INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of "sectioning" the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

Wilhite, Alan Wade

FOUNDATION TECHNIQUES FOR THE DEVELOPMENT OF A COMPUTER-AIDED ENGINEERING SYSTEM FOR AEROSPACE VEHICLES

North Carolina State University at Raleigh PH.D. 1985

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark __✓__.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages __✓__
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) __47__ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages _____
15. Other _____

FOUNDATION TECHNIQUES FOR THE DEVELOPMENT OF A COMPUTER-AIDED ENGINEERING SYSTEM FOR AEROSPACE VEHICLES

by

ALAN W. WILHITE

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

RALEIGH

1 9 8 5

APPROVED BY:

Advisory Committee Chairman

BIOGRAPHY

Alan W. Wilhite was born in Portsmouth, Virginia, on December 7, 1949. He was reared in Norfolk, Virginia, and graduated from Norview High School in 1968. He was a co-op student with NASA Langley Research Center while attending North Carolina State University. He graduated with a Bachelor of Science degree in 1973 and received a Masters Degree in Flight Sciences from George Washington University in 1976.

The author has been working in the Space Systems Division at NASA Langley since 1973. He has published 26 technical papers in the areas of aerodynamics, flight performance, propulsion integration, vehicle design, optimization, and computer-aided design. His work has been in support of the Space Shuttle, advanced launch vehicle, and orbital transfer vehicle technology programs.

He is presently computer-aided design group leader in the Vehicle Analysis Branch and is a member of the AIAA Computer-Aided Design/Computer-Aided Manufacturing (CAD/CAM) technical committee.

ACKNOWLEDGEMENTS

I wish to thank the Program Integration Team (PIT) for their support in developing my dream. In particular, I would like to thank Dr. James Schwing of Old Dominion University for his guidance and support and for his invaluable research in the user interface that has replaced the one presented in this dissertation. Also, thanks are extended to Vicki Crisp of the Kentron Corporation for her helpful suggestions and the coding and re-coding of ARIS and to Kenny Jones of the Computer Science Corporation for developing the program communication pre-compiler. Special thanks are extended to Dr. Fred Dejarnette and Dr. John Perkins for their guidance and constant reminders of the deadlines. Finally, I would like to thank my wife, Pat, and children, Jason and Adam, for their support and understanding.

TABLE OF CONTENTS

TABLE OF CONTENTS (con't)

INTRODUCTION

In the 1960's, the Space Systems Division was created at NASA Langley Research Center to develop advanced technologies for spacecraft, space stations, and especially space transportation, i.e., the Space Shuttle. In the late 1960's groups were formed to evaluate aircraft company proposals for the Space Shuttle in the areas of aerodynamics, heating, weights, flight control, and flight performance.

The results of these groups were invaluable to the Shuttle program, but it was very expensive and time consuming to conduct these studies. For a complete design analysis, each group had to work on the same configuration simultaneously. In general, each group depended on data generated by the other groups. Starting with initial assumptions and engineering approximations, the discipline data would change as detailed analyses were completed and experimental data became available. Many iterations through the disciplines were needed before a configuration analysis could be completed. Complete system optimization was nearly impossible because a single iteration could take days.

In order to reduce the design cycle time, several aircraft companies developed large synthesis programs for the design and analysis of Space Shuttle configurations.[1,2] These programs closely coupled the engineering disciplines in order to conduct parametric and optimization studies. Although the design process was automated with these programs, they had limited success. Each program was developed for a single Space Shuttle configuration, which made them difficult to apply to the ever changing design requirements (such as final orbital conditions, stability, and structural/material selection) and range of Shuttle

concepts (such as two-stage fully reuseable, stage and a half partially reuseable, and the solid/external cryogenic tank/orbiter hybrid that was chosen). The programs lacked flexibility because of fixed design logic, restrictive data communications between subroutines, and lack of generality in the analysis routines. Changing one analysis routine usually meant that more detailed data requirements from the other disciplines were needed. Because of the constantly changing requirements and the data dependency between the analysis routines, the design synthesis programs were in a constant state of revision. Interactive operating systems were not available at this time, thus programs were executed by cards in a batch mode. The designers and analysts could not interact with the design process until a final design was established by the computer. This final problem restricted one of the most important engineering design contributions--creativity.

In the early 1970's, the Optimal Design INtegration (ODIN) system[3,4] was developed to integrate into one system the independent programs of each specialist. The ODIN system was executed in a batch mode. It included a data management system to communicate information between the various engineering programs and an executive control system for creating a design cycle which consisted of sequencing through the individual analysis programs, looping through a sequence of programs based on design constraints, jumping from one sequence to another after a design constraint was satisfied, and optimizing selected design parameters. Conceptually, this system provided a sound foundation for computer-aided design because the specialists could use their programs in which they had confidence, and the design system could adapt to any

configuration or design problem. In practice the ODIN system had two major flaws. First, the development of a design was very difficult because the design cycle could rarely be defined until after several design iterations were completed. In a batch (non-interactive) system, determining the execution sequence of analysis to solve a design problem sometimes took weeks. Again, in a batch process, a deck of geometry cards would be submitted and several hours later a hardcopy of the plot of the geometry would be returned, often with mistakes. These geometry iterations alone could take days.

With the advent of minicomputers and low-cost graphics equipment, the Aerospace Vehicle Interactive Design (AVID) system was developed in 1976.[5,6] This system was very similar to the ODIN system in concept. The data management system was similar but was extended to be used in an interactive computing environment for real time data viewing and editing, and a library was created so that data could be entered into or extracted directly from the database by the analysis programs. An interactive geometry system was developed that reduced geometry generation time from days to hours. Finally, a new executive program was developed to allow the interactive execution of analysis programs. The design cycle is guided by real-time results of the analysis programs. Once a design cycle becomes repetitive, sequences can be developed for batch processing.

Based on the success of the AVID system, a dedicated computer was purchased, and a number of general analysis programs have been developed or acquired for geometry, aerodynamics, heating, flight control, operations, and costs. Major problems still exist, however, because the

number of data elements to be transferred between the analysis programs has increased from several thousand to several million. The number of interactive terminals and distributed computers has also increased (a terminal and/or computer on every desk is now typical), allowing each individual specialist to participate in the design process simultaneously.

The purpose of this dissertation is to develop a methodology to integrate data, programs, and engineering specialists together in the present computer environment. First, engineering design is discussed. An historical perspective of past and current design integration approaches and data management systems is presented. Then a system architecture for program/program user integration is developed along with a data management system to support this architecture. The approach is given in detail and lessons learned from the different phases of implementation will be summarized.

ENGINEERING DESIGN AND COMPUTER AUTOMATION

The engineering design process is illustrated in Figure 1. Working with a set of predefined requirements, an experienced designer develops a configuration that may meet these requirements. This trial configuration is then analyzed by the appropriate engineering disciplines. Performance results (size, weight, cost, etc.) subject to constraints arising from the analysis procedure (maximum loads, material selection, structural arrangement, propulsion selection, etc.) are compared to the initial requirements. If the configuration does not meet the requirements, the characteristics of the configuration are modified in a heuristic way in the early design stages because the consequences of the changes are not known.[7] As the configuration is iterated through the analysis, comparison, and reconfiguration cycle, a general knowledge about the trades of performance and constraints is gained. Through this knowledge, the final configuration can be defined and may be optimized. For practical design, where only small changes to a configuration are required, the design process is well defined and a specific handbook method may be applied; but for revolutionary design (space programs and new projects), the design process, analysis tools, and configuration must evolve together because the initial configuration may be drastically different from the final configuration.[8]

In addition to the iteration cycle for the configuration, results must be iterated by the engineering specialists to complete the analysis of one configuration. As shown in Figure 2, a simplified airplane design cycle, the aerodynamics engineer needs geometry, which is predefined, and the center-of-gravity location, which is not initially

known. A guess of the cg location is used until the weights engineer computes the cg location. The performance engineer needs the aerodynamic results, propulsion specifications, and the weight of the vehicle. The weights engineer needs the performance results and propulsion specifications for loads analysis. It is obvious that the design data must be iterated and passed from one engineer to another before analysis results are completed for just one configuration.

There arebasically three levels of design; conceptual, preliminary, and detailed.[9-11] At the detailed design level, each subsystem and part of the configuration must be thoroughly analyzed and tested. Results from this phase of the design are part drawings used in the manufacturing process.[9] This level of design requires more resources than the other two because part design and definition are labor intensive and analysis results are verified with tests of prototype models.

Because of the investment required at the detailed level of design, the majority of research and development in design automation has been directed towards this level. The term computer-aided design (CAD) in the current literature does not address engineering discipline integration and configuration iteration automation. Computer-aided design is associated with electronic drafting systems for the development of 2-dimensional and 3-dimensional drawings for mechanical, architectural, structural, and electronic applications.[12] These system are tied to the manufacturing process through the generation of tapes (files) for numerically controlled machines that automatically mill the defined parts. This process of combining computer-aided design with computer-aided manufacturing is called CAD/CAM.

The first two levels of design are used to define and evaluate the configuration for the final design level. In the first level of design, the conceptual level, the configuration needs and requirements are evaluated, a market analysis is conducted, and a potential set of solutions are defined. Rough order of magnitude engineering analyses are used to determine if the solutions can be physically obtained. In the second level of design, preliminary design, sophisticated engineering techniques are used to reduce inaccuracies to determine the best configuration for the final design cycle.[10]

Currently automating the conceptual and preliminary levels of design with computerized systems does not appear to be very attractive as compared to the resource intensive detailed design level that ultimately interfaces with the manufacturing process.[11] Engineering companies do not sell configuration designs but sell the products resulting from the detailed design process. On the other hand, it is at the lower levels of design where the products are first developed. At these lower levels, the products can be easily enhanced and optimized while the cost of change is relatively low. These optimized designs may result in reducing re-engineering tasks at the detailed level and in development of a more competitive product. If the analysis tools are not readily available through automation, there is little chance of making radical changes to an existing design or replacing the design with a unique idea because substantial time and cost has already been invested. On the other hand, if the analysis tools are readily accessible that can reduce the time and cost involved in verifying a new concept, innovation can be encouraged at these lower levels . Because of these potential benefits, techniques are now being developed for

combining "applications software, graphics hardware, and data management capabilities" into integrated computer-aided engineering (CAE) systems.[12]

PAST APPROACHES TO COMPUTER-AIDED ENGINEERING

Many previous approaches for engineering discipline integration for analysis and design have had three distinct components. The first component is that of geometry definition and presentation since geometry permeates almost every engineering model. Data management, the second component, communicates data between the engineers and their analysis programs. The final component is executive management which controls and directs the design process and allocates system resources.

Geometry

Because there are a number of companies involved in the development and marketing of geometry (CAD/CAM) systems for all the various levels of design and applications, it will be assumed that geometry is commercially available to perform any required task. For example, there is ANVIL 4000 mechanical drafting and manufacture interface, PATRAN-G for finite element modelling for the large structural analysis programs, and the Configuration Development System (CDS) for conceptual studies of aircraft, to name just a few used at Langley Research Center.

Data Communication and Program Coupling

The data communication and analysis program coupling in past and current computer-aided multidisciplinary systems can be generalized as: closed-coupled integration, close-coupled interfacing, loose-coupled integration, and loose-coupled interfacing.

Single programs that perform design synthesis[1,2,13-15] are classified as being close-coupled integration systems (Fig. 3a). Close-coupling means that the path through the disciplines to resolve parameter iteration, design constraints, and/or optimization is usually fixed. In these single programs, the executive consists of the internal program logic that calls the various subroutine modules for engineering analysis. The data transfer is usually tightly integrated through the use of common global blocks and data files. The main advantages of these systems are: a very fast execution speed that allows parametric studies and optimization, tightly controlled data management so no data interpretation errors occur between the various modules, and small enough size for a small group of engineers (1 to 5) to use. The disadvantages of close-coupled integration derive from the difficulties of integrating all the analysis pieces into one computer program and also from the difficulties in adapting to evolving requirements because the analysis techniques are deeply embedded in the program and data changes often affect much of the program. Many of the systems are developed to perform complete system synthesis; thus it is difficult to analyze a vehicle for a single discipline or conduct just a partial study. The SIZE[16] system tried to eliminate these problems by working with a library of analysis modules that can be precompiled and managed by a customized executive system. Each engineering group develops its own analysis modules to support this system. A special purpose data management system was developed to communicate data between the analysis modules. The SIZE system had a good architecture for developing an automated design system for conceptual studies but only a limited number of design parameters could be used.

Close-coupled interfacing leads to the coupling of independent analysis programs that were used by the individual specialists (Fig. 3b). With interfacing, the data coupling is external to the analysis programs. The interfacing of two programs is accomplished by having the first program write the input file for the second program. If this input file meets all requirements of the program, then the second program does not have to be altered except to create an input file for the next program. By repeating this process between the various programs, a fixed path through the programs can be established for analysis and design. This interfacing technique is relatively easy to implement and has been successful in coupling programs.[17,18]

As programs are linked into these close-coupled systems, a network of programs evolves with fixed execution paths. Therefore, a disadvantage to close-coupled interfacing is that the design cycle is restricted to the fixed path of the linked programs. The path may be appropriate for the first intended application (e.g., an airplane), but may be wrong for another application (e.g., a space shuttle). Also, as the network of analysis programs grows, it becomes more and more difficult to couple new programs into the system. A new program may require input from several programs that are not directly "linked". A separate program (called an intermediary program) must be written to read the output from several programs and create an input file for this new analysis program.

Because of the problem of coupling new programs with de-centralized information and because the design cycle is predefined by the program network, many of the current systems have centralized the data, which leads to loosely coupled programs. Loose-coupled systems allow programs

to be individually executed and execution paths through the programs to be defined externally.

Loose-coupled integration is the technique employed by most business applications today (Fig. 3c). A central database management system is used for data communication to all the separate programs. The programs are developed from their inception to communicate directly with the database management system for both input and output. This approach is the one being developed by the Independent Programs for Aerospace vehicle Design (IPAD) study.[19] The main advantage of loose-coupled integration is that the programs (analysis techniques) can be developed independently and can later be coupled together to form a complete CAD system. Each program is independent of the others as long as the required input data is resident in the central database. The main disadvantage is the complexity of integrating existing programs with the database management system. For a program that has been developed without any considerations for future database integration and has a large data input with many analysis options, the integration task is difficult for anyone but the program developer. The difficulty arises because the internal program variables that must be integrated with the database are almost never documented and must be deduced by comparing the input procedure with the computer code. For programs that have been poorly structured, this variable identification can be a formidable task. For a company-wide CAD system where data standards can be enforced and program development is dedicated to this CAD system, it can be advantageous to integrate the programs. However, because of the small budgets usually associated with conceptual and preliminary CAD systems, the software overhead of integration could be prohibitive.

To eliminate the integration software overhead, analysis programs can be interfaced to a central database for data communication (loose-coupled interfacing). Instead of communicating directly with the database for input, a pre-processor program is used to retrieve data, transform the data, and format the data into an input file for the analysis program (Fig. 3d). The advantage of interfacing with a pre-processor program is that it requires no knowledge of the internal coding of the analysis program. Only the program input requirements are needed, and these are usually well documented. Because no modifications are made to the input and analysis sections of the program, there is no risk of developing "bugs" in a production program. Finally, if problems with the coupled program do occur, the input and analyses program can be examined and executed independently by the specialist responsible for the program. The disadvantages of interfacing are that a pre-processor program must be written for each analysis program and additional computer overhead (time) is required for writing an input file. Program output data to be communicated to other programs can be handled in two ways. First, an output subroutine can be added to the analysis program to generate a file of data. This file is read by a post-processor program which stores the data in the central database. The second method involves integrating the desired output results directly with the central database in the analysis program. The second approach involves just as much work as the first, but the post-processor program development is eliminated. Loose-coupled interfacing has been used by several design systems.[5,20-24]

Executive Management/User Interface

The third major component of past engineering synthesis systems is the tool to manage program execution and incorporate the engineering specialist directly into the computer-aided environment.

With single program synthesis systems, executive management is accomplished mostly through the main routine in a program that loops through the analysis during iterative design cycles, jumps from one iteration to another as design constraints are satisfied, and continues the design process based on directions given by optimization algorithms.[1-3,13-15] In most of these synthesis systems, the only control the user has over the design cycle consists of options to select various design cycles that have been preprogramed into the system. A typical example in these systems is vehicle sizing in which the vehicle can be sized to carry a specified payload on a given mission, the payload weight can be calculated for a fixed vehicle size to meet a specified mission, or the mission can be calculated for a fixed vehicle and payload size. It is obvious that the synthesis programs are not very general from the number of different design systems for various classes of vehicles which can be found in the literature.

