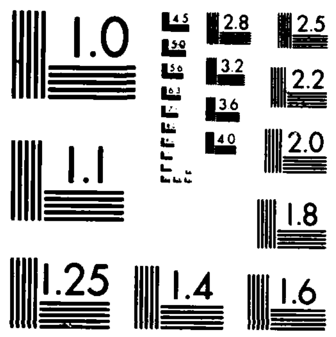
AD-A174 596   APPLYING ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE)   1/1
CONCEPTS FOR COMPUTER (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF SYST   K M AUSTIN
UNCLASSIFIED   SEP 86 AFIT/GLM/ENC/86S-2   F/G 9/2   NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

2

APPLYING ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE) CONCEPTS FOR COMPUTER INTEGRATED MANUFACTURING SYSTEMS (CIMS) SOLUTIONS

THESIS

Kathleen M. Austin
Captain, USAF

AFIT/GLM/ENC/86S-2

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GLM/ENC/86S-2

APPLYING ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE) CONCEPTS FOR COMPUTER INTEGRATED MANUFACTURING SYSTEMS (CIMS) SOLUTIONS

THESIS

Kathleen M. Austin
Captain, USAF

AFIT/GLM/ENC/86S-2

The contents of the document are technically accurate, and no sensitive items, detrimental ideas, or deleterious information is contained therein. Furthermore, the views expressed in the document are those of the author and do not necessarily reflect the views of the School of Systems and Logistics, the Air University, the United States Air Force, or the Department of Defense.

AFIT/GLM/ENC/86S-2

APPLYING ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE) CONCEPTS FOR COMPUTER INTEGRATED MANUFACTURING SYSTEMS (CIMS) SOLUTIONS

THESIS

Presented to the Faculty of the School of Systems and Logistics
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Logistics Management

Kathleen M. Austin, B.S.B.A.
Captain, USAF

September 1986

Preface

The purpose of this research effort was to compare the concepts of Ada Programming Support Environments (ASPE) and Computer Integrated Manufacturing Systems (CIMS) to see if the similarities would justify applying APSE concepts to help solve CIMS problems. There is an immediate need for this kind of integrated, interdisciplinary research because of the vast amounts of money the Department of Defense is investing in each of the concepts.

Department of Defense funding is coming under greater and greater scrutiny. It is time for us to examine even more ways we can work smarter instead of harder. Interdisciplinary research is one way we can work smarter. Using the principals of the General Systems Theory that we all learned during the Logistics Systems Overview course, we should look to see if the research efforts we attempt are actually re-inventions of the same conceptual wheel.

During my studies at AFIT, I took many additional courses in the School of Engineering. Time and time again, I was struck by the similarities of the Engineering School courses to the Logistics School courses. The "language" of the courses in each school was different, but the concepts and the intent were the same. One of my criteria for determining a thesis topic was to find an area from

each school and integrate the two into a thesis with "crossover" potential: understandable by professionals from each school. Another of my criteria was to do meaningful research, i.e., research that would contribute to and extend the current body of knowledge. I found not only a meaningful thesis topic with two areas of extreme personal interest, but also an outstanding thesis advisor with similar interdisciplinary interests.

A list of acronyms has been included to assist the reader. This list immediately precedes the abstract and Chapter I.

This thesis effort could not have been possible without a great deal of help from others. I wish to thank my daughters, Britt and Kirsten Peschke, for their patience and understanding during the long months of this research effort. Thank you, Professor Dan Reynolds for rejuvenating my husband, so that he could keep me going. Phyllis Reynolds, thank you for your superb typing support. I also wish to thank Professor Karyl Adams for her extremely valuable assistance and ideas. I am deeply indebted to my thesis advisor, Maj Pat Lawlis, for her patience, knowledge, interest, and support. Most of all, I would like to thank my mother for her prayers and support and my husband, Dick Peschke. Without him, none of this would have been possible. Thank you for always being there for me.

— Kathleen M. Austin

Table of Contents

Page

List of Figures

List of Tables

List of Acronyms

ACM	Association for Computing Machinery
AFIT	Air Force Institute of Technology
AJPO	Ada Joint Program Office
AML	A Manufacturing Language
ANSI	American National Standards Institute
APSE	Ada Programming Support Environment
APT	Automatically Programmed Tools
ARCADE	AFIT Research Concept for an Ada Development Environment
CAD	Computer Aided Design
CAD/CAM	Computer Aided Design and Computer Aided Manufacturing
CAIS	Common APSE Interface Set
CAM	Computer Aided Manufacturing
CEC	Commission of European Communities
CIMS	Computer Integrated Manufacturing Systems
CNC	Computer Numerically Controlled
COMPACT II	Computer Programs for Automatically Controlling Tools
COPICS	Communications Oriented Production and Inventory Control System
DARPA	Defense Advanced Research Projects Agency
DBMS	Database Management System
DNC	Direct Numerical Control
DoD	Department of Defense

DOD-1	DoD Programming Language 1
DDR&E	Director of Defense Research and Engineering
E & V	Evaluation and Validation
HOL	Higher Order Language
HOLWG	Higher Order Language Working Group
ICAM	Integrated Computer Aided Manufacturing
ISO	International Standards Organization
KAPSE	Kernal Ada Programming Support Environment
KIT	KAPSE Interface Team
KITIA	KAPSE Interface Team from Industry and Academia
MAP	Manufacturing Automation Protocol
MAPSE	Minimal Ada Programming Support Environment
NATO	North Atlantic Treaty Organization
NAVELEX	Naval Electronics Systems Command
NBS	National Bureau of Standards
NC	Numerically Controlled
RTSM	Run Time Support Monitor
SIGPLAN	Special Interest Group of the ACM for Programming Languages

Abstract

This thesis compares the concepts of Ada Programming Support Environments (APSE) and Computer Integrated Manufacturing Systems (CIMS) to see if they are similar enough to apply APSE concepts for solving CIMS problems. After establishing a discussion baseline for each of the concepts, the objectives, design guidelines and pictorial representations of an APSE and a CIMS are first compared. Then the two concepts are compared at the database, interfaces and toolset level. The major differences between an APSE and a CIMS are also identified. Finally, after establishing that similarities do exist between APSE and CIMS concepts, recommendations are made for applying APSE concepts for CIMS solutions that will contribute to achieving Air Force as well as U.S. manufacturing goals.

APPLYING ADA* PROGRAMMING SUPPORT ENVIRONMENT (APSE) CONCEPTS FOR COMPUTER INTEGRATED MANUFACTURING SYSTEMS (CIMS) SOLUTIONS

I. Introduction

The United States' market share in electronics has slipped below 10 percent, less than a third of its share in 1965. Additionally, steel production is down 35 percent and the U.S. share of auto production has been cut in half. In 1979, Japan could produce and ship a car to our country for $1500 less than Detroit could produce a similar car; by 1983, that figure had grown to $2500 and today's figure is about $1900. The manufacturing world is much more competitive now than just a few years ago and America has fallen behind (Port, 1986:100; Mitchell, 1986:103).

With the emergence of Japan, Korea, Brazil, Malaysia and Taiwan as key competitors with the United States, new standards of quality, reliability, productivity and performance have been set. Unless our country is able to improve, we will fall even farther behind (Meredith, 1985:42).

This research effort examines two concepts, Ada Programming Support Environments (APSE) and Computer

*Ada is a registered trademark of the U.S. Government (Ada Joint Program Office).

Integrated Manufacturing Systems (CIMS). Each of these concepts offers productivity improvements in its principal area. This paper explores these concepts to determine if combining them could lead to synergistic improvements in productivity.

Background

According to a recent issue of BusinessWeek, our factories were "engines of innovation" during the years immediately following World War II, but have been neglected since then. Now high technology, in the form of CIMS, is "reinventing the factory" and the factory is once again becoming the key component of international competition (Port, 1986:100-101).

This new manufacturing age is not yet mature. Port, using information from Dataquest, Inc., presents the investment figures for factory automation in Table 1-1. These investments show that a tremendous amount of money is being spent on elements which automate the factory and the production process but, so far, little has been invested in integrating all of these systems. The investments in Table 1-1 are investments in "islands of automation," with nothing to bridge the gaps between them. CIMS bridges the gaps in these islands of automation by integrating them (Port, 1986:102; Gunn, 1986:50-58).

Paying attention to the way factories really function instead of indiscriminately embracing the automated

TABLE 1-1

FACTORY AUTOMATION INVESTMENTS (Port, 1986:102)
($ in millions)

	1980	1985	1990*
Factory computers and software	$ 935	$ 2,861	$ 6,500
Materials handling systems	2,000	4,500	9,000
Machine tools and controls	3,000	4,800	7,000
Programmable controllers	50	550	3,000
Robots and sensors	68	664	2,800
Automated test equipment	800	2,000	4,000
Total spending	$6,853	$15,375	$32,300

*Estimates

equipment aspect or fancy computer technology is the major challenge facing the U.S. manufacturing industry today. What appears to be lacking in the factory automation movement is an integrating environment which can pull the many elements of automated equipment together into a whole which supports the total manufacturing system and not just isolated parts. "CIMS builds on the premise that management should work to optimize the whole business process rather than individual functions or elements" (Willis and Sullivan, 1984:28).

With CIMS, corporations will be able to master and control all of their production resources as well as link the manufacturing function with all of the other functions of the business. A framework for CIMS is shown in Figure 1-1 where computing technology is used to tie together elements of the production function such as computer aided design (CAD), robotics, computer aided manufacturing (CAM), automated materials handling, manufacturing planning and control systems, and group technology. This new age of manufacturing recognizes the importance of information, along with the traditional manufacturing resources of manpower, money, material and machines (Gunn, 1986:56-57; Peschke, 1985:1-2).

Manufacturing is not the only area experiencing a problem with "islands." In the 1970s, the Department of Defense (DoD) discovered that over 500 computer languages and dialects were in use in its weapon systems. These programming "islands" have contributed to the software crisis." Characteristics of the software crisis include:

1. The lack of support for the entire software life cycle;

2. The lack of trained programmers in industry and DoD;

3. Customized languages and applications that do not meet the applications requirements;

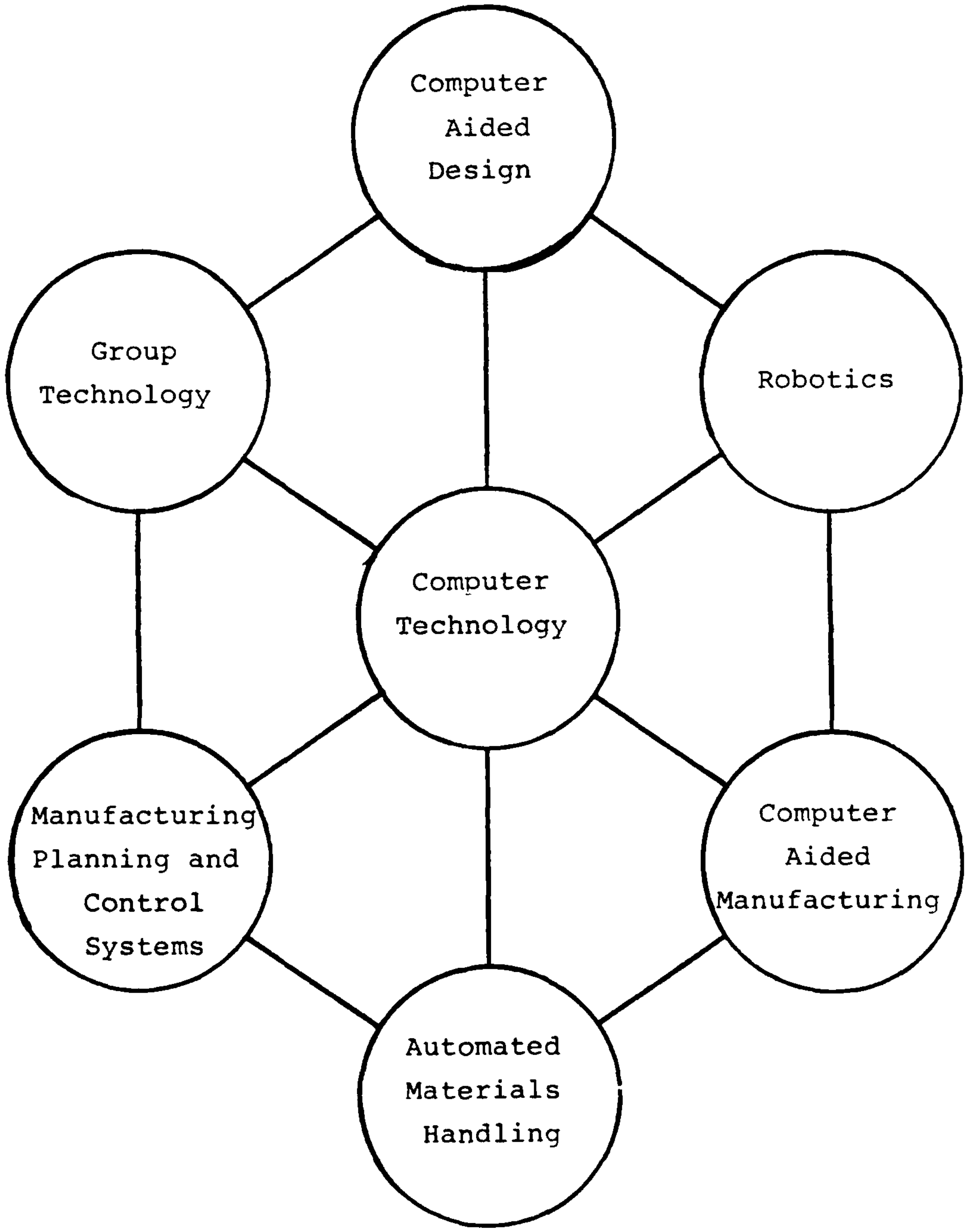

Computer Aided Design
Group Technology
Robotics
Computer Technology
Manufacturing Planning and Control Systems
Computer Aided Manufacturing
Automated Materials Handling

Figure 1-1. CIMS Framework (Gunn, 1986:56)

4. The failure to use good engineering techniques to develop software;

5. The lack of a central control point for language usage; and

6. Software that can seldom be reused from one system to another.

Since the DoD estimated in 1974 that its software budget would be over $3 billion annually (currently DoD spends $10 billion per year, with the Air Force spending 5 percent of its budget or $3 billion per year), a program was established to reduce system development costs, increase software reusability (portability), increase the portability of programmers, increase software reliability and maintainability, and support the management of change and complexity. This program is known today as the Ada Effort (Booch, 1983:12; Canan, 1986; Sammet, 1986:723; Wallace, 1986:2).

