., The Multics System: An Examination of Its Structure, Cambridge, MIT Press, 1972.

(3) Organick, E.I., Computer System Organization (The B5700/B6700 Series), New York, Academic Press, 1973.

STANDARDS FOR COMPUTER SYSTEMS

5. DATA BASE MANAGEMENT SYSTEMS

INTRODUCTION

TYPES OF DATA BASE MANAGEMENT SYSTEMS

- CODASYL
- Self Contained Packages
- Host Language Approach
- Relational Concept

CENTRALIZED VS DISTRIBUTED DATA BASES

- Areas of Consideration

DEVELOPMENT OF A DBMS VS A COMMERCIAL DBMS

ASSESSMENT OF SYSTEMS AND SOME POPULAR COMMERCIAL PACKAGES

RECOMMENDATIONS

REFERENCES

- See Appendix on Data Base Management File Structures

INTRODUCTION

A data base management system (DBMS) is a generalized tool for manipulating large data bases. It provides a flexible facility for accommodating different data files and operations while demanding less programming effort than use of conventional programming languages. DBMS possess the following general properties:

* Software which facilitates such operations as data definition, data storage, data maintenance, data retrieval, and output.

* Software which facilitates reference to data by name and not by physical location.

* Software which is general, rather than specific to a particular set of application programs or files.

Since the early 1950's, when generalized file handling routines were first developed, the technology of DBMS has matured considerably. Within the last ten years, a great number of DBMS packages have appeared on the market. No precise count of operational DBMS exists, but it is estimated that at least 200 are now available.

The use of DBMS to control large data bases and provide information to multiple users has already gained acceptance in the data processing world. A recent survey (1) of DBMS usage, just on IBM 360/370 computers in the United States, reported 3,900 DBMS installations as of 1976.

The off-the-shelf DBMS packages do not provide the same set of functions, and the implementation of functions differs widely in depth and strength of effectiveness (2). There are as yet no standards in the area of DBMS as a total package. Many groups are concerned about standardization and are actively working in this area. The CODASYL Data Base Task Group (DBTG) report (3) has been published by the Programming Language Committee of CODASYL as a part of the 1976 COBOL Journal of Development (4). This report represents a specification of a data base management system; future national and international standards will certainly be influenced by this report. The ANSI/X3/SPARC Data Base Study Group has been meeting since 1972; see Interim Report (5). Part of their charge is to develop a basis for DBMS standardization.

In planning the use of data base software for CAM, the Air Force should recognize the severe difficulties that stem from the lack of standard systems, the technical complexity of data base packages and the consequent problems of training and applications analysis, and the rather high costs in storage and processing time that may be unacceptable in some applications. Although available data base packages may be categorized by a similarity of concept, such as the CODASYL or network approach, none of the available packages are even close to being identical in their commands, language, and functions. No de facto standard data base systems exist or are likely to develop in the next three years. The transferability of data base packages between different computers, particularly between minicomputers and large machines, is very limited. Fundamental differences may be presented in the same package because of machine dependent factors, such as the available mass storage.

TYPES OF DATA BASE MANAGEMENT SYSTEMS

Although there are many DBMS packages in the market with different functions and strategies, for the purpose of this study, the total DBMS technology can be described as four broadly different approaches:

1. CODASYL Data Base Task Group Specification
2. Self-Contained Approach
3. Host Language Approach
4. Relational Approach

Inherent in this classification is the data organization which the data base management system supports. The three favored data model approaches are: network, hierarchical, and relational. (See Figures 1, 2 and 3). The CODASYL DBTG supports a network structure. Most of the self-contained type systems support a hierarchical structure. The non-CODASYL host languages are distinguished from the CODASYL host language types becuase of the two popular packages; IMS which is hierarchical, and TOTAL which supports networks. The relational approach models the relational data organization which has a tabular orientation. The characteristics of the four approaches are discussed below.

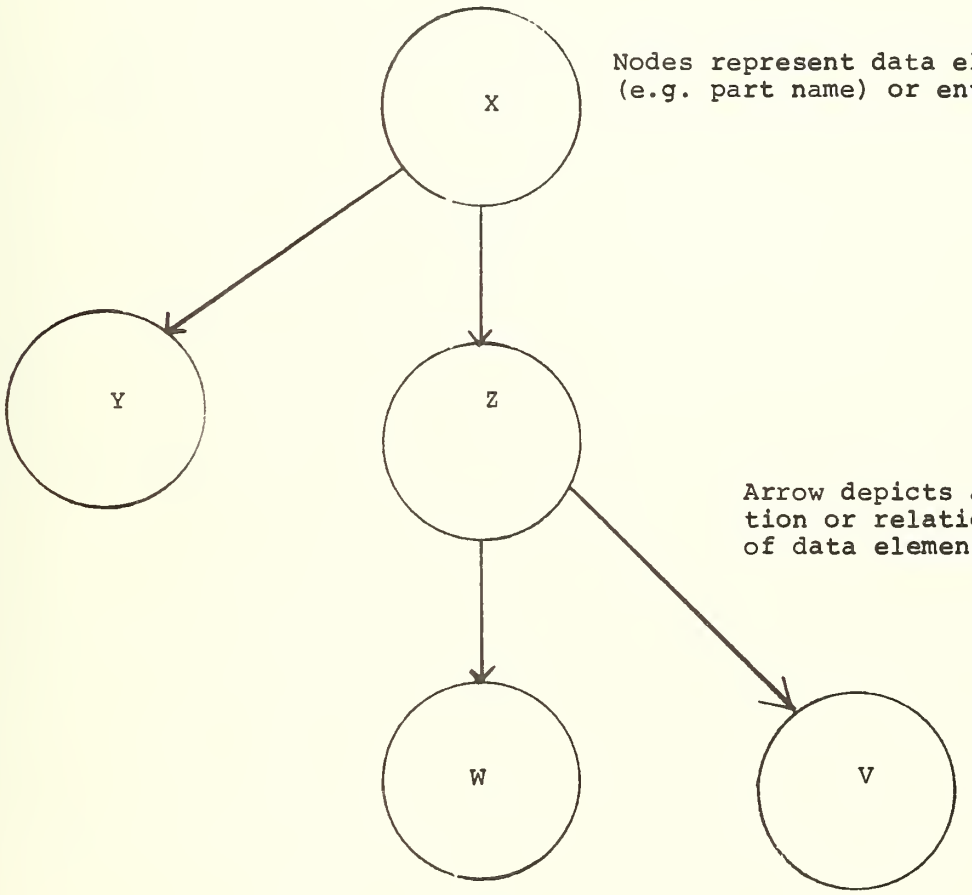


Figure 1

HIERARCHICAL DATA STRUCTURE ILLUSTRATION SHOWING SIMPLE SUPERIOR/SUBORDINATE ASSOCIATIONS

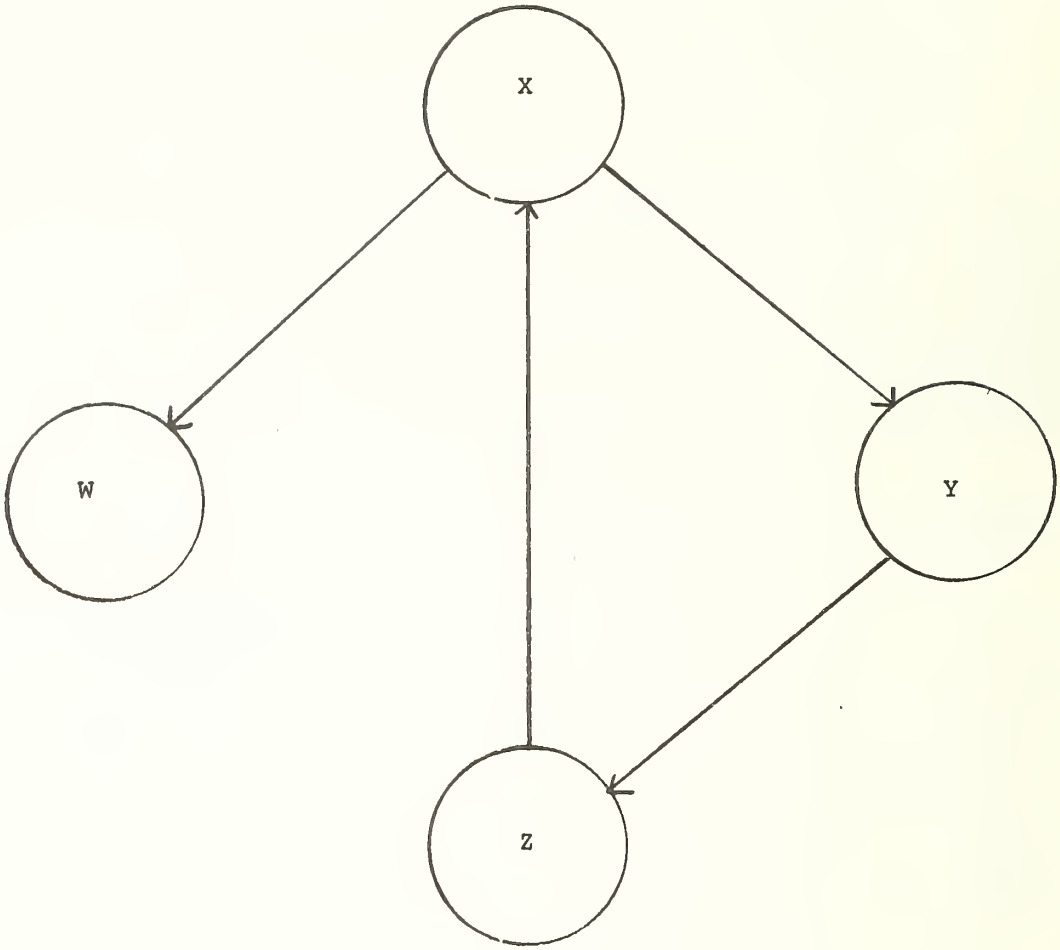


Figure 2

NETWORK DATA STRUCTURE ILLUSTRATION SHOWING
ARBITRARY ASSOCIATIONS OF DATA ELEMENTS

A Relation → A Particular Tabular Association

Date element X	Y	Z
values	associated values	associated values

Figure 3

RELATIONAL SYSTEMS REPRESENT COMPLEX DATA ASSOCIATIONS IN SIMPLE TABLES

CODASYL Data Base Task Group Specification

The CODASYL Data Base Task Group (DBTG) specification as published in 1971 consists of two parts: (1) syntax and semantics of a data description language (DDL) for describing the structured data base, (2) the definition of data manipulation language (DML) statements to augment COBOL (for retrieving and updating data in the data base).