In several design systems that utilized independent computer programs for engineering analysis, the executive consists of a design control language that is used to couple the programs in a specified sequence.[3,16] Looping, jumping, and optimization are available in many of these systems, and are similar to the preprogramed design logic in the single program synthesis programs. This trend of specifying the complete design process with optimization in which the computer solves the design problem was precipated by the batch operating systems that

were available at this time. User interaction was very time consuming because results were available in printed form only (graphical form was available in the next day delivery service), and input had to be punched on cards.

With the advent of minicomputers, interactive operating systems, and low-cost graphics equipment, the trend has shifted from monolithic and highly automated executive systems to the engineer-in-the-design-loop user interface systems.[5,19-23,25] The main objective of the user interface is to provide an environment in which the engineer can execute programs in response to the assimilation of real-time engineering analysis results. If the design problem has to be solved by many iterative cycles, the user interface provides an environment similar to that of the batch synthesis systems. With the user interface, the design cycle can be interactively established and then programmed. This approach saves time and reduces boredom by eliminating repetitive tasks. The second objective of the user interface is to provided a direct path into the design data for review and manipulation. Thus the user interface should provide total control of all the resources of the design system - analysis programs and data.

PERSPECTIVE OF CURRENT DATABASE MANAGEMENT SYSTEMS

A database management system controls the structure of and access to a central repository of data. With a central database, redundancy (duplicate data copies) can be reduced, standards in the representation of data can be enforced, security restrictions can be applied, and data can be shared.[26] One of the major objectives of any database management system is to provide independence between data and the application programs. Data independence can be defined as the isolation of data from the programs so that changes in one do not affect the other. Thus, software maintenance of application programs is minimized when changes or additions are made to the system.

To obtain this data independence, the national standard for the development of database management systems recommends a three layer (schema) approach (Fig. 4).[27] The internal schema defines the access structures for the data. Various structures are provided so that the data can be stored or accessed easily through sorting, stacking, or queueing or accessed quickly by inversion or hashing. The conceptual schema defines the overall structure (defined later) of the data in the central database. Finally, the external schema is a subset (and possible reorganization) of the conceptual schema and defines the data to be accessed by each application program.

The structure of the data in the database is the main difference between current database systems. The three data structures are relational, network, and hierarchical.

The relational model is illustrated in Figure 5. This model is defined by tables called relations. In the figure, the relations are:

VEHICLE, which lists current concepts, SUBSYSTEMS that are typical for any vehicle development, and PACKAGING, which positions the parts in each vehicle. The columns of data are called attributes, and each row of data is called a tuple. The advantages of a relational database are that the structure is very simple to define (tables) and that a natural query language exists based on set theory.[28] For example, to retrieve all parts for the SHUTTLE, the following commands are used:

```
JOIN VEHICLE AND PACKAGING OVER V# GIVING V_AND_P
SELECT S# FROM V_AND_P WHERE VNAME EQ SHUTTLE
    GIVING SH_V_AND_P
JOIN SH_V_AND_P AND PARTS OVER S# GIVING SH_PARTS
```

where V_AND_P, SH_V_AND_P, and SH_PARTS are temporary relations. The JOIN command combines the relations VEHICLE and PACKAGING into one relation, V_AND_P, where the vehicle numbers, V#, match. The SELECT command builds another relation that contains only the SHUTTLE vehicle. The final JOIN command satisfies the query. The disadvantages of using the relational data structure are that some natural data structures are not easily defined in relational form and the systems are traditionally slow because the system, not the user, structures the data internally.

The second data structure, network structure, must be defined both logically and internally. The network structure can model relations (trees) and of course networks as shown in Figure 6. The tables are called records, the columns are called fields, and the physical connections between the records are called sets. The advantages to the network structure are the flexibility in data structure and the ability to customize the internal structure. The disadvantages are the complex language for constructing the structure and the procedural data language

which is used to "walk" through the database record by record using the access paths shown in Figure 6. Most network systems do not support an interactive query language.

The final structure is the hierarchical structure (Fig. 7). This structure was one of the first structures to be used by a database management system. The tables are called records, the columns are called fields, and linkages between records are called segments. Because the structure is simple, both interactive and program interfaces are supported. The problems with the hierarchical structure is that only tree structures can be modelled and data redundancy (duplicate record occurrences) can be a problem. (E.g., SEAT in Figure 7 is duplicated.)

The "right" type of data structure was a major topic of controversy between the developers of these systems in the late 1970's. Today, most systems claim to support some or all the capabilities of all the data structures. Research is being conducted that combines the best features of the three models into a unified data structure and data manipulation (query) language.[26]

There are a number of other qualities about data management systems other than data structures. There are data security techniques that protect the data against both intrusion by non-authorized users and intentional destruction. Security mechanisms range from passwords to physical devices such as voice or fingerprint validators. There are integrity mechanisms to ensure that the database is accurate at all times. It is impossible to ensure that all data entered into the database is completely correct, but it is possible to check the plausibility of the data. Integrity mechanisms include checking the

data between upper and lower bounds or comparing with a set of possible values. Another feature is a backup and recovery system for re-establishing the database after a hardware or software failure. This system is very important for banking, airline reservations, or other business applications where each transaction is critical. A data dictionary is sometimes supported that provides the system administrator with a description of each entity in the database and a cross-reference guide between the data and application programs, which aids in reviewing, modifying, or adding data or programs. Finally, there are mechanisms to share the database simultaneously among all users (called concurrency) because this is the main purpose of most central database systems.

If the main purpose for database management systems is to integrate independent computer programs through a central database and provide data independence and an array of data structures and utilities, why is there little application of these commercial systems to an engineering environment? The main reason that database management systems have not been used in engineering environments is because these systems have been developed exclusively for business applications. Several studies have categorized the differences between engineering and business database applications.[29-33]

The first difference is the programming language. Engineering applications are written in FORTRAN which has very limited support for data structures. Business applications are written mostly in COBOL and PL/1. The network structure discussed previously is based on the CODASYL DBTG[26] standard which is an extension to the COBOL language. Interfacing these systems to FORTRAN programs is either very awkward or

not supported. Also, because business programmers use mostly character or decimal data, engineering data types such as integer, real, and complex are usually not supported.

The typical application for engineering is for integrating large programs for analysis and design. Large groups of localized data are accessed, modified, and then replaced with moderate frequency. Geometric data is an example of data that is retrieved, manipulated, and then replaced. A typical business application (banking, airline reservation, inventory, etc.) involves integrating small programs where extremely small groups of data are consistently being updated, but the database remains constant in size. For example, in a banking application many updates to accounts will be made daily, but only a relatively small number of accounts will be added or deleted.

Database management systems were developed for business data manipulation. When data is being updated, the database system locks out other users to the data until the integrity of the data is checked and data replaced. For business applications, there is no problem with these quick transactions. For engineering applications, the transactions of large data groups can severely degrade multi-user performance because of the long lockout times. If a data integrity error occurs, the system restores the database to its original form and cancels the transaction request. To perform this rollback, the system records the operations of every transaction. When an integrity check is positive, the system simply applies the operations in reverse order. For small transactions, there is usually no problem, but for large transactions, the overhead can be quite significant.

In an engineering environment, localized areas of the database are used by the various specialists. These areas are retrieved as "work copies". These copies are modified after much analysis and then released back into the database. Thus, only the last saved copy needs to be restored if an integrity error occurs. These work copies have several purposes: 1) to modify an existing data set for revisions, 2) to serve as a guide in creating a new data area, and 3) to move data between public and private areas. Because the engineering update activity is characterized by operations on a localized data set, many of the utilities used for concurrency are not needed until a data set is to be placed back into the central database.

There are two approaches on how to support the engineering data activity. The first is to provide all the features of a database management system plus the features applicable to the engineering environment.[19] The second is to support local databases for the engineers and provide a database merging utility to construct a central database from local databases of completed analyses.

The first approach has been studied for more than 10 years at a cost of more than 15 million dollars. To provide data security, multi-user concurrent access, data integrity, backup and recovery, the database management system became very large and computational overhead precluded an interactive design environment. The architecture was redesigned around two separate data management systems. The large multi-featured system was used to handle company wide transactions, and a relatively small system was used for the actual engineering design process.

In this paper, the approach of utilizing a single high performance data management system to couple analysis programs into a design system to support multi-user projects is explored. An overall computer-aided engineering system architecture is designed, tools to support program coupling and user interface are defined, and a data management system to manage the complete system of programs, data, and users is developed.

PRESENT COMPUTER-AIDED ENGINEERING SYSTEM ARCHITECTURE

The present system architecture was developed from experience in developing a conceptual design system, experience in using other systems, and past experiences of previous computer-aided engineering (CAE) systems that have been reported in the open literature. Figure 8 illustrates the architecture of a single-user CAE (computer-aided engineering) system. This single-user system is provided for explanatory purposes and will be expanded to support a multi-user environment.

Single-User System

User Interface

The user interface is used to provide an interactive environment between the user and the CAE system. The user interface consists of a communication language that allows all the capabilities of the system to be accessed by the user. The command language, interactive system presentation to the user, execution error checking and system recovery and multi-user capabilities for the present CAE system have been defined and developed by Dr. James Schwing of Old Dominion University and Donald McMillan, his research assistant.[25]

Program Library

The program library consists of all the programs that have been coupled to the system. For each program, there exists the FORTRAN

source file, the executable run file, appropriate graphics and math libraries, and the procedure for creating the executable run file.

The purpose of the program catalog is similar to that of the card catalog in a typical library - to thoroughly document all available resources. The program catalog consists of the following:

1) program name
2) version number
3) date last modified
4) description
5) keywords
6) source file name
7) executable run file name
8) library names
9) procedure name for building the run file
10) description
11) custodian's name

An example of the program catalog is given in Figure 9.

Procedure Library

The technique used for program execution is with procedure (or command) files that are available on most modern interactive operating systems. These files consist of a list of operating system commands that are executed in sequence. The system commands can consist of any operating system command such as file handling and program compilation, loading, and execution. These procedure files can usually be processed either interactively or in a batch mode. The user interface uses this facility extensively for interfacing with the operating system.

Examples of the procedure files in the library are presented in Figure 10. An entry exists for each system command and consists of the following:

1) procedure name

2) system command number

3) system command

4) system command description

In the command line, the command to execute a program on the PRIME computer is SEG followed by the program name and the input file name if needed. The input file in the procedure files is the template name as described in the data communications section.

Associated with the procedure files is a procedure catalog to identify each procedure. An example of the procedure catalog is presented in Figure 11. The procedure library consists of the following:

1) procedure name

2) description

3) date created

4) custodian's name

For each program, there may be more than one procedure for execution since a program can take on various characteristics based upon the input provided. For example, a generalized trajectory program can be used to simulate vehicle takeoff, vehicle mission maneuvers, and landing. Thus a different procedure could be used for each of these cases that specifies a different input data stream.

Configuration Database Library

A configuration database is the repository for all the data that is used or created by the engineering analysis programs that have been coupled to the system. The database can consist of mission requirements data, geometry definition data, input options and default input values for analysis programs, results generated by the analysis programs, and a wide variety of miscellaneous data. This data can take the form of scalars, arrays, tables, and sequential files in various data types, e.g., integer, real, and character.

The initial configuration database is called the golden configuration database. It consists of an example of input and output data for each of the analysis programs. To use the CAE system, a copy of this golden database is first made and modifications are made to this database to analyze existing configurations or design new configurations. With the concept of new and old configuration databases, checks can be made on the input and output data of analysis programs by comparing the two databases.

Associated with the configuration databases is a configuration database catalog that is used for identification and database recall. An example of the catalog is presented in Figure 12. The catalog consists of the following:

1) configuration database name
2) origin configuration database name (SYSTEM is the golden database)
3) date copied
4) date last modified
5) description
6) custodian's name

Data Dictionary

Just as important as the configuration database is a description of each of the data elements in the database. This information is stored in the the data dictionary. The data dictionary facility in current database management systems is used to store the schemas (data structure definitions) and cross-reference information that shows the data flow through the programs. The data dictionary is invaluable for large systems when changes are made to the application programs or when new programs are added to the database system.[26]

In this CAE system, not only is the data dictionary is not only used to visualize the data flow through the system, it also has an expanded role in program and interactive data communications. A detailed description of the data dictionary will be presented later.

Template Library

Templates are windows into the configuration databases. They are used to define the data in the database that is extracted for analysis program input or replaced by results generated by an analysis program. These same templates are used to define the database input and output data for review and manipulation. Templates can also be used to generate data reports as the design cycle progresses. The template consists of a template catalog and the templates. Templates will be presented in greater detail later because, like the data dictionary, they are used in the database communications.

Activity Log

When a design activity is completed for the day, or temporarily suspended, the activity log is used to record comments about the activity to serve as a reminder for the next design session (a must feature for aging technologists). It can also be used to restart a previous design activity. The activity log consists of the following:

1) user's id
2) session comments
3) configuration database name
4) date and time

An example of a log is presented in Figure 13.

Multi-User CAE System

By comparing Figure 14 with Figure 8, it can be seen that the multi-user CAE system is very similar to the single-user system. The main difference is that in a multi-user system, a work environment containing local copies of configuration databases, programs, procedures, and templates is created for each user. The activity log is moved to the user work environment for personal use and a catalog is added to the global database to identify the various users of the system.

With local procedures and templates, the standard global procedures and templates provided by the system administrator can be customized to the user's personal tastes. For example, procedures for the execution of single analysis program can be combined to form a personal design sequence of program executions to solve a given iterative problem. Also, a template that identifies many input parameters for a program can

be culled to only a few pertinent parameters which reduces interactive data processing time and errors. These personalized features can be used to increase productivity in an experienced user and greatly simplify a system for the novice user.

To maintain strict standards and provide a fixed design environment, only the system administrator can modify the elements of the global database. Because changes in the global database may have an impact on the user databases, the system administrator may have to modify local databases as well.

With this architecture, all the users are isolated from each other. Each user can make changes to the configuration database without affecting the other users. Once a specialist is satisfied with his results, the configuration database can be released to the other specialists or upon approval of the system administrator it can be stored in the global database.

This isolated approach seems to be a better approach than providing concurrent access to a single database. In the concurrent approach, a specialist can change a value to an input parameter. Before he has a chance to run his analysis, a second specialist could change the value again. Thus, the results would not be dependent on his assumed input parameter. In an even worse scenario, the input parameter could be changed by another specialist after the results were computed. In this case the results would not even be dependent on the current parameter value. This problem of data dependency will be discussed later.

For this isolated approach to work, a database compare feature is needed to identify differences between configuration databases. On a

particular project, each specialist may begin with the same configuration database. If two specialists need to communicate their results, the database compare facility would indicate all differences that must be scrutinized to determine if there is any conflicting data between the two configuration databases. A simple example is that one specialist may have changed the geometry. For this case, a decision on the correct geometry would have to be made by either the specialists or the project leader.

This database could also be used by the system administrator to compare the databases of all the specialists working on the same project to determine project progress and identify any potential major conflicts.

In order to release a configuration database that is representative of all the analyses performed by the various disciplines, the specialists would have to discuss each of the database conflicts and agree on a compromise database. In this case, the data may not be "correct" because the results of each specialist may depend on the results from the other specialists, which may have been changed in the compromise database construction. In order to produce a "correct" database, the database may have to be passed from one specialist to another until all conflicts are satisfied. The term "correct" may mean a converged solution or simply a database acceptable to each of the specialists. The problem of obtaining a converged solution is a major one in any multidisciplinary project where data must be communicated between the specialists. This architecture was constructed to reduce data communication time and errors in interpreting the communicated data

through the electronic configuration database and to warn about possible data conflicts. Solution convergence is an area of future research.

Architecture Discussion

The system architecture was designed to provide management, visibility, flexibility, and performance. As discussed in the architecture description, there is a catalog that locates, describes, and provides a point of contact for every active entity in the system (data, programs, and users). There is as much meta data (data describing data) as there is actual configuration data.

This meta data makes the system visible to the user. The work activity log guides the design process by providing past procedures that solved similar problems. The program and procedure catalogs can aid in the selection and use of programs to generate needed data. The global and local databases provide a list of all data generated by the users for that project. Finally, the data dictionary reduces mistakes of interpreting and applying data because a detailed description including the physical units is provided.

By using independent computer programs and a central database system for data communications, there is flexibility in creating a design environment to model engineering problems. The programs can be executed in any order to satisfy design requirements or design constraints that arise during the design cycle. Flexibility is also provided by the addition of new programs or data. Data can be added with absolutely no effect on the system because the data management system provides data independence.

As demonstrated in several attempts to use a commercial business database system in an engineering environment, the system response was slow, the definition of the database was difficult, and trying to use unsupported data types was impossible.[29,30] The present system was designed for maximum performance with multiple single-user systems with an umbrella management system, single purpose databases (a separate database for each project/activity combination), and a data management system designed especially for this system. The localized database system eliminates concurrency checking; thus a small high speed data management system can be used. A single purpose database minimizes database size which results in reduced search times through the database and less disk space because data inversions are not needed to reduce search time of individual user data sets.

The following section is a discussion of the development of a data management system for this CAE system.

