NASA CR-172,582

NASA Contractor Report 172582

EXECUTIVE CONTROL SYSTEMS IN THE ENGINEERING DESIGN ENVIRONMENT

Patricia W. Hurst

UNIVERSITY OF VIRGINIA
Charlottesville, Virginia

Grant NAG1-242
May 1985

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

ABSTRACT

An Executive Control System (ECS) is a software structure for unifying various application codes into a comprehensive system. It provides a library of applications, a uniform access method through a central user interface, and a data management facility. This research report is based on a survey of twenty-four Executive Control Systems designed to unify various CAD/CAE applications for use in diverse engineering design environments within government and industry. The goals of this research were to establish system requirements, to survey state-of-the-art architectural design approaches, and to provide an overview of the historical evolution of these systems.

Foundation for design are presented and include environmental settings, system requirements, major architectural components, and a system classification scheme based on knowledge of the supported engineering domain(s). An overview of the design approaches used in developing the major architectural components of an ECS is presented with examples taken from the surveyed systems. The evolution from early efforts to the current state-of-the-art ECS are presented as three stages of developments: embryonic, batch environment, and conversational environment. Attention is drawn to four major areas of

ECS development that are central to advancing the state-of-the-art and which include inter-disciplinary usage, standardization, knowledge utilization, and computer science technology transfer. For each system included in the survey, a snapshot description is given with references to source documentation.

ACKNOWLEDGEMENTS

I would like to thank my committee, Professors T.W. Pratt, W.N. Martin, and C.C. White, for their guidance and patience in the preparation of this report.

I would also like to thank Professors J.L. Phaltz, J.C. Knight, and T.M. Sigmon for their comments on the preliminary draft of this report which led to the amplification and strengthening of Section 6.

I would like to thank the IPAD Office of NASA Langley Research Center for their financial support of this research. Additionally, I would like to thank individuals within that office for giving me access to their personal collections of historical and current documentation.

This research was supported in part by NASA grant NAG-1-242.

TABLE OF CONTENTS

TABLE OF CONTENTS--continued

1.0 THE NEED FOR EXECUTIVE CONTROL SYSTEMS

An Executive Control System (ECS) is a software structure for unifying various application codes into a comprehensive system. It provides a library of applications, a uniform access method through a central user interface, and a data management facility for creation of data and movement of data between the applications. Used by various engineering organizations throughout government and industry, the ECS provides an engineering design environment within which a wide range of application codes, and utilities for their effective usage, are placed at the fingertips of the engineer user for improved accessibility, efficiency, and ease of use.

The genesis of the ECS in engineering design is found in the 1960's when various individuals and groups within government and industry recognized the need for improving access to the large variety of application codes being constructed for various aspects of engineering design. Subsequent efforts have spawned a variety of systems used by diverse engineering design groups. Examples of currently used systems include ANOPP, PRIDE, and AVID developed by NASA Langley Research Center, DYSCO developed by Kaman Aerospace Corporation, NICE developed by Lockheed Missiles and Space Company. and ISAS developed by the Boeing Company. Among the dozens of systems which have

been built, there is wide variance with respect to user interface, data communication, and auxiliary features.

New development efforts are continuing within government and industry, with a growing recognition of the potential benefit that such systems could bring to the engineering design communities. Unfortunately there is little in the published literature to provide a general overview of the systems requirements, design approaches, and major systems components of the existing Executive Control Systems. As a result, new designs often reinvent concepts and repeat mistakes needlessly.

The central goal of this thesis is to provide a comprehensive overview of the history and state-of-the-art of ECS design, so as to provide a basis for understanding and advancing the design of future systems. The context of computer aided design (CAD) or computer aided engineering (CAE) development and the major problems leading to ECS construction are outlined in this section. The following section gives a more detailed explanation of the goals, approach, and structure of this survey.

CAD/CAE in Engineering Projects

CAD/CAE has been increasingly employed within government and industry since the early 1950's when computers were placed in engineering departments for use as calculators [1]. As the power and sophistication of computer systems has increased during the last three

decades, so has their range of utilization. Today CAD/CAE has a broad definition which includes computerized methods utilized throughout the design, planning, and manufacturing phases of a product. Accompanying this broad definition are a wide variety of software tools for use in such areas as engineering design and analysis, data base management, management information, graphics, drafting, numerical control machining, and robotics. Although used to varying degrees depending on the size and needs of particular engineering environments, CAD/CAE has become a standard tool for increasing both productivity of engineering manpower and utilization of machine tools and equipment.

While the CAD/CAE tools are used effectively within each of the product phases, they currently do not provide the power or breadth required for unified usage across the phases. A major factor is the information gap existing between the planning, design, and manufacturing phases. Project management activities of planning require frequent input from the other phases, yet the type of information required is not currently captured by the CAD/CAE tools used in design and manufacturing. The interface between design and manufacturing, which in times past could be fairly well achieved with conventional engineering drawings, is becoming more complex as the design data bases are accessed by manufacturing for information

required to fabricate and assemble parts. It is becoming a necessity for engineering data bases to accommodate manufacturing requirements and thus contain change and release information for configuration control in addition to geometric data [2].

The government's concern for a more orderly, comprehensive, industry-wide development program to integrate CAD/CAE tools for unified usage across the phases of product development is illustrated by the IPAD program initiated in the mid 70's. An original goal of IPAD was to establish the various tools under one software umbrella system with automated transfer of data between phases. For example, information required by project management activities would be automatically collected by the system as the engineers progressed through design and analysis. Also information required by manufacturing would be captured on the engineering data base during design activity. However the state-of-the-art in CAD/CAE at that time would not support the interface requirements between design and manufacturing. Much effort has since been devoted to bridging this gap, and it continues to be an important objective of CAD/CAE.

The Engineering Design Phase

Within the engineering design phase similar problems exist. Many of the design and analysis tools are stand-alone application codes written in diverse computer

environments. These tools execute as independent units within the framework of the host operating system. Typically each has its own input-output requirements and format and, if conversational, its own style and language for conversing with the engineer user. In order to accomplish particular design scenarios, multiple stand-alone codes are often used by the engineer in sequential fashion with looping back to a previous step as required. In these multiple application scenarios, there are three problem areas with tend to erode the benefits of CAD/CAE:

(1) Data flow between application codes.

In general, data which is the output from one or several applications provides the basis for data input to a subsequent application. However, compatibility of data at these interface points frequently does not exist and extra steps must be taken to combine data, generate additional data, or translate data to different forms.

(2) Difficulty of use of multiple application codes.

Another problem area in using multiple applications is the learning time required to use each effectively. This involves familiarity with such application elements as terminology, input/output requirements and form, user interface language, and engineering algorithms embodied in the code. This also involves familiarity with the host operating system Job Control Language (JCL) for executing the applications and also with utilities for creating/

modifying data as required. The host computer may not be the same for all of the applications and thus an engineer may need expertise in the use of more than one operating system. Martin Marietta, in an early 80's study of in-house computers and design aids used by circuit designers, found that the typical designer had to learn more than 25 computer languages, procedures, and techniques if Computer Aided Circuit Design (CACD) tools were used [3]. The same study found that over 100 situations could arise during a computer-aided design scenario that could stop the progress of a designer unless the right information was known and applied. Many designers felt they could not afford the time or effort required and used CACD only if the job demanded it.

(3) **The continuous evolution of application codes.**

The set of available engineering applications is continually changing. New application codes are added to reflect advances in engineering and software methodologies while others become obsolete and are no longer used. If the engineer is to utilize the improved tools as they become available, then the problems of data flow and learning time associated with multiple application scenarios are compounded.

Executive Control Systems in Engineering Design

To reduce and control problems in these three areas, the Executive Control System (ECS) has been introduced as

a higher level tool which provides for an automated engineering design environment. Within this environment are the various CAD/CAE applications to be used in design scenarios, mechanisms for communicating and interfacing data, and a central engineer-machine interface which allows for their effective usage. The central interface is a powerful component of the ECS in that it abstracts those capabilities of the host operating system typically required for executing applications in an arbitrary user defined sequence. Functions which the user would request the operating system to perform via a JCL sequence are performed within the ECS design environment as a result of user requests via the central interface. This environment provides an evolving set of available applications into a comprehensive system and places them at the fingertips of the engineer. The ultimate goals of introduction of an ECS are to improve performance and increase productivity of the users in the wake of a continually expanding myriad of engineering applications.

Other Applications of the ECS Concept

The ECS approach to unifying various applications within a single operational framework is not unique to engineering design and has been utilized in various domains. For example, the Harvard Programming Development System [4] uses the ECS approach to provide a software development environment. This system consists of an

applications set of software tools (e.g., editors, compilers, loaders, symbolic debuggers, tracing facilities, pretty printers, and probing packages), data management facilities, and a central executive interface to the user for executing the various applications and utilizing the data management facilities. Another example is the Executive Information System [5] marketed by Boeing Computer Services, which provides an embryonic Decision Support System generator. Applications which have been individually available for supporting financial decision-making situations were unified into a collection and made available through a central command language which acts on a common set of data. The Interactive Financial Planning System [5], marketed by Execucom Systems, is a similar system.

2.0 GOALS AND STRUCTURE OF THE SURVEY

This research report is a survey of Executive Control Systems designed for use in various engineering design groups within government and industry. The information on which it is based was acquired from a variety of sources including literature search, telephone inquiries to large engineering companies and selected government engineering installations, and individual references. Major support was provided by the IPAD Office at NASA Langley Research Center. Twenty-four systems were identified and studied to provide the basis for this report.

The primary goals of this research were to establish the requirements for these systems and the state-of-the-art in architectural design approaches, as seen in those systems currently being used or under development. The documentation was reviewed and evaluated with emphasis being placed on the following focal points:

(1) Establishing system requirements, as evidenced by the features and capabilities provided by each system.

(2) Identifying the basic characteristics or design philosophies which could be used to distinguish different types of systems.

(3) Identifying the major architectural components common to all systems.

Sections 3 and 4 are the central sections of this report. Section 3 provides a discussion of the foundations for architectural design, which includes environmental settings, system requirements, major architectural components, and a system classification scheme based on knowledge of the supported engineering domain(s). Section 4 presents a composite of the design aprroaches used in developing the major architectural components of Executive Control Systems.

A secondary goal was to provide an overview of the historical evolution of these systems. From the documentation available on selected embryonic systems developed during the 1960's and early 1970's, several evolutionary development trends emerged which are summarized in Section 5.

It is hoped that in satisfying these research goals, a foundation could be established both as a starting point for developing more advanced systems in the future and also as a general source of information for interested groups. Section 6 draws attention to three major areas of ECS development that are central to advancing the state-of-the-art. Section 7 summarizes the main points of the survey.

This report does not attempt to provide any form of a "shopping list" of systems for a prospective engineering user group, nor was any attempt made to evaluate the

relative merits of the individual systems. Thus in discussing specific features or design approaches, limited enumeration of example systems is given. For readers desiring additional information, snapshot descriptions of each system are given in Section 8, along with references to the available literature, to provide guidance in determining possible sources.

References to source documents have been limited in this report due to the frequent mention and discussion of the various systems. References in Section 1-7 are given only for information which is not related to specific systems. However, a full reference list for each system accompanies its snapshot description in Section 8.

3.0 FOUNDATIONS FOR DESIGN

The purpose of this section is to establish a foundation and perspective for viewing, in the next section, the various architectural design approaches taken in ECS development. The discussion is organized into four parts: environmental settings, requirements, system architectural, and system classification. The environmental settings for development and usage of these systems provide an understanding of both the motivation behind their development and the resources available for their design and implementation. The requirements and system architecture overviews provide an organizational structure for subsequent presentation of the various design approaches. A scheme for classifying systems based on "domain knowledge" is presented in the final part of this section.

3.1 Environment

The environmental settings for development and use of these systems are diverse. This diversity is seen in four main areas: the user communities, the types of engineering design applications to be accessed through the system, the host hardware, and the design/development effort. These areas are overviewed in the following sections.

3.1.1 Users

ECS user communities are found in various engineering design domains. Examples are structural analysis, aircraft noise prediction, electronic circuit design, and fluid/solid mechanics. These communities are typically found in large engineering companies, such as Lockheed Missiles and Space Company or Martin Marietta Corporation, and also in engineering branches of government agencies, such as NASA and the U.S. Army, and their contractors. To a lesser extent, users are also found in smaller engineering firms which have access to an ECS through a time-sharing network.