Three important characteristics of the CODASYL DBTG specification (see Fig. 4) are as follows:

* The data relationships are explicitly defined in the data base. Records that are logically related are tied together by either pointers or by indexes. The relationships are defined when the data base (schema) is defined. The advantage of this architecture is that the relationships can be carefully worked out by the people who understand the data. The disadvantage is that it can be a nontrivial task to change the relationships.

* The Data Definition Language (DDL) is separated into two parts: (1) the Schema DDL is totally language-independent and used to describe the data relationships as mentioned above, and (2) the Sub-Schema DDL which is fashioned around the language of the user's program and restructures the data base for the particular requirements of the program. Thus, this separation permits multiple-language interface, data independence, a smaller view of the data to a program, and protection for the remainder of the data base not used in a given application.

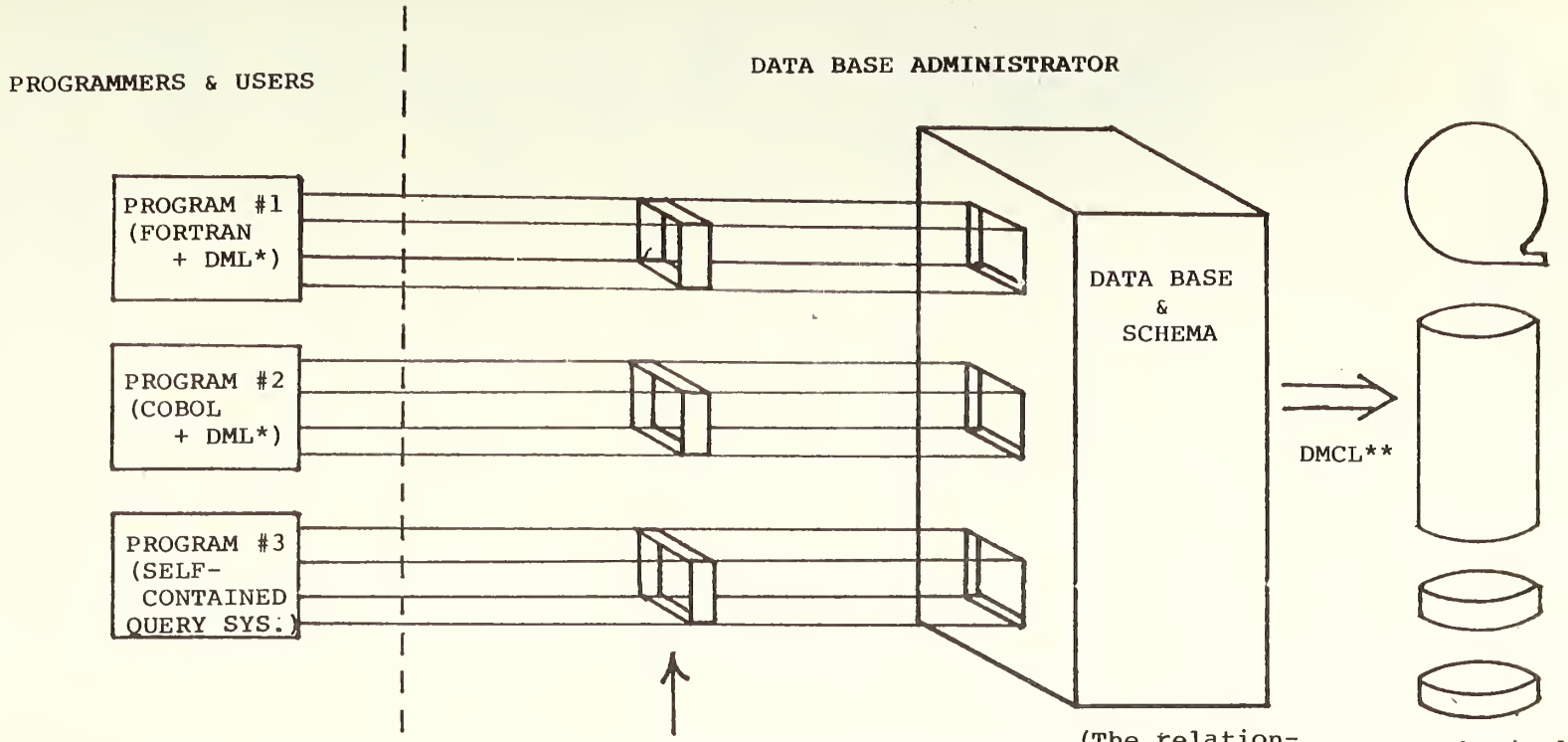
* The Data Manipulation Language (DML) has been designed to help the application programmer "navigate" within the data base. Any given record in the data base can be related to a number of other records, and it might be accessed by any of several paths. The application programmer must know where his program is operating, and how it should retrace its steps when a search proves unfruitful.

The CODASYL DBTG approach adopts the network data model. A network is a more general structure than a hierarchical structure because a given node may have any number of immediate superiors as well as any number of immediate subordinates. Therefore, this approach provides the most powerful means of handling complex data structures, but querying and reporting may prove to be a complex matter.

Self-Contained Packages

The majority of the commerically available data base software packages are of the self-contained type. Typically, these systems possess three major processing capabilities: data creation, data update, and data retrieval and report formatting. A self-contained user language is provided to accomplish all three processing tasks. These systems are aimed at handling a certain set of data base functions in such a way that conventional procedural programming is not required. The capability to specify in detail the search method and data retrieval the programmer wishes is replaced by preprogrammed or built in processing algorithms so that the amount of writing required by the user is minimized. The self-contained systems are optimized on their interrogation and update functions. As a result they represent the most advanced DBMS in the area of user language capabilities.

The very reason for the success of the self-contained DBMS, ie. their high level, non-procedural interrogation language, becomes a large disadvantage in those cases where the user wishes to exercise control over the sequence of detailed steps the system uses to process his requirements. Some systems also provide external programming interfaces where the user can enter his own



*CODASYL Specification
Schema DDL
Sub-Schema DDL
Data Manipulation Language (DML)

**Device Media Control Language not specified.

The Sub-Schemas (as defined by the Sub-Schema DDL*)

1) restructure the data base for the particular requirements of the program
2) prevent program access to unauthorized portions of the data base
3) allow the use of different procedural languages and self-contained languages

(The relationships between the data as defined by the independent Schema Data Definition Language (DDL*))

Figure 4

routines written either in FORTRAN, COBOL, PL/I or assembly language to perform processing not inherently supported by the system. However, this does not yield the same capabilities as a host-language DBMS with its appropriate data manipulation language (DML). The majority of the self-contained data base management packages model the hierarchical data structure with repeating groups. Typically, the system employs the inverted indexed technique to facilitate quick retrieval.

Characteristics of these systems are that they are:

* End-user oriented. The user-language is easy, natural and English-like. Very little application programming is necessary. However, the user is paying for an added layer of software with less efficiency and flexibility.

* Easy to install. After the data base has been created it is relatively easy to change the structure. The data base can be built an application at a time, without requiring that the whole data base be defined at the outset. These capabilities are largely a result of the implementation of an inverted or partially inverted file system for storing the data. However, the (partially) inverted file structure results in difficulty in handling of queries that specify records located in different branches and/or at different levels of a hierarchy, and in addition, results in considerable storage space required for the indicies.

* Easier to formulate unanticipated ad-hoc queries. Self-contained systems permit the user to ask the question directly, and he has no need to call on a programmer as an intermediary. For those applications that self-contained systems can handle, they offer considerably reduced set-up time and a vast reduction in the time required to prepare a new interrogation or update to the data base. However, the end-user must be aware of the data structure supported by the system; if the needed data elements are not keyed or inverted, the system either searches sequentially or refuses to respond. Another caution on a hierarchical tree structure, if the data elements requested in the queries are not of the same hierarchy, no "hits," or even erroneous ones, will be made.

Host Language Approach

Although the CODASYL DBTG specification is a host language type, we have treated it as a separate entity because of two distinctly different packages that are already widely used; IMS by IBM, and TOTAL by Cincom (See Table 1). The host language approach is characterized by the following features:

* The system is designed as a tool for the experienced programmer.

* System functions are invoked from within host programming languages (e.g., COBOL, FORTRAN, PL/I, assembly language).

* The supported data structures generally permit more user control, even down to the physical storage level, than those found in self-contained data management systems.

Host language DBMS generally lack high level language constructs for conditional data, updates and retrievals, as are found in the self-contained type. Typically, this is because the emphasis has been placed on defining logical relationships among records or group of records in large interrelated data bases, rather than on generalized functions. However, these systems do interface to separate Report Program Generators (RPG)/ Query packages (this provides some aspect of the interrogation capabilities

inherent in the self-contained systems) while providing the flexibility to the user of specifying the details of how his request is to be processed through the use of the procedural-DML.