PRESENT CAE SYSTEM DATA MANAGEMENT

A relational information system (ARIS) was developed specifically for the present CAE system. As discussed earlier (see PERSPECTIVE OF CURRENT DATABASE MANAGEMENT SYSTEMS), many of the current database management systems have adopted one of three data models - relational, network, or hierarchical. The relational model was selected for this study because its data structure is easy to understand by engineers (tabular form), there is a natural query language for interactive processing, the model could be enhanced to support any FORTRAN data structure, and its capabilities were a good match for the present system. This relational system was developed because there were no commercial systems available when the development started. Developmental relational systems were available but these systems were used mainly for studies of query optimization and data structures and not for applications.[34-36] Because there was an opportunity to enhance the relational model for engineering and distributive database applications, a relational information system called ARIS was developed.

A detailed description of ARIS is given in the appendix. The following discussion is devoted to enhancements to and applications of the relational model to the present CAE system.

Data Entities

Data entities in the relational model are called attributes and can be defined as the columns of data in the relation (2-dimensional table).

Because the FORTRAN language is the main language for engineering computations, the data types and structures of attributes were enhanced to conform to the FORTRAN language standards.

Any FORTRAN data type can be stored and retrieved in an ARIS database. Interactively, only real, integer and character data types are printed (double precision, complex, and logical have not been implemented). New to relational systems, FORTRAN data arrays have been implemented, and up to 3-dimensional arrays can be declared.

A concept unique to relational database systems is variable type and dimension attributes, which were developed for parameter type relations (Fig. 15). With this variable type attribute, the type and dimension are declared at the time when the data is stored in the database. Thus, scalars and arrays can easily be stored in a relation.

A final data type called file is actually not a data type but a specification of an attribute. The file data type was created to add sequential ASCII files (text or source data) to the database by just providing the file name and not physically storing the data in the database.

For large text files, the file specification means that the external text file can be manipulated by the computer systems editor, thus eliminating the need to develop special editing features for the database system. The ARIS can still search for character strings in the same way it searches character type attributes, and the text files are also printed with interactive queries. With this capability, paragraphs of text can easily be saved, manipulated, and retrieved.

After evaluating the data requirements for the coupling of several programs, it was determined that in some cases, several large

bulk data files were shared between programs. A typical example is geometry. The geometry system (generation, display, and analysis) has its own internal structure that no other program uses. There are two ways to incorporate the geometry file. The data can be placed in a relation as shown in Figure 16a or in a file with just the file name in the relation as shown in Figure 16b. This file data type is best used for these large data files because relation retrieval can be slow compared to just reading a data file because of the database system overhead. Also, data files that can change drastically in size are best handled by the operating system of the computer, which can efficiently allocate and reclaim storage as the file grows and shrinks. The file data type is handled by the database as a repeating data array, and thus no special handling techniques are required for retrieving the records from the data file. To denote that a data file is owned by a database, the database name is appended to the file name when it is created.

Distributive Databases

As shown in the multi-user CAE system (Fig. 14), there are a number of independent configuration databases - global and each user's. In order to distribute the data from one database to another, make copies of complete databases, or make changes to database structure, several techniques were developed using and enhancing the relational algebra language.

In the relational model, a concept of permanent and temporary relations is used to solve complicated queries. For example, to find all the data generated by AWW and NHG, the query might take the form

```
SELECT DB_NAME FROM DB_CATALOG
       WHERE INITIALS EQ AWW GIVING TEMP1

SELECT DB_NAME FROM DB_CATALOG
       WHERE INITIALS EQ NHG GIVING TEMP2

UNION TEMP1 AND TEMP2 GIVING ANSWER

PRINT ANSWER
```

In the two SELECT commands, the relation DB_NAME is searched, and the results are stored in separate temporary relations called TEMP1 and TEMP2. The union operator combines the two relations to form the third relation ANSWER, and the final command prints the results.

In ARIS, the permanent database and the temporary database are treated as one large database with the permanent database searched first for a relation and then the temporary. By appending an extender to a relation (.P for permanent and .T for temporary), relations that exist in both databases can be accessed separately. As an example, to determine all the customized procedures developed by AWW and PRW, the local procedure catalogs of these two users can be combined by opening the AWW database as a permanent database and the PRW database as a temporary database. The command

```
UNION PR_CATALOG.P WITH PR_CATALOG.T
```

satisfies the query. By using this command on every user's database, all the customized procedures in the system can easily be viewed.

There are four commands that are used in this distributive database system that use this permanent/temporary facility. The TPCOPY command makes a copy of the database (from temporary to permanent). A copy can be used to begin a revision of an existing database or make a copy of another user's database or a global database.

The UNION command adds tuples from one user's relation to another (duplicate tuples from the temporary are eliminated). Duplication is determined by a comparison of primary key values. The primary key in relations is the attribute (or groups of attributes) whose value(s) must be unique for each tuple. The primary key attribute(s) is declared when the relation is defined.

The REPLACE command replaces tuples in one relation from another based on the same primary key value in both relations. This command is used to update information from another source, while retaining data in the permanent database that is not common to each relation.

The final command is RCOPY which copies a relation from one database to another and erases the old relation. This command updates a work activity database to be compatible with another work activity.

There are also utilities to perform these activities by unloading the database, database schema, relation data, or relation schema to an ASCII file. Thus database definitions and relation data can be communicated across different types of computers by copying the file from one computer to another and loading the ASCII file into the database system. This system has been developed for the PRIME and CDC computers.

Data Communication

Data communication between analysis programs, users, and the configuration database is through the data management system ARIS. As shown in Figure 17, the ARIS system is divided into 3 layers - the host computer software, the command library, and the interactive interface.

To convert the system to another computer, 95 percent of the code that needs to be changed is the interface between the system and the

host computer software. This conversion consists of the time and date functions, sorting routines (an internal sorting is provided for relations with less than 1000 tuples), and disk random access routines. The other 5 percent is to correct for the differences between the FORTRAN 77 compilers.

The interactive interface consists of the query parse routine, the executive routine, and the command structuring routine. The query parse routine resolves the query into component parts (tokens). The executive routine determines the desired function from the first token and executes the associated command structuring routine. Finally, the command structuring routine (one for each command) processes the tokens,extracts the data needed by the command routine, and then executes that routine. With the interactive commands, databases can be created, relations can be defined, and data can be entered, reviewed, changed, and deleted.

The middle layer of the system consists of the command subroutines that process the relation commands. There exists a subroutine for every interactive command. Thus, all the relational operations are available to the application programs that are available interactively through the command subroutine library.

The library can be incorporated into the engineering computer programs for complete system integration. The library may also be used by a pre-processor program that retrieves the data from the database and formats the data into a file that will be used as input by the analysis program - data interfacing. As mentioned earlier, the integration/interfacing question depends on the level of performance desired and the difficulty in integrating the data interface directly

into an analysis program. In either case, the command library can be used.

Data Communication Utilities

To ease the burden of data communication, integration and interfacing, two utilities have been developed - the reviewer for user/database interactive communications and the data formatter for program/database communications. Both the formatter and reviewer use the data template as a guide for specifying a subset of the configuration database and the data dictionary for describing each data entity in the database subset.

Template

A template is a list of data entities in the configuration database that is used to specify data to be processed by the reviewer and/or the formatter.

When the ARIS was developed, it was assumed that all data communications would be through the interactive query language for user communication and through the subroutine library for program communication. Because the configuration database became quite large (10^6 data elements for only the aerodynamics, propulsion, trajectory, and vehicle sizing analysis programs), editing the entire database was impractical. Because the analysis programs accessed a subset of the configuration database for input and output, special editing programs would have to be developed for each analysis program.

To reduce software overhead and programming mistakes in developing the data communications with the configuration database, the template was developed. The data template is very similar in concept to the

external schema in sophisticated database management systems[26] that were developed for program communication only. In a paper entitled "A General Purpose Data Entry Program", a system was developed for program data communications with a custom data system.[37] This system was developed to avoid the computational overhead of large database management systems and to provide the data structures not supported by the database systems available to the author of the system.[36] Within this system, a template was developed. The template contains the specifications for the data to be entered or altered and all the information needed to guide the data entry task which includes the sequence of data to be edited and the description of the data. This template has the essential ingredients to support an interactive input data reviewer - a definition of the subset of the database to reviewed, a definition and structure of the reviewing sequence, and a description of each of the data entities.

The overhead and data structure problems described in Reference 37 do not exist with the present database. ARIS was developed to support the data structures in the FORTRAN analysis programs, and many of the extra features in current database management systems were not implemented to reduce computation overhead.

In the present system, the template only describes the subset of the database and the sequence in which the data will be processed. The description of the data entities is located in the data dictionary described in the next section.

The data communication utilities support two types of data - parameter data that consists of scalars and arrays and record data that

simulates FORTRAN file structures. Template specifications for parameter and record data are illustrated in Figure 18 and consist of the following:

1) template name
2) sequence number (data retrieval/storage ordering)
3) relation name
4) data type (parameter or record)
5) number of data items (parameters or attributes)
6) parameter names (or attribute names)
7) number of key attribute values
8) attribute values

where the number of data items specifies the number of parameters selected from the relation or the number of attributes in the relation that comprise the fields of the record. The last three items specify the random access order of processing the relation (FORTRAN file) - the key field, the number of records to be accessed, and the value of the key field in each record to be accessed. If the key attribute name is blank, then the relation is accessed like a sequential FORTRAN file.

Associated with the templates is a template catalog. An example of the catalog is presented in Figure 19. The catalog consists of the following:

1) template name
2) description
3) input, output, or report generator
4) program name
5) custodian's name

6) date last modified

The program name is included to inform the user which template can be used to review the input before the program is executed or review the output after the program has been executed.

Data Dictionary

The data dictionary provides an inventory of all data entities that are stored in the configuration database. It contains a list of all relations in the database, a description of each relation and each entity in the relation. As mentioned in the previous section, two types of relations are currently supported by the data communications utilities: parameter and record. In addition to the data communication utilities, the ARIS can be used to support any data structure that can be modelled with the enhanced relational model with the ARIS communication subroutine library.

Each relation (parameter or record) is described in the data dictionary catalog by:

1) relation name
2) textual description
3) type (parameter, record, or relation)
4) number of parameters or attributes
5) custodian's name
6) number of programs that can create this data
7) creator program names
8) number of programs that use this data
9) user program names
10) date created

Each parameter or data field (attribute) in the record (relation) for each relation is described by:

1) relation name
2) parameter name or data field (attribute) name
3) textual description
4) physical units (pounds, feet, etc.)
5) data type (real, integer, or character)
6) number of characters if character data type
7) dimension (none, 1, 2, or 3 subscripts)
   - 1, 1, 1 for scalars
   - 1, 1, 1 for 1-dimensional arrays
   - 1, m, 1 for 2-dimensional arrays
   - 1 ,m, n for 3-dimensional arrays

Examples of the data dictionary catalog and data dictionary data are given in Figures 20 and 21.

The creator and user program names are used to generate a cross-reference listing between data, and programs can be generated. This list is useful when developing the data communications between new programs and the system. Also, when a program requests input data from the database and that data is missing, this cross-reference list can be used to instruct the engineer where this data can be generated, either from output created by an analysis program, from an external data file, or input by hand through the keyboard. Finally, the data dictionary is used with the interactive data reviewer to identify each data entity through a description, physical units, and data type.

Reviewer

The reviewer is the interface between the user and the configuration database. It is used to review and update the data in the database as specified by the data template.

A typical example of a screen produced by the reviewer (Fig. 22) shows that two versions of the data are presented along with the description and the physical units of the data. The description and physical units are provided by the data dictionary (Fig. 21).

Parameter reviewer. - Rather than inventing a system that was unfamiliar to the users, a line editor format was selected for the parameter reviewer that is similar to the text editors that exist on the host computer system. A typical reviewer screen of information is shown in Figure 22.

To move from screen to screen, the command

Nn

is used. The value n can be positive or negative. To go to the next screen, the command N is used. To review two screens from the current screen, the command N2 is used. To see the last screen a large number for n is used, and to go to the first screen, a large negative is used.

To compare the differences between two separate databases, the command

DIFFON

is used. Only the lines in which there is a difference in value between the present and old values will be displayed. All lines can be displayed by turning off the difference command with the

DIFFOFF

command.

To change data, the following commands are used:

```
*,P,0       --- change all present values to the old values

2,P,0       --- change present value to old value on line 2

4,P,55      --- change present value to 55 on line 4

6,P,'NEW'   --- change present value to 'NEW' on line 6

8,P,M       --- modify present value on line 8
```

With the modify command, the value is displayed and can be modified by the keyboard by typing below the value. As examples:

```
 124.65E-02      --- displayed on screen
? 9      3       --- 9 and 3 keyboard input
 129.65E-03      --- resulting change
```

and

```
 129.65E-03      --- displayed on screen
?     ###        --- delete characters
 129.653         --- resulting change
```

and

```
 129.653         --- displayed on screen
? ^11<           --- insert '11'
 11129.653       --- resulting change
```

For character data type, only the first 16 characters are displayed on a reviewer screen. To view the entire character string, the modify command is used to display up to 132 characters.

Other commands include:

```
R       --- Redisplay the current screen

Q       --- Quit editing and do not save changes

E       --- End editing and save changes
```

A number of other commands have been identified but have not been implemented. These commands include value arithmetic (multiplication, division, addition, and subtraction), change values in a range of lines or elements in a array, and value search. An example of the parameter reviewer will be presented in the section entitled Sample Problem.

Record reviewer. - The record reviewer is very similar to the parameter reviewer, but the data is presented in columns across the screen (Fig. 23). There is a problem in presenting record data because only 7 columns of data can be presented on a 132 column screen and and 5 columns on a 80 column screen, assuming a 16 field width minimum for real data. Thus, the reviewer is used much like a typical spreadsheet program such as Visicalc or Lotus 1-2-3.

To display the desired columns of information the following commands are used:

```
Cn          --- displays the relative columns numbers
                from present where:
                   n =  1, displays the next group of
                           columns
                     = -1, displays the previous group
                           of columns
                     = 99, displays the last group of
                           columns
                     =-99, displays the first group of
                           columns
D,1,9,2,-,4 --- displays columns 1, 9, 2, 3, and 4
```

The data editing commands are similar to the parameter editor commands except the column number(s) must be specified:

```
*,P,0      --- change all present line and column
               values to old
1,P,0,*    --- change all present values to old in
               line 1
2,P,0,3    --- change present value to old value in
               line 2/column 3
2,P,3,55   --- change present value to 55 in
               line 2/column 3
```

To display and update character strings greater than 16 characters, the modify command must be used.

The other commands to redisplay the screen, quit, and end are identical to the parameter commands.

Reviewer uses. - There are many uses for the reviewer. First, the data dictionary is the template for the entire database. Thus it can be

used by the database administrator or an engineer to review the complete database or compare versions of the database: his personal versions, his version with other specialists, or his versions with permanent versions that reflect various stages of a large project. Second, any template (program input or program output) can be used to review the data and compare it to previous cases. Finally, summary templates can be constructed and used with the reviewer for final reports or management information that needs to be reviewed interactively.

Formatter

The formatter was originally viewed as a utility that used a template to specify data to be retrieved from the database and then automatically create an input file for an analysis program - a general system pre-processor.

After reviewing the data requirements of the various application programs, the data generated by one analysis program was usually in the wrong form for input to the next program. In many cases the data must be converted to different physical units, materialized from existing data (e.g. gross weight is the sum of component and fuel weights), or completely restructured like geometry data that is different for almost every application.

Because the transformations and conversions of data can be very complicated, the logic of a computer program is needed. Because of this complexity, the idea of an automatic formatter for input file creation was dropped.

The current formatter is a program that generates FORTRAN code that can be used by any program to retrieve data from the configuration

as desired. The data template is used to define the data and the retrieval sequence. The data dictionary is used to specify the data structure and data type. The formatter uses the template and data dictionary information to automatically create code for communicating data between the configuration database and the application program, thus freeing the system administrator from writing the code by hand using the ARIS subroutine library. Kenny Jones from Computer Sciences Corporation developed the precompiler program.

Figure 24 shows an example of an input subroutine developed utilizing the FORTRAN code generated using the template T_EXAMPLE (Fig. 18) and the data dictionary (Fig. 21). As shown in this subroutine, all the code to specify the data types and database retrieval is generated. The implementing programmer must integrate this data with the analysis program. A labeled common statement was used to communicate this retrieved information in the program. Other techniques could have been used such as including parameters in the subroutine statement or writing a file that is read by another subroutine in the analysis program.

Problems can arise when the variable names in the program are different than the data names in the configuration database. Currently, the problem is being solved by changing all the pre-compiled variable names to the program names using the host source editor. A more elegant approach would be to add the program variable names to the template specification so the formatter could make this name substitution automatically. Other problems such as data transformations/conversions from the configuration database to the application program must be implemented by the system programmer.