The user group for a particular ECS is also varied and in some instances evolves over time. It may be a single research group composed of many or only a few individuals, several groups within the same company, or a variety of users nationwide. ACTION was developed and is used solely by a small research group at NASA Langley Research Center for structural analysis. NICE was developed in-house by Lockheed and is used by various groups within that company for fluid/solid mechanics. ISAS was initially developed by the Boeing Company for in-house structural analysis but has evolved to incorporate applications for a variety of engineering domains and is commercially available nationwide to a host of users. ANOPP, which was developed by NASA for the aircraft noise

prediction community, has since been applied to a different engineering domain within Hughes Helicopter Company by replacing the original collection of applications with a different set.

The users of an ECS also vary as individuals with respect to level of formal training, working experience, and general expertise. Knowledge is required in three main areas in order to use an ECS. These include: (1) engineering knowledge in formulating a problem, selecting appropriate applications, and analyzing the results, (2) applications knowledge in understanding the embodied engineering algorithms and determining the input/output requirements, and (3) ECS knowledge in properly communicating through the central interface and selecting among available features. In each of these areas, individual users may range from novice to expert in terms of training, experience, and general expertise.

3.1.2 Applications

The collection of applications "installed" in an ECS and thus available for use is typically limited to those appropriate for the user community. Thus, a system such as LSSIAP which is used by Martin Marietta Corporation engineers to design attitude control systems for large space systems would contain only those design/analysis

applications and auxiliary applications, such as graphics, used by those particular engineers.

The source, size, and general software nature of the installed applications vary but are typically one or more of the following:

Third party. Many of the applications have been developed outside of the ECS environment, often by a different company or government agency, as independent "stand-alone" programs. Such a program may be a large marketed program, such as the MSC/NASTRAN structural analysis program available through the MacNeil-Schwendler Corporation, or it may be a program obtained through a personal reference. These programs vary with respect to size, theoretical complexity, and software construction complexity and quality. In most cases, it is undesirable to "tamper with" these programs, and thus the ECS must accommodate installation of these programs with minimal modification.

Tailored. These applications have been developed specifically for a particular ECS environment, and thus the code has been written to take advantage of utilities provided within the ECS and/or conform to imposed standards. Systems which offer a wide variety of utilities available to the engineering application programmer are ANOPP, DYSCO, AND IAC.

Personal. These applications are those developed outside of the ECS environment as independent programs but within the user group. Thus, there is usually access to knowledge about the software construction of the program and theoretical details. Although modification to these programs in order to install them into a ECS may be feasible, it is often undesirable.

3.1.3 Hardware

Host hardware environments which support the ECS are varied, but because of the large size and computational intensity required by many engineering applications, a mainframe computer is uniformly a component. Examples are the Control Data 6600 and Cyber series, the IBM 360 and 370 series, and the DEC VAX series.

An ECS is designed to abstract the host operating system features and allow utilization of hardware configuration components to varying degrees. Thus, the design of an ECS is based on certain assumptions about the environment provided by the target configuration. There are three main types of environments supported:

Batch. The batch ECS is designed for environments supporting those features typically found in the traditional configurations of the 1960's and 1970's. The basic user input to the ECS is a set of prefabricated card images stored on a file or existing as a physical card

deck. In modern environments, the user typically fabricates the input interactively and stores it on a file. ANOPP, ODPIN, and PRESTO are examples of the batch ECS.

Conversational. The conversational ECS is designed to utilize the modern communication features of interactive computing. The general model for this ECS is to receive a user instruction, interpret and execute it, request the next instruction, and so on. Virtually all of the recent ECS development, such as AVID, DIGIKON, DYSCO, and IAC, have utilized these features.

Distributed. The distributed ECS is designed to accommodate more than one computer tied together in a network for executing applications. In these environments, the user utilizes a less powerful computer for tasks such as constructing input or viewing output and utilizes the powerful computer for computationally intensive tasks. Examples of this type of ECS are ACTION and NICE.

3.1.4 Development

The development effort required for the various systems varies with motivation, purpose, and general resources available. The simpler and less comprehensive systems are typically designed and developed by personnel within the user group. The resulting ECS executable code,

without any installed application, may be quite small, for example, NASA's IDEAS contains about 1,000 lines of code. More comprehensive systems may be developed either in-house, typically by engineers as for AVID and NICE, or by contract to a software/engineer firm as for ANOPP, ISSYS, DIGIKON, DYSCO, and IAC. These systems vary in size depending on the capabilities provided. DYSCO consists of approximately 5,000 lines whereas ANOPP consists of approximately 31,000 lines.

3.2 Requirements

Although the diversity of users, applications, hardware, and development groups mentioned above necessarily lead to differing sets of requirements for Executive Control Systems, there are several major requirements that are prominent in the design of almost every system. These requirements are presented in this section in two categories, primary and secondary.

3.2.1 Primary Requirements

The primary requirements are those which have been satisfied to some extent by many of the surveyed systems and which prominently influence the overall character and architectural design of the individual systems. These requirements provide the structural format for presenting various design approaches in Section 4. They are catergorized into the following: central user interface,

operating system abstraction, execution control, and data movement.

Central User Interface

The system must provide a communication interface between the user and system so that the user may specify the tasks to be performed and have access to the resulting output. This interface should provide a central "top-level" control for executing the engineering design applications available and for utilizing other capabilities provided by the system. The manner in which the user communicates to the system and the learning time needed for effective usage are important factors influencing user acceptance, frequency of usage, and user productivity. The interface should exhibit uniformity. ease of learning, and ease of use.

Operating System Abstraction

In providing a complete engineering design environment, the functions and capabilities of the host operating system typically used in multiple application design scenarios should be abstracted and made available to the user through the ECS central interface. It is undesirable for a user to be in the midst of an ECS sequence of executions and have to terminate, returning to the operating system JCL level, in order to perform a JCL task(s) before continuing the ECS sequence. This break would require reconstruction of the ECS environment and

reduce the overall ECS benefits. Thus, the user should be able to perform JCL level functions without terminating execution of the ECS.

The potentials for abstraction depend upon the offerings of the host environment and their utility to the engineering design environment. Potentials common to all operating systems include JCL file maintenance capabilities such as attachment, detachment, cataloging, deletion, and utilities for creating and modifying file contents. Another potential is found in the distributed processing environment where JCL sequences direct both file transfers between different processors and execution of application on various processors. Since these potentials expand as new features are added to the host environment, the ECS interface component should be expandable to accommodate those improvements beneficial to the engineering design environment.

Execution Control

The system must allow the user to make multiple selections from the engineering design applications and other capabilities available and to execute them in arbitrary order. Four basic considerations are: uniform execution of applications, user defined execution sequence, automatic design, and restart.

Uniform Execution of Applications. The system should abstract execution of the application so that the user

provides only information about "what" is to be executed and is freed from the details of "how" the execution occurs. Details, such as the storage medium for the executable code and the JCL stream required for execution, should be internal to the ECS. From the user's view, this abstraction should be uniform and consistent across applications, with automatic generation of any required JCL streams.

User Defined Execution Sequence. The system should allow the user to both select applications/capabilities and define the sequence of their execution. Flexible control mechanisms which provide for repeating, skipping, and conditionally branching among sequences are desirable. Also desirable is the capability to save sequences and use them later as "building blocks" in various combinations.

Automatic Design. In multiple application scenarios the result from one application often will influence the input to another or will determine the subsequent execution sequence. For example, output value A from one application could be multiplied by 10 to produce the input value B for another application. A more complex example is an optimization problem where a sequence of applications may be executed repetitively, each time altering the input parameters as a function of the results obtained thus far. The ECS should provide for both automatic inspection of current results and subsequent

modifications to the input parameters and/or the execution sequence.

Restart. In some design scenarios, there are two types of restart capability which are useful: pause/restart and checkpoint/restart. In the former, an executing application may pause temporarily and return control to the higher level central user interface. The user may then perform other tasks, such as querying and updating data, before the application restarts execution at the point where the pause was invoked. In the latter, the computational environment consisting of data and perhaps control information is saved either during execution of an application or between successive application executions. Execution continues usually with several environments being saved or checkpointed. Subsequently, after the user has evaluated the results of the design sequence, the sequence can be restarted at the point where the checkpoint occurred using one of the saved environments. Typically, the user will change some of the data or control information during the restart process. Thus initial portions of a design sequence can be "salvaged" and need not be repeated.

Data Movement

There are two types of data movement required for engineering applications; data created by the user to be used as input to an application and data which is output

from one application to be used as input to another. The system must provide for this movement and, for a complete design environment, should also provide for creating, inspection, and modifying the data. Although a distinction is not always made, applications typically require two types of input data, low volume and high volume. Because of differences in size and usage, their requirements also differ.

Low Volume Data. A typical application requires a small number of input values for internal control purposes. Examples of internal usage include initialization conditions, termination conditions, and selectors for computational options. These may be supplied directly by the user or by a previously executed application. Often they will be a function of computational results obtained thus far. The ECS should provide a means for the user to create and modify low volume data and also move it between applications.

High Volume Data. An application may also require high volume input data such as results from aerospace wind tunnel tests or structural grid point values. This type of data may have been prepared externally to the ECS environment or may be the output of another application. The ECS should provide aids in creating special types of frequently used high volume data as well as mechanisms for

modifying high volume data and moving it between applications.

3.2.2 Secondary Requirements

The secondary requirements are those which have less influence on the overall character and architectural design approach of an ECS, but are nonetheless important from the software quality viewpoint and, ultimately, also to the user. These include considerations for user help aids, error recovery, library modification, and efficiency. Selected design approaches are mentioned in this section and are not elaborated further in this report.

User Help Aids

Help aids are particularly of benefit to the user of an on-line conversational ECS. These aids include tutorial information concerning the use of the system and informative error messages. These aids should be tailored to the target users of the system and should take into account individual differences in style, expertise, and experience. Help aids in the form of informative error messages are also of benefit to the user of a batch ECS.

Error Recovery

In an ECS environment, errors may have several origins: error in communicating with the central interface, invalid input to an application, run time error

(e.g., memory space exceeded), or a software error. System recovery from any error is desirable so that the user may make corrections and repeat the execution sequence or perform other tasks. There are tradeoffs between the benefits of recovery and software costs. Most of the current systems limit error detection to those originating from user input error. Conversational systems allow the user to reenter input data and batch systems necessarily terminate the executing application. For example, the batch system ANOPP allows any application to abnormally "quit" with the ECS performing housekeeping functions which would have been performed by the application upon normal termination. The DYSCO system extends error detection to include those with software origins by utilizing "safe programming" coding techniques.

Library Modification

As new applications become available for engineering design, the ECS should be designed to add them to the "library" of those available with minimal modifications to the ECS software. Similarly, the deletion of obsolete applications should be easily performed.

Efficiency

As for any software system, efficiency with respect to execution time and storage requirements are important factors. For simpler systems, these considerations are usually negligible. However, for larger systems offering

more complete engineering design environments, these factors become more important and should influence the architectural design approach taken.

3.3 Architectural Components

Viewing the ECS from a software architecture perspective, there are three major components. These are the executive component, the data management component, and the library component. In more complete systems, there may also be auxiliary components. Although there may be varying degrees of interaction between the components in a particular ECS, their purposes can be viewed as separable and distinct.

3.3.1 Executive Component

The executive component provides the central point of interface between the user and the system. Through this interface, the user selects the engineering applications to be executed and provides instructions as to their order of execution, input sources, and output destinations. The interface also provides access to all other ECS features and capabilities available to the user. The interface mode may be command language, menu, promptive message, or a combination.

3.3.2 Data Management Component

The data management component provides for storage of information and data for use by other system components or the user. It should provide mechanisms for creating and storing data, accessing data, and changing data. Three approaches to data management are prevalent in the surveyed systems. These include utilization of the host operating system's file manager, development of a specialized file partition manager, or incorporation of a generalized data base manager.