Host language type systems can be thought of as extensions to the programming languages. The method chosen to interface the host language data management system with the programming language is usually through the facilities of the CALL statement in the programming language.

Host language type systems provide powerful data management functions for manipulating data, programmable through the flexibility of a programming language and considerable user control over the physical storage structure.

Relational Concept

With commercially available DBMS, the variety of data representation characteristics which can be changed without logically impairing some application programs is still quite limited. Some people feel that the present data base management systems require entirely too much knowledge on the part of the user on how the data base is structured and how the data should be accessed (for the case of host language systems) or are too limited by preprogrammed algorithms (for the use of self-contained systems). Instead, the user, be he an application programmer, manager, engineer, or other - should simply have to specify what data is desired, not how it is to be retrieved. The main problem with present systems is that the data relationships are structured, which favors some types of access at the expense of others, i.e. the application programs are not independent of the data base. (See Appendix for a discussion of some of the characteristics of various file structure techniques.)

Three of the principal kinds of data dependence are:

1) Ordering dependence - e.g. records of a file concerning parts might be sorted in ascending order by part serial number - these systems fail if this ordering is replaced by a different one (e.g. if a search is desired by part material - aluminum, brass, steel etc.) The same is true for a stored ordering implemented by means of pointers.

2) Indexing dependence - from an informational standpoint, indices are redundant components of the data representation, requiring large additional storage capacity from the data structure.

3) Access path dependence - many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. Application programs developed to work with these systems tend to be logically impaired if the trees or networks are changed in structure. Or, if a query is made for data in other than the structured form in the data base, then a time consuming, total and complete search of the data base may be required. In general, the user (or his program) is required to exploit a collection of user access paths to the data.

The relational data base is proposed as a possible solution to these problems. This concept is relatively new (Codd 1970) (6). The approach is based on the premise that users of data base management systems are becoming increasingly concerned with the information content of their data, as opposed to specific representation details. That is, there is a trend toward data base user interfaces that deal with information in application terms rather than with the bits, pointers, and lists that are used to represent information on computer mass-storage devices.

The relational approach to data base management can be characterized as follows:

* Simplicity of user interface. The relational user is presented with a single, consistent data structure and requests can be formulated strictly in terms of information content, without reference to most system-oriented complexities.

* Data independence. The user is relieved of concerns for knowing specific information storage and access strategy.

* Flexible response to ad-hoc queries. Since all information is represented by data values in relations, there is no preferred format for a question.

The most serious question regarding the relational approach is whether it can be implemented to form an efficient and operationally viable DBMS. Many prototype systems exist but no commerical systems exist that are truly relational. These systems are summarized in Table 1.

CENTRALIZED VS. DISTRIBUTED DATA BASES

There is no clear-cut best answer regarding which approach to use. Each approach has its advantages and shortcomings. But the answer seems to depend upon the particular needs and application environment. Data base management packages also need to be selected in the context of some architectural configuration. Two opposite data base architectures can be identified: the centralized data base approach and the distributed data base approach.

Centralized data base - A central data base is usually maintained using a large-scale third generation type mainframe. Data may be generated centrally or bulk entered from several remote data entry stations. The centralized approach allows centralized control of the data bases, which is necessary for efficient data administration. The data base management system for the central computer would need to have full facilities for storage and maintenance of data. In particular, some of the mandatory features should be powerful control functions for data validation, update control, centralize data dictionary capabilities to manage the centralized data base. Retrieval and output reports can be optionally weighed against very end-user oriented query language facility versus transaction invocation via predefined process written in a programming language such as COBOL. The data base management system for the centralized architectural approach can be all of the above four types: CODASYL DBTG-like, non-CODASYL self-contained, non-CODASYL host language, or the relational approach.

Distributed data base approach - The development of computer networks has led to the prospect of distributed data bases. Distributed data bases also include distributed processing which generally consists of remote stations distributed throughout remote locations. The remote stations evolved from intelligent terminals to, at present, minicomputers installed with their own secondary storage. Distributed data bases can have numerous configurations. One scenario might be identified as follows: The distributed information system might be a multi-level hierarchy of processors, generally matching, at each level, the organizational structure and complexity of the manufacturing system. The network could be composed of a number of mini-computers so that processing logic and storage (distributed data bases) would be placed at or near the points where transactions occur. A common design would be used for the numerous data bases and for the data base management systems, so that the total data base could be distributed throughout the system. However, due to the data base being stored under a common data base structure, using common data definitions, any portion of the total data base would be accessible from any mode in the network.

This modular design allows modules to be added and others deleted to meet the needs of a particular situation. This would give the network

DATA BASE MANAGEMENT SYSTEM	SYSTEM 2000	ADABAS	IMS/VS	TOTAL	IDMS
TYPE OF SYSTEM	Self-contained	Self-contained	Host-language	Host-language	CODASYL
VENDOR	MRI Systems Austin, Texas	Software AG Reston, Virginia	IBM	CINCOM SYSTEMS Cincinnati, Ohio	CULLINANE CORP. Boston, MA
COMPUTERS	IBM - 360/370 UNIVAC - 1106/1108/1110 CDC 6000	IBM 360/370 UNIVAC Series 70	IBM 370	IBM 360/370, IBM 3 UNIVAC 70, 9000, CDC 6000 HON 200/2010, NCR H60 VARIAN, INTERDATA, DEC 11	IBM 360/370 UNIVAC 70, 90 PDP 11/45
MAIN MEMORY REQU.	144-210 K bytes	120 K bytes DML 150 K bytes DDL	450 K bytes	35 K bytes (not including buffers)	50-65 K bytes (includes 15 K buffers)
USER LANGUAGES	Own Language COBOL, FORTRAN, PL/1	Own Language COBOL, FORTRAN, PL/1	COBOL, PL/1	COBOL, FORTRAN, PL/1	COBOL, FORTRAN, PL/1
FILE STRUCTURE AND ACCESS METHOD	Hierarchical, Sequential, Inverted	Inverted Sequential Access Method (ISAM), Inverted Direct (IDAM), Inverted, Network	HDAM - hierarchical HSAM direct access HIDAM method HISAM	Full file inversions, Chains, Network, Random/Calculation	Chains, Networks, Rings Hierarchies Random/Calculation Full file inversions
TELE PROCESSING INTERFACES	Has own TP package	INTERCOMM, CICS, TASK/MASTER	CICS, DC, INTERCOMM, TASK/MASTER	CICS, ENVIRON/1, INTERCOMM TASK/MASTER	CICS, DC, INTERCOMM
REPORT GENERATOR QUERY FACILITIES	Has own RPG package and query	ADA WRITER, ADASCRIPT, EASYTRIEVE	CULPRIT, MARK IV, ASI-ST	SOCRATES, MARK IV, ASI-ST, CULPRIT, EXTRACTO, EASYTRIEVE	CULPRIT
NUMBER OF INSTALLATIONS	> 100	> 50	> 500	> 750	> 100

TABLE 1 - DATA BASE MANAGEMENT SYSTEMS

DATA BASE MANAGEMENT SYSTEM	INGRES	NOMAD	DMS 1100	IDS - II	EDMS
TYPE OF SYSTEM	Relational	Relational	CODASYL	CODASYL	CODASYL
VENDOR	University of California at Berkeley	National C.S.S. Inc.	UNIVAC Roseville, Minnesota	Honeywell Information Sys. Phoenix, Arizona	Xerox Information Systems Group El Segundo, CA
COMPUTERS	DEC PDP 11 with Unix	Time share service	UNIVAC 1106, 1108, 1110	H 9GS) 200, 400, 600 H-6000, Large Series 60's	XDS SIGMA 6, 7, 9, 560
MAIN MEMORY REQU.	----	----	16-65 K words (36 bit)	2 K words Monitor 10 K words/Partition	12-18 K words (32 bits)
USER LANGUAGES	QUEL	----	COBOL, FORTRAN	COBOL, FORTRAN	COBOL, FORTRAN
FILE STRUCTURE AND ACCESS METHOD	Relational	Relational and Hierarchical	ISAM, Direct Random/Calculation Inverted Tables, Pointer array	Random/direct, Hierarchy, Networks	Direct, Random/Calculation Indexed, Chains, Network
TELE PROCESSING INTERFACES	----	----	Transaction Interface Package	----	----
REPORT GENERATOR FACILITIES	----	----	----	MDQS Report Generator (on-line request only)	----
NUMBER OF INSTALLATIONS	> 6	> 5	> 40	> 170 (including IDS-I which is not CODASYL)	> 50

TABLE 1 - DATA BASE MANAGEMENT SYSTEMS

the ability to withstand severe damage to some of the processors (or some of the storage units) with the remainder being able to continue operation. (Due to the centralization of the data that has occurred as a result of the installation of DBMS some plants have experienced severe problems in the total shut down of the operations when something goes wrong with the system. The solutions organizations are arriving at turn out to be approximations to the network philosophy - some companies have logically and physically subdividided the data base to cause it to reside on different storage units - others have installed additional minicomputers to allow for continued data-taking if the main system goes down). The distributed communications in a network structure would provide at least two independent paths between any two modes, so as to provide automatic alternate routing for messages. Two kinds of data base management software can be considered in this scenerio:

1) There are data base management systems specifically built for the minicomputer. For example, Hewlett-Packard has developed a package called IMAGE, and Data General has a data base management system (INFOS) for its Eclipse series. Varian and Harris have signed contracts with Cincom to offer TOTAL. Cullinane offers IDMS which is precompiled on an IBM 360 and the object module run on DEC's PDP 11/45. Other prototype systems which are not yet commerically available are operational for DEC's PDP-11.