The use of the formatter in program interfacing and integration is shown in Figure 25. For program interfacing, the formatter is used to create the code for retrieving the configuration data. The pre-processor program transforms and converts this data (if necessary), and then creates an input file for the analysis program. The analysis program reads this input, computes, and creates a results file. The formatter is also used to create code for the post-processor program to store data into the configuration database. The post-processor program reads the results file, transforms and converts the data, and stores the results into the configuration database.

The integrated program coupling is similar to the interfaced coupling, but all the database communications and data transformations/conversion subroutines are located in the application program.

IMPLEMENTATION

As shown in the CAE system architecture (Fig. 14) the majority of the system consists of data. Thus, a large effort was devoted to the development of the data management system (ARIS) that supports the data system - configuration data and meta data.

The global database in the CAE system is implemented as a directory of files in the host computer. The directory consists of the meta database which describes the other files in the directory: the configuration databases, the program files, and the library files (Fig. 26). The meta database consists of relations defined for the program library (Fig. 9), the procedure library (Figs. 10 and 11), the configuration database catalog (Fig. 12), the data dictionary (Figs. 20 and 21), the template library (Figs. 18 and 19) and the user catalog.

The program files consist of the source file, the executable run file, and the procedure file that is used to build the executable run file. The library files consist of the source files of the library used by the analysis programs that are not supported by the host computer system administrator.

The configuration databases are implemented as separated ARIS databases. They contain all the data that is described by the data dictionary (Figs 20 and 21).

The following procedure illustrates how a new program is integrated into the system (also shown schematically in Figure 27). The CAE system administrator must do the following:

1) Define input and output requirements of the program.

2) Check the data dictionary for compatible input and output data with existing data entities.

3) Develop the definitions for any new relations (attributes and/or records) that are needed.

4) Enter the relation definitions and data entity descriptions into the data dictionary.

5) Enter program source, executable run file, procedure file, and library source if needed to the global directory.

6) Enter program description into the program library.

7) Enter execution procedure and description into the procedure library.

8) Develop input and output templates, and enter the templates and description into the template library.

At this time the ARIS and formatter are used to do the following:

1) To create a golden database (proven input/output data values for program demonstration), compile a configuration database, copy the golden example into the new database, and place null values into the newly defined data entities.

2) Enter the golden database into the global directory and update all previous configuration databases.

3) Precompile input and output code using the formatter.

The system administrator must now complete the following:

1) Complete the development of the input and output routines (incorporate the precompiled code into the application program if necessary).

2) Enter programs and definitions into the program library if pre- or post-processors were developed.

3) Enter new data values into the golden configuration database using the reviewer and/or program execution.

4) Execute new program and check results.

The local databases now must be updated with the new changes. The local databases are implemented similarly to the global databases in which a directory is created for each user. The user environment consists of a configuration library which consists of copies of the global configuration databases and a user description of these copies, subsets of the global templates with a user description, personalized procedures

developed by combining global procedures, and a log for design session comments.

SAMPLE PROBLEM

A relatively trivial problem is presented to demonstrate the many features of the integration system. The programs used (Fig. 9) are:

1) GEO_DIG - digitize an aircraft shape from an engineering drawing with an interactive graphics tablet

2) HABFRMT - convert digitized geometry to Hypersonic Arbitrary Body (HAB) geometry format[38]

3) IMAGE[39] - plot HAB format vehicle

4) GEO_PROP - compute the HAB geometric properties: areas, volumes, center-of-area location, area moments and products of inertia

5) WTS_BAL - compute weights and center-of-gravity location

6) HYPERPRE - preprocessor to create an input file for HYPER

7) HYPER - compute hypersonic aerodynamics

The program and data flow are shown in Figure 28. The body and wing geometry is digitized from an engineering drawing with a graphics tablet using the program GEO_DIG. The HABFRMT program is then used to convert the digitized data to HAB format to be used by two standard programs: IMAGE for plotting the panelled vehicle and GEO_PROP for computing the geometric characteristics. By applying a unit weight (pound per square foot) distribution on the surface areas of each of the components, the total vehicle weight and center-of-gravity location of the total vehicle can be computed with WTS_BAL. A very simple hypersonic aerodynamics program, HYPER, that uses vehicle geometric parameters (wing area, leading edge sweep, trailing edge sweep, etc.) is

finally used to compute the trim conditions of the vehicle. To create the input file for HYPER, the preprocessor HYPERPRE is used.

The command to start the system is

AIDES AWW

where AIDES is the present Aerospace Integrated Design and Engineering System and AWW are the users initials. If the initials are not found in the users catalog (Fig. 14), then the user information must be entered into the system.

The current databases and the last log entry are then displayed:

```
CURRENT ACTIVE DATABASES:

   PRESENT = MARTIN2 - MMC1 WITH   NEW WING
   OLD     = MARTIN  - MARTIN TASK II SSTO

LAST LOG ENTRY > CCV TECHNOLOGY WEIGHTS - 01/10/84


** enter return to continue **
```

The data for the active databases and log is found in the database catalog (Fig. 12) and the activity log (Fig. 13). The executive menu is displayed after return is entered:

```
**********************
*                    *
* A I D E S EXECUTIVE *
*                    *
**********************


E - TO EXECUTE PROCEDURES

     E, PROCEDURE name1 [, PROCEDURE name2] ...


L - TO LIST PROCEDURES

     L (return)         - to list all procedures
     L, PROCEDURE name  - to list procedure commands
```

U - TO DISPLAY UTILITIES MENU

Q - TO QUIT

>E,GEO_DIG,IMAGE,GEO_PRP,WTS_BAL,HYPER

All the above commands are self explanatory except the utilities menu. The utilities menu consists of commands for:

1) listing the global and local configuration database catalog, procedure catalog, program catalog, or template catalog,

2) making a copy of a configuration database, procedure, or template,

3) editing the local catalogs, log, procedures, or templates,

4) activating a different configuration database (present or old).

The command E followed by the five procedure names executes all the procedures in the present system (Fig. 10). First, the vehicle is digitized using the geometry in Figure 29. The template T_DIG_OUT has been incorporated into the GEO_DIG program for data communications. The next procedure command allows the user to review the digitized data (a simpler example with less data will be presented later). Then the digitized data is transformed into panel geometry and parameter data by the HABFRMT program. Thus, with this geometry procedure, geometry can be manipulated in graphical form in the GEO_DIG program and in digital form by the reviewer. Sometimes this direct data manipulation is necessary because the digitizer and graphics screen resolution are not adequate.

The IMAGE procedure displays the panel geometry (Fig. 30).

The WAB procedure computes the geometric properties. In this procedure, the reviewer is first used to edit the unit weights for each component:

```
********************
*                  *
*      AIDES       *
*                  *
* R E V I E W E R  *
*                  *
********************

Template = T_UNIT_WTS
```

SCREEN 1

L#	P_VALUE	O_VALUE	DESCRIPTION	UNITS
1	8 65	8.87	WNG COMP WT	LB/FT2
2	5.42	5.42	BDY COMP WT	LB/FT2
3	6.42	6.42	TAIL COMP WT	LB/FT2
4	4.55	4.55	BFLP COMP WT	LB/FT2

```
EDIT
>1,P,9.65     (change present value on line 1 to 9.65)
>2,P,6.50     (change present value on line 2 to 6.50)
>R            (re-print the screen)
```

SCREEN 1

L#	P_VALUE	O_VALUE	DESCRIPTION	UNITS
1	9 65	8.87	WNG COMP WT	LB/FT2
2	6.50	5.42	BDY COMP WT	LB/FT2
3	6.42	6.42	TAIL COMP WT	LB/FT2
4	4.55	4.55	BFLP COMP WT	LB/FT2

```
EDIT
>E        (save changes and end reviewer session)
```

After the unit weights are edited, the geometric properties of the

vehicle (volumes, areas, center-of-gravity locations, and moments of inertia) are computed by the GEO_PRP program and the weight properties (weight and weight center-of-gravity locations) are computed by the WTS_BAL program.

In the hypersonic procedure, HYPER, the reviewer is used to change the geometric parameter data (Fig. 15), the center-of-gravity location, and the hypersonic input parameters. Then the preprocessor, HYPERPRE, creates an input file for the HYPER program called HYPIN. The hypersonic program computes the pitching moment and plots the results as shown in Figure 31.

With this CAE system, the effects of geometry and weight distribution on hypersonic trim can be determined.

DATA INTERDEPENDENCE

One of the major problems in maintaining the integrity of the database in a loosely coupled system is data interdependence. The flow of programs and data in the sample problem is illustrated in Figure 28. Suppose that the geometry has been generated, the weight properties have been computed, and the next step is to review the input data to the HYPER program. Figure 15 shows the geometry parameters. If the wing area, STOTAL, is changed in the reviewer, then the geometry has changed, and all data associated with the geometry data (like weight properties) is now in error because it is based on the previous wing area. This problem of a data entity affecting one or more other data entities is called data interdependence.

Business data systems do not have the degree of flexibility in order of program execution that is needed by an engineering system. The path through the applications in business systems is rigid to maintain the highest degree of data integrity possible. Mechanisms are available to check each data entry for correctness and no data can be entered into the database unless all integrity checks are passed.

In an engineering system, the user should know the upper and lower bounds on a data value, or he should not be using the application program. The usual business data integrity mechanisms of data comparisons against bounds, ranges, or statistics are not very useful and mostly represent extra system overhead.

A data interdependence integrity mechanism, on the other hand, would be very useful in a loosely coupled system as a warning system against improper data. In the above wing area example, the engineer

might have looped through all the programs many times, and learned that the center-of-gravity location did not change significantly as the wing geometry changed. Thus, a quicker iteration loop would be just to change the wing parameters with the reviewer and compute the aerodynamics. Once a wing geometry was found that satisfied the trim condition, then a full loop through all the programs would be completed to verify the final results. In this case, the data interdependence integrity mechanism would be used as a warning system, not a control of the system.

A cursory attempt was made at defining an integrity mechanism for this data interdependence problem. The implementation of the integrity system is based on the flow of data through the programs (Fig. 32). The integrity mechanism assumes that all the output of a program is a function of all the input. Thus, for example, if any data is changed in the unit weights relation, UNIT_WTS, then the data in the weights properties relation, WT_PROP, would have a warning tag. If the HYPERPRE program is executed directly after the unit weights relation has been changed with the reviewer, then a warning would be displayed that the weight properties may be in error, and this error could be corrected be executing the WTS_BAL program. The program/relation relationship is established in the relation catalog (Fig. 20).

In the case where the output of a program is changed, it is assumed that all the output and the input data that created this output are in error. For example, if the geometry characterics, GEO_CHAR, are changed by the reviewer, the relations DIG_OUT and XYZ'S are tagged. Becase the relation XYZ'S is used as input to the program GEO_PRP, then the relation GEO_PROP is tagged. This tagging process is continued through the

data flow of the system. A change in geometry creates a tag on all the data used by the program that need geometry information or information based on geometry.

The data dependency graph (Fig. 32) is defined from the data flow through the programs, which is defined by the creator and the user program list in the relation catalog (Fig.20). Any time data is updated in the database by programs or interactively by a user, the data dependency graph is processed at the appropriate position and the affected relations are tagged.

STATUS

After defining the requirements for the system, all effort was directed towards data management because almost all the functions of the system depended on this development. The ARIS system (in the appendix) was developed in two years. It has been used in the development of several application programs that have been integrated into the system: geometry, mission modelling, rocket engine database, and an aerodynamic database.

A prototype system was developed exclusively around the ARIS system using several orbital transfer vehicle analysis programs.[6] The result of this prototype was that complete database visibility was needed which resulted in the development of the reviewer. Data input into the database was just as important as output; thus, the capability of replacing data with no internal database checking was implemented. Replacements can be made when no more tuples are added to the current relation and the primary key attributes are not changed. This replacement feature speeds storage by 50 percent and happens approximately 75 percent of the time. Finally, it was found that integrity interdependency tags with date and time on individual data entries made the system prohibitively slow with its overhead. The new system being developed uses a compiled interdependency graph and operates at the relation level.

The final details of the user interface are close to completion. Dr. Schwing of Old Dominion University is implementing the utility programs for managing the configuration databases and the interative

command structure for the user. A single-user system should be operational soon.

CONCLUSIONS

An approach for coupling independent engineering programs for design and analysis of aerospace vehicles was developed. It consists of a loosely coupled network of engineering programs that communicate through a relational information system. The system architecture consists of the engineering programs, a user interface for managing the system, a catalog system for maintaining data about the programs, data, and users, individual work areas for each engineer, and a global database that is used as a repository of all engineering data created for central use. A relational information system was developed for this system to communicate engineering data with a FORTRAN language interface. Finally, two utilities were developed to aid in interactively editing the database and communicating data to the analysis programs.

The system architecture provides management of the complete system through utilities to maintain data about the database, programs, and users and an executive for executing programs. The system is completely visible to the user because descriptions of each entity of the system are provided. The global configuration databases can be reviewed and copied, and the local configuration databases can be edited. Flexibility is provided by the data management system because there is minimum impact on the system when new programs or data are added to the system, and the programs can be executed in either the batch or interactive mode in any logical order. Finally, data interdependence is a major problem in loosely coupled systems, and a cursory attempt is made to define an integrity mechanism for alleviating this problem.

REFERENCES

[1]"Space Shuttle Synthesis Program (SSSP), Final Report." General Dynamics Convair Division Report No. GDC-DBB70-002, December 1970.

[2]Garrison, J. M. "Development of a Weight/Sizing Synthesis Computer Program." McDonnell-Douglas Astronautics Company, MDC-E0746, February 1973.

[3]Glatt, C. R. and Hague, D. S. "ODIN--Optimal Design Integration System." NASA CR-2492, February 1975.

[4]Glatt, C. R., Hague, D. S., and Watson, D. A. "Dialog: An Executive Computer Program for Linking Independent Programs." NASA CR-2296, September 1973.

[5]Wilhite, Alan W. "The Aerospace Vehicle Interactive Design System." Presented at the 19th Aerospace Sciences Meeting, AIAA Paper 81-0233, January 12-15, 1981.

[6]Wilhite, Alan W., Johnson, S.C., and Crisp, V. "Integrating Computer Programs for Engineering Analysis and Design." Presented at the AIAA 21st Aerospace Sciences Meeting, AIAA Paper 83-0597, January 10-13, 1983.

[7]Gott, B. "The Scope of Computer-Aided Design," Computer-Aided Design. Proceedings of the IFIP Working Conference on Principles of Computer-Aided Design, North Holland Publishing Co., 1973, pp. 1-18.

[8]Gregory, S. A. "Design and the Design Method," The Design Method. Plenum Publishing Corp., New York, 1966, pp. 3-10.

[9]Heldenfels, R. R. "Integrated, Computer-Aided Design of Aircraft." Presented at the AGARD Conference on Aircraft Design Integration and Optimization, CP-147-Vol.1, Oct. 1973.

[10]Meyer, D. D., Anderton, G. L., and Crowell, H. A. "The Design Process." Presented at the AIAA Aircraft Systems and Technology Conference, AIAA Paper 78-1483, August 21-23, 1978.

[11]Woodson, T. T. Introduction to Engineering. McGraw Hill Book Company, 1966.

[12]Meyers, Ware. "CAD/CAM: The Need for a Broader Focus," Computer, Vol. 15, No. 1, January 1982, pp. 105-117.

[13]Oman, B. H. "Vehicle Design Evalution Program (VDEP)." NASA CR-145070, January 1977.

[14]Gregory, T. J. "Performance Trade-Offs and Research Problems for Hypersonic Transportations." AIAA Journal of Aircraft, July-August 1965.

[15]Roch, A. J. "Missle Integrated Design Analysis Systems (MIDAS)." Presented at the AIAA 19th Aerospace Sciences Meeting, AIAA Paper 81-0285, January 12-15, 1981.

[16]DeBilzan, C. C. and Pickett, H. E. "SIZE: The Aerospace Corportation's Modular Vehicle Design Program." Presented at the AIAA/SAE 11th Propulsion Conference, AIAA Paper 75-1275, September 29 - October 1, 1975.

[17]Wennagel, G. J., Loshigian, H. H., and Rosenbaum, J. D. "RAVES: Rapid Aerospace Vehicle Evaluation System." Presented at the 1975 ASME Winter Annual Meeting, November 30 - December 4, 1975.

[18]Wennagel, G. J., Mason, P. W., and Rosenbaum, J. D. "IDEAS, Integrated Design and Analysis System." (Preprint) 680728, Soc. Automot. Eng., October 1968.

[19]"IPAD: Integrated Programs for Aerospace-Vehicle Design." NASA CP-2143, September 1980.

[20]Leondis, Alex. "Large Advanced Space Systems Computer-Aided Design and Analysis Program," NASA CR-159191, 1980.

[21]Vos, R. G., et al. "Development and Use of an Integrated Analysis Capability." AIAA Paper 83-1017, 1983.