The Ada Effort is a three-pronged approach to achieve a standardized programming language, support modern software engineering methods and define and enforce a common software support environment. An environment is defined by this researcher as the system software providing integrated and interactive aid, support and/or control for accomplishing the current objective. The Ada Effort has been led by the DoD, but has the support and active participation of industry and academia both here and abroad. To date, the Ada programming language has been developed, standardized and become the mandated programming language for DoD and NATO mission-critical computer systems. An Ada Programming Support Environment (APSE) requirements document (STONEMAN) has been published, and industry and the DoD are working to develop an APSE meeting of all the STONEMAN requirements (Booch, 1983:11-21; DeLauer, 1983:11; Lawlis, 1986:1-2; Wallace, 1986:3-8).

Scope of the Thesis

This research looks closely at a project initiated to produce recommendations for CIMS standards within the Air Force. It compares this with the work done by the DoD to produce recommendations for APSE standards. The document produced for the CIMS project was published in 1977 and the APSE document was published in 1980. Although these are not new documents, neither has yet to be updated. This is probably because a true CIMS has yet to be implemented, APSE implementations are just beginning to appear, and evaluations must be completed before updates are meaningful.

One of the recommendations presented in the CIMS volume was that the then DOD-1 programming language (now Ada) be evaluated for computer aided manufacturing applications (Evans and others, 1977:159). The time has come. That evaluation is the focus of this thesis.

Research Question

The research question is defined as:

How similar are the APSE and CIMS concepts, and would the similarities justify applying APSE concepts for solving CIMS problems?

Research Objectives

The research objectives are as follows:

1. Compare the objectives of APSE and CIMS.

2. Compare the design guidelines of APSE and CIMS.

3. Compare the pictorial representations of APSE and CIMS.

4. Compare the component levels of APSE and CIMS by examining the database, interfaces and toolset of each.

5. Identify major differences between APSE and CIMS.

6. If deemed appropriate, recommend ways APSE concepts can be applied to CIMS in order to help solve the manufacturing crisis within the Air Force and U.S. industry.

Organization

The approach used in this research effort is to describe the development of both CIMS and the APSE, compare their concepts, and then determine how, if at all, APSE concepts can be applied to CIMS. A chapter is devoted to each step in this approach.

Chapter II defines Computer Integrated Manufacturing Systems and describes its functions, elements and components. The history of CIMS traces the use of computers for business and industrial applications from the 1950s to the present. Following the history of CIMS is a description of the Air Force CIMS program, which is known as the Integrated Computer Aided Manufacturing (ICAM) project. The history of ICAM and the ICAM standards proposal are then discussed in depth.

In Chapter III, the history of the Ada program is traced from the recognition of the need for a standard DoD language, through language development, test, validation and adoption. Following the Ada language discussion is the history of the APSE, including development and requirements generation. An appendix, detailing the development of the Ada and APSE requirements documents, is included to chronologically summarize the evolution of the Ada program within the Department of Defense.

Having now established a foundation for discussion, Chapter IV compares APSE and CIMS concepts. First, the comparison is at the macro level, addressing the objectives, design guidelines and pictorial representations of each. Following that, APSE and CIMS are compared at the component level by examining the database, interfaces and toolsets requirements of each.

Chapter V contains the conclusions based on the research objectives. It also makes recommendations for future work in this research area.

II. Computer Integrated Manufacturing Systems (CIMS)

Before comparing the APSE and CIMS concepts, a foundation for discussing the concepts themselves must be established. Although computers have been a part of business since the 1950s, the integration of computers into manufacturing has not yet matured to a working concept. In this chapter, CIMS is defined and described.

Introduction

> The key issue in manufacturing corporations in the Western world is how to respond to the problem of loss of competitive strength and industrial vitality. Western industry has lost market shares, millions of factory jobs, and its head start in equipment and process technology. (Skinner, 1985:219)

According to Dr. Wickham Skinner, the Harvard Business School business professor considered to be the prime mover in the "back to manufacturing" movement, the real paradox in the issue addressed above is that while professional management really started in the factory with Frederick Taylor, the factory now is the most poorly managed area of American business functions. He believes that long-term manufacturing strategy has been ignored by corporations and that is the cause of the problems facing manufacturing corporations. Strategic manufacturing planning would focus on properly using all the corporation's resources (men, machines, money, materials and information) to achieve the corporation's competitive goals (Skinner, 1985:214-239).

Computer Integrated Manufacturing Systems (CIMS) provides a generic manufacturing strategy that any corporation can adapt to its specific needs. Stan Manchuk, of McDonnell Douglas - Tulsa, views CIMS as a "pervasive manufacturing strategy" whose development represents "a conscious long-term implementation of strategic, operational and tactical plans to achieve high productivity and efficiency in plant operations" (Manchuk, 1984:34). Thus CIMS, because of its impact on manufacturing organizations and its potential to help resolve critical business problems, should be examined in detail.

The purpose of this chapter is to define and describe CIMS, trace its history and describe the most extensive investment in CIMS to date, that of the Air Force Integrated Computer Aided Manufacturing (ICAM) Project.

CIMS Defined

In January 1984, Industrial Engineering magazine began a twenty-four month examination of CIMS issues. Their purpose was to address the key CIMS issues, provide direction and information to manufacturing companies, and to "encourage a sensible and logical approach to the development of a CIMS" (Sadowski, 1984b:36).

Dr. Randall Sadowski, Purdue University professor and editor of the Industrial Engineering CIMS series, defines CIMS as a "truly integrated CAD/CAM system, encompassing all the activities from the planning and design of a product to its manufacture and shipping" (Sadowski, 1984b:36). He views CIMS as a concept that combines current (and future) manufacturing technologies with business management and control, in order to achieve the automated factory of the future. The CIMS goal is to optimize the business as a total system rather than to optimize only segments of the business, which would sub-optimize the total system (Sadowski, 1984b:35-36).

CIMS Functions. Based on Sadowski's CIMS definition, the manufacturing activities of interest range from initial product design through the production process to shipping the finished product. For this thesis, manufacturing is defined to be "the conversion of a design into a finished product," and production is defined to be "the actual making, the physical act of doing what is necessary to make the product" (ICAM Industrial Coalition definitions as quoted by Young and Mayer, 1984:29). The functions needed to accomplish CIMS manufacturing activities are:

1. Computer Aided Design and Computer Aided Manufacturing (CAD/CAM), which is the computerized design and drafting of both products and the manufacturing activities;

2. Production scheduling and control, which is the Master scheduling activity along with all the planning for material management and requirements, starting with product conception;

3. Process automation, which is computerized control of process, test and inspection actions, as well as computerized equipment performing certain processes;

4. Process control, which is the automated tracking of equipment activities, and the reporting of the need for operator intervention;

5. Material handling and storage, which is the computerized inventory handling system allowing storage and retrieval of finished and purchased parts based on the manufacturing system's requirements;

6. Distribution management, which is the order processing, warehousing, transportation, sales reporting and invoicing activities;

7. Maintenance scheduling and control, including the scheduling of preventive maintenance, the reporting of equ [illegible]ment needing unscheduled maintenance or repair actions and the record keeping of equipment usage rates; and

8. Finance and accounting, which includes the reporting of operating actions, forecasting future requirements and tracking and analyzing cost data.

These functions exist today in various stages of maturity (manual systems to computerized systems) and acceptance by industry. The CIMS goal is to tie them together in a logical system capable of optimizing the complete manufacturing system (Sadowski, 1984b:34-37,40; Willis and Sullivan, 1984:30-32; Young and Mayer, 1984:28-29).

CIMS Elements. Central to CIMS is information flow, control and management. While this may seem surprising for a manufacturing-oriented subject, the truth is that manufacturing today is "75% [an] information handling system and only 25% a materials transforming system" (Skinner, 1985:63). Based on this one can identify basically two flows in manufacturing: information and material. Both enter a factory, are processed and transformed in a sequence/sequences of operations, and leave the factory. In CIMS, one is concerned with the operations that occur to and with both information and materials.

CIMS Components. CIMS has two major categories of components, hardware and software. These categories are part of the elements of CIMS (information and materials can be/are processed by hardware and software), as well as being a part of the CIMS functions (since most of the functions are/can be automated). Hardware runs the components, elements and functions; software controls them (Sadowski, 1984b:36; Willis and Sullivan, 1984:29-32; Young and Mayer, 1984:29).

History of CIMS (Sadowski, 1984a:34-42)

Computers have been involved in business and industrial applications for over thirty years. Only recently have computers found their way out of accounting-type activities and into the operational elements.

1950s. Industrial computer applications began in the 1950s, primarily in finance and administrative areas. These applications were repetitive types of calculations such as those required for payrolls and general ledgers. The well defined, sequential processing nature of these calculations fit the state of computer technology at the time.

1960s. As computer capabilities increased and costs decreased, more systems were developed with increased processing speeds, more input/output devices, more non-scientific programming languages and the systems were no longer limited to sequential processing. During the late 1950s and early 1960s, the punch card was used in administrative and financial applications as well as in simple manufacturing applications. The manufacturing applications included simple inventory accounting, parts listings, labor reporting and job tracking. Additionally, FORTRAN came into use during this time period, for the engineering aspects of manufacturing. Unfortunately, as computer hardware was becoming more advanced, available and affordable, software was virtually nonexistent for many manufacturing applications.

In the late 1960s, minicomputers emerged. Although applications software did not exist for the minis, they gained wide acceptance for engineering and statistical manufacturing (process monitoring and control) applications. Many industrial and manufacturing companies began writing their own software, primarily for inventory accounting, operations scheduling and requirements planning. The benefits of in-house software development included customized applications programs. However, this customized software was the beginning of "islands of automation," where each application stood alone. Although optimized for a specific application, these various applications programs could not interact with or support each other, much less optimize the manufacturing process as a total system.

<u>1970s</u>. In the early 1970s, IBM published eight volumes on a systems concept for a production inventory control system. This communications-oriented production information and control system (COPICS) presented a view of the flow of data in a manufacturing organization from sales forecasting to shop floor control. According to Sadowski, "This concept (COPICS) provided the outline and defined the boundaries which allowed the intelligent development of

software for the control of manufacturing systems" (Sadowski, 1984a:36). The software systems that resulted from the COPTICS concept were mainly in the requirements planning area and were widely adopted. Only a limited number of production and inventory control systems were produced. This resulted in another iteration of in-house applications programs for production and inventory control which were not integrated with the requirements planning programs. Still greater numbers of islands of automation were created. Engineering functions, such as CAD and CAM, and factory automation were not included in commercial or in-house applications because of a lack of software and system support.

1980s. In the late 1970s and early 1980s, computerized manufacturing systems were a target of not only hardware and software producers, but also the consulting community. The main drivers for this targeting were the technological advances in both hardware and software, the recognition of the need for such systems for a company's survival, the increased affordability of hardware and software and the recognition of this area as one that had been totally ignored. By 1981, 283 different computerized manufacturing systems were available, most under $50,000 (but that figure does not include the cost of hardware or the modifications to a company's existing software so the new applications could access the company's files) (Sadowski, 1984a:40). Unfortunately, even these systems were not generic, but rather were customized for specific companies' applications, thus continuing the existence of the islands of automation.

Many of these manufacturing systems applications failed. Reasons cited for the failures include:

1. Not understanding what a manufacturing system was;

2. Not understanding the need for accurate records and timely feedback;

3. Placing the responsibility for the systems on computer technicians, rather than manufacturing managers;

4. Numerous data sources containing inconsistent and contradictory information; and

5. No disciplined relations existing between the manufacturing systems (islands of automation) and the organization's functional elements such as marketing, engineering and physical distribution.