2) Another approach is relatively new. It is the concept of a Data Computer. It consists of hardware solely used for the accessing of data. A prototype is being built by Computer Corporation of America for the ARPA network. A similar concept is the "back-end" computer concept where the data base is maintained as the "back-end" computer, usually a minicomputer, and interfaced to a host computer, usually a large-scale third-generation type where user request language is translated.

Many advantages would accrue from a geographically dispersed approach to data base management, including:

* Better provisions for protection than with centralized systems.

* Flexibility and localized control of data processing activities.

* Data validation at local sites, resulting in cleaner data input to the central computer data base.

* Flexibility and potential of a distributed architecture.

The distributed data base concept is not without problems. For example, does the user have to know the data location, does the request language need to be different to access different distributed data bases, etc. Most importantly, there is no fully operational distributed data base system as yet.

Areas of Consideration

1) There are no standards in data base management as a total package. The CODASYL DBTG report lends itself to be a potential candidate for standardization, but many feel that the CODASYL DBTG approach is not universally accepted and should not be standardized. This is a result of the general feeling that data base management systems are very much an evolving technology where many developments are yet to come. It is felt that the standardization of the CODASYL data base task group (DBTG) report would provide sufficient inertia to the system so as to impede the development of new and perhaps better data base management systems. In addition, the

specifications of the DBTG report are felt to be too incomplete. The proposed data manipulation language (DML) is felt to be too procedurally oriented (therefore, not easily used by the non-programmer) and to have shortcomings with respect to data independence, data integrity, and compatibility (Guide-Share report). The CODASYL specifications make no provision for handling existing sequential and index sequential file structures. Nor do they define a device media control language (DMCL) which is the storage structure language used to describe the mapping of the data onto physical storage media. And while the specification of a totally language-independent data definition language (DDL) would theoretically allow access to the data base by either a host-language request or a self-contained-like query, only the host-language data manipulation language (DML) has been specified. CODASYL has not yet developed the specifications for query and reporting languages. As mentioned earlier, the application programmer must know how to "navigate" in a complex data base environment. Query and reporting languages will have to do such "navigating" automatically, and as a result could be quite complex in their development. However, this difficulty exists for any type of complex data base structure (i.e. a network or graph structure), and is not limited to CODASYL-like systems. Some report program generators (RPG)/ Query packages are available and interface to the commerical CODASYL systems. These are not as powerful, as yet, as the self-contained system query capabilities. In addition, CODASYL does not specify the recovery techniques to be used after a system goes down, nor the method to be used for restructuring and/or reorganizing the data base. All of these features are left up to individual vendor and/or user to supply.

2) The possibility exists that there will be standards in each of the different DBMS approaches. The rationale is that since there are different programming languages for different applications, e.g. COBOL, FORTRAN, PL/I, BASIC, it is conceivable that there may be different data base management systems under consideration for different applications.

3) The contemporary large-scale data base management systems are built with separable functional modules. For example, a data base management system may consist of a nucleus plus the following kind of functional modules:

   * the data definition language for specifying the logical structure,

   * the data dictionary/directories for ease of managing data description,

   * the teleprocessing message handling for on-line interactions,

   * the user language processor for user interface to manipulate the data,

   * the protocols for invoking procedures on the data base system,

   * the data access methods for physical sotrage accesses,

   * the report writer for formatting fancy reports.

   Each of these areas may potentially be considered for standardization. Already, the commercial world has been marketing data base management software in optionally upgradable and pluggable modules. Adjunct packages such as report writer, query languages, teleprocessing front-end, data dictionary and various utilities such as

bulk load, sort, etc. have started to appear in the marketplace. These adjunct packages usually operate in conjunction with a specific data base management system. Although the interfaces of these packages are not as yet flexible, standardizing the interfaces of a data base system leads to the concept of interchangeable parts.

DEVELOPMENT OF A DBMS VS A COMMERCIAL DBMS

The development of a data base management system is considered too involved, too expensive, and too risky to attempt. Years of problems of cost overruns, unattained goals, and expensive maintenance have plaqued new development efforts. There are, at present, a number of commerical data base management systems available, that, while they fall short of meeting the requirements of an idealized DBMS, effectively provide for the recording, retrieving, and updating of large, complex stores of data. It is recommended that the Air Force choose one of these commercial systems with the expectation of updating or even converting to an entirely new system in several years as major advances in DBMS occur. To wait for these advances, or to attempt to develop new systems (which would entail a great expenditure of resources in a not-well-understood field to obtain questionable improvements) would cause a major setback to the overall project. Experience and a clearer understanding of the problem of what is really needed from a DBMS in an Integrated Computer Aided Manufacturing system can be better obtained from using an existing commercial DBMS in a working system rather than attempting to develop a DBMS for a non-working system.

At present, there are no working commerical packages of the relational DBMS type. This area is considered too experimental to be implemented in the Air Force project. Much more research and development is required to see if these relational systems can provide the theoretical advances they promise.

There are a number of commerical systems (both host-language and self-contained) available and the choice of a system should include such considerations as

flexibility - in terms of use on a number of different computers in a distributed system all addressing the distributed data base.

portability - in terms of being able to move a DBMS or a data base or portions of a data base from an existing hardware/software complex to another.

adaptability - in terms of the ability to change data definitions easily (i.e., to add, delete, lengthen, shorten, or change the relative location of fields within records, records within sets or files, or relationship indicators (pointers or indices)), to do all of this without having to make changes in application programs and without having to dump and reload the whole data base.

ASSESSMENT OF SYSTEMS AND SOME POPULAR COMMERCIAL PACKAGES

Self-contained systems - System 2000 marketed by the MRI systems corporation and ADABAS, distributed by Software AG, are among the most popular self-contained DBMS. As mentioned previously, these self-contained systems originated with their own internal language with no connection to any of the procedure-oriented languages (such as COBOL or FORTRAN). However, most self-contained systems now provide interfaces to allow use of COBOL, FORTRAN, and PL/1 in formulating data requests, but the use of these procedural languages does not result in the same capabilities or

efficiencies as obtained with host-language DML. Even though these self-contained systems are very good in query and reporting capabilities, they are not recommended for the Air Force project because, due to the limitations of a hierarchical file structure, System 2000 does not handle complex data structures (networks), ADABAS, however does have a network file structure that is like the CODASYL approach; the limited capabilities of the built-in processing algorithms which are only partially corrected for by providing interfaces to procedural language programs; and, their rather large size somewhat restricts their use in a distributed system.

Host-Language Systems - The host-language systems such as IBM's Information Management System (IMS) and CINCOM's TOTAL are embedded in a host language (COBOL, PL/1, or FORTRAN (TOTAL only)) and therefore are built upon the facilities of a procedural language.

IMS is a hierarchical based file structure which means network-type relationships are difficult to handle, but IMS does allow network-like structures. This probably results in a considerable overhead in additional pointers and indices and is probably partially responsible for IMS requiring the largest amount of main memory (450K bytes) of any of the DBMS. IMS does not have a FORTRAN host-language capability.

IBM will provide the hardware, operating system, data base management system, data communications package, and query and reporting facilities. Further, IBM makes frequent improvements to these, and gives them good support. IBM is not currently implementing the CODASYL DBTG specifications and has no plans to do so.

User comments on IMS include good recovery, flexibility in data orgnization and administration, versatile file structures, and that changes to data relationships can be achieved via rule redefinition without requiring major program modification or data reentry. However, IMS is also reported to be a very complex product requiring much application software support, and it has large core requirements. IMS is not recommended for this project because it is specific to IBM equipment and produces a sole source condition incompatible with the objectes of a portable system. In addition, the DBMS is so large that it does not fit in with the concept of a distributed system, which has been given as a potential Air Force objective.

TOTAL is the most successful data base management system in terms of number of installations (>750). It, like the CODASYL DBTG specifications, was derived from Integrated Data Store (IDS), the grandfather of the data base management systems. TOTAL does not, however, conform to the CODASYL DBTG specifications but conceivably could be converted (TOTAL is similar to the CODASYL specifications in the way data is structured and the way the data relationships are expressed).

TOTAL does allow file inversions, chains, and networks so that complex data structures can be easily represented and quickly retrieved. Users report that the system requires small amounts of core ($\sim$ 35K bytes) and is inexpensive and easy to install. It was developed with small users (DOS environment) in mind. But while it is simple to use, the system is somewhat awkward for large multi-file use since when one file is being accessed, all other files are locked out. Hence, simultaneous processing of several data files is impossible. Also the system's performance degrades with the addition of new variable data records over a period of time.

The major drawbacks seen with TOTAL at present are its inability to efficiently handle multi-file access. However, an interactive query package has recently been added, and future developments could easily make TOTAL a reasonable alternative to a CODASYL based system.

CODASYL Systems - The data base management systems built along the guidelines of the CODASYL specifications are considered to be the most promising, at present, for implementation by the Air Force. The CODASYL specifications represent the most comprehenisve effort to form a "common" (not standard) and machine independent approach in the development of a DBMS. No real standards are expected in this field for at least five to ten years due to the present lack of knowledge and understanding about how a data base should really be structured and accessed and what all the requirements are for the "best" data base management system.

The CODASYL specifications have gone the farthest in providing the basis for a common, modular architecture for DBMS. This approach of carefully partitioning the system to develop a modular architecture has two very important advantages:

1) by partitioning the system, interfaces can be carefully defined and eventually standardized,

2) a common architecture for data base management systems should facilitate the development of distributed systems with distributed data bases.