[22]de Kruyf, J., Ferrante, J. G., and Dutto, E. "ESABASE, A Computer-Aided Engineering Framework Facilitating Integrated Systems Design." ESA Journal, Vol. 6, 1982, pp.415-429.

[23]Dror, B. "Computer-Aided Design at Israel Aircraft Industries," Computers & Graphics, Vol 3., Nos. 2/3, 1978, pp. 93-105.

[24]Fulton, R. E., et al. "Application of Computer-Aided Aircraft Design in a Multidisciplinary Environment." Presented at the AIAA/ASME/SAE 14th Structures, Structural Dynamics, and Materials Conference, AIAA Paper 73-353, March 20-22, 1973.

[25]Schwing, J. L. "User Interface for Integrated Computer-Aided Design Systems," NASA Research Grant NCCI-74, June 1984.

[26]Date, C. J. An Introduction to Database Systems. Addison-Wesley Publishing Company, February 1982.

[27]Study Group on Data Base Management Systems. Interim Report. ANSI/X3/SPARC, pp. II16-II28, February 1975.

[28]Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." CACM 13, No. 6, June 1970.

[29]Sidle, Thomas W. "Weaknesses of Commercial Data Base Management SYSTEMS in Engineering Applications." Presented at the 17th Design Automation Conference, 1980, pp. 57-61.

[30]Felippa, C. A. "Database Management in Scientific Computing-I. General Description," Computers & Structures, Vol. 10, 1979, pp. 53-61.

[31]Bandurski, A. E. and Jefferson, D. K. "Data Description for CAD." Presented at the ACM SIGMOD Workshop, 1975.

[32]Grabowski, H. and Eigner, M. "Employing a Relational Data Structure in a CAD System." Proceedings of the Interactive Techniques in Computer Aided Design, September 21-23, 1978, pp. 367-377.

[33]"Engineering and Scientific Data Management." NASA CP-2055, May 18-19, 1978.

[34]Astrahan, M. M. et al. "System R: A Relational Data Base Management System," Computer, Vol. 12, No. 5, May 1979, pp. 42-48.

[35]Stonebraker, M., Wong, E., and Kreps, P. "The Design and Implementation of INGRESS," ACM Transactions of Database Systems, Vol. 1, No. 3, September 1976, pp. 189-222.

[36]Erickson, Wayne J. "RIM -- A Relational Database Management System." CDC VIM 34 Mpls, Control Data Corp., 1981, pp. 1-80 through 1-85.

[37]Jacky, J. P. and Kalet, I. J. "A General Purpose Data Entry Program," Computing Practices, Vol. 26, No. 6, June 1983, pp. 409-417.

[38]Gentry, Arvel E. "The Mark IV Supersonic-Hypersonic Arbitrary Body Program," AFFDL-TR-73-159, November 1973.

[39]Glatt, C. R. "Image: A Computer Code for Generating Picture-Like Images of Aerospace Vehicles." NASA CR-2430, September 1974.

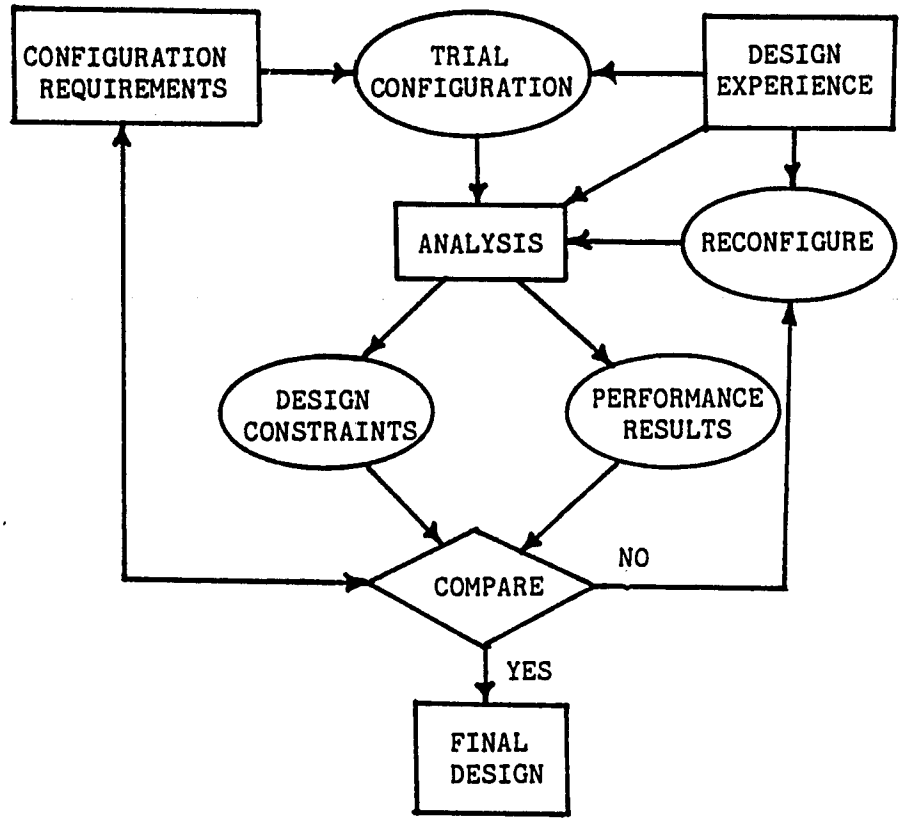


Fig 1 - The design process

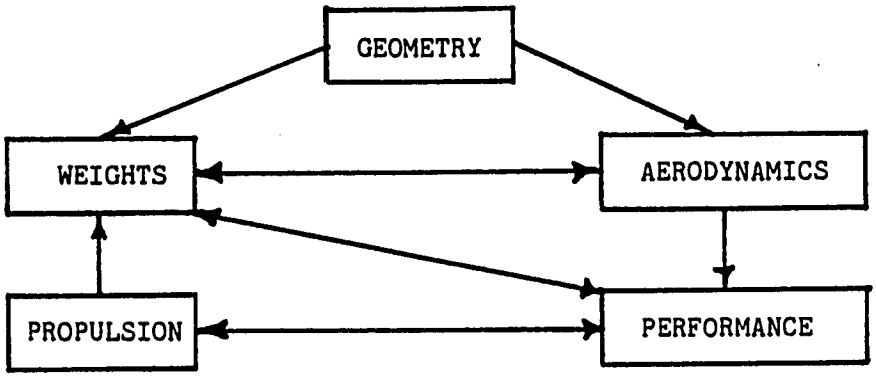


Fig 2 - Data communication between the engineering specialists

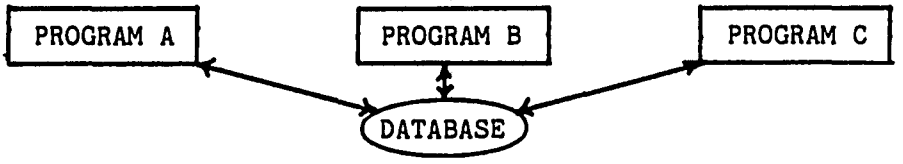
DATA TRANSFERED BY GLOBAL AND DATA FILES

(a) Close-coupled integration

DATA TRANSFER THROUGH PROGRAMS BY INPUT FILES

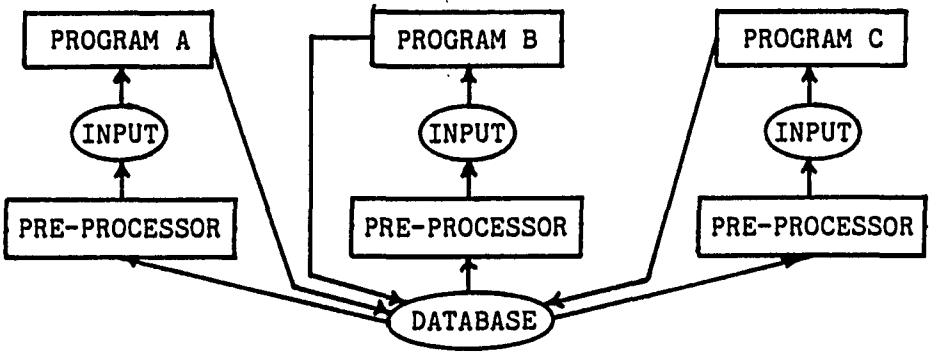
(b) Close-coupled interfacing

DATA TRANSFERED THROUGH A CENTRAL DATABASE



(c) Loose-coupled integration

DATA TRANSFERED THROUGH A PRE-PROCESSOR AND CENTRAL DATABASE



(d) Loose-coupled interfacing

Fig 3 - Program coupling techniques

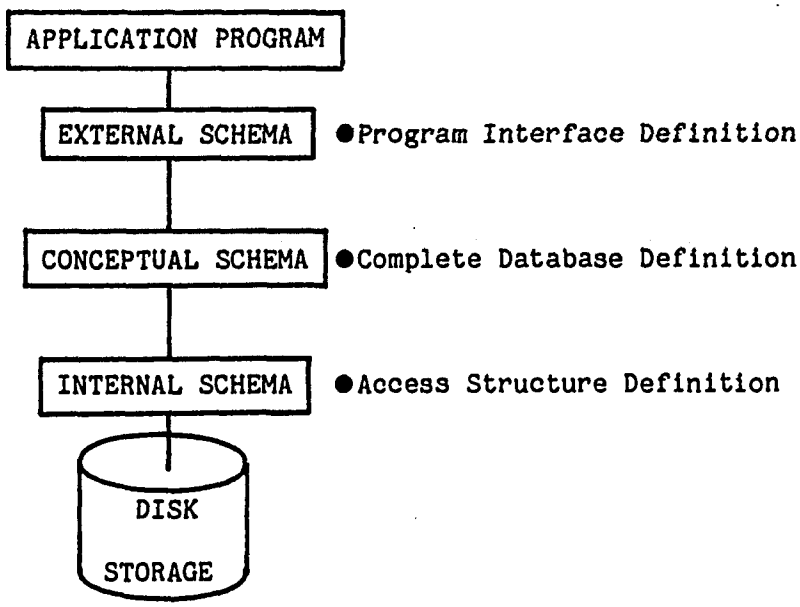


Fig 4 - Three schema database architecture

VEHICLE

V#	VNAME	ENGINEER	DATE
V1	SHUTTLE	REHDER	01/05/75
V2	FIGHTER	NAFTEL	06/17/84
V3	TRANSPORT	CRUZ	11/06/73

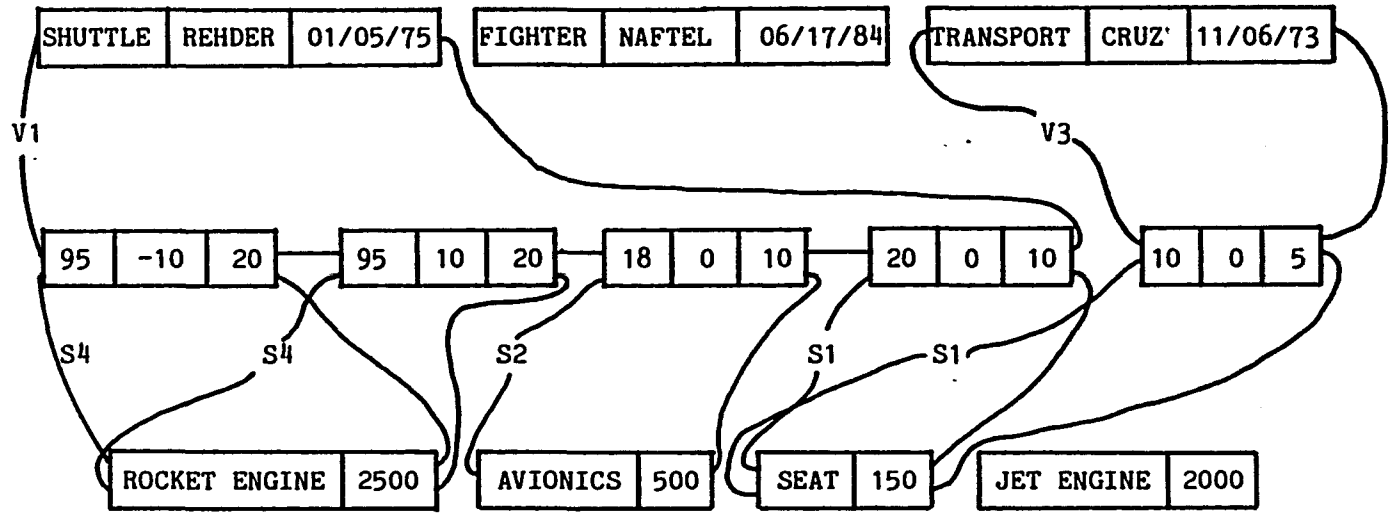
SUBSYSTEMS

S#	SNAME	MASS
S1	SEAT	150
S2	AVIONICS	500
S3	JET ENGINE	2000
S4	ROCKET ENGINE	2500

PACKAGING

V#	S#	XCG	YCG	ZCG
V1	S1	20	0	10
V1	S2	18	0	10
V1	S4	95	-10	20
V1	S4	95	10	20
V3	S1	10	0	5

Fig 5 - Mass properties relational database



RECORDS: VEHICLES, PACKAGING, SUBSYSTEMS
SETS : V#, S#

Fig 6 - Mass properties network database

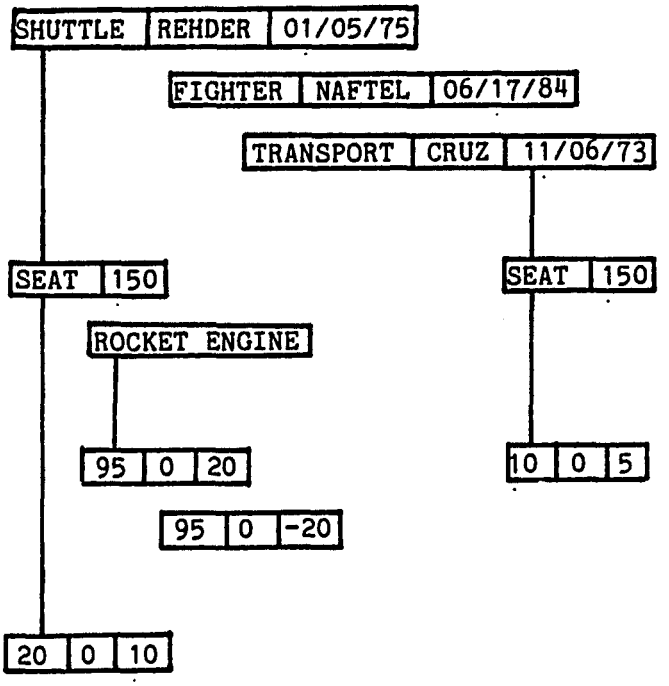


Fig 7 - Mass properties hierarchical database

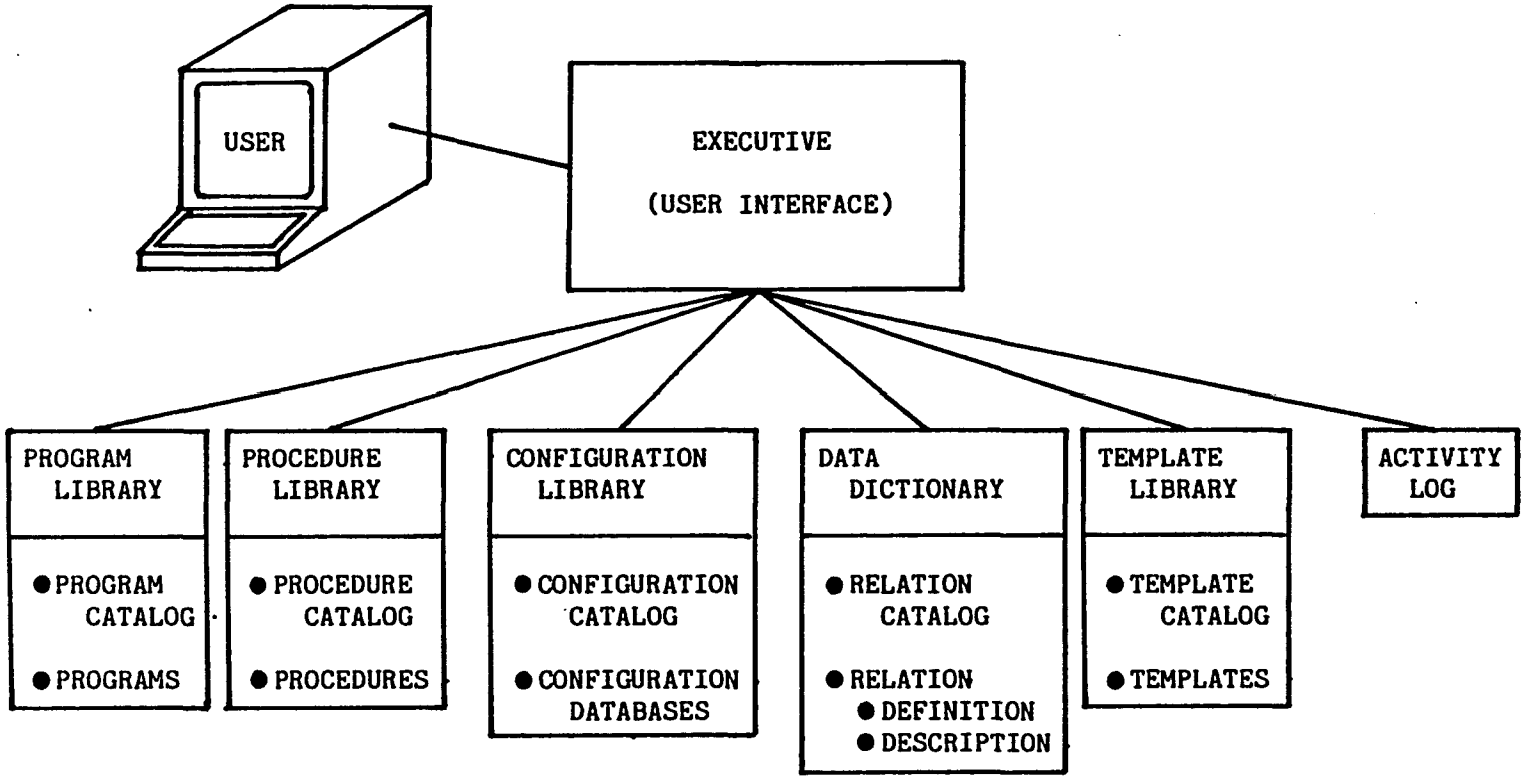


Fig 8 - Single-user CAE architecture

PROGRAM_CATALOG

P_NAME	VER	DATE	DESCRIPTION
GEO_DIG	1	01/06/84	DIGITIZE WING/BODY/TAIL/BODY FLAP
HABFRMT	1	01/06/84	CONVERT DIG_OUT TO HAB FORMAT
IMAGE	1	01/06/84	DISPLAY HAB GEOMETRY
GEO_PRP	1	01/06/84	COMPUTE PROPERTIES FROM HAB GEOM
WTS_BAL	1	01/06/84	QUICKIE WEIGHTS AND BALANCE
HYPERPRE	1	01/06/84	PREPROCESSOR FOR HYPER HYPERPRE.F77
HYPER	1	01/06/84	QUICK HYPERSONICS FROM BOEING