3.3.3 Library Component

The library component is conceptually the collection of engineering applications available to the user for execution. These include specific codes such as for structural analysis, circuit design, graphics, and specialized input data generations. From the user's view, these application appear to be a uniform part of the system. However, from the architectural view these applications could be embedded within the ECS software of could exist external to the ECS environment. Current systems support three architectural types of applications, which are termed here "independent applications", "interfaced applications", and "integrated applications", and are explained more fully in Section 4.

3.3.4 Auxiliary Components

An auxiliary component provides a collection of logically related functions or capabilities which significantly enhances the usefulness or performance of the ECS. An auxiliary component may be invisible to the user and provide for such enhancements as software efficiency, flexibility, and lower software development cost. The component may instead be visible to the user and provide for such enhancements as creation and management of engineering tables.

3.4 System Classification

There are several schemes by which the ECS could be classified, for example, by interface mode, by engineering domain(s) supported by the specific applications available in the library, by completeness of the engineering design environment, or by host hardware environment. Such classification schemes are used above and in Section 4. However, in recent years there has emerged a type of ECS which departs significantly from its ancestors and from the current mainstream of ECS development. The mainstream ECS is designed to be a "generalist" and thus does not embody knowledge about specific engineering domains. The emerging ECS is designed for a specific engineering domain and thus incorporates knowledge about that environment and its processes. In this paper the former is termed a

"General ECS" and the latter a "Domain ECS". Although only two of the surveyed systems, DYSCO and SUPER-CAD, fall into the Domain ECS class, they are sufficient to suggest the underlying characteristics of such systems.

3.4.1 General ECS

The General ECS is designed to support engineering design at-large. The library component is designed to accommodate any application which meets its architectural requirements and a specific engineering domain is supported by installing specific applications into the library. The executive component is designed to execute any application in the library without regard as to content, purpose, or contextual validity. The executive may be tailored with promptive messages and actions related to specific library applications, but the executive design only provides "slots" where code may be inserted and does not incorporate knowledge about specific applications in any systematic way. Similarly, the data management component is designed to accommodate various data structures without regard as to purpose and usage. The executive and data management components perform functions in a way analogous to an operating system. That is, functions are performed in arbitrary order as defined by the user without consideration of contextual meaning and purpose. The system thus provides a "tool box" of

features and applications and performs as a faithful servant taking instructions from its master, the user. The system performs few, if any, tasks automatically without direct instruction.

3.4.2 Domain ECS

The Domain ECS is designed to support a specific engineering design domain. It is similar to the General ECS is that is also provides a "tool box" of features and engineering applications, but it performs as a guide instead of as a servant. Knowledge about the design process(es), the steps typically followed, and the data required are incorporated into the system. With this knowledge, the system may assume more responsibility for validating the order of application executions, for locating and validating the required data, and for automatically performing various sub-tasks without explicit user instruction. The user still directs the design scenario, but the manner is less rigorous and less demanding of detail. The ECS expects a higher level instruction from the user to provide a general direction or goal. The system uses its internal map, which shows the steps required, to proceed along the path, pausing to acquire more information from the user as necessary.

In the Domain ECS, the role of the executive component is expanded beyond that of providing the central

user interface. It is the primary owner and user of the domain knowledge. Functions which are common to several applications, perhaps those appropriate at a certain point in the design process, may be elevated to the executive component level and be automatically performed under executive control. These may include executing special applications to generate input data, translating data into different formats, locating and validating required data, or performing an engineering algorithm. An example of the latter is found in DYSCO, the only implemented Domain ECS among those surveyed. In this system, the design process captured is that of defining the components and associated forces of a structure and performing a dynamic analysis utilizing a specific engineering coupling algorithm. At a specific point in the process, just before a user selected solution analysis method is applied, the executive automatically accesses the required data describing the structure, performs the coupling procedure, and passes the resulting data to the solution method chosen. The user need not request this step nor specify the input source or output destination; the executive extracts this information from the preceeding steps and the context.

The other components of a Domain ECS are similar to those of the General ECS. The data management component may reflect no domain knowledge. but may be used as a tool

for the executive and library components. The library component may become a partitioned collection of applications, as is the case with DYSCO, with each sub-library being appropriate at specific points in the design process.

4.0 ARCHITECTURAL DESIGN APPROACHES

This section presents an overview of design approaches used for the major components of the ECS. The discussion is organized around the major architectural components of an ECS as outlined in Section 3.3. For the executive component, the requirements outlined in Section 3.2 form a convenient basis for the presentation.

4.1 Executive Component

The executive component provides mechanisms through the central user interface for satisfying the ECS requirements. The user has access to the features available in the ECS through the interface and thus may perform tasks such as executing applications, specifying control flow, and manipulating and querying stored data.

4.1.1 Central User Interface

There are three basic modes by which the user may interface to the executive. These are the command language mode, the menu mode, and the promptive message mode. Although one mode is predominantly utilized in a particular ECS, the ECS may also utilize other modes to a lesser extent.

Command Language Mode

In the command language mode, the user provides instructions to the executive by specifying a series of

commands. A command is typically composed of a key word(s) identifying the function to be performed with an argument list providing additional information. The commands are analogous to sentences and, forming a language for user communication, they are often referred to as an Engineering Command Language (ECL) or an Engineering Design Language (EDL).

Each ECS utilizing this mode supports a unique language with its own repertoire of key words, syntax for argument lists, and style of usage. The number of commands for a language varies, but is typically in the range of 20 to 50. An ECS may assign a distinctive name to its language such as CLIP found in the NICE system, DIALOG in ODIN, and ESCORT in PRESTO. The batch systems, such as ANOPP and PRESTO, utilize only the command language mode. However, this mode is also used in systems developed for conversational environments, such as NICE, DIGIKON, and IAC.

The style and syntax of these languages exhibit influences from programming languages and mirror their evolution. Variable names and types and also algebraic and conditional operators are often similar to those of FORTRAN, as is prevalent use of the GO TO construct. Recently developed command languages incorporate features similar to those found in block oriented languages, such as ALGOL and PASCAL, and include DO WHILE and BEGIN...END.

The following excerpt from an ANOPP input sequence demonstrates some of these influences:

```
      ATTACH /A296/F1
      ATTACH /B001/F2
      PARAM ICOUNT = 0
   10 PARAM B = ICOUNT * 100
      EXECUTE JETLAG ( IN=F1, OUT=F2, INIT=B, DIFF=DIFF)
      PARAM ICOUNT = ICOUNT + 1
      IF ((ICOUNT .LT. 50) .AND. (DIFF .GT. .05)) GO TO 10
      CONTINUE
```

A strong JCL influence is seen in several of the command languages, such as in the SAVES, PICASSO, and ISSYS systems. The basis for this influence is the philosophy that modern operating systems satisfy most requirements and commands should therefore resemble the host JCL for more direct implementation [6]. This collection of "pseudo JCL" commands is then augmented with additional commands to overcome inadequacies. For example, the ISSYS language was patterned after the Control Data Network Operating System (NOS) JCL. In the following excerpt from an ISSYS input sequence, the CALLS are agumenting commands and the others are "pseudo" NOS commands for file manipulations:

```
GET, ALOAD/UN=FILE1.
GET, DRPROPT,DMASST.
CALL(ISSYS(XQ=SETPR))
CALL(ISSYS(XQ=TRSG))
EXIT.
REWIND,ISERR.
COPYSBF,ISERR,OUTPUT.
```

Menu Mode

In the menu mode, the user provides instructions to the executive by selecting an item from a menu of options. The menu mode is typically used in a hierarchial manner with selections from the main menu and subsequent sub-menus that eventually culminate in the desired function to be performed, such as executing a specific application or updating a data file. Simpler conversational systems, such as ACTION, PRIDE, and IDEAS, frequently utilize this mode as well as more complete conversational systems, such as AVID and ISAS. As an example, the main menu of AVID is:

```
1 - DATA BASE MENU
2 - DIGITIZE VEHICLE
3 - COMPUTE VEHICLE DATA AND COORDINATES
4 - PLOT VEHICLE
5 - VOLUMES AND AREAS
6 - SIMPLE HYPERSONICS
7 - ROCKET ENGINE SELECTION
8 - MASS PROPERTIES
RETURN TO TERMINATE AVID
```

An advantage of the menu mode is the natural provision for "slots" where code can be easily inserted for specialized messages to the user concerning the particular menu option chosen. ACTION utilizes this concept and issues "reminder" messages to the user on prerequisite actions which should have been taken, such as required data input and file attachments. If the user indicates that the actions have not been performed, the

executive returns to a higher level menu to allow their accomplishment.

Promptive Message Mode

In the promptive message mode, the user provides information in response to a descriptive message. This mode is mostly used in the conversational General ECS for secondary communication, although EASYCACD uses it for primary communication as well. However, it is effective as the primary communication mode for a Domain ECS, such as DYSCO, where the executive component guides the user through a design scenario and relies on previous responses and domain knowledge to determine the next step in the design process and needed information.

A frequent criticism of the promptive message mode is the tedium for the experienced user. Typically, the user must wait for each message before responding, although a series of answers can be anticipated. EASYCACD overcomes this tedium by allowing the user to input a series of anticipated answers in response to a single question. The system bypasses the associated promptive messages and issues prompts only as required. Thus, as the user gains experience and can anticipate required information, the promptive messages are automatically reduced.

4.1.2 Operating System Abstraction

The extent and nature of the mechanisms provided to abstract the host operating system varies among the systems. By definition, an ECS must abstract the execution of an engineering application, and typically the ECS will automatically construct any JCL stream required. Frequently provided abstractions allow the user to access and manipulate files prior to executing an application. These include file attachment, detachment, deletion, purge, and copy function. For example, the ANOPP command ATTACH /FILE10/A attaches the permanent file known to the operating system as FILE10 and assigns the local name A to be used within the ANOPP environment.

4.1.3 Execution Control

Mechanisms provided by the ECS for user control of the execution sequence allow for uniform execution, user defined sequences, automatic design, and restart.

Uniform Execution of Applications

In a particular ECS, the user selects an application for execution via either a command, a menu selection, or a response to a promptive message. Regardless of interface mode, the user supplies the "name" of the application and supporting information such as source of input data. The user may also have the option, as is the case in ISAS, of executing the application in batch or interactive mode.

There are four architectural approaches used in executing an application:

(1) The application may be a subprogram callable directly by the ECS code.

(2) The application may be an independent program placed into execution on the same processor as the executing ECS via a JCL stream. The ECS pauses and waits until execution is complete before continuing with the next user command.

(3) Same as (2) except the ECS does not pause but immediately continues executing with the next user command.

(4) The application may be an independent program placed into execution on a different processor in a distributed network via a JCL stream. The ECS immediately continues execution with the next user command, not waiting for completion of the application.

A system may use only one approach, e.g., in ANOPP all applications are integrated into the ECS as subprograms, or may use a combination, e.g., ACTION uses (3) and (4). Regardless of the approach taken, the ECS automatically generates any JCL stream required. Thus from the user viewpoint, the applications are selected and executed in a uniform manner.

User Defined Execution Sequence

In any ECS, the user has ultimate control over the execution sequence. However, the ease and flexibility of defining the sequence varies depending on the central interface mode. In the conversational menu and promptive message systems. the user iterates through the menus or promptive sequences, selecting the next task upon completion of the previous one. There is no mechanism provided by these systems (ordinarily) to automate or predefine a sequence. Due to their simplicity, they are easy to use and flexible for design scenarios which are not complex or which are not frequently repeated. However, for complex or repetitive scenarios the command languages provide more ease and flexibility in usage. In these systems, the user is provided a rich assortment of mechanisms for predefining sequences, repeating sequences, altering sequences, and saving sequences.

In the command language systems, the execution sequence is defined by a sequence of commands. In batch systems, the sequence of commands is completely predefined by the user and is input as a single unit to be executed from beginning to end. In the conversational systems, a single command may be supplied as a unit for execution before the next command is given or a sequence of commands may be given as a unit. For example, IAC provides a "BEGIN...END block construct for a sequence of commands to

be executed as a unit. In these systems, provision is made for temporarily or permanently saving a sequence of commands for later use in a "building block" fashion. For example, the ANOPP commands STARTCS and ENDCS enclose a sequence to be saved. The CALL command is used to bring a saved sequence into execution. Several sequences, each with a different identifier, can be saved on various files. When such a sequence is "called" into execution, parameter substitutions can be made in a manner similar to that found in assembler language macro expansion. The command "CALL SEQ1 (A = B, 10 = 20)" would access the saved sequence identified by SEQ1, inspect the contents, replace the variable name A with B and the integer 10 with 20 for all occurences, and then execute the resulting sequence of commands. With these commands, a user may build a personal library of command sequences to be used repeatedly in constructing complex scenarios. Other systems which provide for saving sequences are ODIN, IAC, and NICE.