Data base management systems, in general, were originally designed either as a host-language system or a self-contained system according to expected applications and many that were developed were specially tailored for the unique applications of that individual company, that is, there are many unique DBMS solutions. Now, the trend of the successful system is to modularity: the self-contained systems have interfaces to procedural languages; the host-language systems have interfaces to report generation/query systems; both types of DBMS have provided interfaces to teleprocessing packages, and data dictionary/directories. Thus, all of the DBMS appear to be moving in the direction that the CODASYL specifications had originally outlined. The CODASYL specifications specifically and comprehensively attack this problem of modular partitioning and definition, rather than backing into it as the other commercial systems appear to be doing. Whereas all of the systems seem to be coming to the same end, CODASYL, alone has attempted to charter the path and define the architecture. Thus, the CODASYL sepcifications are most in line with the philosophy of correctly defining the logical modules and then standardizing on the interfaces connecting them. The CODASYL specifications provide the type of common architecture necessary for the distributed computer network. But although a number of CODASYL-type systems are available, they are by no means identical. The specifications themselves are in a state of change by the Data Description Language Committee. Additionally, it appears that TOTAL is widely implemented and is a reasonable alternative to the CODASYL approach. Hence, a competetive procurement should be used to select a single DBMS to suit ICAM requirements for the near future.

RECOMMENDATIONS

1. A common data base management system will be critical to the integration of ICAM software. In particular the DBMS provides the interface between all applications programs.

2. The Air Force should not attempt the development of any new general purpose data base management system due to the expenditure of resources required without any guarantee of success.

3. Functional specifications should be prepared for the competetive procurement of a commercially available data base software package to support all near-term ICAM projects. The specification should require the package to be available on all hardware systems that would be considered for CAM applications in the first few years of the program. Emphasis should be placed on obtaining modular architecture, well defined interfaces, portability of applications programs, integratability of ICAM modules, and future adaptability to a computer network system with distributed data bases. The evaluation for selection should include a benchmark demonstration of performance on a typical CAM application.

4. The Air Force should initiate participation in NBS FIPS Task Group 24, which has begun to consider government-wide needs for data base standards, and in ANSI efforts, such as the ANSI/S3/APARC Study Group, that is identifying the need for ANSI standards.

5. The Air Force should monitor the continuing research and development work with relational data base management systems.

REFERENCES

(1) International Data Corp., "The Data Base Management Software Market on IBM 360/370 Systems," International Data Corp. #1685, 214 Third Avenue, Waltham, Mass., 02154, May 1976.

(2) Fong, E., Collica, J., and Marron, B. Six Data Base Management Systems: Feature Analysis and User Experience. NBS Technical Note 887, Nov., 1975.

(3) CODASYL Programming Language Committee, Data Base Task Group Report, Available from ACM, April 1971.

(4) CODASYL Programming Language Committee, COBOL Journal of Development, 1976.

(5) ANSI X3/SPARC/Study Group - Data Base Systems, "Interim Report" ACM/SIGMOD Newsletter: fdt. 7,2 (Dec. 1975).

(6) Codd, E. F. "A Relational Model of Data for Large Shared Data Banks", Comm. ACM 13,6 (June 1976) pp. 377-397.

Berg, J. L., ed. Data Base Directions, NBS Special Publication 451, Sept., 1976.

Sibley, E. H., The CODASYL Data Base Approach: A COBOL Example of Design of Use of a Personnel File. NBSIR 74-500, Feb., 1974.

(7) "Data Base Management System Requirements, Nov. 11, 1970," by the Joint Guide-Share Data Base Requirements Group. Order from Share Secretary, Suite 750, 25 Broadway, New York, NY 10004, price $1.50.

(8) "The Debate on Data Base Management" by Richard G. Canning. EDP Analyzer, March 1972, Vista California 92083.

(9) "The Current Status of Data Management" by Richard G. Canning. EDP Analyzer, February 1974, Vista California 92083.

(10) "Introduction to Feature Analysis of Generalized DBMS," Communications of the ACM, May 1971 p. 302-318.

(11) "Concepts of Data Base Management" Honeywell's manual for their IDS Technical Presentation.

(12) "Data Base Design" The Manual for AMR International, Inc.'s Course on data base design. Copyrighted 1973 by AMR International, Inc.

STANDARDS FOR COMPUTER SYSTEMS

6. SOFTWARE TESTING AND TOOLS

INTRODUCTION

The most thoroughly tested software pieces are usually the system components; compilers, editors, file management procedures, schedulers. This is hardly surprising considering that systems software is crucial to all aspects of a marketable system. Although this brief discussion will begin with an examination of typical systems testing procedures, there remain many aspects which arise mostly in applications. These topics are covered later in the discussion.

SYSTEM TESTING

Two aspects of system testing are easily visible even in the most cursory examination.

Performance measurement is fundamental to any operation involving expensive components such as cpu, discs and memory. The natural questions of What helps best? and Who pays for what? occur over and over. Each question involves some aspect of measurement (hence, testing) of a computer system. It is quite important that a system have a fine enough clock to allow meaningful measurements of user and system states. Without such a hardware clock a system is tuned only with difficulty, and (importantly) billing of services can become confused.

Language processor testing is another significant domain for system checkout and testing. Several checkout schemes have been mentioned earlier: for COBOL, FORTRAN, BASIC, and MUMPS. For languages such as these which are heavily used on their systems the investment in language test routines is quite justified, specifically since it also promotes program transferability among processors on distinct and different systems. Testing also assures conformance to an acceptable performance standard: that is, it shows a capability to handle required language features.

APPLICATIONS TESTING

There is a vast users' area over which the tag "testing" can be attached. For sake of convenience it is often the case that static (or textual) program features are treated as distinct from dynamic (or executable) behavior.

Static Testing

Static testing encompasses several labels. At this level of the lexicon, names must be accounted for, e.g., external system names should not conflict. A common problem along these lines occurs when one module in a system is rewritten and external storage maps are changed. The new maps may not agree with other module-maps unless some monitoring is made of storage definitions, and enforcements made to maintain consistency.

Syntactic testing is obvious, and every compiler does it with greater or lesser degrees of success. A number of points may be worth mentioning. First, the compilation facilities can serve as good enforcers of any system "standards" that are required for transportability or clarity. The compiler is an especially good and effective place for enforcement, in that failure to comply can imply failure to get any work done. Secondly, many compilers have extremely poor error message and diagnostic facilities. For some reason this is especially true for COBOL, and it seems to be more the fault of the compilers than the language. Some test programs have been written to test compiler diagnostics, but further work could be done on this aspect. The problem is easy to ignore but important to the everyday programmer. Thirdly, some languages such as early PL/1 have conventions, defaults design choices

which make compilation errors less apparent, and sometimes, almost invisible!

Semantic and functional testing are more wishes than current realizable technology. Questions arise on the meaning of primitive operations, machine dependencies (e.g., word size), and representations. Functional testing, or proof-of-correctness asks whether a program corresponds to its original specifications; this is an extremely difficult problem and little progress has been made of a practical nature.

Dynamic Testing

The first aspect of dynamic testing is one of correspondence. Has the correct problem been solved? Comparing actual runs against true answers may reveal the ultimate bug--having solved the wrong problem.

Performance measurement has been mentioned regarding the need for a good hardware clock for accounting and tuning. Similar requirements apply directly to applications programs. Three other points are worth mentioning:

(i) Program conversion and modularization--Help find "related" code to load together;

(ii) Learn variations in efficiency, isolate bottlenecks and non- critical parts;

(iii) Subsetting. Given cases of interest, chart "live" segments in a large program, thereby limiting the code to that of immediate interest.

The third area of dynamic testing could be called functional- empirical. One wants to thoroughly exercise a program [Huang, 1975]. In addition, the instrumented code can be tested against standard test cases to check that the latter are truly thorough. Weaknesses in test data are usually sins of omission, so parts of the program may not be used. This shows up immediately when program segment counters are zero.

Mathematical Software Testing

In CAM system utilization, any numerical errors arising during design or production management would be reflected in the finished product. It is important then that CAM engineers have confidence in the numerical performance of software as well as in the logical correctness of the programs. Although there are no standards for the numerical quality of mathematical software testing, there are testing practices that are somewhat de facto standards. We review some of these practices in this section.

Mathematical software according to Cody [1] denotes those computer programs that implement mathematical algorithms. Mathematical theorems are usually established about the theoretical nature of the algorithms and their convergence properties. In general, such results do not concern themselves with finite machine arithmetic. Very precise theoretical error bounds can usually be established for these theoretical function approximations [3].

However, when machine considerations such as word length, radix, floating or fixed point arithmetic are introduced, the theoretical algorithm must be restructured for a particular implementation in order not to lose the mathematical properties required to assure the theoretical error bounds. The restructuring of the theoretical algorithm for a particular implementation

might be referred to as the machine algorithm. By an implementation of a theoretical algorithm we mean the restructuring of the computation to make use of particular machine optimization of computations and the programming language used.

With respect to the testing of mathematical software there are two divisions. These are:

(1) Programming Languages Supporting Mathematical Functions
(2) Scientific/Engineering Support Mathematical Software

These divisions arise because, for language support mathematical software, i.e., mathematical function routines, there has emerged what appears to be a consensus approach, or de facto standard, for testing the mathematical function routines such as exponential, sine, cosine, etc. However, for general scientific software there are no general standards, but there are two approaches that might be referred to as test or benchmark problem sets and roundoff error analysis, (See Cody [1]). These latter approaches will not be considered here since we are concerned only with the mathematical function libraries that impinge on the language standards.