SOURCE	RUN	LIBRARY	PROCEDURE	KEY	NAME
DIGVEH.F77	DIGVEH.SEG	PLOT10	DG	GEOMETRY	AWW
HABFRMT.F77	HABFRMT.SEG		DG	GEOMETRY	AWW
IMAGE.F77	IMAGE.SEG	PLOT10	IMAGE	HAB	AWW
WAB.F77	WAB.SEG		WAB	HAB	AWW
WTBAL.F77	WTBAL.SEG		WAB	WEIGHTS	AWW
HYPERPRE.F77	HYPERPRE.SEG		HYPER	PREPROP	AWW
HYPER.F77	HYPER.SEG	PLOT10	HYPER	AERO	AWW

Fig 9 - Program catalog

PROCEDURES

P_NAME	P#	COMMAND LINE	DESCRIPTION
DG	1	SEG DIGVEH.SEG	EXECUTE DIGITIZING PROGRAM
DB	2	SEG REVIEWER.SEG T_DIG_OUT	REVIEW DIGITIZED OUTPUT
DB	2	SEG HABFRMT.SEG	FORMAT HAB GEOMETRY AND GEOMETRIC PARAMETERS
IMAGE	1	SEG IMAGE.SEG	DISPLAY HAB GEOMETRY
WAB	1	SEG REVIEWER.SEG T_UNIT_WTS	REVIEW UNIT WEIGHTS
WAB	2	SEG WAB.SEG	COMPUTE GEOMETRIC CHARACTERISTICS
WAB	3	SEG WTBAL.SEG	COMPUTE UNIT WEIGHT
WAB	4	SEG REVIEWER.SEG T_WTS_PROP	REVIEW WEIGHTS DATABASE
HYPER	1	SEG REVIEWER:SEG T_G_W_HIN	REVIEW GEOMETRY, WEIGHTS,HYPER AND HYPER INPUT
HYPER	2	SEG HYPERPRE:SEG	CREATE INPUT FILE HYPIN FOR HYPER
HYPER	3	SEG HYPER.SEG	COMPUTE HYPERSONIC TRIM

Fig 10 - Procedure files

PROC_CATALOG

P_NAME	DESCRIPTION	DATE	NAME
DG	DIGITIZE GEOMETRY	01/07/04	AWW
IMAGE	DISPLAY HAB GEOMETRY	01/07/84	AWW
WAB	COMPUTE WEIGHTS AREAS AND BALANCE	01/07/84	AWW
HYPER	COMPUTE HYPERSONIC AERODYNAMICS	01/07/84	AWW

Fig 11 - Procedure catalog

DB_CATALOG

DB_NAME	ORIGIN	M_DATE	DESCRIPTION	NAME	C_DATE
EXAMPLE	SYSTEM	01/05/84	BENCHMARK INPUT DATA	SYSTEM	01/05/84
MARTIN	EXAMPLE	01/08/84	MARTIN TASK II SSTO	AWW	01/06/84
MARTIN2	MARTIN	01/10/84	MMC1 WITH NEW WING	AWW	01/08/84

Fig 12 - Configuration database catalog

ACTIVITY_LOG

USER_ID	SESSION COMMENTS	DB_NAME	DATE	TIME
AWW	START MARTIN STUDY	MARTIN	01/06/84	10:30
AWW	INCREASE BODY FOR P/L	MARTIN	01/06/84	16:20
AWW	COMPLETED P/L STUDY	MARTIN	01/08/84	11:10
AWW	NEW STUDY-RESIZE WING	MARTIN2	01/08/84	16:08
AWW	CCV TECHNOLOGY WEIGHTS	MARTIN2	01/10/84	11:14

Fig 13 - Activity log

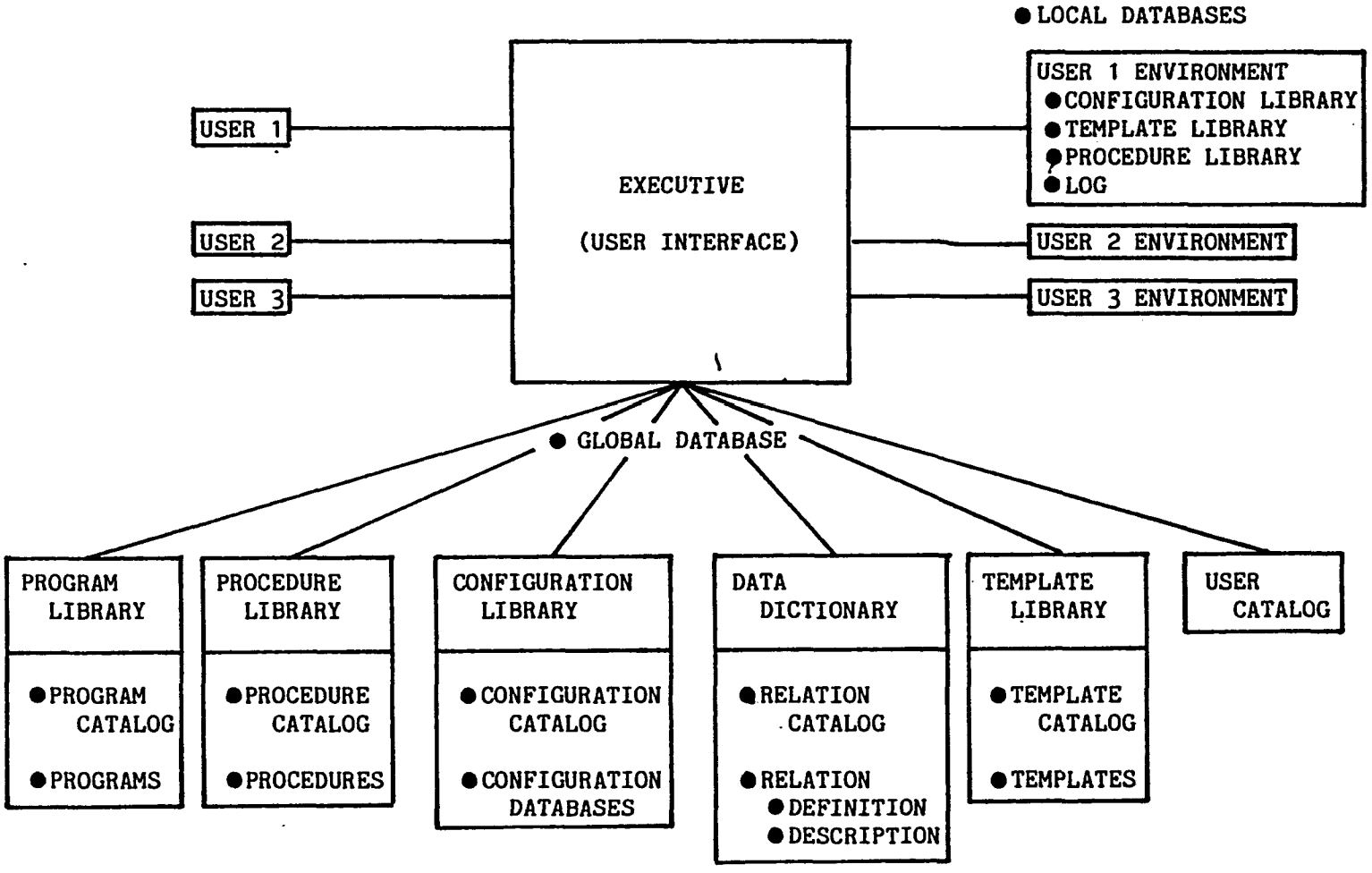


Fig 14 - Multi-user CAE architecture

RELATION GEO_CHAR

P#	PNAME	PVALUE
1	STOTAL	1088.082
2	SWLE	49.5
3	SWTE	20.0
4	TAPER	0.21
5	WALOC	66.11
6	XLBODY	61.9
7	XLNOSE	42.7
8	CAMBER	0.068
9	HBODY	12.12
10	WBODY	18.97
11	POWER	0.43
12	BFL	1.0

Fig 15 - Geometry parameter relation

RELATION XYZ'S

X	Y	Z	STATUS	NAME
0.0000	0.0000	0.0000	3	NOSE
0:0000	0:0000	0:0000	0	NOSE
0:0000	0:0000	0:0000	0	NOSE
0:0000	0:0000	0:0000	0	NOSE
0:5000	0:0000	-0:5000	1	NOSE
0:5000	0:5000	-0:5000	0	NOSE
0:5000	0:5000	0:5000	0	NOSE
0:5000	0:0000	0:5000	0	NOSE
1:0000	0:0000	-0:7500	3	BODY
⋮	⋮	⋮	⋮	⋮

a) Geometry stored in relational form

RELATION XYZ'S

FILE_NAME
GEOM_XYZ

b) Geometry stored in a file referenced by a relation

Fig 16 - Two ways of storing bulk data files in ARIS

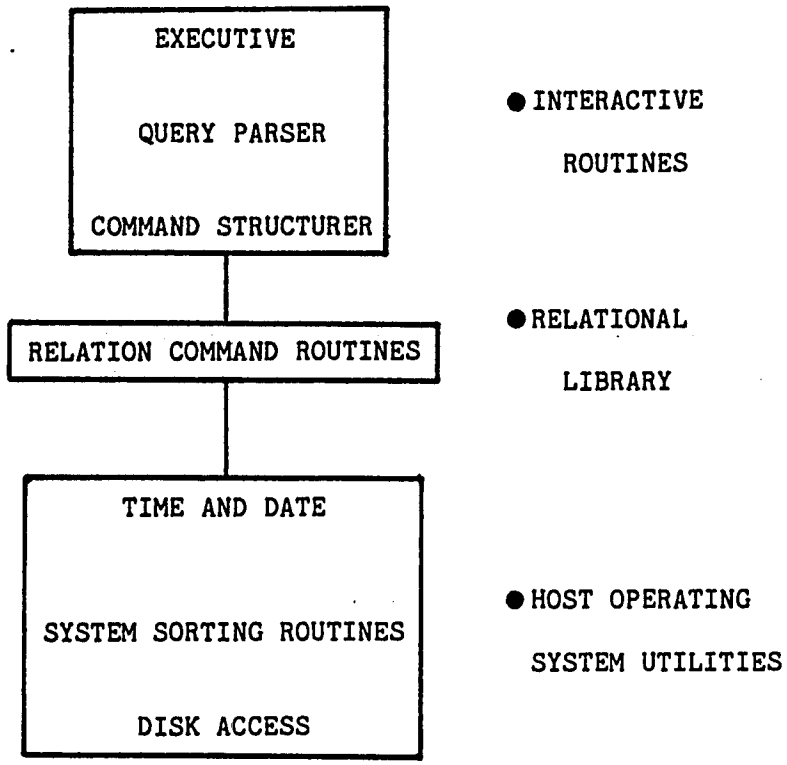


Fig 17 - ARIS software layers

TEMPLATES

T_NAME	SEQ#	RELATIONS	P/R	N_I	ITEMS	N_VAL	VALUES
T_DIGOUT	1	DIG_OUT	PARA	0			
T_HYPGEOM	1	XYZ'S	RECORD	0		0	
	2	GEO_CHAR	PARA	0			
T_HABFILE	1	XYZ'S	RECORD	0		0	
T_UNIT_WTS	1	UNIT_WTS	PARA	0			
T_GEO_PROP	1	GEO_PROP	PARA	0			
T_UNTS_GPRP	1	UNIT_WTS	PARA	0			
	2	GEO_PROP	PARA	0			
T_WT_PROP	1	WT_PROP	RECORD	0		0	
T_G_W_HIN	1	GEO_CHAR	PARA	0			
	2	WT_PROP	RECORD	1	CG	1	TOTAL
	3	HYPER_IN	PARA	0			
T_EXAMPLE	1	WT_PROP	RECORD	0		0	

Fig 18 - Template description

TPLATE_CATALOG

T_NAME	DESCRIPTION	I/O/RG	PROGRAM	NAME	DATE
T_DIG_OUT	DIGITIZE OUTPUT	OUTPUT	GEO_DIG	AWW	01/06/84
		INPUT	HABRFMT	AWW	01/06/84
T_HYPGEOM	HAB GEOM FILE AND PARAMETERS	OUTPUT	HABFRMT	AWW	01/06/84
T_HABFILE	HAB GEOMETRY FILE	INPUT	IMAGE	AWW	01/06/84
		INPUT	GEO_PRP	AWW	01/06/84
T_UNIT_WTS	UNITS WEIGHTS INPUT	REPORT		AWW	01/06/84
T_GEO_PROP	GEOMETRIC PROPERTIES	OUTPUT	GEO_PRP	AWW	01/06/84
T_UWTS_GPRP	UNITS WEIGHTS AND GEOM PROP	INPUT	WTS_BAL	AWW	01/06/84
T_WT_PROP	WEIGHT PROPERTIES	OUTPUT	WTS_BAL	AWW	01/06/84
T_G_W_HIN	GEOM, CG, AND INPUT	INPUT	HYPER	AWW	01/06/84
T_EXAMPLE	RECORD EXAMPLE	REPORT		AWW	01/06/84

Fig 19 - Template catalog

RELATION

REL_NAME	DESCRIPTION	P/A	N_P/A	NAME	NC	C_NAME	NU	U_NAME	DATE
DIG_OUT	DIGITIZING OUTPUT	PARA	28	AWW	1	GEO_DIG	2	IMAGE GEOM_PRP	01/06/84
XYZ'S	HAB GEOMETRY FILE	ATTR	1	AWW	1	HABFRMT	2	IMAGE GEO_PRP	01/06/84
GEO_CHAR	GEOMETRIC CHARACTERISTICS	PARA	12	AWW	1	GEO_DIG	1	HYPER	01/06/84
GEO_PROP	GEOMETRIC PROPERTIES	ATTR	8	AWW	1	GEO_PRP	1	WTS_BAL	01/06/84
UNIT_WTS	COMPONENT UNIT WEIGHTS	PARA	2	AWW	1	USER	1	WTS_BAL	01/06/84
WT_PROP	WEIGHTS PROPERTIES	ATTR	3	AWW	1	WTS_BAL	1	HYPER	01/06/84
HYPER_IN	HYPER USER INPUT	PARA	2	AWW	1	USER	1	HYPER	01/06/84