Analyzing the failures made it clear that a viable manufacturing system must be integrated, contain an overall database management system, data communications, and data communications management system, and have the capability of working on-line in an interactive mode (in direct communication with the computer so the information in the database is always current). The ideal computer integrated manufacturing system would be company-wide and have the capabi'ity of successfully communicating with and accessing a company-wide database.

Past experiences have defined the concept of what a computer integrated manufacturing system should be, but so far neither the required total systems architecture nor the required integration software is available.

The Integrated Computer Aided Manufacturing (ICAM) Project (Evans and others, 1977)

The Air Force, more than any other organization, has taken the lead in CIMS development through its ICAM project.

History of the ICAM Project. In 1973, the Air Force, recognizing the problems in attaining true computer integrated manufacturing systems, developed a plan to attempt the identification and grouping of the major manufacturing functions. After these functions were identified and grouped, the Air Force planned an organized approach to integrating the functions. The response from industry at the conclusion of the identification effort was positive, however there was disagreement publicly and privately as to the direction any follow-on effort should take.

Another study was done in 1975, at the direction of Deputy Secretary of Defense W. P. Clemens, to identify cost savings opportunities in the defense industry through the use of computers. The study concluded that the computerized integration of manufacturing subsystems was the key to cost savings.

A result of the 1975 study was a new long-term Air Force initiative with a goal of totally integratable computer aided manufacturing. Part of this initiative was to evaluate standards (existing and potential) which would be useful for the ICAM project. The standards evaluation project was divided into five tasks:

1. Identifying the current applicable standards;
2. Analyzing the existing standards;
3. Assessing actual industry usage of standards;
4. Recommending optimal standards for developing the computer integrated manufacturing system; and
5. Identifying the standards organizations and recommending the Air Force's role in standards activities (Evans and others, 1977:1).

In June 1977, Evans and others of the National Bureau of Standards (NBS) completed the final technical report on CIMS standards evaluation.

National Bureau of Standards (NBS) Report. According to the NBS report, the goal of a true CIMS can be achieved through the use of formal standards, technical guidelines and ICAM program policy. Using these techniques will allow the definition of the interfaces between the functional manufacturing modules (some of which already exist) and their environment. Integration of the entire system can only occur when the interfaces match and the functional modules can fit together.

Evans and others identified four objectives which they view as essential to the ICAM program's success: software portability, module integration, distributed data processing and exchangeable manufacturing data (Evans and others, 1977:4).

Software Portability. Software portability from a CIMS standpoint is achieved through developing all software in standardized high level languages, following specified programming practices and standardizing documentation practices (Evans and others, 1977:5). These will provide portability in that the modules can be easily moved between machines and users, and the modules can be more easily integrated into a total system without extensive modification. Five languages were reviewed for CIMS adequacy (FORTRAN, COBOL, MUMPS, PL/1 and BASIC). None of the languages met all of the requirements; however, FORTRAN and COBOL were deemed to "suffice for near term applications" (Evans and others, 1977:31). The NBS report recommended the Air Force investigate the use of DOD-1 (which later became Ada), which, although in development at the time, appeared to meet the ICAM requirements.

Integratable Modules. In order to develop a true general CIMS, a capability must exist to use software developed by many different contractors. For example, this software could include requirements planning from one contractor, inventory control from a second, process planning from a third and so on. Each of these application programs must be partitioned into separate modules with well defined interfaces so they can be integrated into a total system.

Figure 2-1 describes a proposed ICAM system. The major functions include ICAM application programs (such as process planning, inventory control and tool selection), a database management system (DBMS), ICAM system software (such as text editors, debuggers and libraries) and host system software (including assemblers, compilers, linkers and file managers). The proposed system also contains three types of interfaces: between the applications programs and the host system, between application programs (through the DBMS), and system software interfaces.

Application Program Interfaces. Application program interfaces are viewed as a key to CIMS software portability. The ICAM standards report assumes a standardized and validated programming language (which implies a standardized run time support monitor) and therefore programs and programming languages can be independent of the host system. According to Figure 2-1, the

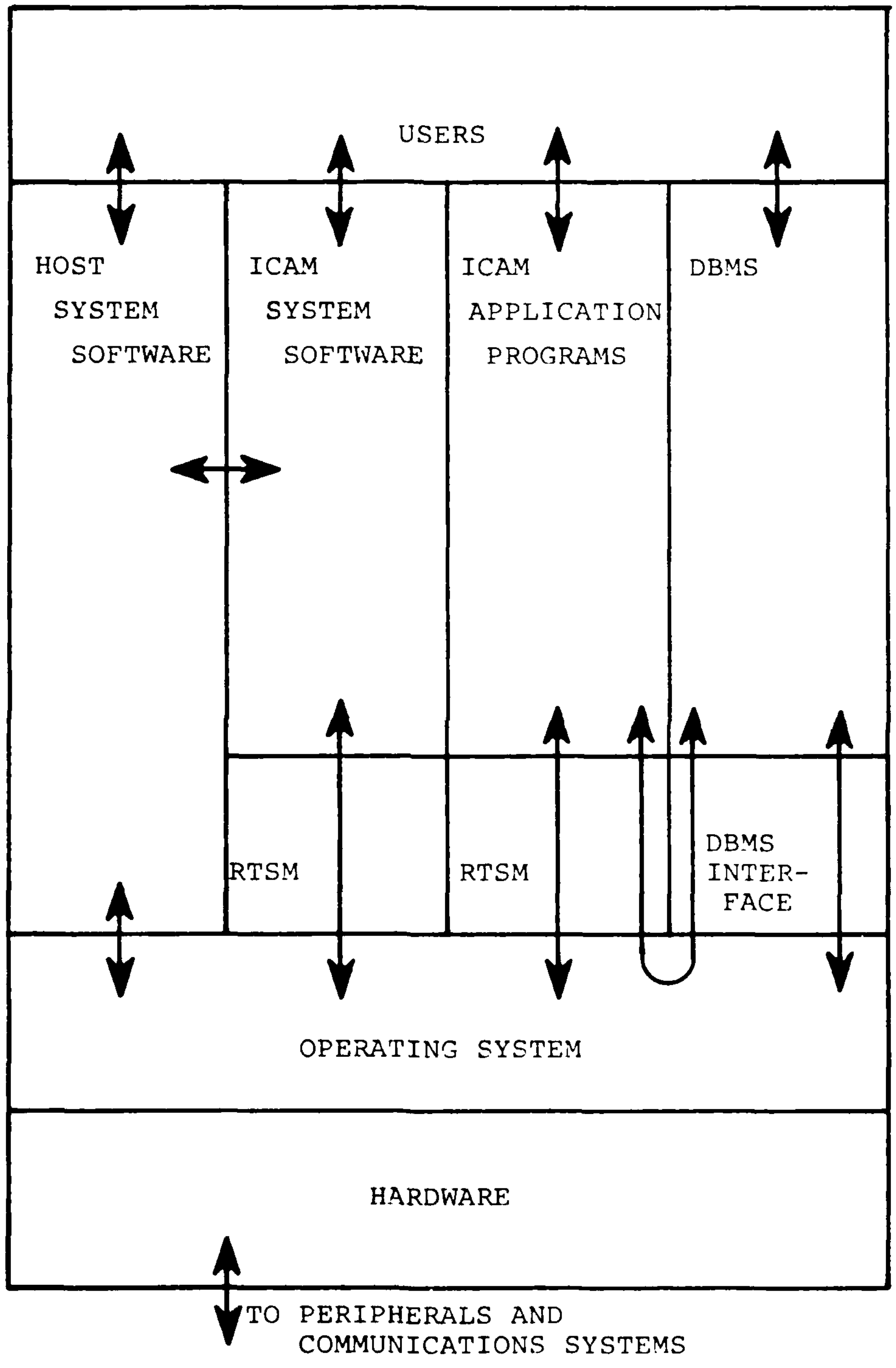


Figure 2-1. ICAM Structure (Evans and others, 1977:32)

applications programs will interface with only the user, the operating system and the DBMS (through the operating system and the DBMS interface). Application programs will NOT interface with either the ICAM or host system software (Evans and others, 1977:32-33).

Database Interfaces. Database interfaces are critical for integrating application program modules. The ICAM standards report proposes a single (but perhaps distributed) database as the sole interface between application programs. The database interfaces with the user, the operating system and, through the DBMS interface, with the application programs (Evans and others, 1977:32-34).

System Software Interfaces. The NBS report recognizes that some system software will be host computer dependent, but warns that overuse of the dependent software will negate the CIMS software portability goal. To overcome this potential portability problem, the report encourages communication between the ICAM system software and the host system software. As Figure 2-1 shows, the host system software interfaces with the user, the operating system and the ICAM system software. Program development tools reside partly in the ICAM system software and partly in the host system software. Interfaces with the operating system include run time support monitors

(RTSMs) for the ICAM system software and ICAM applications programs, as well as interfaces with the host system software, and the interface between operating system and the database management system (DBMS) (through the DBMS interface)(Evans and others, 1977:32-35).

Distributed Data Processing. Evans and others view distributed processing as having tremendous potential for CIMS. One proposed CIMS scenario is a series of mini and micro computers, running dedicated application programs, all integrated through a network for a total computer aided manufacturing system.

This array concept is recommended for CIMS because it would allow for an evolving approach to CIMS in existing factories by networking the factory's current islands of automation. Additionally, it would be a cost effective system, since minis and micros are becoming more affordable.

Exchangeable Manufacturing Data. As noted earlier in this chapter, current manufacturing is only 25 percent material flow and transformation with the balance a flow and transformation of information. Manufacturing data includes, for example, engineering drawings, performance and design specifications and work instructions. Most of this data is currently computerized and stored in word processors and CAD/CAM systems.

With the current requirements for competition in contracting for weapon systems, the DoD is buying more and more data and the rights to use that data for follow-on acquisitions. The capability for exchangeable data will reduce the DoD costs for data as well as encourage defense contractors to be more efficient by using the company-wide database for storing the manufacturing data.

Summary

The implemented CIMS factory will bridge the islands of automation in existence today with fully automated and integrated design activities feeding fabrication activities, which in turn will feed distribution activities. Host computers will oversee and control all associated elements. This CIMS implementation is perhaps the best hope U.S. industries will have for competing against the products of foreign manufacturing firms (Groover and Wiginton, 1986:75).

Computer integrated manufacturing systems can provide a generic manufacturing strategy that any company can adapt for its specific needs. Until a true CIMS exists, however, the manufacturing system cannot be optimized, and it will continue to be the most poorly managed area of our business systems.

To date, the most extensive effort to produce a CIMS has been the Air Force's ICAM project. The ICAM initiative is a planned, controlled, organized approach to achieving a truly integrated computer aided manufacturing system. In order to make CIMS a reality, a common support environment will be required to control and direct each subsystem. No CIMS efforts to date have yielded this environment.

III. Ada

The Ada programming language and its associated support environment appear to conceptually offer the capabilities to produce a common support environment for CIMS. In this chapter, Ada and the Ada Programming Support Environment are described so that Chapter IV can compare the similarities of CIMS and Ada support environments.

Introduction

> Programming languages are neither the cause of nor the solution to software problems, but because of the central role they plan in all software activity, they can either aggravate existing problems or simplify their solutions. (Fisher, 1976:8)

Acknowledging Ada as the DoD standard programming language is merely the tip of the iceberg of the "Ada system," for Ada is so much more than just another programming language. In this chapter, the discussion begins with recognition of the need for a standard DoD programming language and traces the history of the development of DOD-1, later renamed Ada. Paralleling the Ada development was the development of a programming environment to support the Ada language. This environment, called the Ada Programming Support Environment (APSE), and its history are also described in this chapter.

Recognition of the Need for a DoD Standard Language

In the early 1970s, there were at least 500 computer programming languages used by the DoD and its contractors (estimates range from 500 to 1500 different languages and dialects). The software programs generated by these languages cost over $3 billion in 1973 (approximately 46 percent of the total DoD computer costs), with 56 percent of the software costs in embedded computer systems. According to Sammet (Sammet, 1986:722), an embedded computer system is "a computer that is part of a larger--probably noncomputer--system, which need not necessarily be a weapon" (Booch, 1983:11-13; Sammet, 1986:722-723).

Originally, the high number of languages and their dialects was justified by the belief that each must be customized in order to be optimal for a specific system requirement. However, there were no control/coordination points nor standards offices in any of the services. In 1975, after the Office of the Director of Defense Research and Engineering (DDR&E) review of the services' requests for more money to customize more of the languages and dialects, the High Order Language Working Group (HOLWG) was established. The HOLWG, comprised of military and defense agencies' representatives, was tasked to oversee the investigation into language standardization. The HOLWG's goal was to "standardize on a few well-specified, well-maintained languages to ensure the availability of high-quality, production-engineered compilers for computers used by the DoD" (Carlson, 1981:7). In addition to establishing the HOLWG, DDR&E emphasized the language standardization program's priority by stopping all language research and development funds (Booch, 1983:13-14; Carlson, 1981:7; Sammet, 1986:723).