The simplest type of error testing is a direct comparison of computed function values against published tables. There are several difficulties with a naive application of this method. The first difficulty is the entry and storage of the large data set that would be needed in order to perform an exhaustive comparison. It would also require detailed checking of the input data to determine transcription error, and it would of course require editing of the data after entry. The next difficulty is the sparseness of the entries. Approximation procedures would have to be programmed to generate reference values to test the function subroutines at arguments between the table entries. The major difficulty is that these table-generated data points do not provide a sufficient sample of the behavior of the routine under test. Sample sizes of several thousand arguments have been used by some testers. Furthermore, table generated tests do not provide flexibility to the user.

Although the data table methodology is cumbersome and requires manual checking and preparation, the general idea is the same as the methodology used by the function testing community. The difference lies in the fact that the procedures for generating the comparison tables have been automated and allow a wider testing range and flexibility.

The most prominent scheme of accuracy checking is one that involves automatic tabular comparison where the standard table values are generated within the machine as needed. This usually requires the provision of a subroutine to compute standard values for a function to a precision greater than that of the routine under test. With such a routine it is possible to generate a table of comparison values automatically that can either be stored for future use or used immediately at the time of generation. This routine would generate high precision function values for specific arguments or for random arguments.

The emphasis in the mathematical testing community has been on the statistical sampling of the accuracy because of the objective ability to measure this. The approach has been widely used by a number of researchers (See Kuki [3], Cody [4], and Lozier [10] for examples.)

With regard to de facto testing methodologies, mathematical software divides itself into two classes. First the language support elementary function routines and second general scientific routines that are collected into libraries. There is a well- defined procedure for testing the language support function routines. However, there are a number of procedures that rely on performing arithmetics other than floating point that have been used to estimate numerical error. Since the process of developing scientific libraries, especially those that may be used to design critical parts, is a lengthy and expensive one, it is imperative to identify as soon as possible viable numerical software testing procedures and begin using them in evaluating user libraries.

SOFTWARE TOOLS

It is evident from prevailing experience and research that every software production project, regardless of complexity, must include a tool provisioning activity. The toolmaker faces several questions, to be answered in collaboration with his project manager: Is there a commonly accepted set of standardized tools applicable to every project?; What set of special tools can be identified for a project at the outset?; Are necessary tools already available as commercial packages with acceptable cost?; What are the economical approaches to creating special tools and modifying them as may be needed in the course of a project? Corresponding evidence shows there is inordinate difficulty in selecting tools tools from the marketplace shelf. Commercial items are available at reasonable cost, but there is essentially no standardization of tool capabilities. The number of suppliers and the diversity of packages confound the would-be buyer. But equally important, proprietary packages cannot be modified and tailored by the buyer since the source code is usually not delivered with purchase. Although a basic set of tools is identifiable for any project, it appears that that special modifications are warranted in many cases. Furthermore, a general expansion and integration of available tool functions would be well-advised to cope with the widely-recognized problem of software quality control. The following analysis tends to support a recommendation for standardization of basic tools at source code level, so that CAM software production can be conducted with a common set of tools amenable to user extension and specialization.

Types of Tools

The only standard tool for software production today is the high-level language compiler. This statement applies the traditional understanding that a standard is a formal specification produced by a recognized professional group for nearly universal application. Yet, national and international standardization of compilers has only addressed programming language definition, ignoring crucial capabilities such as the form and content of output listings, accuracy and scope of diagnostic messages, debugging features, and interactive and batch modes.

Even so, use and economics of tool design have lead to commonly discernible types. These tools cannot be called defacto standards, for they reflect only similar purpose and function, and not by any means a near equivalence of capability brought about by uniform commercial demand. The following common types have been determined from a survey of commercial packages. Omitted are compilers, assemblers, data base management systems, utility routines, application programs or libraries (e.g. mathematical routines). Also excluded are replacement packages for software normally offered by a hardware

vendor, such as operating systems and I/O access methods. Common tools are: Abort diagnoses--provide full or selective dumps; Breakpoint control--for interactive debugging; Cross reference generator; Data auditor/catalog--analyzes data relationships; Error analysis and recovery--intercept selected abnormal terminations; File or library manager--centralized retrieval and update; Flowchart generator; Program auditor--checks conformance of programs; Program execution monitor--see testing sections, above; Program formatter/documentor-- Rearranges and structures source text; Project manager--scheduling and production aid; Resource monitor--accounting information; Shorthand or macro expander--may also include decision table expansion; Source level translator--e.g. RPG to COBOL; Test data generator; Test simulator--simulates execution and flow, allow user decisions in testing; Text editor.

Minimum Essential Tools

Contemoorary experience and practioners' concensus are sufficient to recommend some tools as essential for almost any software development project. Exceptions may arise if a computer has unusual architecture or limited capabilities (e.g. no mass storage). Minicomputer systems are generally included, particularly since the UNIX system [Ritchie] has demonstrated that a highly effective, interactive programming support system is practical on a low-cost minicomputer.

It is recommended that in general program development be done with support of an interactive computer system. Interactive support increases productivity throughout the changes, debugging, and testing that characterize most projects.

The primary tool is the compiler for the high-level programming language. Again, experience has amply proven enhancements of programming productivity using high-level languages. Only selected procedures critical to system performance need to be assembly-language coded for extreme execution speed. Other essential tools are recommended as a minimum complement for most projects:

Text editor - For entering, correcting, and modifying such texts as program specifications and design documentation. Requires a facility for online storage and recall of named text units for inspection, printing or editing.

Program editor - For entering, correcting, and modifying program texts. With free-form programming languages, one editor could serve both as text and program editor.

Program librarian - For storing all program texts, associated job control statements, common data definitions, and test data, and maintaining a chronological record of modifications between distinct versions. Includes appropriate access controls for members of the project group.

Debugger - For analyzing program behavior during execution on test data input, and deriving execution statistics and traces to help correlate program output with the results of individual high-level language statements.

Project manager - For recording chronologically the activity of the individual project members on defined program modules and deliverables of the project. Standard specifications of functions for each tool type appear feasible and desirable, and would assist those who undertake toolmaking without benefit of prior study and experience. Yet it is clear that

individual projects often may need to create special features that would not be available in standardized tools. Various project requirements or circumstances may dictate such specializations. For instance, large projects with many personnel especially would benefit from extensions to automatically enforce unique design standards and practices that are difficult to ensure through personal communications and code inspections.

Desirable specializations may range in difficulty from minor extensions of extant tools to new composite tools formed by integrating and refining several distinct packages. Both of these cases require the original tool's source code--ideally in a system-standard high level language--and thorough documentation of course. The latter case also requires that the building block tools be carefully designed, with flexible interfaces and modular design, permitting extensive modifications with relative ease.

RECOMMENDATIONS

It is appropriate therefore to recommend studies and development on CAM programming tools, with the following goals:

1. to make widely available a set of CAM building block tools, with standard designs and source code in CAM-system high level language;

2. to evaluate alternatives for interfaces and modular design that would support major modifications of tools without loss of efficiency and performance; and

3. to develop guidelines for rapid and reliable specialization of tools from available building blocks, based upon the characteristics of CAM projects most benefitting from special tools.

REFERENCES

(Computer Validation)

Federal Property Management Regulation 101-32.1305-a Validation of COBOL Compilers.

NBS Special Publication 399, Vols. 1-3, "NBS FORTRAN Test Programs."

MDC/29, MUMPS Validation Program User Guide.

(General)

NBS Technical Notes 874, "Software Testing For Network Services"; 849, "A FORTRAN Analyzer"; 800, "Computer Networking" (above some approaches to quality assurance).

K. V. Hanford, "Automatic Generation of Test Cases," IBM Systems Journal 9, 4(1970), pp. 242-257.

J. C. Huang, "An Approach to Program Testing," Computing Surveys 7, 3(September 1975), pp. 113-128.

(Mathematical Software Testing)

W. J. Cody, "The Evaluation of Mathematical Software," Program Test Methods, EWilliam C. Hetzel, Prentice-Hall, Inc., Englewood Cliffs, NJ (1973), 121.

C. T. Fike, Computer Evaluation of Mathematical Functions, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1968).

H. Kuki, "Mathematical Function Subprograms for Basis System Libraries -- Objectives, Constraints and Trade-Off," Mathematical Software, Academic Press, NY, p187-199.

W. J. Cody, "Performance Testing of Function Subroutine," AFIPS ConProc., Vol. 34, 1969.

N. A. Clark, W. J. Cody, K. E. Hillstrom and E. A. Thieleker, "Performance statistics of the FORTRAN IV (H) Library for the IBM Systems/360," Report ANL7321, Argonne National Laboratories (1967).

C. L. Lawson, "Study of the Accuracy of the Double Precision Arithmetic Operations on the IBM 7094 Computer," JPL Tech. Memo #33-142, Jet Propulsion Laboratory, Pasadena, 1963.

D. W. Lozier, L. C. Maximon, W. L. Sadowski, "A Bit Comparison Program for Algorithm Testing," The Computer Journal, Vol6, N2, pp. 111-117.

A.C.R. Newbery, Anne P. Leigh, "Consistency Tests for Elementary Functions," AFIPS ConProc., Vol. 39, 1971.

W. J. Cody, "Software for the Elementary Functions," Mathematical Software, R. Rice Ed., Academic Press, NY, 1971.

D.W. Lozier, L. C. Maximon, and W. L. Sadowski, "Performance Testing of a FORTRAN Library of Mathematical Functions Routines-- A Case Study in the Application of Testing Techniques," Journ. of Res., NBS, B. Math. Sce., Vol. 77B, Nos. 3 & 4, July- December 1973.

(Software tools)

Brooks, Frederick P., Jr. The mythical man-month, Addison-Wesley Publishing Company, Reading, Mass., 1975, p.128.

Reifer, Donald J., "Automated aids for reliable software," Proceedings of the International Conference on Reliable Software, SIGPLAN Notices 10, 6(June 1975), pp. 131-140.