A command language typically provides flexible control structures used in combination with command language variables. Commands which provide for control structuring include the generic forms "GO TO label", "CONTINUE", "IF condition THEN command(s) ELSE command(s)", "DO WHILE condition command(s)", and "DO UNTIL condition command(s)". Commands may also be

provided to assign values to command language variables, utilizing expressions involving constants and previously defined command language variables. In some systems, such as ANOPP, the command language variable are also visible to the engineering applications. The ANOPP application may query the existence and value of a variable and also may create or modify a variable. By utilizing the control structuring and variable assignment commands, the user can construct complex design scenarios involving repetition and conditional branching.

Automatic Design

Elements of automatic design are provided in those command language systems which support both flexible control structuring commands and command language variables which are visible to the engineering applications. By using these features, the user may construct a sequence of commands which utilizes the variable values set by a completed application to determine the next execution sequence. Also these results can be used to determine the proper setting of variables which will be used as input for subsequent applications. Combining the looping, branching, and assignment commands with the command variable visibility, the user can thus perform some basic elements of automatic design.

Restart

Neither of the two variations of the restart capability, pause and checkpoint, are supported in most of the systems surveyed. However, the pause/restart feature is found in IAC and the checkpoint/restart feature is found in ANOPP. IAC provides a utility subprogram which an engineering application may call, perhaps between design cycles, to temporarily return control to the executive command interface level. The user may then perform intermediate tasks such as querying and updating a data base before returning control to the pausing application. ANOPP provides for checkpoint/restart at the executive command interface level. The CKPNT command saves the current executive environment on a file and the RSTRT command may subsequently be used to reestablish the saved environment. This environment includes information such as names of attached files, command language variable names and values, temporary data created under executive control, and various control data. The surveyed systems do not provide for checkpoint/restart at the application level. However, a particular application may provide this capability to the user, independent of the ECS framework.

4.1.4 Data Movement

The typical engineering application requires two types of input data, low volume and high volume, and also

generates output data of the same two types. There are various design approaches utilized in the surveyed systems for creation, modification, storage, and flow of these data between applications.

Low Volume Data

Design approaches for creation and general movement of low volume data include the use of pre/postprocessors, command arguments, command language variables, and the data management component.

Pre/postprocessors. In conversational systems, a special preprocessor program may be executed prior to the desired application which prompts the user for the input data. The data is typically validated and stored on a file to be subsequently accessed and read by the application. The format is precisely as expected by the application. Usually, as in the PRIDE and ACTION systems, the user selects the preprocessor for execution as any other application. However, in the ISAS system, the executive automatically brings into execution the proper preprocessor for a particular application. A special postprocessor may take the output from one application and either allow the user to modify the data for reuse by the same application or translate it to another format for use by a different application.

Command Arguments. In some command language systems, such as IAC and DIGIKON, the command arguments are used to

pass values of certain input parameters to the application. The executive often passes these values to a special module written for the application for validation before the main code for the application begins execution.

Command Language Variables. In some command language systems, the command language variables are used to pass data between the executive command environment and the application. For example, the ANOPP system passes variable names via command arguments. The command "EXECUTE application (localname = actualname,...)" generates a name translation table. When the application code calls an executive utility module (e.g., ASKP, GETP, PUTP) to access or create a command language variable, a local name is used as an argument. The utility uses the translation table to retrieve the actual name by which the variable is stored. An early system, ODIN, utilized a novel mechanism for input data communication which made no demands on the application code. In the input data file for an application, the user would embed "cues" identifying the command language variable whose value should be used. Prior to executing the application, the executive would scan the input file and replace the cues with the current value of the variables. The application would then read the file in its normal way. A similar scheme was used in passing output data as command language variables but with slight code modification. The PRESTO

system employs the FORTRAN NAMELIST feature in an approach similar to that of ODIN.

Data Management Component. Low volume data may be created, modified, and moved between applications via the facilities provided by the ECS data management component, as discussed in Section 4.2 below.

High Volume Data

Design approaches for satisfying the management requirements of high volume data can be categorized into those providing for user creation of the data and those providing for flow of data between applications.

Data Creation. Three basic design approaches allow user creation of high volume data and include the use of data generators, the ECS data management component, and auxiliary components.

Data generators are special types of preprocessors which utilize a small amount of input data, either directly from the user or from another source, to generate a larger amount of data which will be used as input to an application. For example, generators are employed in structural analysis for generating nodal values for a large structural grid from a "brief" grid description. Generators are increasingly being introduced into interactive computing environments. The facilities of the data management component may be used to create data files in specific format required by an application.

Auxiliary components may be incorporated into an ECS to provide this capability for specific types of data. For example, the ANOPP Table Manager provides commands for the creation and modification of engineering tables which are subsequently accessed by applications via special utility modules.

Data Flow. There are three design approaches which provide for high volume data flow between applications: direct flow, translated flow, and abstracted flow. Wilhite [7] uses the alternative terms close-coupled interface, loose-coupled interface, and loose-coupled integration respectively.

Direct flow of data requires that an application create the data in precisely the format required as input by a subsequent application. This is the method traditionally used in CAD/CAE and is prevalent in many of the surveyed systems. As the library expands with many applications which interact, this approach yields a increasingly complex administrative task of insuring correct correspondence of formats within the application codes. Among the systems utilizing this approach are RAVES and ANOPP.

Translated flow of data requires intermediate processing between execution of the application involved. Pre/post processors are utilized to translate the data which is created as output from one application into the

format required for input to a second application. Typically there is a special processor developed for each pair of interacting applications. The PRIDE system employs the relational data base manager RIM to provide an intermediary data base for storing output data which has been translated by a postprocessor into a generalized format. This data may subsequently be accessed and translate by a preprocessor into the precise format required by a second application.

In abstracted data flow the applications which create and access the data are not concerned with format, storage form, or in what order the data was created. A centralized data base is used for data communication between various applications. These concerns are typically the responsibility of a data base manager through which all creation/access is performed. The application creates/accesses data by specifying some type of indentification which the data base manager internally correlates to a set of predefined formats to accomplish the desired function. Since ordering is not of concern, the application may access only those data items which are required. Abstracted data flow is not apparent in the systems surveyed except in the DYSCO system which obtains the abstraction through combination of the executive and data management components. Much of the input data for DYSCO applications is obtained directly under executive

control, utilizing tables which contain the requirements for the various applications. A table is prepared for each application by the software developer and contains information about each data item which may be required, such as data type, existence criteria which must be met before the user is asked to supply input, and range constraints for input validation. The executive controls the input consistently across applications, directs the storage process, provides an editing capability, and provides access by the application via special utilities. The applications access data by name only, thus obtaining abstracted data flow.

4.2 Data Management Component

The data management component provides for storage and access of data within the ECS environment. The primary focus is on management of engineering data for use by the various applications. Design approaches include the use of a file systems, a file partition manager, or a data base manager.

4.2.1 File System

In the file system approach. files are viewed as indivisible and uninterpreted objects which are to be uniformly treated without regard as to content. Other than those file capabilities provided through central interface abstraction of the host operating system (e.g.,

file attachment), there are not special provisions for creating, modifying, and accessing file contents. The applications utilize directly the host file system utilities, such as the Control Data Cyber Record Manager, or utilize the programming language Input/Output features, such as FORTRAN READ/WRITE statements. The engineer creates and manages the files through the host file system. Uniformity in file usage may be obtained through convention. For example, in the DIGIKON system fixed file units are consistently used for the various types of data such as modeling data, plotting data, or "scratch" data.

4.2.2 File Partition Manager

In the file partitioning approach. each file is divided into a collection of logically independent sub-files or partitions. Each partition is analogous to a separate file in the file system approach and thus may be created, modified, and accessed as a separate unit. The file partition manager provides utilities to the applications for storing/retrieving data and also provides creation/manipulation capabilities to the engineer user through the central interface. A partition is typically composed of ordered records, the contents of which are also ordered, with the creation and access of data being rigidly coordinated as in the file system approach. Partitioning allows various independent data to be stored

on one file(s) and thus reduces the overall number of files required for a design scenario.

An example of a file partition manager is the ANOPP Member Manager (MM). A partition is called a "data member" and is uniquely identified by a combination of filename and member name. A member is composed of variable length records which may be composed of distinct formatted elements (e.g., integer, real array, string) or unformatted "words". In either case, the MM abstracts the internal format or representation of the data such that access is by position within a record. Twelve utilities are provided to the application for data manipulation, such as MMPUTW for writing a specified number of words to a member or MMGETE for reading elements from a member. Manipulation also may be performed by the user through twelve provided commands, such as MEMLIST for listing a specified member or UPDATE for updating a member down to the element level of detail. Other systems which support a file partition manager are CASE, IAC, NICE and DYSCO.

4.2.3 Data Base Manager

In the data base manager approach, the file view remains prevalent but the creation and access of data need not be as rigidly coordinated. Data items are created/ accessed by "name" without regard to order of creation and storage form. Thus items created in a certain order may

be accessed in random fashion. Applications can share information without an agreed on common format and can view data without concern for other data not needed. Various utilities are provided to applications for creating and accessing data, and special query commands are provided the user for the same functions. The data base manager allows for evolving semantic content and storage form without modifying existing applications and user queries. This type of manager is typically developed to support engineering-at-large independent of the ECS environment and is incorporated into the ECS as an independent component.

Although several systems incorporate a data base manager, there is a lack of uniformity in usage and there appears to be no single manager which is prevalent. The relational data base manager RIM is utilized by PRIDE for translating data between applications. RIM is also incorporated into ISAS as a utility for applications which may require it. AVID incorporates the ARIS relational data base manager.

4.3 Library Component

The library is composed of applications available for execution through the central executive interface. It is conceptually viewed as a repository where new applications may be added and obsolete applications may be deleted.

From the architectural view of the relationship between an application and the remaining components of an ECS, there are three design approaches which may be used, termed here the "independent application", the "interfaced application", and the "integrated application".

4.3.1 Independent Application

The independent application, although accessible through the executive central interface, is capable of being executed independently and outside of the ECS environment through an appropriate JCL stream. This type of application is developed independently of the ECS environment and does not utilize any of the utilities which may be uniquely provided by the ECS. When such an application is selected within the ECS environment, the executive provides for generation of the JCL stream required for its execution. Examples of independent applications include the structural analysis NASTRAN and SPAR programs. Menu driven systems, such as ISAS, ACTION, and PRIDE, as well as some command driven systems such as IAC, support this type of application.

4.3.2 Interfaced Application

The interfaced application is similar to the independent application in that it typically is developed independently of the ECS and does not require any of the unique ECS utilities for execution. However, the

application is incorporated into the ECS software as a subprogram directly "callable" for execution. In some systems, the application code may require minor modification to interface the input/output properly to the executive.

4.3.3 Integrated Application

The integrated application is developed for execution within a specific ECS and utilizes the unique utilities provided by the system. These applications are subprograms within the ECS and could not be executed externally to the supporting ECS software. For example, ANOPP was designed to support integrated applications and provides a rich assortment of utilities for dynamic storage usage, data movement through command variables, retrieval/interpolation of engineering tables, and file partition manager functions. Other systems which were designed to support intergrated applications include DYSCO and IAC.

4.4 Auxiliary Components

Auxiliary components are not commonly incorporated into an ECS; however, their occurrence is found in some systems which are designed to accommodate integrated applications. Two prominent examples are the inclusion of a Dynamic Storage Manager, within IAC and ANOPP, and a

Table Manager, within ANOPP. The Dynamic Storage Manager component is provided to overcome the FORTRAN restriction of static storage allocation. A large block of storage is statically allocated for use by applications or other system components with subsequent expansion or release. The Table Manager component allows for creation of engineering tables with subsequent query/interpolation capabilities. Such auxiliary components are intended to increase overall system performance and reduce the total coding requirements.