The HOLWG's first task was to determine what the DoD language requirements actually were. Four months after the HOLWG was established, a language requirements document, STRAWMAN, was distributed to military and other federal agencies, industry, academia and the European computing community. Responses to STRAWMAN were incorporated into WOODENMAN and in January 1976, the next revision, TINMAN, was distributed. The TINMAN requirements included support for modern software engineering methodologies, including abstraction, information hiding, modularity, localization, uniformity, completeness, and confirmability. Additional requirements were for real-time control and exception handling. Above all, the DoD was interested in life cycle support, reliability, portability, reusability and suitability for use in its embedded computer systems. TINMAN was viewed as a complete language requirements document and was given international distribution (Booch, 1983:14-15; Carlson, 1981:7-8; Sammet, 1986:723).

Another HOLWG task in 1976 was to establish policies to control and coordinate the existing DoD computer languages. DoD Directive 5000.29, Management of Computer Resources in Major Defense Systems, required only DoD-approved languages to be used in major systems unless it could be proven cost-effective to use a non-approved language for a specific application. By enforcing DoD-approved languages, the HOLWG was able to establish focal/control points for those languages. DoD Directive 5000.31, Interim List of DoD Approved High Order Languages (HOL), released in November 1976, allowed seven languages (Table 3-1) to be used while the HOLWG was working to finalize the language standardization program (Booch, 1983:14-15; Carlson, 1981:7-8; Sammet, 1986:723).

The TINMAN review in 1976 was done in two ways. The first method was to review twenty-three existing languages (Table 3-2) against the TINMAN requirements. None of the existing languages was able to meet more than 75 percent of the TINMAN requirements; and all were categorized as inappropriate. The language review concluded:

1. None of the existing languages met the requirements for a common DoD embedded computer systems language.

2. There should be only one common language.

3. It was feasible to create a new language meeting the TINMAN requirements.

TABLE 3-1

DOD DIRECTIVE 5000.31 APPROVED LANGUAGES
(Booch, 1983:15)

Dept/Agency	Language
DoD	FORTRAN COBOL
Army	TACPOL
Navy	CMS-2 SPL/1
Air Force	JOVIAL J3 JOVIAL J73

TABLE 3-2

LANGUAGES EVALUATED AGAINST TINMAN
(Sammet, 1986:723)

Used for DoD and/or Embedded Computers			
CMS-2	CS-4	HAL/S	JOVIAL3B
JOVIAL J73	SPL/1	TACPOL	
Used for Process Control and/or Embedded Computers (mainly European)			
CORAL 66	LIS	LTR	PEARL
PDL2	RTL/2		
Research-Oriented Languages			
ECL	EL-1	EUCLID	MORAL
Widely Used and/or General Languages			
ALGOL60	ALGOL68	COBOL	FORTRAN
Pascal	PL/1	SIMULA67	

4. The new language should be based on an existing language.

The HOLWG finally determined that Pascal, ALGOL68 and PL/1 were appropriate as base languages.

The second aspect of the TINMAN evaluation was to consolidate comments to the evaluation of the requirements document itself, which resulted in a further revision, IRONMAN, during 1977 (Booch, 1983:15-16; Carlson, 1981:8; Sammet, 1986:723).

Thus, DoD, after discovering the high cost of computer software, the dominance of embedded computer systems and the uncontrolled proliferation of computer languages and dialects, determined that one new, standard language, common to all DoD systems should be developed. "It then set out with a refreshingly new approach to getting such a language" (Lawlis, 1982:2).

Development of DOD-1

Based on the results of the TINMAN reviews, the HOLWG began development of the common language, DOD-1, with program management under the aegis of the Defense Advanced Research Projects Agency (DARPA). Phase I, the language design effort, was based on the requirements defined in IRONMAN. Rather than hire a single contractor to develop DOD-1, DARPA held an international design competition to reach the world's best language designers. Of the seventeen teams proposing to design the new language, four were chosen for the six-month Phase I effort. (Interestingly, all four based their design on Pascal.) At the end of Phase I, eighty review teams around the world spent one year evaluating the designs in typical embedded computer systems (Booch, 1983:16-17; Carlson, 1981:8; Sammet, 1986:724).

In 1978, based on the Phase I design evaluations, the HOLWG issued STEELMAN, the final language requirements document. STEELMAN incorporated the evaluation comments and corrected deficiencies discovered during Phase I. Two of the four Phase I design teams were selected to continue language development, according to the requirements in STEELMAN, during Phase II. In 1979, both final language designs were sent out for public comment, and one version (from the French design team) was finally selected as "Preliminary Ada" (Booch, 1983:17-19; Carlson, 1981:8; Sammet, 1986:724).

The DoD had searched for a new name for the new language. It hoped that a non-military name would increase the general appeal of the language. Augusta Ada Byron, Countess of Lovelace, and the daughter of the poet, Lord Byron, is viewed as the world's first computer programmer. Her mathematical work with Charles Babbage led her to suggest a method for programming Babbage's difference and analytic engines. DOD-1 became Ada in May 1979 (Booch, 1983:18-19).

Ada Test and Validation

Ada's development did not end with the selection of the final language design; the selection marked the beginning of Phase III, Ada Formal Test and Evaluation. The preliminary language reference manual was published in a highly respected computer journal of the Association for Computing Machinery (ACM), the SIGPLAN Notices of the Special Interest Group for Programming Languages (SIGPLAN). It was also sent to over 2000 experts for comment. Additionally, the HOLWG contracted for development of a compiler validation facility which would be working before any production Ada compilers were completed (Booch, 1983:19; Sammet, 1986:725).

The rationale behind the compiler validation facility was to employ extremely tight configuration control on the Ada language. The HOLWG wanted no dialects, subsets or supersets of Ada. It set out to prevent them through the compiler validation activities, as well as applying for American National Standards Institute (ANSI) and International Standards Organization (ISO) standardization recognition. Additionally, the DoD applied for, and received, a trademark for Ada and also made the language reference manual a military standard (MIL-STD-1815a). Consequently, no changes in Ada are possible without the DoD's approval (Booch, 1983:19-20; Sammet, 1986:724-725).

The Ada Joint Program Office (AJPO), established in 1980, as the follow-on to the now defunct HOLWG, is tasked with reviewing and overseeing Ada policy within the DoD. The Under Secretary for Defense made Ada mandatory for mission-critical system development and upgrades within DoD as of 1 July 1984 (DeLauer, 1983:1; Hicks, 1985:1). The AJPO also promotes the acceptance, standardization and recognition of Ada, not only within the DoD, but also within industry and academia, both in the U.S. and abroad. AJPO success with this promotion endeavor can be measured by the adoption of Ada as a NATO standard effective 1 January 1986; the mandate for all United Kingdom Ministry of Defense real time systems to be written in Ada effective July 1987; and the funding of fifteen Ada software projects by the Commission of the European Communities (CEC) between 1984 and 1986 (Lawlis, 1986:1-2; Lieblein, 1986: 736-737; Sammet, 1986:727).

Ada Programming Support Environment (APSE) Development

In the 1977-1978 timeframe, during Phase II (continuing design effort) of the Ada language, a parallel effort was started by the HOLWG to examine the programming environment for Ada. Early on the HOLWG realized

> . . . a programming language alone was not sufficient to ensure the desired improvements in software

development, but rather the language needed to be coupled with a quality support system. (Booch, 1983: 18)

The environment, along with providing the quality support system, would also assure life cycle support for software projects; DoD software projects are extremely long-lived (Booch, 1983:18; Freedman, 1985:3).

The concept of programming environments did not begin with Ada. In general, a good programming environment can be viewed as a set of integrated, interactive tools and methodologies to assist in developing software programs. Existing programming environments contain interactive tools such as compilers, linkers, loaders, text editors, support libraries and path testers, but they are not necessarily integrated. Reasons for the small number of tools currently available include the large number of computer programming languages available (a lot of basic tools exist for many languages, but few languages have extensive tools) and the machine dependence of the languages (this ties into the number of languages as well as to the number of different machines which support the different languages). Booth believes "the degree to which a complete and coordinated environment is available will affect the development of software far more than any individual programming language" (Booch, 1983:352). Obviously, a complete and coordinated programming environment will increase programming productivity (Barstow and others, 1984:xi; Booch, 1983: 352-353; Freedman, 1985:3-7).

The HOLWG circulated SANDMAN, which focused on the managerial and technical issues of programming environments, in early 1978. In June 1978, approximately sixty people from the military, industry and academia attended a joint HOLWG-University of California at Irvine conference to discuss SANDMAN, discuss the reviewers' comments to it, and also to review the revised document which was named PEBBLEMAN. By December 1978, a revised PEBBLEMAN was publicly distributed (Booch, 1983:18; Freedman, 1985:3).

STONEMAN, the APSE requirements document based on feedback from PEBBLEMAN, was first issued in November 1979, with the final version published in February 1980. STONEMAN defines an APSE in terms of the needs associated with Ada programming (Booch, 1983:18; Freedman, 1985:3).

APSE Requirements

The STONEMAN document primarily specifies APSE requirements, although it also provides assessment and evaluation criteria for APSE designs and offers APSE design and implementation guidance. The purpose of an APSE, according to STONEMAN is

> . . . to support the development and maintenance of Ada applications software throughout its life cycle, with particular emphasis on software for embedded computer systems. (DoD, 1980:1)

Given that Ada is the common language to be used for all mission-critical DoD computer systems, and that the STONEMAN APSE requirements will mandate complete, coordinated APSEs, the potential benefits of an APSE include reduced development costs for tools and compilers as well as improved portability for programs, tools and programmers (Booch, 1983:352-353; Freedman, 1985:3-8; DoD, 1983:1).

A DoD goal for an APSE is that both programs and tools be portable within an APSE. STONEMAN addresses an approach to portability by stating requirements for the layers (levels) of an APSE (Booch, 1983:352-353; Freedman, 1985:3-8; DoD, 1983:1).

APSE Structure. Figure 3-1 represents the layered structure of an APSE. The innermost level is the host computer hardware and operating system. On top of the host computer system is the first level of an APSE: the "kernal" or Kernal Ada Programming Support Environment (KAPSE).

KAPSE. The KAPSE essentially controls the resources of the host computer for the APSE. It contains the database, run time support functions and the communications functions the APSE needs. The purpose of the KAPSE is "to allow portable tools to be produced and to support a basic machine-independent user interface to an APSE" (DoD, 1980:10). The KAPSE is essentially hidden from the user, but it allows access control from the tools in the

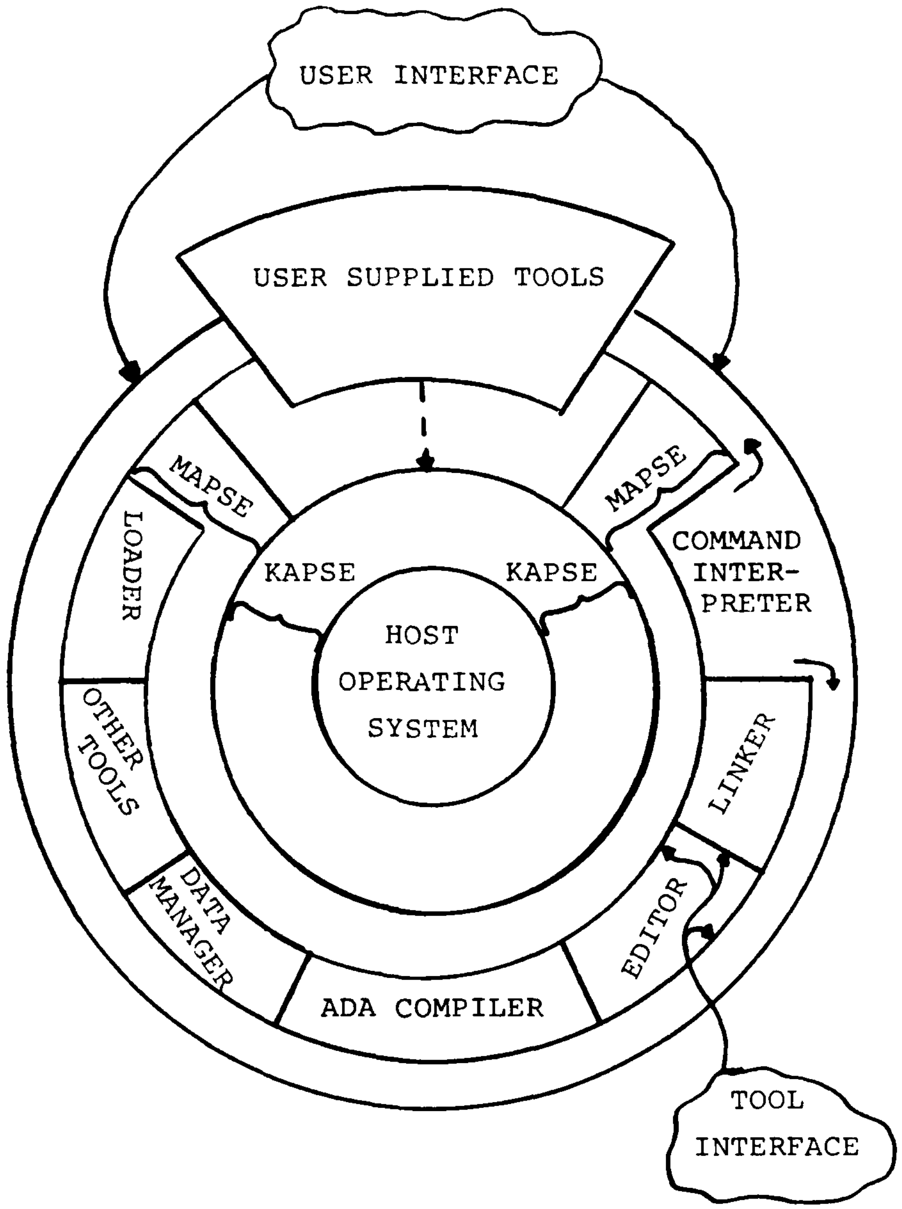


Figure 3-1. APSE Structure (Babich and others, 1981:28)

next outer layer of the APSE, through the KAPSE, via the standard machine independent interface to program libraries and peripherals (Booch, 1983:354; Freedman, 1984:31; DoD, 1983:1,10).