Ritchie, Dennis M. and Thompson, Ken. "The UNIX Time-sharing system," Comm. ACM 17, 7(July 1974), pp. 365-375.

Van Dam, Andries, and Rice, David E. "On-line text editing: a survey," Computing Surveys 3, 3(Sept. 1971), pp. 103-105.

Wichmann, B.A. "A syntax checker for ALGOL 60," NPL Report NAC 53, August 1974, Division of Numerical Analysis and Computing, National Physical Laboratory, Teddington, Middlesex, England.

STANDARDS FOR COMPUTER SYSTEMS

7. DOCUMENTATION STANDARDS

FIPS PUB 38 DOCUMENTATION REQUIREMENTS

PROGRAM COMPLEXITY	Software Summary	Users Manual	Operations Manual	Program Maintenance Manual	Test Plan	Functional Requirements Document	System/Subsystem Specification	Test Analysis Report	Program Specification	Data Requirements Document	Data Base Specification
One shot-single use programs*	X										
Small limited purpose programs	X	X									
Multipurpose-general programs	X	X	X	X	X			**		***	***
Large scale-multiuser programs	X	X	X	X	X	X		**		***	***
Large scale systems Common data base Multi application programs	X	X	X	X	X	X	X	X		***	***
Totally integrated systems Multi disipline users Multi contractor development	X	X	X	X	X	X	X	X	X	X	X

* Additional documentation may be required by local policy.
** Test Analysis Report may be prepared informally.
*** May or may not be needed depending upon project.

Figure 1 Project Complexity Influences Documentation Requirements

INTRODUCTION

One of the recent developments in software has been the emphasis on control of the complete generation cycle, and an examination of the dependencies that should exist in this cycle. Documentation preparation should be treated as a continuing effort evolving from preliminary requirements-drafts, through change and reviews to the final documentation and continuation documentation of the delivered software products. Since clear and complete documentation is a keystone for portable and maintainable software modules, definitive guidelines for its preparation are of vital importance to the Air Force program. A documentation administrator should be assigned to work with the contracting officer to define and enforce requirements for clear documentation including source code of system components which should be Air Force property. The documentation administrator should have a clear idea of what is in the CAM system; therefore a model should be constantly kept of system components to catch any omissions. The model should be available to users for feedback on its adequacy and degree of coverage in current documentation.

The extent of documentation should depend on the size, complexity and value of the project. Special requirements are necessary for certain well-defined components; for example, interactive processors (editors, language translators, networking modules) should have available on-line "help" files to show how to use them. A user should be able to run these interactive components without shutting off his terminal, but rather, using it to advantage.

Documentation should spell out clearly the specific software compatibilities and incompatibilities; i.e. does compiler X read files of type Y. In addition, compatibilities should be spelled out as specific mandatory requirements in early stages of design documentation, and the design requirements written to preclude as many undesirable conflicts as possible. The extent, detail, and formality of software documentation must be included in all contractual arrangements for software procurement.

Automated aids for development and maintenance of ICAM documentation would be a great assistance to both contractors and the documentation administrator. The development of such aids should be considered as an early ICAM project.

SOFTWARE DOCUMENTATION GUIDELINES

In reviewing various guidelines for software documentation a growing tendency is noted toward the development of a full life cycle management system for the software creation process. NASA documentation standards for part of the Appollo project are a good example. Entitled "Procedures for Management Control of Software Development for Appollo" the guidelines address each functional step from requirements analysis to coding, testing and maintenance. Specifications are made for the documentation required for each functional step.

Considerable progress has been made in Federal standards for software documentation: FIPS PUB 38 is prehaps best suited for the development of large systems, providing as it does a checklist of items worthy of detailed attention in a project. The documentation categories of GIPS PUB 38 begin with functional requirements, pass through the natural stages of a project, and end with test plans and analyses. The standard recognizes that not all documentation categories are needed for every project. Rather as the size, complexity and visibility of a project increases so does its need for more extensive documentation. Figure 1 has been abstracted from FIPS PUB 38 to emphasize this concept.

CAM-I has established documentation standards to assure the availability of detailed information on the software products developed. The standard defines the structure, content and use of ten separate documents to be developed within each software project. The functional content of the CAM-I specified documents is quite similar to those in FIPS PUB 38 although the latter are more completely defined.

Three other differences are noted:

FIPS PUB 38 breaks out separate Test Plans and Test Analysis Reports rather than embedding the functions in other documents. These two are very important for validating the applications module and for assuring that the coding meets portability requirements.

CAM-I describes a Project Status Report necessary for good Air Force management control of the software development process. Also included is a Project Prospectus which ICAM may find quite useful as a brief descriptive outline of a module that can be sent to all prospective users or included in press releases, etc.

FIPS PUB 38 contains an in depth specification for defining a module's interdependence with various data bases. In the distributed processing, integrated systems environment envisioned for ICAM due attention must be given to these data requirements.

A number of miscellaneous recommendations exist for bits and pieces of program development. For example, there is FIPS PUB 24 on flowcharting. In addition, a large number of texts and articles exist. Yourdon, pages 23-24, provides some excellent common sense on documentation and maintenance of program modules, including advice on the use of variables in original codings.

The Department of Defense has issued in December 1972 a manual on Automated Data Systems Documentation Standards which has been implemented by all three services.

Smaller programs and projects in the CAM undertaking may find the work of the American Nuclear Society (244 East Orden Ave., Hinsdale, Illinois 60521) useful. Two documents of the Society are referenced at the end of this chapter.

RECOMMENDATIONS

1. The Air Force should extend FIPS 38 in order to have more detailed guidelines for computer program documentation and to software. The guidelines should use the framework of FIPS 38, and may incorporate useful sections of the CAM-I, NASA, and American Nuclear Society publications.

2. The Air Force should establish a documentation administrator to specify and maintain system and software documentation. Since there are many disparate CAM interests, a tight rein on documentation in the first development stages combined with industry participation (as practical) could promote a clarification of standard CAM system components and procedures.

3. Automated aids for production and maintenance of documentation should be considered as early program efforts.

REFERENCES

(1) Guidelines for Documentation of Computer Programs and Automated Data Systems. Federal Information Processing Standards Publications FIPS PUB 38, February 15, 1976. US Department of Commerce, National Bureau of Standards.

(2) Guidelines for the Documentation of Digital Computer Programs. Americal Nuclear Society, ANS-10.3. (Also ANSI N413).

(3) ANS Standard. Recommended Practices to Facilitate the Interchange of Digital Computer Programs. ANS STD 3-1971.

(4) Flowchart Symbols and Their Usage in Information Processing. FIPS PUB 24, June 30, 1973. (Same as ANSI X3.5-1970)

(5) Daniel D. McCracken. "How to write a readable FORTRAN program." DATAMATION (Oct. 1972), pp. 73-77.

(6) E. Yourdon. Techniques of Program Structure and Design, Prentice-Hall, Inc., (Englewood Cliffs, N.J., 1975).

(7) Department of Defense Automated Data Systems Documentation Standards Manual, 4120.17M, December 1972.

(8) CAM-I Standard for Computer Program Documentation, STD-73-SC-01, May 1973.

STANDARDS FOR COMPUTER SYSTEMS

8. MEDIA STANDARDS

INTRODUCTION

Of basic importance to any computer system is the media on which computer readable information is prepared, stored and exchanged. Adherence to formal media standards is a simple economic principle. Consider a computer program of 20 thousand language statements punched onto a nonstandard card deck. A lengthy and costly keypunch task would await anyone wishing to use this program. Fortunately the industry has pretty well standardized the media in common use; punched cards, magnetic tape, punched paper tape, and disk packs.

PUNCHED CARDS

These are the familiar 3 1/4 x 7 3/8 inch heavy paper cards that are as common to a computer programmer as nails to a carpenter. ANSI Standard X3.11 describes the physical attributes and quality of these while ANSI Standard X3.21 defines the size and location of the rectangular holes. It should be remembered that for punched cards to be readily transportable it is necessary that a specification be made to the coding of characters on the card. See the Hollerith Punched Paper Card Code.

MAGNETIC TAPE

Specifications for 1/2 inch wide magnetic tape and reels are given in ANSI Standard X3.40 while format and recording data are detailed in ANSI X3.14 and X3.22. Together these standards enable mechanical, magnetic and recording format interchangeability of data among various systems and equipment utilizing the American National Standard Code for Information Exchange, X3.4. Magnetic tape written in this manner provides the best means of exchanging computer data. It is also a convenient method for use in archieval storage and distribution of ICAM developed software. A recent DATAMATION article details recommended procedures for maintaining good quality control over a magnetic tape based archieval record storage facility.

MAGNETIC DISK PACKS

ANSI Standard X3.46-1974 provides the general, magnetic and physical requirements for interchangeability of six-disk packs among various disk drives. However, ANSI leaves the formating and recording of data to the manufactures' discression. As a result absolute compatibility is not guaranteed. The six-disk pack is giving way to a twelve-disk pack for which an ANSI standard is yet unavailable.

PUNCHED PAPER TAPE

Two ANSI Standards exist for describing punched paper tape. This media is most extensively used for the numerical control of machine tools. However, some use is seen for data storage in minicomputer applications. ANSI X3.29 details the physical characteristics and acceptance test procedures for one inch wide and eleven-sixteenths inch wide unpunched, oiled paper tpae. ANSI X3.18 covers the physical dimensions of the tape as well as its perforations. Caution is advised that to insure portability of paper tapes one must specify the format and coding of the data as well as the physical characteristics.