5.0 HISTORICAL PERSPECTIVE

The genesis of the ECS in the engineering design environment is found in the 1960's when individuals sought their own personal automated procedures for executing a sequence of independent programs. By the end of the decade, various groups in industry and government recognized the need to provide a framework for incorporating engineering programs into comprehensive systems for improved accessibility, efficiency, and ease of use. Subsequent efforts have spawned a variety of systems for diverse engineering disciplines and environments. The evolution from early efforts to the current state-of-the-art ECS can be broadly divided into three stages of developments: embryonic, batch environment, and conversational environment.

5.1 Embryonic Stage (1960's)

A primary feature characterizing CAD/CAE tools has historically been the close operating relationship between the user and the computer [1]. In the early 1960's, the computer was still viewed as a large and efficient calculator which could be programmed to carry out iterative, trial and error sequences of calculations. In many environments, only one user at a time could be served by the machines which were often operated directly by the

users. As their size and power increased by orders of magnitude, computer systems had to be operated and managed by a specialist staff for utilization efficiency. This decade saw the emergence of time-sharing systems which allowed many programs to be run simultaneously (as far as the user could tell) and an increase in accessibility through interactive or remote job entry terminals. As the sophistication of the computer environment grew, so did the range and complexity of CAD/CAE tools. The role of the computer was expanded beyond that of a calculator to include automated aids for solving very large problems. For example, solution of a large scale optimization problem could involve 12,000 constraints and variables and yield in excess of 100,000 lines of output [8]. To perform an engineering design task, it was often necessary to execute a sequence of large programs each requiring individual JCL streams and unique input data formats (which frequently were incompatible). In this atmosphere of change and increasingly complexity, the engineer frequently came to view the computer as a rather remote facility surrounded by jargon speaking experts.

To reduce the efforts and tedium required to execute sequences of programs, individual engineers sought their own personal solutions. Template JCL streams for multiple executions were constructed and saved, to be later tailored for specific instances of usage. These templates

grew in number and complexity and were shared with colleagues to form small but personal libraries of JCL streams. These libraries were embryonic attempts to centralize and automate the process of selecting desired programs, ordering their execution, and specifying the source/destination of input/output data.

As a result of steady growth in digital computer applications, by the end of the decade nearly all engineering disciplines were aided to some extent by CAD/CAE tools. Difficulties in using a sequence of tools became a significant problem, recognized not only at the individual engineer level but also at higher organizational and inter-disciplinary levels. A typical large company engineering environment in the late 60's is that described for the Product Engineering Department of Grumman Aerospace Corporation in [9]:

> "...Each group had some collection of applicable computer programs and manual procedures, but integration of analysis efforts relative to consistent or planned data formats was hard to find. In addition, computer programs had not been structured to perform a specified flow of analyses needed to perform the overall engineering analysis functions in a disciplined or organized manner...A very large number of engineering man hours were being wasted, by today's standards, in data calculation, acquisition and transmittal, and in manually manipulating data from one format to another, from one axis system to another, etc."

A milestone in laying the foundation for the future ECS was born out of this atmosphere and became known as

IDEAS. Within Grumman, it was recognized that an organized approach was essential to give greater confidence that structural drawings would reach manufacturing according to a schedule and that changes in the primary structure would not occur after release of the drawings. Developed during the 1967-1968 time period within Grumman, IDEAS was a collection of computer programs, each run in batch mode, and each with well defined procedures and sequences for their use by various engineering groups. The programs were those which were required for aerospace vehicle sizing and which had a direct effect on schedules for the release of structural drawings to manufacturing areas. To monitor the progress of the various groups using the programs for a design effort and to insure inter-group interface requirements were satisfied, an IDEAS room was utilized to act as a "command and control post". The lead personnel from the groups would meet each day in the IDEAS room to chart activity and progress against the master schedule. Used for the first time on the F-14 aircraft, the IDEAS approach reduced by eighteen months the time period which would have been required with pre-IDEAS procedures. By 1973, IDEAS encompassed seventy six (76) engineering programs used by the inter-disciplinary groups involved in design.

IDEAS is representative of this time period in two respects: the concern for more efficient utilization of CAD/CAE tools and the organizational concepts subsequently defined. Defining the frequently used sequences of tools and the interactions of their data requirements laid the foundations for development of the centralized framework of the ECS.

5.2 Batch Environment Stage (1970's)

The logical next step in development was the centralization of the CAD/CAE tools into one engineering system. The RAVES system, initiated in 1973 by Grumman, was based on the IDEAS concepts and was one of the most comprehensive systems of this era. Intended to embody major analysis efforts from all aerospace vehicle design disciplines, tools or applications were categorized as preliminary design, point design (for the proposal effort), and detail design. Each application had an "exec" which prompted the user for information regarding the source/destination of input/output data. Data flow between applications was direct, that is, each program generated data in the precise format required for a subsequent application. The development of ISAS by the Boeing company paralled RAVES and was intended to reduce flow time in preliminary structural design analysis. Both of these early systems employed elementary executive

components which performed "routing" functions for the selected applications and utilized a file system for the data management component.

Although interactive computer environments were increasing during the 1970's, the typical engineering application was executed in batch or non-conversational mode. Corresponding to this typical usage, the ECS developed during this decade was predominately designed for batch execution and employed the command language mode for the executive central interface. As previously mentioned, RAVES was an exception and employed the promptive message mode to a limited extent. The command languages of early systems, such as ATLAS, were limited in scope and generally provided only for the selection and execution of applications and the designation of source/destination of the input/output data. To augment the language, however, sequences of FORTRAN statements including COMMON block declarations were allowable inclusions.

By the middle to latter part of the decade, the command languages had become more complex and sufficiently powerful to eliminate the need for FORTRAN statement augmentation. Flexible control structures, allowing for looping and conditional branching, became prevalent, as well as methods for saving command sequences for later use as "building blocks". The systems typically utilized the

file system approach to data management, provided low volume data movement via command variables, and supported independent and interfaced applications. Auxiliary components were not typically utilized by systems of this time period, which includes DIGIKON, ODIN, and PRESTO.

ANOPP was developed during this time period as one of the more advanced command languages systems and departed from the others in several respects. A file partition manager was utilized as a data management component, with creation, modification, and general access capabilities. Auxiliary components were utilized for dynamic storage management and engineering table management. ANOPP supported applications which were highly integrated. The system continues to be actively supported and used by several groups. It remains one of the more sophisticated systems.

In addition to the command languages influenced by programming languages, a separate "thread of evolution" is seen in systems which employed command languages based on the JCL of the underlying operating system. An early system of this type was SAVES, developed in the early 70's to automate the use of programs for preliminary structural design. This system was an extension of the individual innovations of the 60's where JCL template streams were saved for later instantiation. The PICASSO and ISSYS systems were developed in the late 70's and incorporated a

command language based on the Control Data Network Operating System (NOS).

5.3 Conversational Environment Stage (1980's)

Beginning in the late 70's and continuing into the 80's, a surge of interest in conversational computing emerged which has exerted a strong influence on development of the ECS. Applications which heretofore operated solely in a batch environment are gradually being modified to incorporate conversational characteristics. Conversational preprocessors and data generators are also being developed as "front-ends" for large, computationally intensive applications [10]. In parallel with this interest, and perhaps as a result of it, there has been a significant increase in the number of Executive Control Systems being developed throughout government and industry. These systems invariably offer a conversational central interface which incorporates the menu, command, or promptive message mode.

The majority of these systems are less sophisticated than the command language systems of the previous decade and perform basic "routing" functions through a menu mode central interface. They most often support independent applications and thus automatically generate the JCL stream required for execution. The data management component is typically a file system, with a data base

manager sometimes made available through the library component for ad hoc usage. These systems are designed for a relatively small, single discipline, user group with a limited number of applications anticipated. Systems of this type include ACTION, CASE, IDEAS (NASA), LSSIAP, and EASYCACD. PRIDE is also of this general type, but, as a research prototype, it places primary emphasis on utilizing the RIM data base manager as a means of translating data between applications.

Although much of the current activity is devoted to development of the simpler menu mode system, there is also continuing development of the more sophisticated ECS. Earlier systems, such as RAVES, ISAS and DIGIKON, have undergone modification to accommodate the newly developed conversational applications, preprocessors, and data generators. A newly developed command language system, IAC, maintains the flexible control structures and capability to save command sequences typical of the batch ECS while offering a conversational interface. IAC also offers an auxiliary dynamic storage component to aid integrated applications. The NICE system provides both a batch and conversational command interface mode. Recently developed systems of this category, such as IAC, CASE, AVID, NICE, and DYSCO, typically incorporate a file partition manager for data management. However, there is interest in utilizing a data base manager, such as in

PRIDE and AVID. The command language mode is most frequently chosen for the central interface in the more sophisticated systems; however AVID utilized both the menu and the promptive message mode.

A departure from the traditional General ECS has also emerged in the form of the Domain ECS utilizing the promptive message mode of interface. In this ECS the executive component captures the engineering process and incorporates knowledge about the data structures and applications to effectively guide the user through the design scenario. One such system, DYSCO, has been implemented and another, SUPER-CAD, has been proposed.

If the recent surge in ECS development is a reliable predictor for future efforts, it appears that the remaining years of this decade may yield additional systems, both for single discipline and inter-disciplinary design communities. One such major project is the ongoing design effort within the U.S. Army to develop a comprehensive system, 2GCHAS, for inter-disciplinary design and analysis of helicopters.

6.0 EVALUATION

Throughout the evolution and development of the ECS, the engineering communities have utilized diverse and innovative design approaches in satisfying the requirements and overall goals of the systems. Benefits have been reaped and have led to a growing recognition of their potential benefit to the engineering communities. Accompanying the relatively recent surge of interest in conversational computing, there has been a significant increase in ECS development efforts. It thus seems timely and appropriate to view the current state-of-the-art with an eye on the future in an attempt to uncover areas where further exploitation can be accomplished and where new advances in computer science technology can be employed. With this perspective, there are four major areas which invite scrutiny and consideration for further investigations: inter-disciplinary usage, standardization, knowledge utilization, and computer science technology.

6.1 Inter-Disciplinary Usage

Early in the history of the ECS there was recognition of the need for comprehensive systems which could provide an engineering design environment and which could spand the boundaries of the various disciplines involved. This is seen in the organizational concepts of Grumman's IDEAS

and in the original goals of the IPAD project. In some sense, this has been accomplished by the current General ECS which is designed for engineering-at-large and thus can accommodate any engineering application and its associated data structures. At least two of the systems, ISAS and RAVES, are in practice used across various disciplines. The 2GCHAS system is being designed for interdisciplinary usage. Although the practicality of inter-disciplinary use of an ECS has been demonstrated, there are limitations in the current data movement methods which inhibit realization of broader benefits. Each discipline views the product being designed from distinct vantage points and thus each requires input data and yields resulting output data which is distinct and generally incompatible with that of the other disciplines. Hartung [11] views this process as "Islands of Automation" which will eventually bump into other Islands causing yet another interface problem. For an inter-disciplinary design environment, automated data movement with compatible interface between the disciplines is required. As yet, however, compatibility of data has not been fully realized within a discipline and is obtained only by ad hoc methods such as special pre/postprocessors. Research efforts, such as found in the PRIDE and AVID systems, suggest that a centralized and standardized data base of design data from which the various applications may

retrieve and deposit data may be a key to automated data movement within and across the various disciplines.

6.2 Standardization

Only a few of the current systems reach a user community beyond that of the original development group although, by nature, the General ECS can accommodate the needs of a variety of groups and disciplines. Those which have include AVID, ANOPP, DYSCO, and ISAS. While it is true that not all of the systems offer a complete "tool box" of features, there is redundancy and overlap of functional capabilities among the simpler systems and among the more sophisticated systems. Many offer the same basic functions but differ only in the design strategies chosen and the implementation style.

This lack of standardization has implications for future development and utilization potential. Much time and effort is spent designing, developing, and maintaining functionally redundant systems. As new systems are introduced and as personnel transfer to different environments with distinct systems, additional learning time is required to gain new expertise. This is particularly true for the sophisticated command language systems.