MAPSE. Above the KAPSE is the Minimal Ada program Support Environment (MAPSE). As its name implies, the MAPSE contains the minimal set of tools needed by the APSE to perform programming support functions over the life cycle of a program or project. The MAPSE tools control the host computer's resources through the standard KAPSE interface, "just as a general controls his army through his lieutenants" (Freedman, 1984:31,33). The minimal tools prescribed by STONEMAN for a MAPSE will be described later in this chapter (Booch, 1983:355-356; Freedman, 1984:31-34; DoD, 1980:1,10-11,36-41).

APSE. The top level of the APSE contains tools that go beyond the scope of the MAPSE. STONEMAN does not prescribe tools for the APSE level, but describes some of the tools' capabilities including object creation, modification, analysis, transformation, display, execution, and maintenance. Examples of the types of tools one might find at the APSE level are those that impose modern software engineering methodologies as well as those that may meet the unique requirements of specific projects. The STONEMAN description of APSE level tools will appear in a later section of this chapter (Booch, 1983:356; Freedman, 1985:33; DoD, 1980:1-2,11,24-26,42).

APSE Portability Issues. A major DoD goal throughout the development of Ada and APSEs has been portability of both software programmers and software programs. Portability is impossible if either of the two are host computer or host operating system dependent. A means of achieving portability is through a common standardized KAPSE interface with the host computer. To address this issue, the AJPO established the KAPSE Interface Team (KIT) and the KIT from Industry and Academia (KITIA) in 1982. The teams' goal was to develop a common interface at the point where the KAPSE relates to the host operating system (Sammet, 1986:728).

By 1983, the teams had developed a draft Common APSE Interface Set (CAIS); the current version is a proposed military standard. The CAIS consists of a set of standard KAPSE interfaces written in Ada. Included in the CAIS are six functional support areas:

1. Node support (for managing files, directories, processes and devices);

2. Structure support (for the groupings of directories, partitions and configurations);

3. File input-output support;

4. Process support (which includes process control, definition, communication, analysis and interrupts);

5. Device support (for interactive and non-interactive devices);

6. Utilities support (for text, list and string manipulation).

Compliance with the CAIS will provide transportable and interoperable tools and programs (Freedman, 1985:214-220).

APSE Components. There are three basic components to an APSE, the database, the user and system interfaces and the toolset. Information is held in the database for each project throughout the project's life cycle. The interfaces allow integration and coordination of all the APSE components with the user and the host computer system. Software tools for developing, maintaining and controlling the Ada programs are contained in the toolset (DoD, 1980:1)

Database. The database is key to an APSE; it contains all the information for each project's entire life cycle. In an APSE database, each separately identifiable collection of data (an object such as a program file) has the capability of being related to one or more other objects. Additionally, each new iteration of an object is kept in the database in order to provide an historical audit trail throughout the life cycle of the project, as well as to provide configuration control on the objects themselves. Since several different projects can exist in the database, the APSE can group objects under their specified project and control user access to only designated projects. The APSE can also generate management level reports on the database itself (DoD, 1980:16-17).

Interfaces. The second APSE component, interfaces, can be viewed from three perspectives: the interfaces between the user and the APSE, the interfaces within the APSE, and interfaces between the APSE and the host computer system.

According to STONEMAN, the user may initially connect to the APSE through the host computer operating system from a variety of terminal devices, ranging from the most primitive teletype-like terminal to the most sophisticated terminals imaginable. The user then communicates with the APSE using a command interpreter. This user interface can be either primitive or sophisticated, depending on the capabilities (sophistication) of the user's terminal (DoD, 1980:21).

The interfaces within the APSE (intra-APSE communications) have access to the database so that the tools involved may use the objects in the database. Additionally, one tool may invoke another tool. For example, while using a text editor tool, the user may wish to invoke a compiler.

Interfaces between the APSE and the host computer operating system (the commands used by software tools to access the database and other system functions) are standardized via the CAIS in the KAPSE. The CAIS promotes tool portability from one APSE to another. Because of CAIS standardized interfaces, the only portion of an APSE which must be customized for different computer systems is the system dependent implementation of how the CAIS commands are translated into host system commands.

Toolset. The final APSE component, the toolset, contains tools for development, application program management and project management as well as all the project-unique tools that may be developed. The tools are coordinated and provide life cycle support for a project from conception (original requirements specification) through deployment, modification and project phaseout. STONEMAN envisions a comprehensive APSE as providing tools for requirements specification, overall system design, program design, program verification and project management (DoD, 1980:8).

The minimal tools for an APSE (located at the MAPSE level) include a text editor, pretty printer (formats text), Ada compiler, linkers, loaders, set-use static analyzers (a debugging tool to track the changing values for variables), control flow analyzers (debugging tools to trace controls in a program), dynamic interface routines (to allow access to all available system terminals), file administrators (to allow file transfer and comparison), command interpreters (to allow the tools to be used as well as the user's commands to be understood), and a configuration manager (to aid in controlling the history and persistence of items in the project database) (DoD, 1980: 37-38).

The APSE toolset will not be closed. New tools can be built with existing tools and added to the APSE. The tools in the APSE will always be able to communicate with each other. The communication will be such that project history and configuration control can be continuously monitored and maintained.

Additional tools (at the APSE level) as described in STONEMAN can provide supplemental capabilities such as syntax-directed program editing (for automatically correct Ada syntax), documentation assistance (with the capability of documenting Ada programs while they are being written as well as implementing military documentation standards), project control (a tool for tracking progress against review dates and budgets), configuration control (to assess the impact of any program changes/modifications as well as to check that all relevant parts of a system are modified), and project verification (to automatically verify system programs). They can also provide performance measurements (to determine program and programmer efficiencies), fault reporting (to handle error reports during program maintenance), requirements specifications (which could trace the requirements from the conception of a project through the design and coding stages), and tools to aid system and program design. APSE level tools can also impose specific software design methodologies and provide project unique tools (DoD, 1980:42-44).

APSE Evaluation and Validation (E & V)

In 1983 the AJPO organized an APSE E & V Task to establish the capability for evaluating and validating APSEs and APSE-component. Validation is the capability for assessing APSE or APSE-component conformance to established standards (such as CAIS). Evaluation is the capability for assessing the qualities of the attributes of an APSE or APSE-component for which no standards have yet been established. These qualities can include reusability, efficiency, and maintainability. The main benefits of the E & V Task will be

> . . . to encourage the development of quality APSEs, to promote interoperability and transportability and to provide users and developers with a uniform and comprehensive means for assessing and selecting APSEs suitable for their specific applications and methodologies. (DoD, 1985:4)

The goals of the E & V effort are to establish a set of E & V requirements and a minimal set of E & V tools to assess APSEs, along with a feedback system for comments, discussion and wide distribution of E & V team actions. The long-term E & V objective is to establish an interactive database of E & V results available to all potential APSE users and designers (DoD, 1985:4-5).

Summary

Once the need for a common language was recognized, it was developed by the best minds available in a controlled, disciplined fashion. The control continued from requirements specification through development, test and deployment. When the DoD recognized that the language itself was only a part of the larger system, it proceeded to develop the APSE and CAIS, again in a controlled, disciplined manner.

As this chapter has shown, Ada is more than just another computer language. The Ada system consists of the language itself and the APSE, which includes support for the entire software lifecycle. With the development of the Ada system, the DoD and industry and academia, both here and abroad, are involved in changing the way hardware and software is viewed. Software in the Ada system must no longer be hardware dependent; it can much more easily be standardized, portable, interoperable, reusable, reliable, maintainable, controlled, efficient, modifiable, suitable for embedded computer systems, supported over its entire life cycle, robust, integrated and user friendly.

IV. Comparison of APSE and CIMS

In this author's opinion, the overriding benefit of the Ada system's development, even over the cost efficiencies that should result from increased productivity from the use of Ada, is the concept of the APSE itself, and the generalizability and applicability of APSE concepts for other than strictly computer programming language support environments. For example, a 1985 Air Force Institute of Technology (AFIT) thesis (Adams, 1985:121-122) concluded that APSE concepts were directly applicable to a display environment supporting the interactive generation of alphanumerics and symbology for aircraft simulation. In this chapter, another parallel with APSE concepts will be established; computer integrated manufacturing systems concepts have much in common with Ada environment theory.

Introduction

The purpose of this chapter is to compare the requirements for an Ada Programming Support Environment (APSE) as embodied in the STONEMAN document (February 1980) with the June 1977 National Bureau of Standards (NBS) technical report on standards for computer aided manufacturing. While these two documents may appear dissimilar according to their titles, the concepts they address really are quite similar. Recall the Chapter III definition of an embedded computer system: "a computer that is part of a larger--probably noncomputer--system, which need not necessarily be a weapon" (Sammet, 1986:722). Computer integrated manufacturing systems (CIMS), also referred to by the National Bureau of Standards (NBS) report (Evans and others, 1977) as integrated computer aided manufacturing (ICAM) is, by definition, made up of embedded computer systems as well as other functional systems. Additionally, recall the purpose of an APSE is to

> . . . support the development and maintenance of Ada applications software throughout its life cycle with particular emphasis on software for embedded computer applications. (DoD, 1980:1)

Based on these underlying definitions that apply to both APSEs and CIMS, it becomes not only appropriate to compare these two documents, but mandatory. If the two documents are parallel in scope and objective, it makes sense for government and industry to combine their concepts for multi-purpose applications.

The document comparisons will first be done at the macro level, comparing scopes, objectives and goals of APSEs and CIMS. Following this top level assessment, the analysis will be broken down into three areas, aligned with the three components of an APSE: database, interfaces, and tools.

APSE versus CIMS

Objectives. According to STONEMAN, the objectives of an APSE are to provide cost-effective support throughout the entire project life cycle. The support provided to the project team includes all functional areas of a software project, "particularly in the embedded computer system field" (DoD, 1980:8).

CIMS objectives, according to the ICAM standards report, include software portability, integratable modules, distributed data processing and manufacturing data that is exchangeable (Evans and others, 1977:4).

Design Guidelines. Admittedly the two objectives appear, at first glance, to have little similarity. However, further research into the objectives provides evidence of common ground. Both are referring to support for the complete life cycle of embedded systems. The design guidelines for an APSE include software quality (reliability, maintainability, performance, evolution and responsiveness to changing requirements) and simplicity. That is, the APSE structure shall be straightforward, easy to understand and written in only one language, Ada. Additional guidelines include life cycle support, project team support, user helpfulness, uniform communications protocols and system and project portability. Portability is a key common goal for both APSEs and CIMS (DoD, 1980:14-15).

Portability. Portability from an APSE system and project standpoint is related to the APSE itself. The APSE will be standardized where it interfaces with the host computer by use of the CAIS. This means the APSE will be able to easily move from one computer system to another with just a new system dependent "kernel" implementation. For example, as only a relatively small part of one student's masters thesis work, the current prototype APSE developed at AFIT, called the AFIT Research Concept for an Ada Development Environment (ARCADE), while resident on the UNIX operating system, is in the process of being ported to the VMS operating system (Linski, 1986). With the APSE portable, all of the APSE tools and application programs (projects) are also portable.

Software portability from a CIMS standpoint is achieved through developing all software in standardized high level languages, following specified programming practices and standardizing documentation practices (Evans and others, 1977:5). Furthermore, the ICAM project indicates that software dependence "should be minimized with interfaces provided through machine independent software written in a high level language" (Evans and others, 1977:35). However, the ICAM standards report does not go as far as STONEMAN in describing how software portability can be achieved.

Additional Design Guidelines. Other STONEMAN APSE design guidelines include integrated and granular tools, which are software tools that can communicate with each other as well as being able to combine in different fashions to meet the user's requirements. Also, the APSE must be open-ended, that is, the tools can improve, update, and modify existing tools as well as create new tools. This concept of open-endedness allows for the inclusion of application-specific software, such as that which could provide ICAM support.

Robustness. The final two STONEMAN design guidelines, robustness and hardware usage, are perhaps the most critical for both an APSE and for a CIMS. Robustness means the software can protect itself from system and user errors, i.e., handle errors where possible, and, when failure is inevitable, fail gracefully. The software must be able to react to unforeseen circumstances and provide a way to diagnose and recover from error conditions.

Normally when something unforeseen occurs during the execution of a computer program, control passes from the program itself to the computer operating system and the program is aborted. In a critical embedded system this could be disastrous. A typical example would be a hospital life-support system (with an embedded controlling computer). If a failure should occur, we expect the life support system to continue to operate (albeit in a degraded mode) and also to signal the danger to the system operator. Ada and the APSE provide the method for supporting this requirement.