Fig 20 - Relation catalog

REL_NAME	P/A_NAME	DESCRIPTION	UNITS	TYPE	NCHAR	DIM_1	DIM_2	DIM_3
DIG_OUT	NCROSS	NUMBER OF X-STATIONS	NA	INT	0	1	1	1
DIG_OUT	XCROSS	CROSS SECTION STATIONS	FT	REAL	0	20	1	1
DIG_OUT	NCRPTS	NUMBER OF CROSS SECTION POINTS	NA	INT	0	1	1	1
DIG_OUT	YCROSS	Y_CROSS SECTION POINTS	FT	REAL	0	20	20	1
DIG_OUT	ZCROSS	Z-CROSS SECTION POINTS	FT	REAL	0	20	20	1
DIG_OUT	NPLAN	NUMBER OF PLANFORM POINTS	NA	INT	0	1	1	1
DIG_OUT	XPLAN	X-PLANFORM POINTS	FT	REAL	0	20	1	1
DIG_OUT	PLAN	Y-PLANFORM POINT	FT	REAL	0	20	1	1
DIG_OUT	NSIDET	NUMBER OF SIDE TOP POINTS	NA	INT	0	1	1	1
DIG_OUT	XSIDET	X-SIDE TOP POINTS	FT	REAL	0	20	1	1
DIG_OUT	ZSIDET	Z-SIDE TOP POINTS	FT	REAL	0	20	1	1
DIG_OUT	NSIDEB	NUMBER OF SIDE BOTTOM POINTS	NA	INT	0	1	1	1
DIG_OUT	XSIDEB	X-SIDE BOTTOM POINTS	FT	REAL	0	20	1	1
DIG_OUT	ZSIDEB	Z-SIDE BOTTOM POINTS	FT	REAL	0	20	1	1
DIG_OUT	NWING	NUMBER OF WING PLANFORM POINTS	NA	INT	0	1	1	1
DIG_OUT	XWING	X-WING PLANFORM POINTS	FT	REAL	0	12	1	1
DIG_OUT	YWING	Y-WING PLANFORM POINTS	FT	REAL	0	12	1	1
DIG_OUT	ZWING	Z-WING PLANFORM POINTS	FT	REAL	0	12	1	1
DIG_OUT	NWAIRF	NUMBER OF WING AIRFOIL POINTS	NA	INT	0	1	1	1
DIG_OUT	XWAIR	X-AIRFOIL POINT	NA	REAL	0	20	1	1
DIG_OUT	ZWAIR	Z-AIRFOIL POINT	NA	REAL	0	20	1	1
DIG_OUT	NTAIL	NUMBER OF TAIL PLANFORM POINTS	NA	INT	0	1	1	1
DIG_OUT	XTAIL	X-TAIL POINTS	FT	REAL	0	12	1	1
DIG_OUT	YTAIL	Y-TAIL POINTS	FT	REAL	0	12	1	1
DIG_OUT	ZTAIL	Z-TAIL POINTS	FT	REAL	0	12	1	1
DIG_OUT	NTAIRF	NUMBER OF TAIL AIRFOIL POINTS	NA	INT	0	1	1	1
DIG_OUT	XTAIRF	X-TAIL AIRFOIL POINTS	NA	REAL	0	20	1	1
DIG_OUT	ZTAIRF	Z-TAIL AIRFOIL POINTS	NA	REAL	0	20	1	1
XYZ'S	FILENAME	HAB GEOMETRY FILE NAME	NA	FILE	80	1	1	1

Fig 21 - Data description

REL_NAME	P/A_NAME	DESCRIPTION	UNITS	TYPE	NCHAR	DIM_1	DIM_2	DIM_3
GEO_CHAR	STOTAL	WING REFERENCE AREA	FT2	REAL	0	1	1	1
GEO_CHAR	SWLE	WING LEADING SWEEP	DG	REAL	0	1	1	1
GEO_CHAR	SWTE	WING TRAILING SWEEP	DG	REAL	0	1	1	1
GEO_CHAR	TAPER	WING TAPER RATIO(CTIP/CROOT-REF)	NA	REAL	0	1	1	1
GEO_CHAR	WALOC	THEORETICAL WING APEX LOCATION	FT	REAL	0	1	1	1
GEO_CHAR	XLBODY	TOTAL BODY LENGTH	FT	REAL	0	1	1	1
GEO_CHAR	XLNOSE	BODY NOSE LENGTH	FT	REAL	0	1	1	1
GEO_CHAR	CAMBER	BODY CAMBER	FT	REAL	0	1	1	1
GEO_CHAR	HBODY	BODY HEIGHT	FT	REAL	0	1	1	1
GEO_CHAR	WBODY	BODY WIDTH	FT	REAL	0	1	1	1
GEO_CHAR	POWER	POWER LAW BODY FACTOR	NA	REAL	0	1	1	1
GEO_CHAR	BFL	BODY FLAP LENGTH	FT	REAL	0	1	1	1
GEO_PROP	C_NAME	COMPONENT NAME	NA	CHAR	8	1	1	1
GEO_PROP	S_AREA	SURFACE AREA	FT2	REAL	0	1	1	1
GEO_PROP	F_AREA	MAXIMUM FRONTAL AREA	FT2	REAL	0	1	1	1
GEO_PROP	S_AREA	SIDE AREA	FT2	REAL	0	1	1	1
GEO_PROP	P_AREA	PLANFORM AREA	FT2	REAL	0	1	1	1
GEO_PROP	VOLUME	INTERNAL VOLUME	FT3	REAL	0	1	1	1
GEO_PROP	CG	XYZ CENTER-OF-AREA LOCATION	FT	REAL	0	3	1	1
GEO_PROP	I	AREA MOMENTS/PRODUCTS OF INERTIA	FT2	REAL	0	6	1	1
UNIT_WTS	C_NAME	COMPONENT NAME	NA	CHAR	8	1	1	1
UNIT_WTS	C_U_WTS	COMPONENT UNIT WEIGHTS	LB /FT2	REAL	0	1	1	1
WT_PROP	C_NAME	COMPONENT NAME	NA	CHAR	8	1	1	1
WT_PROP	WEIGHT	COMPONENT WEIGHT	LB	REAL	0	1	1	1
WT_PROP	CG	XYZ CENTER-OF-GRAVITY LOCATION	FT	REAL	0	3	1	1
HYPER_IN	PSTAG	NEWTONIAN COEFFICIENT	NA	REAL	0	1	1	1
HYPER_IN	PRINT	PRINT FLAG (=YES FOR PRINTING)	NA	CHAR	4	1	1	1

Fig 21 (con't)

```
**********************
*                    *
*       AIDES        *
*                    *
* R E V I E W E R    *
*                    *
**********************
```

Template = T_UNIT_WTS

SCREEN 1

L#	C_VALUE	O_VALUE	DESCRIPTION	UNITS
1	8 65	8.87	WNG COMP WT	LB/FT2
2	5.42	5:42	BDY COMP WT	LB/FT2
3	6.42	6:42	TAIL COMP WT	LB/FT2
4	4:55	4:55	BFLP COMP WT	LB/FT2

```
EDIT
>1,P,9.65     (change present value on line 1 to 9.65)
>2,P,6:50     (change present value on line 2 to 6:50)
>R            (re-print the screen)
```

SCREEN 1

L#	C_VALUE	O_VALUE	DESCRIPTION	UNITS
1	9 65	8.87	WNG COMP WT	LB/FT2
2	6.50	5:42	BDY COMP WT	LB/FT2
3	6:42	6.42	TAIL COMP WT	LB/FT2
4	4:55	4:55	BFLP COMP WT	LB/FT2

```
EDIT
>E        (save changes and end reviewer session)
```

Fig 22 - Parameter reviewer example

```
******************
*                *
*      AIDES     *
*                *
* R E V I E W E R *
*                *
******************
```

Template = T_EXAMPLE

COLUMN	DIMENSION	NAME	DESCRIPTION	UNITS
1		C_NAME	COMPONENT NAME	
2		WEIGHT	COMPONENT WEIGHT	LB
3	(3)	CG	XYZ CENTER-OF-GRAVITY	FT

SCREEN 1

COL#	1	2	3	4	5
L#	C_NAME	WEIGHT	CG(1)	CG(2)	CG(3)
1P	BODY	2463.0	62.0	0.0	2.05
10	BODY	2463.0	62.0	0.0	2.05
.		.	.	.	.
2P	WING	486.0	68.0	0.0	-3.05
20	WING	436.0	68.0	0.0	-3.05
.		.	.	.	.
3P	TAIL	127.0	85.0	0.0	4.36
30	TAIL	127.0	85.0	0.0	4.36
.		.	.	.	.
4P	TOTAL	3076.0	63.9	0.0	1.32
40	TOTAL	3026.0	63.8	0.0	1.41

```
EDIT
>1,P,3,2555  (change present value on line 1/column 3 to 2555)
>D,1,2       (display columns 1 and 2)
>R           (re-print the screen)
```

Fig 23 - Parameter or relation reviewer example

SCREEN 1

COL#	1	2
L#	C_NAME	WEIGHT
1P	BODY	2555.0
10	BODY	2463.0
2P	WING	486.0
20	WING	436.0
3P	TAIL	127.0
30	TAIL	127.0
4P	TOTAL	3076.0
40	TOTAL	3026.0

```
EDIT
>E        (save changes and end reviewer session)
```

Fig 23 (con't) - Record or relation reviewer example

```
      SUBROUTINE INPUT
C
C     ROUTINE TO INPUT DATA INTO THE HYPERSONIC PROGRAM
C
C
      COMMON / C_INPUT / STOTAL, SWLE, SWTE, TAPER, WALOC, XLBODY,
     +                   XLNOSE, CAMBER, HBODY, WBODY, POWER,BFL,
     +                   PSTAG, PFLAG, XCG
C                                                              /*AIDE
C           TEMPLATE T_G_W_HIN                                 /*AIDE
C                                                              /*AIDE
C           RELATION GEO_CHAR                                  /*AIDE
      REAL STOTAL, SWLE, SWTE, TAPER, WALOC, XLBODY,XLNOSE     /*AIDE
     +     CAMBER, HBODY, WBODY, POWER,BFL                     /*AIDE
C           RELATION WT_PROP                                   /*AIDE
      DIMENSION CG(3)                                          /*AIDE
C           RELATION HYPER_IN                                  /*AIDE
      REAL PSTAG                                               /*AIDE
      CHARACTER*4 PFLAG                                        /*AIDE
C                                                              /*AIDE
C                                                              /*AIDE
      CALL DBIN1(STOTAL, SWLE, SWTE, TAPER, WALOC, XLBODY,     /*AIDE
     +          XLBODY, CAMBER, HBODY, WBODY, POWER,BFL)       /*AIDE
      CALL DBIN2(CG,NRECORDS)                                  /*AIDE
      XCG = CG(1)
      CALL DBIN3(PSTAG,PFLAG)                                  /*AIDE
      RETURN
      END
```

Fig 24 - Precompiled code incorporated with an input subroutine

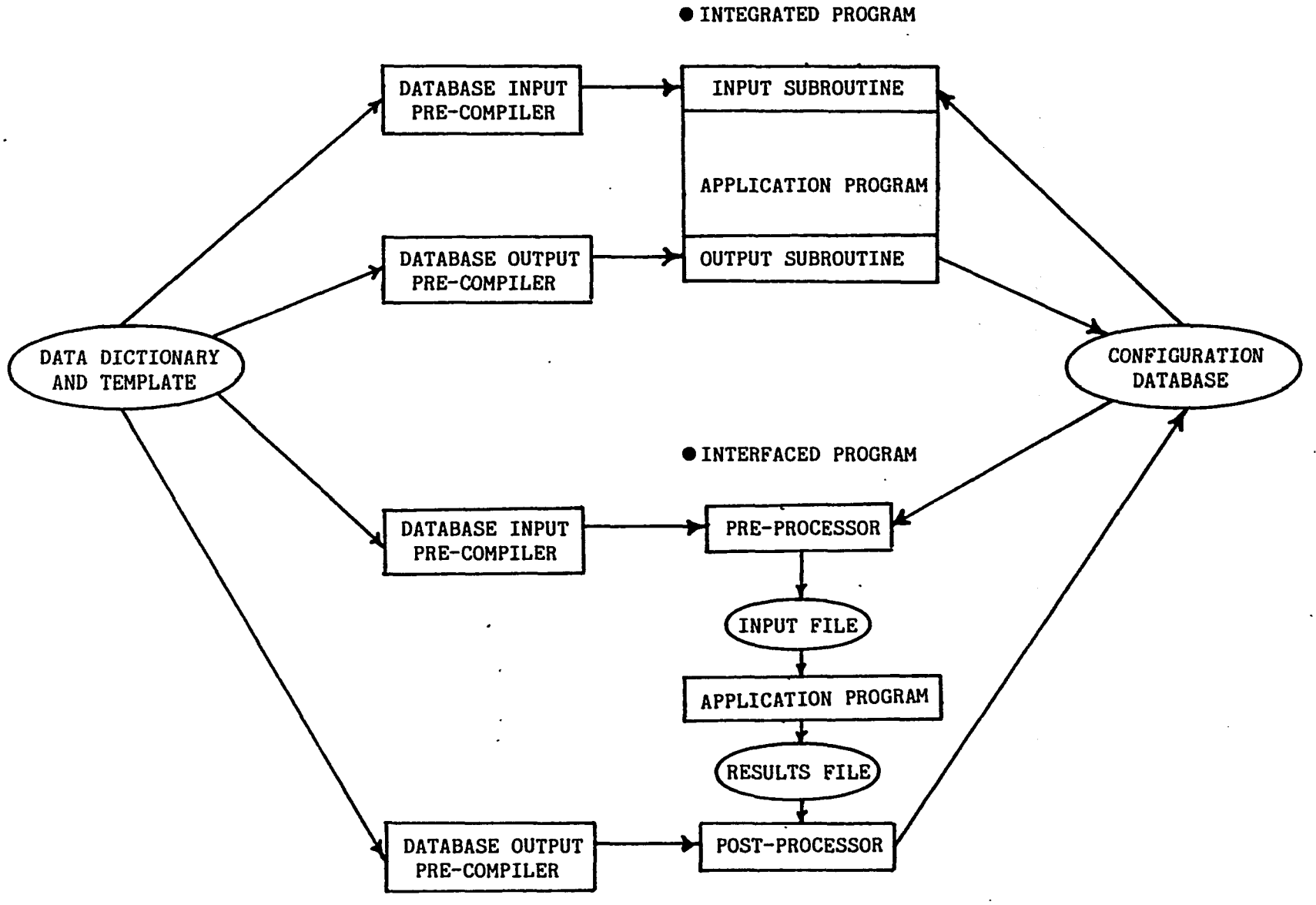


Fig 25 - Program integration and interfacing with the formatter

```
META DATABASE
     PROGRAM CATALOG
     PROCEDURE LIBRARY
     CONFIGURATION CATALOG
     DATA DICTIONARY
     TEMPLATE LIBRARY
     USER CATALOG


CONFIGURATION DATABASE 1

CONFIGURATION DATABASE 2

CONFIGURATION DATABASE 3
          .
          .
          .
          .

 PROGRAM FILES
     PROGRAM1.SOURCE
     PROGRAM1:SEG
     PROGRAM1:PROC
     PROGRAM2:SOURCE
     PROGRAM2:SEG
     PROGRAM2:PROC
          .
          .
          .
          .

 LIBRARY FILES
     PLOT10.SOURCE
     DI3000:SOURCE
          .
          .
          .
          .
```