As development of new systems continues, portability issues in three main areas must be addressed. The first

concerns the applications. Each application code which has been developed outside of any ECS environment must be modified to interface with each ECS within which it is to be used. Duplicated effort will thus be expended for popularly used engineering codes. Integrated applications developed within a specific ECS environment to take advantage of the available utilities can rarely be modified for a different ECS without extensive recoding. Standardization would reduce these problems and allow a higher level of application portability.

The second area is data management. Among the various file partition manager and data base manager components being developed and incorporated into the systems, there is no single one (or even few) which has gained common acceptance. Among the file partition managers there is a high level of functional redundancy, yet each has been developed uniquely for a particular ECS. Within the few systems which incorporate a data base manager, its usage is not coordinated and controlled to the level required to parallel its usage in commercial environments. Although engineering data is generally acknowledged to be more diversified and thus the handling is more complex than for commercial data, a standardized data management component would enhance portability and reduce the number of special pre/post processors currently being required for data flow.

The third area is the host computer and operating system. It has long been acknowledged that if significant effort is expended in developing a software system then it is desirable for the software to be transportable to other computers and/or operating systems with minimum effort. As a result of this view, the functional requirements and the architectural design approach of the ECS are frequently restricted in order to accommodate portability. Portability is obtained at the expense of not taking full advantage of hardware and operating system features. For example, characters may be packed four per word to accommodate IBM, DEC and Control Data installation; but there could be a serious degrading of efficiency on a Control Data computer which allowed a packing of ten characters per word. Another example is found in the associated variable feature available to a FORTRAN application running under the DEC TOPS operating system and which provides the current number of records written to any specific file. In the DYSCO ECS, utilization of this feature could have significantly reduced the coding requirements for a certain group of modules but was restricted due to a portability requirement. It is not reasonable to expect that a standardized ECS, which must accommodate a variety of operating systems, would overcome these problems. However, if a standard operating system was available on the various computers found in

engineering environments, a standardized ECS could be developed to exploit those features provided. There is current interest in standardization of the UNIX operating system. If this effort is succesful and if UNIX becomes readily available for various scientific computers, it may provide a powerful framework for ECS standardization.

Hartung [11] has suggested that perhaps there is a growing market for third party software vendors who might bring a universally usable system of this type to the marketplace. In the current atmosphere of proliferation, these issues and the feasibility for standardization should be considered by the various engineering user groups.

6.3 Knowledge Utilization

There are broadly three types of engineering knowledge which can currently be exploited in the ECS design environment. The first is mathematical knowledge which is independent of the human factor - individual differences in expertise and experience found among design engineers - and which has traditionally been formalizable as algorithms. The second is domain knowledge, which is also independent of the human factor and which has traditionally not been formalized. The third is expert knowledge which is dependent upon the human factor and which has traditionally defied formalization.

Mathematical knowledge has been and continues to be exploited in the ECS; however, the key to gaining significantly more benefit from the future ECS is exploitation of the second and third types of knowledge.

Mathematical Knowledge. The first type of knowledge is the "backbone" of engineering design and has been the basis for most engineering software since the early days of computing. By its sheer nature, this knowledge is easily represented by equations or other mathematical algorithms, and its correctness, accuracy, and appropriateness are based on theoretical considerations rather than being dependent on the individual engineer user. Virtually all of the mainstream CAD/CAE applications are implementations of various mathematical algorithms; as new or improved algorithms are formulated, new or improved applications follow soon thereafter. Efficient execution of these algorithms has been a primary focal point for past CAD/CAE research efforts in the quest not only for improved algorithms and software implementation methodologies, but also for improved hardware architectures, such as those supporting parallel designs. Exploitation of this type of knowledge has been and should continue to be of critical importance to CAD/ CAE systems, in particular, the ECS.

Domain Knowledge. Domain knowledge deals with how the mathematical algorithms are used in a particular engineering domain to obtain a design solution(s). In the context of the ECS, domain knowledge is knowledge about using various engineering applications in an efficient, effective, and technically appropriate way. While a great deal of engineering judgement is required in the overall use of these tools (e.g. formulating the approach, selecting specific applications, determining parameter value settings) a large part of the effective usage of these tools is not dependent upon the human factor and is the same regardless of the specific engineer user. This includes knowing such things as what is a meaningful order of execution, how to prepare or locate input data, what intermediate processing is needed, and what is to be done with output data. As a simple but frequently encountered example, consider a two-application scenario where the pattern of execution is always the same and is such that the output from one application is used as input for the other, but the output must first be reformatted by a special processor. The knowledge required to perform this pattern of execution is generically the same for any user and varies only with usage context, yet in today's typical ECS environment the burden of supplying the (often redundant) details and insuring their validity is a user responsibility. The Martin Marietta study [3] cited

previously in Section 1, found over 100 situations which could arise in a CACD scenario and which would stop progress unless the right information was known by the user. The right information, more often than not, is precisely that domain knowledge required to use the mathematical applications correctly in an integrated fashion. Although this information has generally not been centralized and organized with automation in mind, it is available and offers no hurdle to formalization. Capturing this type of knowledge would free the user from knowing the details associated with integrated usage of applications and would significantly increase the power of the ECS.

The General ECS, which has dominated development efforts, is by nature limited in capturing domain knowledge. With the emphasis on generality, the General ECS usually provides little support for capturing domain assumptions such as the role of an application, what type of input is required and its location, or what intermediary steps are implicit to a particular design process. The basic goal of the General ECS is to service engineering domains "at-large" and thus is in conflict with the goal to exploit knowledge about a specific domain. As a result, the General ECS is not a suitable vehicle for future exploitation of domain knowledge.

The Domain ECS is by nature a suitable vehicle for capturing domain knowledge. The goal of such an ECS is to capture the design process of a particular engineering domain and exploit knowledge about the applications available, such as their role and data requirements. The system performs as a guide and requires less detail in the instructions to accomplish a design task. It makes assumptions about the domain, such as the data structures required and the role of each application, and is capable of performing many functions automatically based on contextual information without user instruction. The Domain ECS is a recent development, with DYSCO being the only implemented system (among those surveyed) which exhibits these characteristics. DYSCO is sufficiently mature, however, to serve as a prelminary model for future development efforts.

Expert Knowledge. Expert knowledge is that knowledge about a domain which resides only within the individual expert engineer and which has in the past defied formalization. It is manifested at those points in the design process when the engineer inspects the results from a CAD/CAE application, assimilates and integrates this data with his/her internally held expertise, and, using engineering judgment, decides what is to be the next step in the design scenario. The way in which this sequence occurs and the knowledge which is utilized appears to be

personalized and varies among individual engineers. Although there are individual differences, a given expert is generally consistent and repetitively applies personal methodologies and expertise in making design decisions. Utilization of this type of engineering knowledge is not currently incorporated into the ECS environment.

Recent developments in "expert system" technology [12], emerging from the Artificial Intelligence (AI) branch of computer science, offer potential for capturing the third type of engineering knowledge. Limited to academic laboratories in the 1970's, this technology is now becoming cost-effective and is beginning to enter into commercial applications. The demonstrated utility of the expert system has been in capturing the judgment and decision process of the individual experts. Although existing expert systems are for other domains, these domains bear many similarities to the engineering judgement and decision process described above. It is reasonable to anticipate that if properly applied, expert system technology could aid in reducing the demands for human resources and expertise in the engineering design process.

The ECS offers a potential framework for embedding expert system technology. From a cursory review of current expert technology, two possible approaches appear. An expert system could be developed for the General or

Domain ECS and selected for execution as any other engineering application. For example, in optimization studies, where several or many different parameters may be varied in search of an "optimal" solution, an expert system could perhaps be developed to select initial parameter settings and to subsequently evaluate their effect and "decide" which parameter changes offer most promise in finding the best solution. A second approach is found in the underlying philosophy of the Domain ECS, which as discussed above, emerged from the view that in limiting the computerized environment to a specific design process, assumptions and knowledge about that process could be incorporated in the executive to yield more automatic deductive mechanisms. The aims of the Domain ECS and the expert system are similar; both are concerned with capturing knowledge about usage of the more basic mathematical knowledge of a domain. Interestingly, the jargons are also similar and overlapping. Because of these basic similarities, the Domain ECS offers a more natural habitat and greater potential for embedding expert system technology, and thus exploiting expert knowledge, in the engineering design environment.

6.4 Computer Science Technology Transfer

Since the early days of computing, the engineer has maintained intimacy with the processes involved in using

the computer as an aid to design. The engineer has been accustomed to participating in the software life cycle by formulating the requirements and specifications for an application. Frequently the engineer also performs the design, coding, testing, and even the maintenance phases of the cycle. During the early stages of ECS development, it was a natural extension of these experiences for the engineer to take the primary role in determining the requirements, specifications, and design of these systems. Based on knowledge and expertise gained as a result of using operating systems, programming languages, and other tools, the engineer-designer has led the evolution of the general ECS up until the present.

However, during the time frame of ECS evolution the field of computer science has rapidly expanded and matured. Advances have been made in areas such as operating systems, programming languages, hardware architecture, networks, software engineering, data bases, and artifical intelligence to produce increasingly powerful and sophisticated tools. In many cases the problems found in the design of an ECS are similar to problems in the more general design of operating systems, programming languages, and database management systems. Through research in computer science, elegant, efficient, and general solutions to many of these design problems have been developed. For example, the UNIX operating

system [13] provides a hierarchical file system, programmable command languages (the "shell"), and clean interface for the insertion and joining of new applications (the concepts of "filters" and "pipes", among others). The Ada programming language [14] provides methods for building application "packages" that can be joined via an ECS into a more reliable, portable, and maintainable software system. Similarly, many recent advances in database systems would be of use in the design of an ECS [15].

Computer science thus provides a wide spectrum of new design concepts that might be used by future ECS designers and that also may provide a software base on which future ECS designs may be implemented with greater ease, reliability, and elegance. An example is found in the UNIX hierarchical file system. A significant portion of the code for current file partition managers is devoted to implementing basic functions of a "one-level" hierarchical file system. These functions satisfy the requirement to minimize the number of physical files required for data storage while also allowing selective access to groups of data which are logically independent. The hierarchical file system supported by UNIX could be appealed to directly for implementing this ECS data management requirement. Not only would the coding requirements be reduced but the testing requirements also, since the

reliability of the UNIX file system has been established through extensive past usage. Also, the UNIX file system provides additional capabilities which are not provided by today's ECS but would undoubtedly be useful. Examples include "multi-level" hierarchical organizations and "links" which allow simultaneous data access by several users. Having benefitted from a great deal of development effort and from the knowledge of designers with diffuse computer science expertise and experience, the design of the UNIX file system is more elegant than the designs of current file partition managers.

Too few of the advances in computer science technology, such as those found in UNIX, have found their way into the design of recent executive control systems. The engineering community should seek partnerships with the computer science community to make greater use of these advances in any new ECS design. In the past too much of the design effort for an ECS has gone into solution of the low-level design problems common to most large software systems (data base management, data movement between applications, user interface) and too little into the unique problems associated with engineering design, such as the utilization of domain knowledge mentioned above.

One of the important conclusions resulting from this survey of the variety of ECS system architectures is that

many of the design problems in an ECS are common problems that arise in any large software system architecture. By utilizing solutions to these common problems devleoped by the computer science research community, the future ECS designer may concentrate his efforts much more fully on the interesting and unique high-level problems of the underlying engineering design applications area.

User communities other than engineering have also utilized the ECS concept to integrate a variety of applications. Decision support systems, in particular, as well as programming design environments, contain many useful concepts relevant to ECS design. It would be beneficial to the future engineering ECS to capture the expertise and experiences with these related systems.

The engineering ECS provides a fertile field for academic computer science research. It is already accepted as a proper vehicle for research in academic engineering environments. Examples include SUPER-CAD which provided the basis for a Master's thesis and AVID which has provided the basis for several doctoral dissertations. Computer science researchers have been active in the evolution and development of integrated systems in the other domains but up until the present have not focused their attention on engineering design. The requirements for the engineering design environment include many found in other domains, but engineering has

complexitites and diversities which post unique problems for which efficient solutions are not readily known. Greater attention by the computer science community to the unique problems posed by the engineering ECS would lead to benefits for both engineering and computer science.