When used in NC applications, punched paper tape has been justly described as the weakest link in the process. This is a result of the many maintenance problems that exist on paper tape punches and readers. It would be unfortunate if the Air Force perpetuated the use of paper tape in large scale CAM Systems. Direct wire link is today far more efficient, reliable, and versatile.

RECOMMENDATIONS

1. Magnetic tape should be used as the primary means of exchanging and storing computer readable information.

2. All magnetic tapes should conform to ANSI X3.40, X3.14 and X3.22 Standards.

3. The use of punched paper cards should be deemphasized as it is an inefficient media of information storage. However, where it is necessary to produce cards the ANSI X3.11 and ANSI X3.21 Standards should be specified.

4. The use of punched paper tape should be avoided for transmitting NC data. Direct wire link from computer to machine controller provides a higher quality system configuration.

REFERENCES

(1) Specification for General Purpose Paper Cards for Information Processing, ANSI X3.11, October 1969, American National Standards Institute, Inc., 1430 Broadway, New York City, New York 10018.

(2) Rectangular Holes in Twelve-Row Punched Cards, ANSI X3.21, October 1976, American National Standards Institute.

(3) Recorded Magnetic Tape for Information Interchange;
200 CPI, NRZI, ANSI X3.14, December 1972
800 CPI, NRZI, ANSI X3.22, December 1972
9 Track 200 CPI, NRZI, ANSI X3.40, February 1976
9 Track 800 CPI, NRZI, ANSI X3.40, February 1976
9 Track 1600 CPI, PE, ANSI X3.40, February 1976
American National Standards Institute.

(4) One Inch Perforated Paper Tape for Information Interchange, ANSI X3.18, March 1974, American National Standards Institute.

(5) Eleven-Sixteenths-Inch Perforated Paper Tape for Information Interchange, ANSI X3.19, March 1974, American National Standards Institute.

(6) Specification of Properties of Unpunched Oiled Paper Perforator Tape, ANSI X3.29, May 1971, American National Standards Institute.

(7) Unrecorded Magnetic Six-Disk Pack, ANSI X3.46-1974, May 1974, American National Standards Institute.

(8) Archival Data Storage, Sidney B. Geller, DATAMATION, October 1974, pp 72-80.

APPENDIX A

DATA BASE MANAGEMENT FILE STRUCTURES

FILE STRUCTURE AND ACCESS METHOD

- Sequential
- Random
- List

COMPLEX STRUCTURES

- Indexed Sequential
- Tree
- Network
- Sets

File Structure and Access Method

With present commerical data base management systems, many of the characteristics of their operations are a result of the particular file structure and access method used. We will describe here, the various methods used with their advantages and limitations. These will be general remarks and do not indicate that some of the limitations cannot be resolved by clever alterations, however, these additional correction factors are usually expensive in terms of main memory, storage space, or retrieval time overhead.

We will partition the structure/access methods into three types - sequential, random, and list.

1) SEQUENTIAL - Here, the record is contiguous, its location is based on the value a record's key has relative to other records. (Storage devices tend to be tapes and cards.)

   Advantages - very rapid access to the next file.

   Limitations - a new file has to be written for each update to a record or if a new additional record is inserted, retrieving records out of their normal sequence is virtually impossible, if a file is to be retrieved by more than one key (e.g. a water pump specification may be under the key-engine parts and the key-aluminum parts), then duplicate files have to be created leading to much data redundancy.

2) RANDOM - records are stored and retrieved on the basis of a predictable relationship between the key of the record and the address of the location where the record is stored (Storage devices are drums and discs). Three general methods are used to determine the address:

   a) Direct Address - the address of the Jones' record (for example) i.e., the disc, track, and sector location (number 3469, for example) is known by the programmer and is supplied at storage and retrieval times.

      Advantages - allows equally fast access to all records.

      Limitation - additional effort is required to maintain these direct addresses.

   b) Dictionary lookup - both the address and the record key are stored in a dictionary (table or index). To locate the "Jones" record, the dictionary is scanned for a match on this name. Then the location address is picked up and the record retrieved.

      Advantage - the system maintains the actual address.

      Limitation - additional time required to scan the dictionary, and additional storage space required for the indices.

   c) Calculation or Randomization - the record key is converted through some kind of hash code process into an address.

      Advantage - can retrieve all records equally fast without having to search a data file or index file, and records can be sorted, retrieved, and updated in place without effecting other records in the storage media.

Limitation - may not yield a unique address for each record, therefore, if overlap occurs - causes a condition called overflow therefore have to use pointers indicating where the overflow record is stored.

3) LIST - The basic concept here is to separate the logical organization from the physical organization. The next logical record desired can be "pointed" to, and need not be the next physical record as in sequential organization. Thus, new records can be placed in any space that is available. There are three basic types of list organization:

a) Simple list - pointers are used to cause a record to be a member of as many lists as desired under any number of different keys (like our water pump example).

Advantage - there is no duplication of the record in the data base and, therefore, no multi-updates, in addition, the record can be stored anywhere in the file where there is space.

Limitation - additional space required for pointers, user's system must take into consideration the length of the lists as well as the number of lists in which a record participates - these factors increase the file maintenance overhead time since if a record is deleted, all of the lists it was involved in have to be readjusted to bypass it.

b) Inverted list - makes available every data item as a key. Such an organization requires a table or index of all data values in the system and contains the addresses of all record locations where those values occur.

Advantage - allows access to all data with equal ease - this gives good query and reporting capabilities - good at handling hierarchical data structures.

Limitation - the index table required can be as large or larger than the data itself, as with the simple list system above, there can be much maintenance required in storing and updating data in large tables - this system has difficulty in handling, requests of records located in different branches and/or levels of a hierarchical structure, or located in network type data structures.

c) Ring - the last record in a list points (by a pointer) back to the first (forming a ring structure).

Advantage - very powerful as it provides retrieve and process capabilities in both directions while allowing branching to to other logically related ring structures.

Limitation - again heavy record pointer overhead - these searches can be quite time consuming if the data base is not "tuned" properly - e.g. if the pointers send you back and forth to different discs for each record in the list to be searched.

Compex Structures

In addition, to the simpler methods of storing and relating records described above more complex relational structures can be defined.

1) INDEXED SEQUENTIAL - The file is organized so that records can be accessed either by use of an index or in a sequential fashion

(of the indices or the data records). Indices containing record keys and addresses may exist for each record in the file.

Advantages - it provides some of the speed of retrieval of the sequential file (once you are at the right location) by using indicies to increase the speed of entering the file at the proper place.

Limitations - still large maintenance problems of sequential files with the additional maintenance overhead of indices.

2) TREE - several layers of indices or records are used to establish a tree-branched hierarchy. Indices may be organized as lists or in sequence, with either method's characteristics.

Advantage - convenient in maintaining large dictionaries - allows the data to be structured to represent rather complex data relationships.

Limitation - only a single entry point into each hierarchical relationship - therefore, can require a long time to search the hierarchy for one piece of data. Does not represent a network related data structure, since no branches of the tree touch.

3) NETWORK - specialized form of a hierarchy where all the branches can be interconnected.

Advantage - permits the storage and retrieval mechanisms of the data management system to start with any record in the file and move in multidirections throughout the hierarchy. The network structure allows the data to accurately model real world manufacturing and business relationships.

Limitation - updating and record deletion can involve large maintenance due to the involved relationships.

4) SETS - this is the CODASYL concept of relating data records. Each set type consists of one record type declared as owner and plus one or more record types declared as members. Connection between the owner record and the member records is made by chains (embedded pointers) or pointer arrays (indices). Both tree and network structures can easily be built from these sets. At the least, sets are connected by one way pointers, but the user may also choose to declare two-way pointers for given sets. These sets can then be searched in either direction with equal efficiency. In addition, the member records can have pointers to the owner record, to avoid stepping through the chain to obtain the owner.

NBS-114A (REV. 7-73)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBSIR-1094(R)	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE

STANDARDS FOR COMPUTER AIDED MANUFACTURING

Third Interim Report

5. Publication Date: January 1977

6. Performing Organization Code: 600.20

7. AUTHOR(S)

Dr. John M. Evans, Jr., et. al.

8. Performing Organ. Report No.

9. PERFORMING ORGANIZATION NAME AND ADDRESS

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

10. Project/Task/Work Unit No.

11. Contract/Grant No.

12. Sponsoring Organization Name and Complete Address (*Street, City, State, ZIP*)

Manufacturing Technology Division
Air Force Materials Laboratory
Wright-Patterson Air Force Base, Ohio 45433

13. Type of Report & Period Covered: Third Interim

14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT (*A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.*)

In the previous interim reports existing and potential standards were identified which will be useful to the Air Force in the development and implementation of integrated computer aided manufacturing systems; and a comprehensive reference data base was provided on all formal and de facto standards that are considered to be relevant to the Air Force Program. This report discusses the utility of these standards to the Air Force Program and in each relevant standards area recommends a best approach to follow either toward adopting existing standards or toward developing needed standards.

17. KEY WORDS (*six to twelve entries; alphabetical order; capitalize only the first letter of the first key word unless a proper name; separated by semicolons*)

CAM standards; computer aided manufacturing; communications; computer systems; operating systems; system integration.

18. AVAILABILITY

[ ] Unlimited

[X] For Official Distribution. Do Not Release to NTIS

[ ] Order From Sup. of Doc., U.S. Government Printing Office Washington, D.C. 20402, SD Cat. No. C13

[ ] Order From National Technical Information Service (NTIS) Springfield, Virginia 22151

19. SECURITY CLASS (THIS REPORT)	21. NO. OF PAGES
UNCLASSIFIED	
20. SECURITY CLASS (THIS PAGE)	22. Price
UNCLASSIFIED	

USCOMM-DC 29042-P74