A CIMS example of robustness would presuppose a power failure in one automated assembly area of a factory. Rather than shutting down the entire automated factory, a robust CIMS would identify the fault to the operator and continue the rest of the factory operations (in a degraded mode).

Hardware Usage. Hardware usage by an APSE will be designed to exploit modern, high capacity hardware (DoD, 1980:15). The NBS report requires the ICAM project to be developed and implemented to at least the fourth generation of computer systems, the most advanced hardware at the time of the ICAM standards report.

Thus, research into the objectives of both an APSE and CIMS reveals they want basically the same things: simplicity; portability; standardized languages, programming and documentation practices; system robustness and exploitation of modern hardware capabilities. Apparent differences in the two documents are only in the level to which the requirements are specified. This is to be expected due to the specific application orientation of the ICAM standards report and the general environment orientation of STONEMAN.

Pictorial Representations. Before beginning the detailed comparison of APSE and CIMS components, a review of the location of the major components of an APSE and a CIMS is indicated.

APSE. Figure 4-1 shows the APSE architecture as described in Chapter III. Figure 4-2 is an adaptation of Figure 4-1 for ease of comparison. At the core is the host operating system which maintains and controls the hardware and the actual system database. The kernal APSE (KAPSE) provides all (and the only) interfaces between the APSE and the host system including access controls to system dependent functions (Babich and others, 1981:28; Booch, 1983:354).

The level above the KAPSE contains the STONEMAN-required minimal tools support and is known as the Minimal APSE or MAPSE. The user does have access to the MAPSE; it contains all the basic tools for program development and support, as well as the database management system. All of the MAPSE tools can communicate with each other (Babich and others, 1981:29; Booch, 1983:354).

The top level of the environment, the full APSE, contains tools which can be software tools supplied by the user (possibly supporting a specific application), as well as tools that go far beyond the minimal requirements in the MAPSE. STONEMAN does not prescribe tools for the APSE

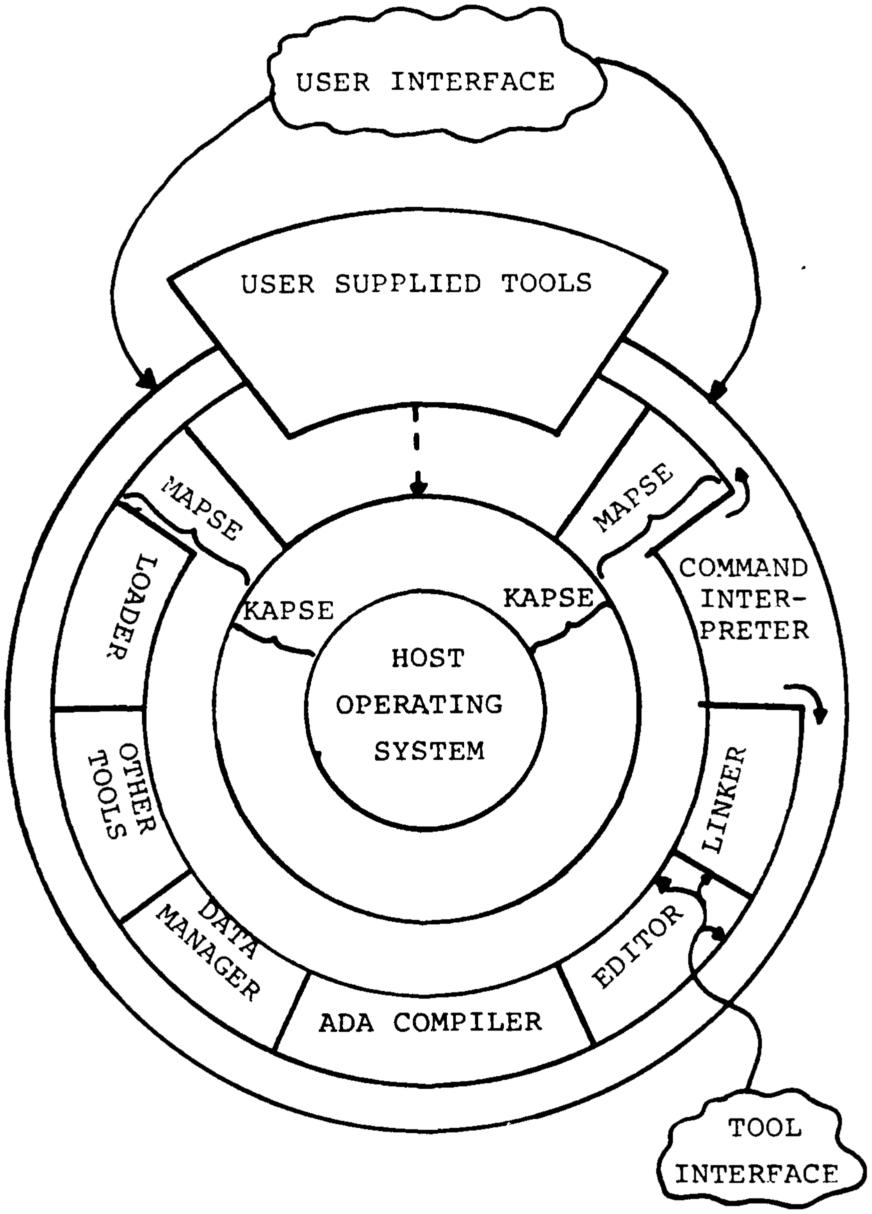


Figure 4-1. APSE Structure (Babich and others, 1981:28)

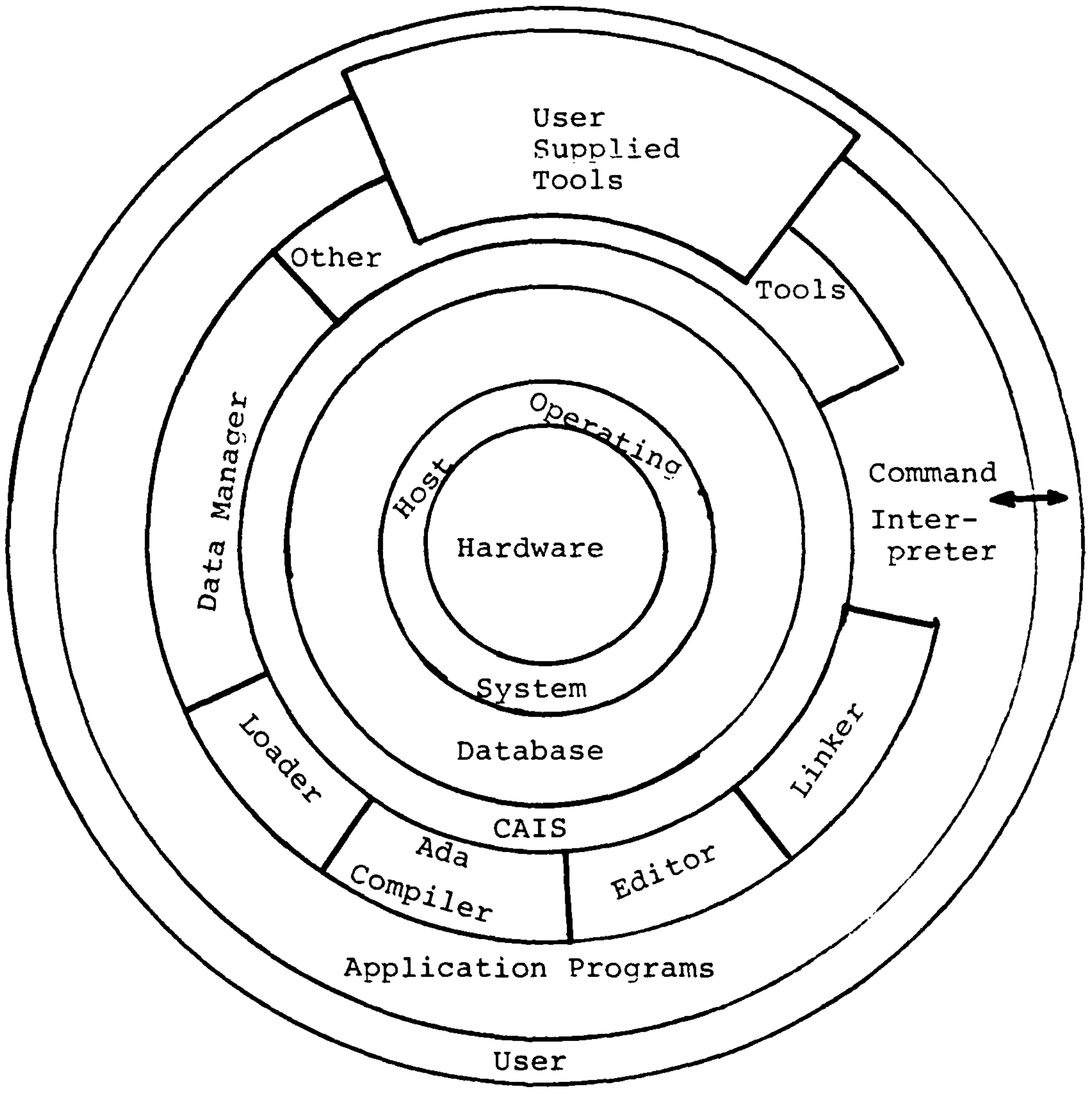


Figure 4-2. Adapted APSE Structure (Adapted from Babich and others, 1981:28)

level, but rather leaves it open to be extended and meet specific user needs. All tools, whether at the MAPSE or full APSE level, can only access host system functions by using standard calls (Babich and others, 1981:27-34; Booch, 1983:356-357).

CIMS. Figure 4-3 shows the ICAM system as described in Chapter II. Figure 4-4 is Figure 4-3 from a slightly different perspective to make it easier to compare with the APSE Figure 4-2 (imagine the left and right ends of 4-3 have been brought together to form the circle in Figure 4-4). At the core is the hardware and the operating system. Interfaces with the operating system include run time support monitors (RTSMs) for the ICAM system software and ICAM applications programs as well interfaces with the host system software and the interface between operating system and the database management system (DBMS) (through the DBMS interface) (Evans and others, 1977:32-35).

The RTSMs, DBMS, oprating system, hardware and part of the host system software comprise the ICAM version of a "kernal." Note that the APSE kernal has one standardized interface while the ICAM kernal has four. Also note the overlapping of the host dependent system software and the similarities with the KAPSE.

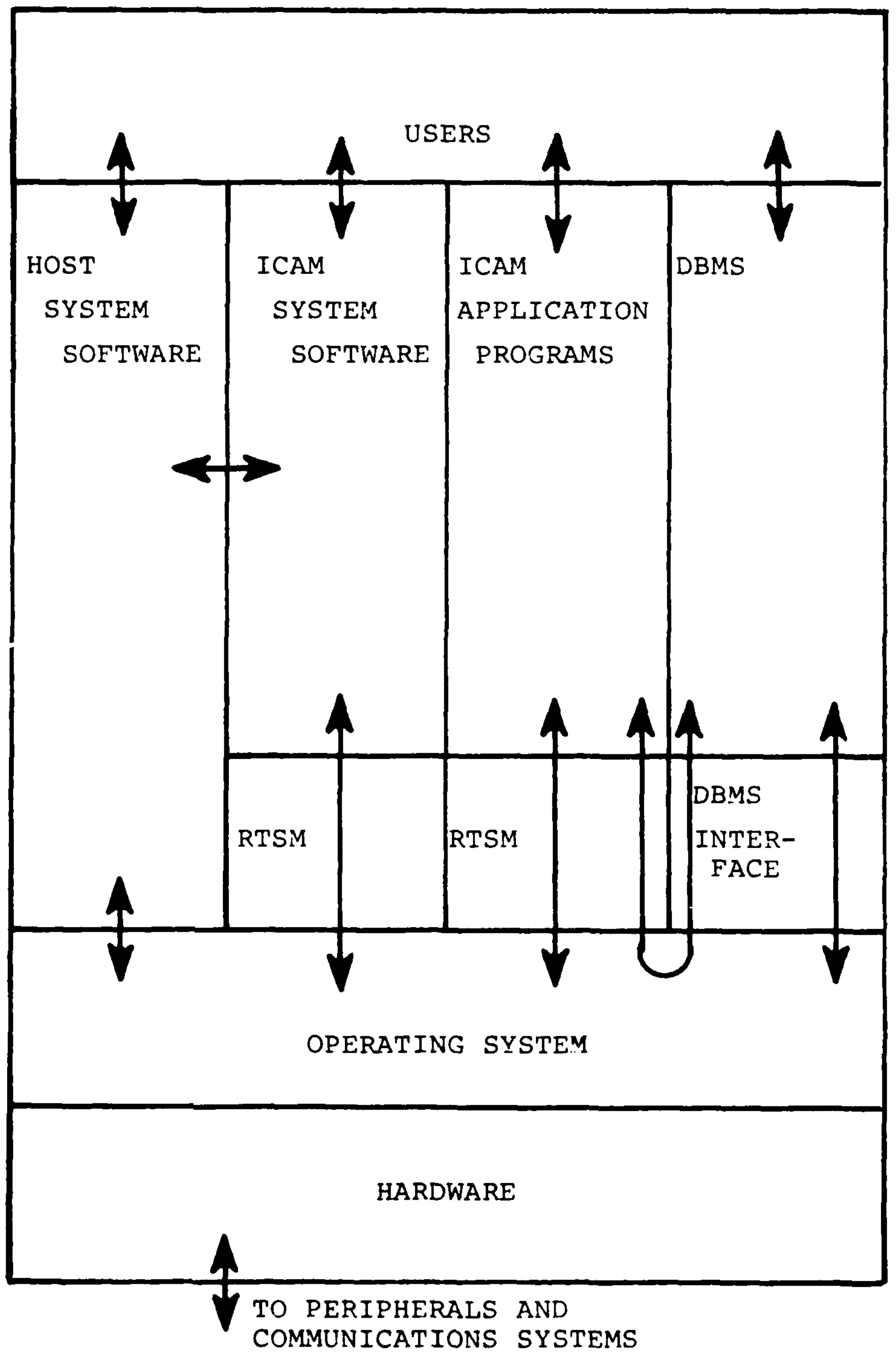


Figure 4-3. ICAM Structure (Evans and others, 1977:32)