7.0 CONCLUSION

The Executive Control System was introduced into the engineering design environment to provide a framework for unifying various engineering applications into a comprehensive system. The benefits include improved accessibility, efficiency, and ease of use. Among the surveyed systems, there is wide variance with respect to the executive, data management, and library components as well as auxiliary components. The components are characterized by the architectural design approaches employed. The central user interface is the most prominent feature of the executive and may be designed to utilize the command language, menu, or promptive message mode. The data management component is designed to utilize a file system, a file partition manager, or a data base manager for creation and general access of data. The library may contain independent, interfaced, or integrated applications. Auxiliary components are not often found in current systems, but those which have been utilized include capabilities for dynamic storage management and engineering table management. Each system may be classified as either a General ECS, designed for engineering-at-large, or a Domain ECS, designed for a specific engineering discipline. Until the 1980's, when there emerged a surge of interest in conversational

computing, ECS development focused on use of command languages for batch environments. More recently we have seen an increase in the number of systems being developed. A significant portion of these new systems utilize the menu mode and offer an elementary set of capabilities and features. Development of more sophisticated systems continues, with predominant use of command languages and file partition managers. Areas which offer potential for future exploitation and increased benefit to the engineering user communities include inter-disciplinary usage, standardization, knowledge utilization, and computer science technology transfer.

8.0 SYSTEM SNAPSHOTS

This section contains a brief description of each of the twenty-four Executive Control Systems included in the survey, ordered alphabetically by acronym. The references to source documentation for each are included.

(2GCHAS) Second Generation Comprehensive Helicopter Analysis System [16]

2GCHAS is an ongoing design effort within the AVRADCOM Directorate of the U.S. Army. The goal of the system is to provide a comprehensive design environment for inter-disciplinary analysis of helicopters.

(ACTION) [17]

ACTION was developed in 1981 by NASA Langley Research Center for a small research group environment. Designed for a distributed environment, it is implemented on a Prime 400 minicomputer linked with a Control Data 6600 mainframe. The conversational system employs the menu mode for central interface with executive generation of JCL streams as required to execute independent applications.

(ANOPP) Aircraft Noise Prediction Program [18.19.20,21,22]

ANOPP was developed by Control Data Corporation under contract for the Aircraft Noise Prediction Office at NASA Langley Research Center with initial delivery in 1977. A sophisticated system for the batch environment, ANOPP was

implemented on the Control Data Cyber mainframe for usage by various engineering groups within the noise prediction discipline. The primary components include an executive which employs the command language interface mode, a file partition manager called Member Manager, and a library of highly integrated applications. The command language provides flexible control structures and common language variables for low volume data movement. Auxiliary components include the Dynamic Storage Manager and the Table Manager.

(ATLAS) Integrated Structural Analysis and Design System [23,24]

ATLAS was developed by Boeing Commercial Airplane Company to support structural analysis and design for aeroelastic vehicle studies involving multiple disciplines. Initiated in 1969, the system was designed for a batch environment. The executive utilizes an elementary command language which can be augmented with FORTRAN statements to expand its power and flexibility. Pre/postprocessors are employed to process input data and manipulate output data. A file system is employed to permit data movement between applications via named random access disk files. The system was developed for the Control Data 6600 mainframe.

(AVID) Aerospace Vehicle Interactive Design System [25,7,26,27]

Developed at NASA Langley Research Center, AVID was initiated in the mid 1970's to support aerospace vehicle design and to overcome the major weaknesses of a predecessor system, ODIN. The executive component employs a menu mode interface. The relational data base manager ARIS provides for a dynamic centralized design data base for inter-application communication, a dictionary of design data descriptions, and a directory of application descriptions.

(CASE) [28]

CASE is currently being developed by Johnson Space Center with expected completion in 1984. The conversational system employs a menu mode interface, with library applications which may be batch or interactive. Preprocessors for each application prompt the user for information required for execution. The SUMS file partition manager is employed to allow tailored construction of a partition by the creating application. Large volume data movement is direct with the creating application and the subsequent retrieving application using corresponding formats.

(DIGIKON) Digital and Continuous Flight Control System [29]

DIGIKON was initially developed by Honeywell, Inc. in 1974 to support modeling and analysis of digital and

continuous flight control systems. Initially designed for a batch environment, the system has been upgraded through various government and industry contracts with the most recent version, DIGIKON IV, supporting conversational features on the Control Data Cyber, the PRIME, and the MULTICS computers. A command language is utilized for the executive interface with the command keyword indicating the specific application to be executed. Preprocessors for the applications prompt the user for any missing arguments in the command statement and also for the data input source. A file system is utilized for data management, with utilities for initializations, printing, editing, copying, and reading. File usage is by convention for specific types of data content such as modeling data or graphics data.

(DYSCO) Dynamic Coupling System [30,31]

DYSCO was initiated in the late 1970's by Kaman Aerospace Corporation for dynamic analysis of structures in helicopter studies. It has continually been upgraded and has broadened its user base to include various government groups. A Domain Executive, the executive captures the engineering design scenario of constructing and analyzing a structure. A file partition manager is employed and, in cooperation with the executive, provides for abstraction in data movement between the highly

integrated applications. The interface mode is promptive message.

(EASYCACD) Easy Computer Aided Circuit Design Programs [3]

EASYCACD was developed in 1982 by Martin Marietta Denver Aerospace to unify circuit design applications, reduce application training requirements, and eliminate the need for user knowledge of the host operating system. The promptive message interface mode is utilized with prompts being automatically reduced as the user gains experience and is able to anticipate question and answer sequences. Preprocessors are used to prompt the user for application input. Implemented for the DEC VAX computer, the UNIX operating system capabilities are utilized for a file system data management components, for job submittals, and for online tutoring.

(IAC) Integrated Analysis Capability [32, 33]

IAC was developed in 1983 by Boeing Aerospace Company under contract for NASA Goddard Space Flight Center. The executive utilizes both the command language mode, which supports approximately 50 commands, and the menu mode. Two types of data management are provided, a file system and a file partition manager based on relational data base concepts. Developed to support thermal, structures, and control technologies, the library supports interfaced and integrated applications. An auxiliary Dynamic Storage Manager component is incorporated.

(IDEAS) Integrated Design Analysis System [34]

IDEAS was developed during the 1967-1968 time period within Grumman Aerospace Corporation as an organizational approach to give greater confidence in the scheduling of structural drawing releases to the manufacturing department. IDEAS is not an ECS but instead is a collection of programs utilized in aerospace vehicle sizing. The usage of these programs was systematically monitored to increase overall productivity. IDEAS is representative of the engineering atmosphere of its day and reflects the recognition of the need for Executive Control Systems in the engineering environment. The later development of the RAVES system was based on the IDEAS concepts.

(IDEAS) Interactive Design and Evaluation of Advanced Spacecraft [35]

The IDEAS system was developed in 1982 by NASA Langley Research Center for analysis of antenna spacecraft for the Land Mobile Satellite System communications missions. This conversational system is composed of an elementary executive utilizing a menu mode interface. A file system is utilized for data management.

(IPAD) Integrated Products for Aerospace-Vehicle Design [36,37,38,39]

IPAD was proposed in the late 1970's by NASA Langley Research Center as a means to unify the automated software tools used throughout the various phases of aerospace

vehicle product developement. From concept through design and analysis to manufacturing, the system was to support the various applications and provide for automated data movement and interfacing. However, the computer science technology of the day was inadequate to realize the objective. Efforts, such as development of the PRIDE system, have continued with focus on bridging the data interface gap between engineering and manufacturing.

(ISAS) Interfaced Structural Analysis System [40]

ISAS was developed by the Boeing Company for rapid data processing activities in order to reduce the flow times for design and analysis of structures. It was initially released in 1974. The executive utilizes the menu interface mode. The library supports independent and interfaced applications with optional batch or interactive execution. Preprocessors developed for the applications prompt the user for input. The data management component incorporates a file system and also the relational RIM data base manager. ISAS has continually been upgraded and currently supports a variety of applications for inter-disciplinary usage. It is available on the nationwide Boeing timesharing network and is implemented for Control Data Cyber mainframes.

(ISSYS) Integrated Synergistic Synthesis System [41]

ISSYS was developed by Kentron International, Inc. under contract for NASA Langley Research Center with

delivery in 1980. Implemented for Control Data Cyber mainframes, the executive command language is based on the Network Operating System (NOS) JCL. The library supports applications for design and analysis of aircraft structures.

(LSSIAP) Large Space Systems Integrated Analysis Program [42,43]

LSSIAP was developed by Martin Marietta Corporation in 1981 to integrate geometry, mass, area, and mission data used in design and analysis of attitude control systems for large space systems. The executive employs a menu interface being performed according to the direct design approach. ODIN was the predecessor of the AVID system.

(NICE) Network of Interactive Computational Elements [44,45,46]

NICE was initiated in the last 1970's by Lockhead Missiles and Space Company to support formulation, implementation, and usage of advanced computational methods in fluid and solid mechanics. Implementation was for a distributed network environment consisting of IBM and DEC VAX minicomputers and Univac mainframes. The system has sophisticated components which support various operating modes such as processor-command, user-directive, processor-directive, and message. The command language CLIP, consisting of fifty (50) commands, provides the central interface. The GAL file partition manager

provides two levels of data management, local and global.

(ODIN) Optimal Design Integration [47]

ODIN was developed by Aerophsics Research Corporation under contract for NASA Langley Research Center in 1971. Implemented for a batch environment on the Control Data 6600 mainframe, ODIN supported the design and analysis of launch vehicle systems. The command language, called DIALOG or ODINEX, provided flexible control structures and utilized preconstructed JCL sequences for executing independent applications. Command language variables were utilized for low volume data movement, with high volume data movement being performed according to the direct design approach. ODIN was the predecessor to the AVID system.

(PICASSO) Program to Integrate Controls, Aerodynamics, Structures, Software, and Optimization [48]

PICASSO was developed by NASA Langley Research Center in 1978 to support a multi-disciplinary analysis and synthesis methodology for a wide range of aerospace vehicles. The command language interface mode is employed and is based on the Control Data Network Operating System (NOS) JCL. A file system is employed for data management.

(PRESTO) Prediction of Electronic Circuits [49]

PRESTO was developed by Boeing Computer Services, Inc. under contract for the Defense Nuclear Agency with delivery in 1975. It was implemented for a batch

environment on the Control Data 6600 mainframe to support analysis of electronic circuits in determining electromagnetic pulse effects. The command language ESCORT is well developed and provides flexible control structures and command language variables for low volume data movement. A file system is used for data management. The library supports independent applications, with predefined JCL streams constructed by the user.

(PRIDE) Prototype Integrated Design System [50,51]

PRIDE was developed by the Integrated Programs for Aerospace-Vehicle Design (IPAD) Office at NASA Langley Research Center in 1982 as a prototype for assessing the use of a relational data management system for data movement between engineering applications. Developed for a conversational environment on the DEC VAX, PRIDE is composed of a menu mode executive, a library of independent applications, and the relational data base manager RIM. Applications may execute in batch or interactive mode with the executive generating JCL streams as required.

(RAVES) Rapid Aerospace Vehicle Evaluation System [9,52]

RAVES was initiated in 1973 by Grumman Aerospace Corporation and was based on the IDEAS collection of programs and organizational concepts. Implemented on the IBM 360 and 370 series, it was designed to support major analysis efforts from all aerospace vehicle engineering

disciplines. The executive performs an elementary routing function. A file system is utilized for data management, with direct data movement between applications so that each application generates data in the format required by a subsequent application. RAVES has continually been expanded and remains an actively utilized system.

(SAVES) Sizing Aerospace Vehicle Structures [6,8]

SAVES was developed by NASA Langley Research Center in the early 1970's to automate the use of programs for preliminary structural design of a complete aerospace vehicle. Designed for a batch environment, it was implemented on the Control Data 6600 mainframe. The executive performed elementary routing functions, with the user creating JCL streams to execute independent programs in sequence and to access required data files. The data management component utilized a file system.