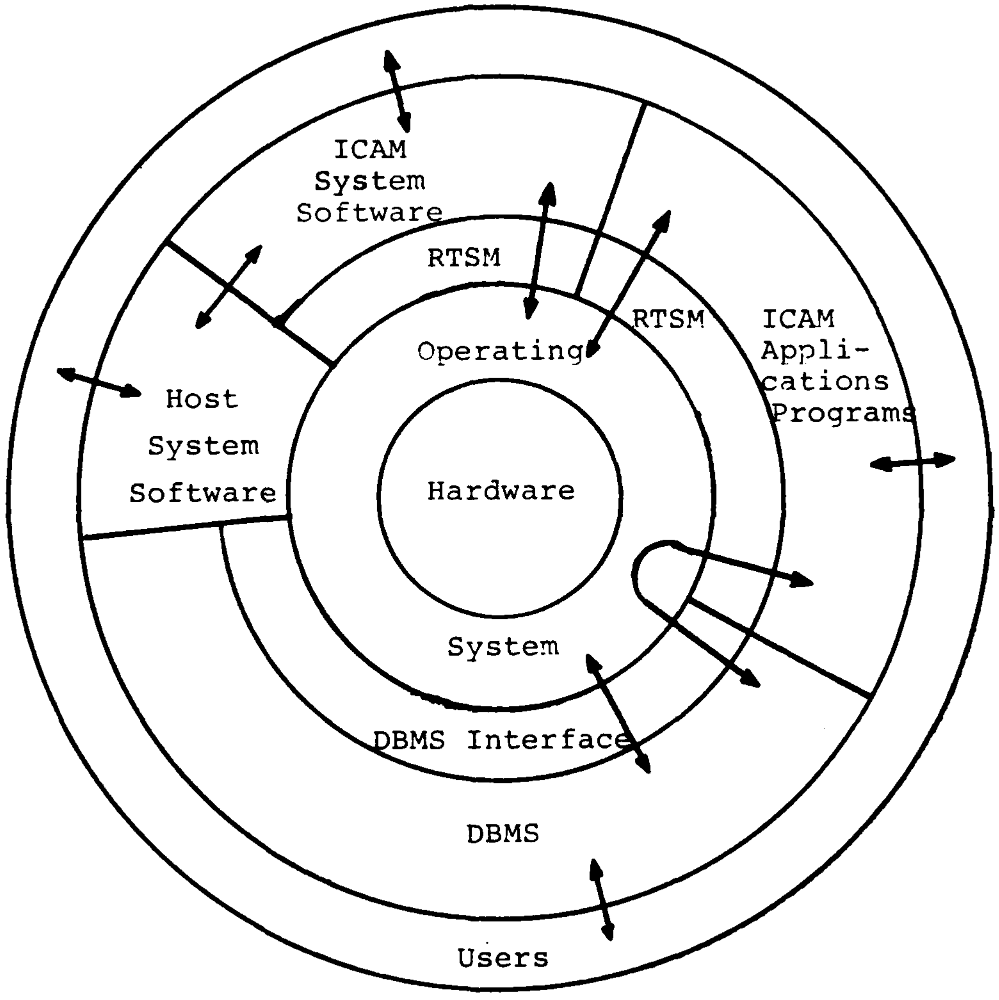


Figure 4-4. Adapted ICAM Structure (Adapted from Evans and others, 1977:32)

Moving away from the core, the next level contains the ICAM system software (which contains all the software tools for developing, modifying and maintaining the ICAM application programs), ICAM application programs, the DBMS and the host system software. Note that the ICAM system software and DBMS contain most of the tools found in the MAPSE; the balance of the MAPSE tools are in the host system software (compilers and linkers) (Evans and others, 1977:32-35).

The outer layer of the ICAM structure is partitioned into host and ICAM systems software as well as a DBMS, while the tools in an APSE are more modularized. The integrated modular tools within the APSE contain applications and high level "system software"; the data manager tool is the APSE equivalent of a DBMS. The APSE modularity illustrates a permeable barrier while the CIMS partitioning (with impermeable barriers) affords much less flexibility. In both the APSE and ICAM structures, the user interfaces with the structure through the outer barrier via a command interpreter.

Hence, the only real differences between the two postulated structures are the extent to which the user and the tools can have uncontrolled access to system resources (the ICAM structure encourages controlled access but permits limited uncontrolled access while the APSE permits no uncontrolled access whatsoever) and the extent of tool integration possibilities (the ICAM structure does not provide for interactions between high level tools from different software sets--such as an ICAM system software module with a host system software module--while the APSE permits total flexibility in the integration of high level tools).

Component Levels

A comparison of the component level of the APSE and CIMS leads to the same conclusion.

Database.
Both the APSE and CIMS use a database as a central component.

APSE.
The database is the real key to an APSE--it contains all the information for each project for the project's entire life cycle (DoD, 1980:16). In an APSE database, each separately identifiable collection of data can be related to other objects. Additionally, all iterations of an object are kept in the database to provide an object's history through the life cycle of the project and to provide configuration control on the objects themselves. Since different projects can exist, the APSE can group objects under specified projects and control user access to only designated projects. The APSE also can generate management level reports about the database (DoD, 1980:16-17).

<u>CIMS</u>. The ICAM standards report says the database is "the most critical element" and requires a single database as the "source of all information for, and hence the interface between, all application programs" (Evans and others, 1977:33).

The NBS technical report recommends a common database management system in order to integrate all software. It further suggests a relational database management system for ease of use (only one data structure for the user to learn), data independence (rather than data that is sequentially sorted, indexed or restricted to certain access paths) and flexibility. An additional benefit to a common relational database management system and common data structure is that *distributed databases (resident on* different computers) are possible, yet each can still access data on another computer system (Evans and others, 1977:194-196,204).

This modular design (of distributed databases) and common relational database management system provides the integrated modules which are a CIMS objective. The APSE also provides for integrated modules through its standardization, objects, projects and relations between/among objects. The words are somewhat different but the intent is the same.

Interfaces. The APSE and CIMS views of interfaces are described in detail in Chapters III and II respectively. Below is a summary of those descriptions.

APSE. The APSE interfaces can be viewed from three perspectives: the interfaces between the user and the APSE, the interfaces within the APSE, and interfaces between the APSE and the host computer system. As discussed in Chapter III, the user communicates with the APSE using a command interpreter. The interfaces within the APSE (intra-APSE communications) have access to the database so that the tools involved may use the objects in the database, and one tool may invoke another tool. Interfaces between the APSE and the host computer operating system (the commands used by software tools to access the database and other system functions) are standardized via the CAIS.

CIMS. The ICAM standards report addresses interfaces in three categories: interfaces between the host system and applications programs, interfaces between application programs through the DBMS and system software interfaces. As discussed in Chapter II, the applications programs will interface with the user, the operating system and the DBMS (through the operating system and the DBMS interface). The database interfaces with the user, the operating system and, through the DBMS interface, with the application programs. The host system software interfaces with the user, the operating system and the ICAM system software.

The equivalent to intra-APSE communications does not exist in the ICAM standards report. Instead, the program development tools reside partly in the ICAM system software and partly in the host system software. Indeed, while the interface concepts addressed by the NBS report are similar to the APSE interface concepts (increasing standardization and portability potential), the recommended CIMS interface actualizations appear to come up short of achieving true portability and standardization.

Toolset. The difference in the APSE and CIMS discussions of tools is in the level of specification. These also were described in more detail in Chapters III and II respectively, and those descriptions are summarized below.

APSE. As discussed in Chapter III, the APSE toolset contains software tools for development, application program management and project management as well as all the project-unique tools that may be developed. The tools are coordinated and provide life cycle support for a project. The APSE toolset will not be closed. New tools can be built with existing tools and added to the APSE.

CIMS. As discussed in Chapter II, the ICAM standards report indicates that CIMS systems software should contain "all of the software tools needed for (CIMS) development, maintenance and operation" (Evans and others, 1977:34). Specialized software for a given project may range from

> . . . minor extensions of extant tools to new composite tools formed by integrating and refining several distinct packages . . . [this] requires that the building block tools be carefully designed, with flexible interfaces and modular design. (Evans and others, 1977:219)

Thus APSE and CIMS toolsets have the same minimum requirements, the same concepts of specialization and the same objective, transportability. The two toolsets do not, however, appear to have the same ease of use and extendability (extensibility) capabilities because of the lack of an interface between/among the tools and because of the partitioned location of the CIMS tools.

Differences

The ICAM standards report, while conceptually similar to STONEMAN, does address some topics not covered by STONEMAN but still technically related to an APSE (again, this is related to the fact that the CIMS document is more specific to an application area than STONEMAN). These topics include numerically controlled (NC) machine part programming languages, computer aided design/computer aided manufacturing (CAD/CAM) interface standards, media standards and di.:.ributed databases.

At the time of the NBS report (1977), the two recommended NC programming languages were Automatically Programmed Tools (APT) and Computer Programs for Automatically Controlling Tools (COMPACT II). Since that time and with the increased use of direct NC (DNC), computer NC (CNC) and robotics, no one standardized language dominates the field. Recently, reports have been published of attempts to standardize and integrate all NC-related language requirements with the current CAD/CAM languages. Grossman suggests merging APT with A Manufacturing Language (AML) with the result being an object-oriented approach for NC, robotics and CAD (Grossman, 1986:515-522).

STONEMAN requires that all APSE-level programs use the standard Ada Language. This standardization on Ada could be applied to all CIMS applications programs and high level (outside the kernal) software too^1s. However, the NBS report also refers to the use of languages in two areas not really addressed by STONEMAN. One is the language of user interfaces, such as CAD/CAM interfaces, or how a user can best communicate with application programs. The second is the low level languages, such as NC programming languages, used to instruct the embedded computer processors themselves. Standardization in these areas may be desirable, but the STONEMAN work implies that it may not be required in order to achieve the objectives of the ICAM project.

Media standards recommendations in the NBS report include using magnetic tape conforming to ANSI standards as the primary information exchange and storage media, deemphasizing the use of punched paper cards and avoiding punched paper tape (Evans and others, 1977:239). STONEMAN does not specifically address media standards since the media standards issues relate more to the system dependent KAPSE area (STONEMAN is more concerned with the higher level software--it does not really delve into hardware issues except to say what hardware must be supported by the KAPSE).

The ICAM standards report does not officiall recommend a distributed database over a centralized database, but does point out several advantages to a distributed system including local, rather than central, data validation; flexibility and localized control; and more potential modularization. Disadvantages are also discussed, the main ones being: whether or not the user must specifically know where each dataset is located and whether or not there can be common communications to all the databases (Evans and others, 1977:196,199).

Distributed databases are not specifically addressed in STONEMAN. Prototype Ada distributed database management systems currently exist as a result of research sponsored by Defense Advanced Research Projects Agency (DARPA) and Naval Electronics Systems Command (NAVELEX) contracts (Chan and others, 1986:G.1.3.1 - G.1.3.20).

Summary

In this chapter, the APSE requirements as embodied in STONEMAN have been compared with the ICAM standards report recommendations for CIMS standards. Both documents are similar in their overall objectives, design guidelines and hardware usage recommendations. Additionally, when compared from an APSE component perspective, the documents have similar requirements for databases, interfaces, and tools. The NBS document addresses four main items which STONEMAN does not: NC programming languages, CAD/CAM interfaces, media standards and distributed databases. Some differences are to be expected however, due to the specific application orientation of the ICAM standards report and the general environment orientation of STONEMAN. The point is that the similarities are significant, indicating that APSE concepts should be applied to future CIMS work.

V. Conclusions and Recommendations

Process control and factory automation applications look a great deal like DoD applications. Ada isn't just the DoD's language. It's the best language for large, complex software development. (Ken Drager, President of Computer Corporation of America, as quoted by Carlyle, 1986:30)

Conclusions

Recall the research question postulated in Chapter I: How similar are APSE and CIMS concepts, and would the similarities justify applying APSE concepts for solving CIMS problems? Based on the discussions in Chapter IV, this chapter provides the answer to that research question, by addressing each research objective.

Research Objective 1. Compare the objectives of APSE and CIMS: Although the specifics are different, at a high level both APSE and CIMS refer to support for the complete life cycle of an embedded computer system.

Research Objective 2. Compare the design guidelines of APSE and CIMS: Both an APSE and CIMS are to be simply designed, portable, based on standardized languages, follow standardized programming and documentation practices, support system robustness and exploit the capabilities of modern hardware.

Research Objective 3. Compare the pictorial representations of APSE and CIMS: The only real differences between the APSE and CIMS postulated structures are the extent to which the user and the tools can have uncontrolled access to system resources and the extent of tool integration possibilities.

Research Objective 4. Compare the component levels of APSE and CIMS by examining the database, interfaces and toolset of each:

Database. Both APSE and CIMS use a database as the central component. Each requires the database to be the source of all information for applications programs. The intent of both the APSE and CIMS databases is the same in that each provides integrated modules which support ease of use, data independence and flexibility.

Interfaces. While the interface concepts addressed in the NBS report are similar to the APSE interface concepts, the recommended CIMS interface actualizations appear to come up short of achieving true portability and standardization.

Toolset. The APSE and CIMS toolsets have the same minimum requirements, the same concepts of specialization and the same objective, transportability. The two toolsets do not appear to have the same ease of use and extensibility capabilities because of the lack of an interface between/among the tools and because of the partitioned location of the CIMS tools.

<u>Research</u> <u>Objective</u> <u>5</u>. Identify major differences between APSE and CIMS: The APSE and CIMS differences appear to be more of a matter of the scope of the two documents (STONEMAN and the NBS report) than of actual conceptual differences.

Standardization in the areas of low level languages (such as NC programming languages) and the languages of user interfaces (such as CAD/CAM interfaces) may be desirable, as is recommended in the ICAM report. However, STONEMAN implies that this standardization may not be necessary to achieve the ICAM project objectives, as long as a standard interface exists between the high level tools and the low level system software. Since STONEMAN really addresses higher level software, the issue of media standards, which is predominantly hardware oriented, is never addressed for an APSE. Distributed databases are not addressed in STONEMAN (the idea was to get implementations of "simple" APSEs before worrying about more complex issues). However, as the work on distributed databases begins to mature, it offers an APSE the same advantages as it offers a CIMS.

Research Objective 6. If deemed appropriate, recommend ways APSE concepts can be applied to CIMS in order to help solve the manufacturing crisis within the Air Force and U.S. industry: Having determined that APSE concepts can be appropriately applied to CIMS, CIMS could benefit greatly from applying the APSE ideas of:

1. Standardizing on Ada as the HOL to be used for high level software tool development (including applications-specific software).

2. Using an APSE-like software environment with flexible intra-APSE interfaces, but a very strict standard (CAIS) interface between the high level tools and the kernal.

These Ada standards could be the key to unlocking the doors that until now have been blocking the realization of CIMS in the Air Force as well as in U.S. manufacturing.

Recommendations for Follow-on Research

Many potential follow-on research efforts can be proposed for this initial comparison of APSE and CIMS concepts. They include the following:

1. Examine ARCADE to determine if it can meet the CIMS requirements.

2. Determine the current status of the ICAM standards and compare them with STONEMAN.

3. Develop a proposed prototype CIMS application using Ada and the APSE capabilities to demonstrate the combined concepts' feasibility for enhancing manufacturing productivity.

4. Compare STONEMAN and the CAIS with standards proposed for commercial CIMS applications, such as General Motors' manufacturing automation protocol (MAP), in order to extend the combined APSE/CIMS concept for potential government applications.

Appendix: Chronology of Ada and APSE Requirements Documents

	Ada Requirements Documents	APSE Requirements Documents
July 1975	STRAWMAN	
August 1975	WOODENMAN	
January 1976	TINMAN	
January 1977	IRONMAN	
July 1977	IRONMAN (Revised)	
June 1978	STEELMAN	
July 1978		SANDMAN
July 1978		PEBBLEMAN
February 1980		STONEMAN

Bibliography

Adams, Karyl A. A Display Environment Supporting the Interactive Generation of alphaNumerics and Symbology with DESIGNS on the Future. MS thesis, AFIT/GCS/MA/85J-1. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1985.

Babich, Wayne and others. "The Ada Language System" (reprinted from Computer), Ada. Waltham MA: SofTech, Inc., 1981.

Barstow, David R. and others. Interactive Programming Environments. New York: McGraw-Hill Book Co., 1984.

Booch, Grady. Software Engineering with Ada. Menlo Park CA: Benjamin/Cummings Publishing Co., 1983.

Canan, James W. "The Software Crisis," Air Force Magazine, 69: 46-52 (May 1986).

Carlson, William E. "Ada, A Promising Beginning" (reprinted from Computer), Waltham MA: SofTech, Inc., 1981.

Carlyle, Ralph Emmett. "Putting Spurs to Ada," Datamation, 32: 30-31 (1 September 1986).

Chan, Arvola and others. "A Database Management Capability for Ada" (reprinted from the Proceedings of the Annual Washington Ada Symposium, March 1985), Proceedings of the First International Conference on Ada Programming Language Applications for the NASA Space Station, edited by Rodney L. Bown. G.1.3.1 - G.1.3.20. Houston: University of Houston - Clear Lake, 1986.

DeLauer, Richard D., Under Secretary of Defense, Research and Engineering. Memorandum for Secretaries of the Military Departments and others on Interim DoD Policy on Computer Programming Languages. Department of Defense, Washington, 10 June 1983.

Department of Defense. "Requirements for Evaluation and Validation of Ada Programming Support Environments." Evaluation and Validation Team Requirements Working Group Working Paper Version 2.0, 6 December 1985.

-----. Requirements for Ada Programming Support Environments (STONEMAN). Washington: Government Printing Office, February 1980.

Evans, John M. and others. Standards for Computer Aided Manufacturing. April 1976 - December 1976. Contract FY145776-00369. National Bureau of Standards, Washington, June 1977 (AFML-TR-77-145).

Fisher, David A. "A Common Programming Language for the Department of Defense - Background and Technical Requirements." Report P-1191. Institute of Defense Analysis, Washington, 1976.

Freedman, Roy S. Programming with APSE Software Tools. Princeton NJ: Petrocelli Books, 1985.

Groover, Mikell P. and John C. Wiginton. "CIM and the Flexible Automated Factory of the Future," Industrial Engineering, 18: 75-85 (January 1986).

Grossman, David D. "Opportunities for Research on Numerical Control Machining," Communications of the ACM, 29: 515-522 (June 1986).

Gunn, Thomas. "The CIM Connection," Datamation, 32: 50-58 (1 February 1986).

Hicks, Donald A., Under Secretary of Defense for Research and Engineering. Memorandum for Secretaries of the Military Departments and others on Implementation of Ada in Department of Defense Programs. Department of Defense, Washington, 2 December 1985.

Lawlis, Maj Patricia K. "Ada - Europe, 5-9 May 1986, Ada, Managing the Transition." Trip Report. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, May 1986.

-----. Arbitrary Precision in a Preliminary Math Unit for Ada. MS thesis, AFIT/GCS/MA/82M-2. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1982.

Lieblein, Edward. "The Department of Defense Software Initiative--A Status Report," Communications of the ACM, 29: 734-744 (August 1986).

Linski, 2Lt David M. "Design and Implementation of a Prototype APSE with a CAIS." Independent Study Prospectus. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June 1986.

Manchuk, Stan. "CIM Project May be Doomed if Key Building Blocks Are Missing," *Industrial Engineering*, *16*: 34-43 (July 1984).

Meredith, Jack. "The Economics of Computer Integ ted Manufacturing" (reprinted from the *1984 Fall Industrial Engineering Conference Proceedings*), *Computer Integrated Manufacturing Systems: Selected Readings*, edited by John W. Nazemetz and others. Atlanta: Industrial Engineering and Management Press, 1985.

Mitchell, Russell. "Painful Lessons," *BusinessWeek*, 100-103, 16 June 1986.

Peschke, Lt Col Richard E. *A Structured Optimization Method for Information Resource Management*. PhD dissertation. University of Houston, Houston TX, 1985.

Port, Otis. "High Tech to the Rescue," *BusinessWeek*, 103-104, 16 June 1986.

Sadowski, Randall P. "History of Computer Use in Manufacturing Shows Major Need Now is for Integration," *Industrial Engineering*, *16*: 34-42 (March 1984a).

-----. "Computer-Integrated Manufacturing Series will Apply Systems Approach to Factory of Future," *Industrial Engineering*, *16*: 35-40 (January 1984b).

Sammet, Jean E. "Why Ada is Not Just Another Programming Language," *Communications of the ACM*, *29*: 722-731 (August 1986).

Skinner, Wickham. *Manufacturing: The Formidable Competitive Weapon*. New York: John Wiley and Sons, 1985.

Wallace, Robert H. *Practitioner's Guide to Ada*. New York: McGraw-Hill Book Co., 1986.

Willis, Roger G. and Kevin H. Sullivan. "CIMS in Perspective: Costs, Benefits, Timing, Payback Periods are Outlined," *Industrial Engineering*, *16*: 28-36 (February 1984).

Young, Robert E. and Richard Mayer. "The Information Dilemma: To Conceptualize Manufacturing as Information Process," *Industrial Engineering*, *16*: 28-34 (September 1984).

Vita

Kathleen M. Austin was born on 17 March 1952 in Columbus, Nebraska. She graduated from Columbus High School in 1970. After graduating from the University of Nebraska at Omaha in December 1977, she worked for Allied Chemical until entering the Air Force in February of 1979. She was commissioned through OTS in May of the same year. Her first assignment was at Rome Air Development Center, Griffiss AFB, New York, where she served as Chief, Program Control, for the PAVE MOVER (now JSTARS) and Assault Breaker Programs. In October of 1982, she was assigned to the Air Force Plant Representative Office (AFPRO) at Hughes Aircraft Company, Missile Systems Group, in Tucson, Arizona. Her duty was as Program Manager for Army and Navy Programs, including the Navy's Phoenix missile, the Army's Tube-Launched Optically-Tracked Wire-Guided (TOW) missile and the Marine Corp's Angle Rate Bombing Set (ARBS). She left Tucson for the Air Force Institute of Technology School of Systems and Logistics in May of 1985. She is a member of the National Contract Managers Association (NCMA) and the Society of Logistics Engineers (SOLE) as well as Alpha Iota Delta and Sigma Iota Epsilon honoraries. She has had two papers published in the AFIT _Compendium of Papers for Systems Production Management_,

Permanent address: 11 Beaver Lodge Road
Columbus, Nebraska 68601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A174596

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GLM/ENC/86S-2	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Systems and Logistics 6b. OFFICE SYMBOL *(If applicable)* AFIT/ENC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS *(City, State and ZIP Code)* Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433-6583	7b. ADDRESS *(City, State and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State and ZIP Code)*	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE *(Include Security Classification)* See Box 19	

12. PERSONAL AUTHOR(S)
Kathleen M. Austin, B.S.B.A., Captain, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr., Mo., Day)*	15. PAGE COUNT
MS Thesis	FROM ________ TO ________	1986 September	96

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB. GR.	
09	02		Programming Languages, Manufacturing, Production, Integrated Systems, Computer Aided Design
13	08		

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

Title: APPLYING ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE) CONCEPTS FOR COMPUTER INTEGRATED MANUFACTURING SYSTEMS (CIMS) SOLUTIONS

Thesis Chairman: Patricia K. Lawlis, Major, USAF
Instructor

Approved for public release: IAW AFR 190-1.
[illegible] 24 Sept 86
LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AFC)
Wright-Patterson AFB OH 45433

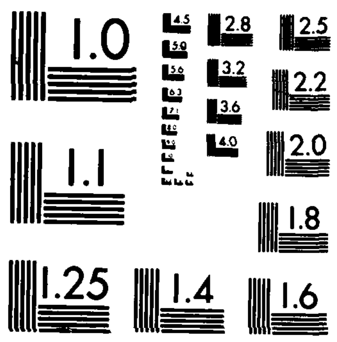
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED [✓] SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER *(Include Area Code)*	22c. OFFICE SYMBOL
Patricia K. Lawlis, Major, USAF	513-255-3098	AFIT/ENC

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A174 596 APPLYING ADA PROGRAMMING SUPPORT ENVIRONMENT (APSE) 2/2
CONCEPTS FOR COMPUTER (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF SYST K M AUSTIN
UNCLASSIFIED SEP 86 AFIT/GLM/ENC/86S-2 F/G 9/2 NL



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

This thesis compares the concepts of Ada Programming Support Environments (APSE) and Computer Integrated Manufacturing Systems (CIMS) to see if they are similar enough to apply APSE concepts for solving CIMS problems. After establishing a discussion baseline for each of the concepts, the objectives, design guidelines and pictorial representations of an APSE and a CIMS are first compared. Then the two concepts are compared at the database, interfaces and toolset level. The major differences between an APSE and a CIMS are also identified. Finally, after establishing that similarities do exist between APSE and CIMS concepts, recommendations are made for applying APSE concepts for CIMS solutions that will contribute to achieving Air Force as well as U.S. manufacturing goals.

END

12-86

DTIC