(SUPER-CAD) Super Computer Aided Design [53]

SUPER-CAD was proposed in a Masters Degree thesis presented to the Air Force Institute of Technology in 1982. The proposed system would support applications for microelectronic design, specifically very large scale integrated circuits (VLSI) and very high speed integrated circuits (VHSIC). SUPER-CAD appears to contain elements of a Domain ECS because it captures and utilizes knowledge about the specific engineering discipline (over and above the algorithms typically associated with CAD/CAE tools).

9.0 REFERENCES

1. Gott, B. Why Can't We Manage CAD?, Conference on Managing Computer Aided Design, Process Industries Division of the Institution of Mechanical Engineers (November 19, 1980), ISBN 0-85298-470-7.

2. LaFavor, S.A. and Doelling, A.E. Some Implications of Interactive Computer Application to Aircraft Development, Winter Annual Meeting of the American Society of Mechanical Engineers, Houston, TX (1975).

3. Grout, J.S. EASYCACD - A VAX Implementation of a Universal User Interface for a System of Computer Aided Circuit Design (CACD) Programs, Proc. IEEE 1982 National Aerospace and Electronics Conf., Dayton, OH (May 1982), I, 1319-1323, NAECON 82CH1765-7.

4. Cheatham, T. Harvard Programming Development System (PDS), ACM Software Engineering Notes, 8, 5 (October 1983), 49-50.

5. Sprague, R.H. and Carlson, E.D. Building Effective Decision Support Systems, Prentice-Hall, 1982.

6. Sobieszczanski, J. Building a Computer-Aided Design Capability Using a Standard Time Share Operating System, Winter Annual Meeting of the American Society of Mechanical Engineers, Houston, TX (1975).

7. Wilhite, A. and Johnson, S.C. Integrating Computer Programs for Engineering Analysis and Design, AIAA 21st Aerospace Sciences Meeting, Reno, NV (January 1983), AIAA-83-0597.

8. Blackburn, C.L. and Dixon, S.C. Automated Procedures for Sizing Aerospace Vehicle Structures (SAVES), Journal of Aircraft, 9, 12 (December 1972), 812-819.

9. Wennagel, G.J., et al. RAVES - Rapid Aerospace Vehicle Evaluation System, Annual Meeting of the American Society of Mechanical Engineers, Houston, TX (1975).

10. Herendeen, D.L. Interactive Pre- and Post-Processors for Finite Element Computer Programs, AIAA Aircraft Systems and Technology Conf., Dayton, OH (August 11-13, 1981), AIAA-81-1629.

11. Schaeffer Bulletin on Engineering and Design, Schaeffer Analysis, Inc., Mount Vernon, NH (January 1983).

12. Gervarter, W.B. An Overview of Expert Systems, U.S. Dept. of Commerce, NBSIR 82-2505, May 1982.

13. Ritchie, D.M. and Thompson, K. The UNIX Time-Sharing System, Communications ACM, 17, 7 (July 1974), 365-375.

14. Barnes, J.G.P. Programming in ADA. Addison-Wesley, London, 1982.

15. Ullman, J.D. Principles of Database Systems. second Edition, Computer Science Press, Rockville, Md, 1982.

16. Kerr, A.W. and Davis, J.M. A System for Interdisciplinary Analysis - A Key to Improved Rotorcraft Design, 35th Annual National Forum of the American Helicopter Society, Washington, D.C. (May 1979).

17. Stack, S.H. Computer-Aided Design System Geared Toward Conceptual Design in a Research Environment, AIAA 19th Aerospace Sciences Meeting, St. Louis, MO (January 1981), AIAA-81-0372.

18. Gilliam, R.E. Aircraft Noise Prediction Program User's Manual, NASA TM-84486.

19. Zorumski, W.E. Aircraft Noise Prediction Program Theoretical Manual, NASA TMX-83199, 1981.

20. Gilliam, R.E., et al. ANOPP Programmer's Reference Manual for the Executive System, NASA TMX-74029, 1977.

21. ANOPP Programming and Documentation Standards Document, Prepared by Control Data Corp., Hampton, VA, NASA CR-144989, 1977.

22. Baucom, P.H. Software Blueprints, ACM Proc. 1978 Annual Conf., I (December 4-6, 1978), 385-392.

23. Miller, R.R. Jr. Structures Technology and the Impact of Computers, Winter Annual Meeting of the American Society of Mechanical Engineers, Houston, TX (1975).

24. ATLAS: An Integrated Structural Analysis and Design System. Control and Systems Manual, D6-25400-0002TN, The Boeing Company (1972).

25. Wilhite, A.W. The Aerospace Vehicle Interactive Design System, AIAA 19th Aerospace Sciences Meeting, St. Louis, MO (January 1981), AIAA-83-0597.

26. Wilhite, A.W. and Rehder, J.J. AVID: A Design System for Technology Studies of Advanced Transportation Concepts, AIAA/NASA Conf. on Advanced Technology for Future Space Systems, Hampton, VA (May 8-10, 1979), AIAA-79-0872.

27. Mangiaracina, A.A. User's Guide, MAF AVID, I, II, III, Michoud Assembly Facility, Martin Marietta Corp., 1980.

28. Defife, J. Personal communication, Johnson Space Center, Houston, TX (August 1983).

29. Mahesh, J.K., et al. Interactive Flight Control System Analysis Program, DIGIKON IV User Reference Manual, II, Honeywell Inc. for NASA Langley Research Center, Contract NAS1-16438, December 1982.

30. Berman, A. A Generalized Coupling Technique for the Dynamic Analysis of Structural Systems, AIAA/ASME/ASCE/AHS Structures, Structural Dynamics, and Material Conf., St. Louis, MO (April 1979), and Journal of the American Helicopter Society (July 1980).

31. Hurst, P.W. and Berman, A. DYSCO: An Executive Control System for Dynamic Analysis of Synthesized Structures, AIAA/ASME/ASCE/AHS 24th Structures, Structural Dynamics, and Material Conf., Lake Tahoe, NV (May 2-4, 1983), AIAA-83-0944.

32. Vos, R.G., et al. Development and Use of an Integrated Analysis Capability, AIAA/ASME/ASCE/AHS 24th Structures, Structural Dynamics, and Materials Conf., Lake Tahoe, NV (May 2-4, 1983), AIAA-83-1017.

33. Vos, R.G., et al. Integrated Analysis Capability (IAC) for Large Space Systems, I, II, Boeing Aerospace Co. for NASA Goddard Space Flight Center, Contract NAS5-25767, June 1980.

34. IDEAS, Integrated Design and Analysis System, Society of Automotive Engineers, Aeronautic and Space Engineering, and Manufacturing Meeting, Los Angeles, CA (October 7-11, 1968), Paper No. 680728.

35. Garrett, L.B. and Ferebee, M.J., Jr. Comparative Analysis of Large Antenna Spacecraft Using the Ideas System, AIAA/ASME/ASCE/AHS 24th Structures, Structural Dynamics, and Materials Conf., Lake Tahoe, NV (May 2-4, 1983), AIAA-83-0798.

36. Development of Integrated Programs for Aerospace Vehicle Design (IPAD), IPEX Preliminary Design, IV, Boeing Commercial Airplane Co. for NASA Langley Research Center, D6-IPAD-70036-D, March 12, 1980.

37. Development of Integrated Programs for Aerospace-Vehicle Design (IPAD), User View of IPAD, 8, Boeing Commercial Airplane Co. for NASA Langley Research Center, D6-IPAD-70036-D, March 12, 1980.

38. Meyer, D.D. Development of Integrated Programs for Aerospace-Vehicle Design (IPAD), Reference Design Process, Boeing Commercial Airplane Co. for NASA Langley Research Center, NASA CR-2981, 1979.

39. Fulton, R.E. CAD/CAM Approach to Improving Industry Productivity Gathers Momentum, Astronautics and Aeronautics (February 1982), 64-70.

40. Grisham, A.F. and Keller, R.W. Interfaced Structural Analysis System Level III User's Manual, I, The Boeing Company, 81205, 1982.

41. Dovi, A.R. ISSYS: An Integrated Synergistic Synthesis System, Kentron International, Inc. for NASA Langley Research Center, NASA CR-159221, February 1980.

42. Farrell, C.E. Integrated Design System for Large Space Systems, AIAA Computers in Aerospace III Conf., San Diego, CA (October 26-28, 1981), AIAA-81-2167.

43. Farrell, C.E. An Interactive, Integrated Computer Program for Determining Large Space System Attitude Control System Requirements, AIAA/AAS Astrodynamics Conf., San Diego, CA (August 1982), AIAA-82-1407.

44. Felippa, C.A. A Command Language for Applied Mechanics Processors, Lockheed Palo Alto Research Laboratory, Palo Alto, CA, LMSC-D633582, 1979.

45. Felippa, C.A. Architecture of a Distributed Analysis Network for Computational Mechanics, Computers and Structures, 13, 405-413, 1981.

46. Wong, D.G., et al. Integrated Structural Analysis and Design Support for Advanced Launch Vehicles, AIAA-82-0675, 1982.

47. Glatt, C.R., et al. DIALOG: An Executive Computer Program for Linking Independent Programs. Aerophysics Research Corp. for NASA Langley Research Center, NASA CR-2296, September 1973.

48. Sobieszczanski-Sobieski, J. and Goetz, R.C. Synthesis of Aircraft Structure Using Integrated Design and Analysis Methods, Status Report, Research in Computerized Structural Analysis and Synthesis Symposium, Washington, D.C. (October 30 - November 1, 1978), NASA Conf. Pub. 2059.

49. PRESTO Digital Computer Code Users Guide, I, II, Boeing Computer Services, Inc. for Defense Nuclear Agency, Contract DNA001-75-C-0225, December 1975.

50. Blackburn, C.L., et al. The Role and Application of Data Base Management in Integrated Computer-Aided Design, AIAA/ASME/ASCE/AHS 23rd Structures, Structural Dynamics, and Materials Conf., New Orleans, LA (May 1982).

51. Fishwick, P.A. and Blackburn, C.L. Managing Engineering Data Bases, Computers in Mechanical Engineering, 1, 3, (January 1983), 8-16.

52. Interactive User-Oriented Aircraft Configuration Generation, Technical Proposal, 1, Grumman Aerospace Corp. for Wright-Patterson Air Force Base, RFP, F33615-76-R-3047, November 1975.

53. Cable, H.S. II SUPER-CAD: An Integrated Structure for Design Automation, Master's Thesis, School of Engineering, Air Force Institute of Technology, Air University, Wright-Patterson Air Force Base, OH, AFIT/GCS/EE/82J-7, 1982.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-172582		

4. Title and Subtitle	5. Report Date
Executive Control Systems in the Engineering Design Environment	May 1985
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Patricia W. Hurst	
	10. Work Unit No.
9. Performing Organization Name and Address	
University of Virginia Charlottesville, Virginia 22904	11. Contract or Grant No. NAG1-242
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report
National Aeronautics and Space Administration Washington, DC 20546	14. Sponsoring Agency Code 505-36-33-01

15. Supplementary Notes

Langley Technical Monitor: John N. Shoosmith
Master of Science (Computer Science) thesis submitted by the author to the School of Engineering and Applied Science at the University of Virginia.

16. Abstract

An Executive Control System (ECS) is a software structure for unifying various applications codes into a comprehensive system. It provides a library of applications, a uniform access method through a central user interface, and a data management facility. This research report is based on a survey of twenty-four Executive Control Systems designed to unify various CAD/CAE applications for use in diverse engineering design environments within government and industry. The goals of this research were to establish system requirements to survey state-of-the-art architectural design approaches, and to provide an overview of the historical evolution of these systems.

Foundation for design are presented and include environmental settings, system requirements, major architectural components, and a system classification scheme based on knowledge of the supported engineering domain(s). An overview of the design approaches used in developing the major architectural components of an ECS is presented with examples taken from the surveyed systems. The evolution from early efforts to the current state-of-the-art ECS are presented as three states of developments: embryonic, batch environment, and conversational environment. Attention is drawn to four major areas of ECS development that are central to advancing the state-of-the art and which include interdisciplinary usage, standardization, knowledge utilization, and computer science technology transfer. For each system included in the survey, a snapshot description is given with references to source documentation.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Engineering design CAD/CAE Executive control systems	Unclassified - Unlimited Subject category - 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	107	A06

For sale by the National Technical Information Service, Springfield, Virginia 22161