Fig 26 - Global database directory

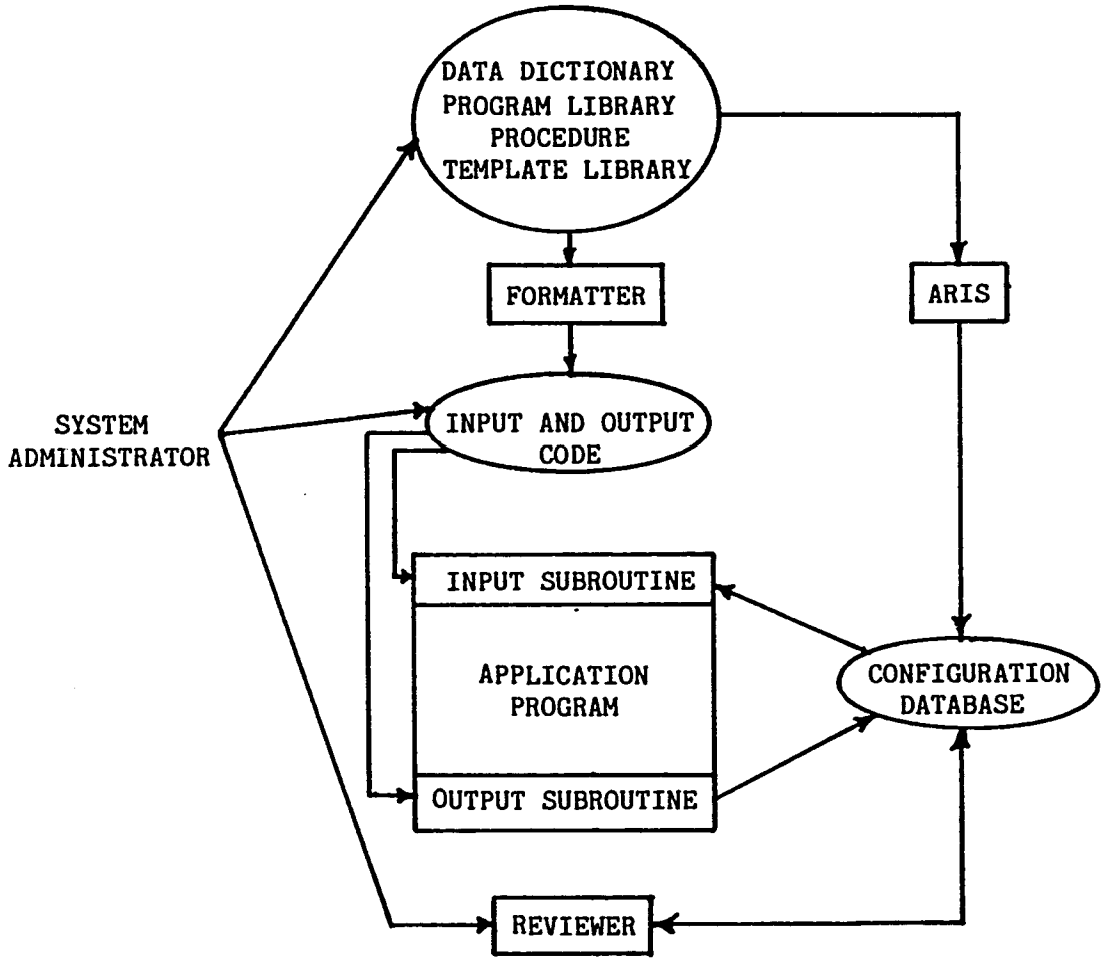


Fig 27 - Installing a new program into the CAE system

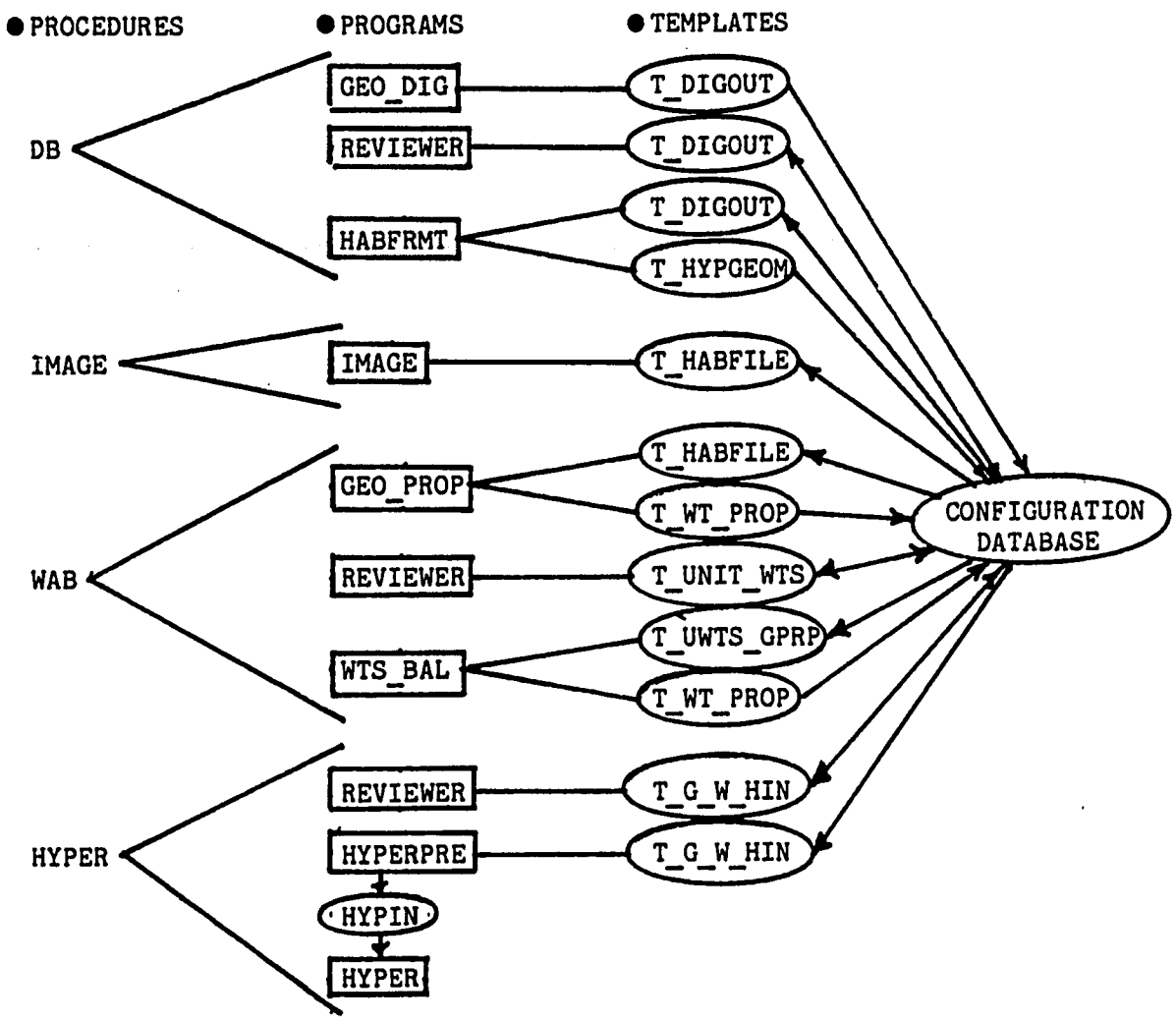


Fig 28 - Program and data flow

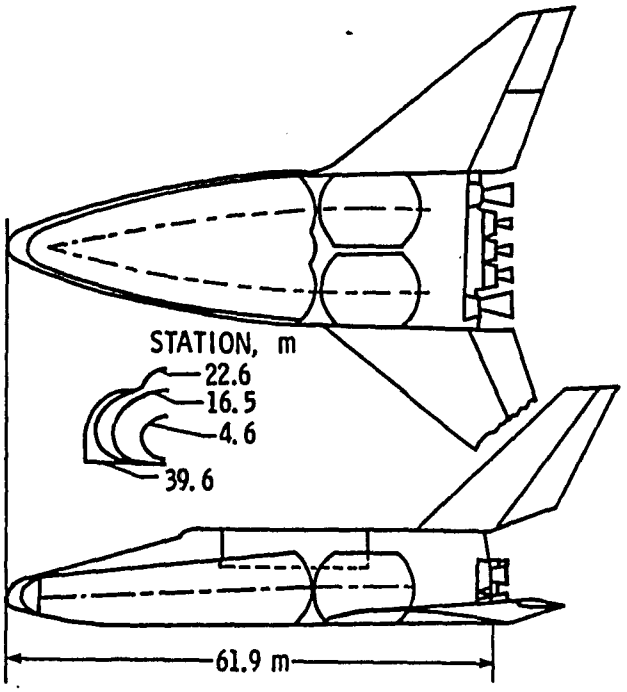


Fig 29 - Engineering drawing to be digitized

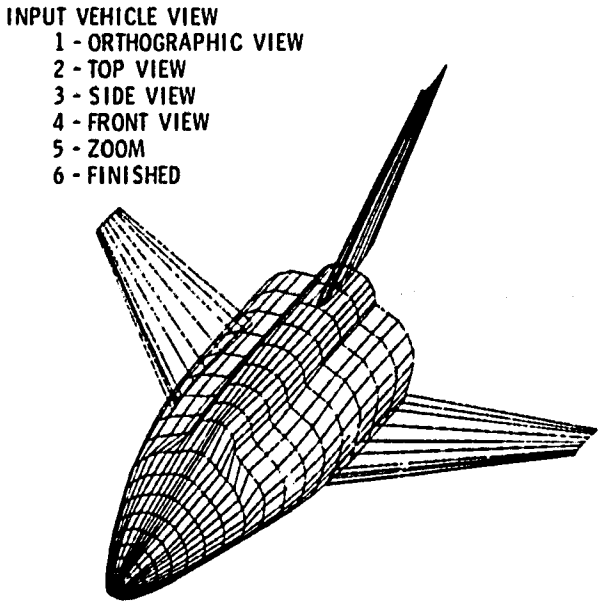


Fig 30 - Vehicle panel geometry

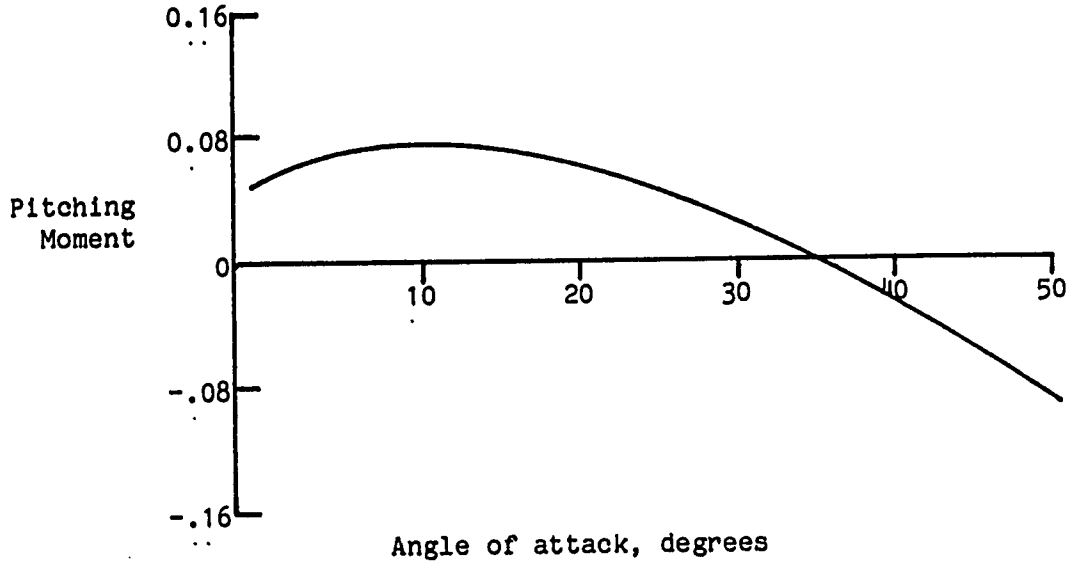


Fig 31 - Vehicle trim aerodynamics

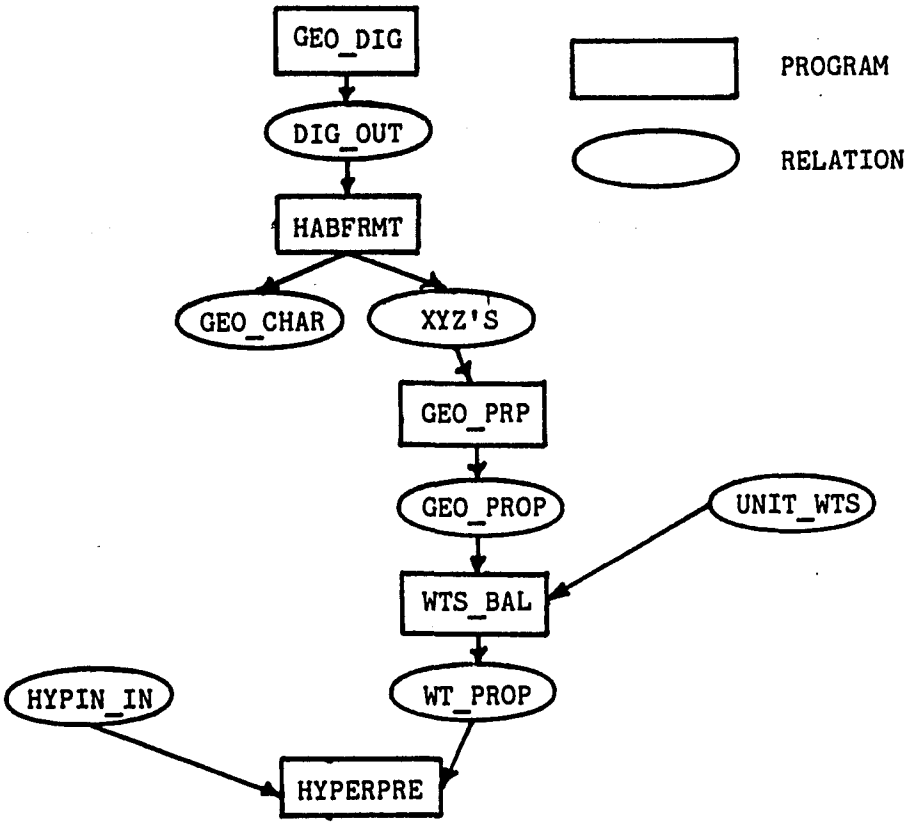


Fig 32 - Program and relation data flow (data dependency graph)

APPENDIX

A RELATIONAL INFORMATION SYSTEM

Relational Data Model

Currently, there are three basic data models used to structure data in database management systems (Refs. A1 and A2). The most recent data model to be introduced, the relational model, is the only one based on mathematical theory.[A3] It is becoming a widely accepted data model since data can be easily structured and manipulated. Also it is easier to understand than the other models, and data is retrieved by simple queries. Based on the evaluation of current data management systems, it was decided to develop a general-purpose relational system which is called A Relational Information System (ARIS) to support the computer-aided engineering framework.

The relational model is based on set theory. Instead of explaining the relational model in strict mathematical terms, a simplistic approach is taken to describe the current relational implementation.

As shown in Figure A1, the relation, TEST, can be represented as a table. The relation consists of six attributes (or columns). The degree of the relation is equal to the number of attributes, e.g., the degree of TEST is six. The number of tuples (or rows in the table) is called the cardinality of the relation, e.g., the cardinality of TEST is 5.

To review and manipulate relations, a query language was developed (based on relational algebra)[A3] for interactive processing along with corresponding library of subroutines for database processing from an application program.

Interactive ARIS

The interactive mode of ARIS receives commands that are entered directly from a user terminal to create and manipulate relational data.

General Comments

The system prompts the user with a right arrow (>) whenever input is required to be entered. Each command line consists of one or more words typed in by the user. The command can be continued to the next line by using the (&) character at the end of a line. The following are examples of the same command:

```
> SELECT * FROM TEST WHERE TEST# EQ 100
```

is the same as

```
> SELECT * FROM TEST &
> WHERE TEST# EQ 100
```

To separate words in a command either a space or a comma or both can be used. The following commands

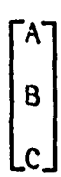
```
> SELECT TEST#, RUN# FROM TEST
> SELECT TEST# RUN# FROM TEST
```

are identical. Multiple spaces are ignored. A character string with blanks must be enclosed by single quotes (') as shown in the following command:

```
> SELECT FROM TEST WHERE NAME EQ 'A W WILHITE'
```

If no blanks are in the character string, single quotes are not needed. A maximum of 50 tokens (words, values, or subscripts) is allowed for each interactive command with a maximum of 132 characters per line.

Many of the interactive commands have options and/or selections that a user can make. In defining the syntax of the commands, an option within a command is given as



This syntax represents an option where either A, B, C or none of these options can be used. The ellipsis syntax

. . .

means that the preceding option can be repeated. The syntax

$$\left\{\begin{matrix} A \\ B \\ C \end{matrix}\right\}$$

represents a selection where either A, B, or C must be used.

Although not obvious now, the syntax will be clarified with the syntax of the interactive commands and the corresponding examples (see the SELECT command).

Initiation and Termination

The following sections of this paper will describe the construction of a database, the construction of relations in a database, the input of tuples into relations, and the various ways relations can be manipulated. When ARIS is first executed, a banner will appear with the date and time. The user must first enter the database name as shown in Figure A2 before any database activities can be performed. The database name consists of one to six characters and must be unique. If no database exists with that name, a virgin database is created.

For the manipulation of two simultaneous databases (e.g., to transfer a relation from one database to another or to save the temporary database) both database names can be entered after the input request by:

database-name-1, database-name-2

These databases are considered to be the permanent and temporary databases respectively. If the temporary database is to be saved, its name must be entered at the initiation of a session.

QUIT: To terminate the program and save the current database(s), the command

QUIT

is used. A statement declaring the names of the permanent database and the temporary database, if a temporary database name was entered at the beginning of the session, will appear on the screen.

HELP: The command

HELP

lists the syntax and abbreviations that can be used for all the ARIS interactive commands. An illustration of each of the above commands is given in Figure A2.

Relation Definition

A database consists of one or more relations. When the program is initiated, the first response requested from the user is the database name (Fig. A3). The database name, DBNAME, is entered. The data description required for the relation TEST from Figure A1 is shown in Figure A3.

CREATE. The data description of a new relation can be added to the database at any time with the CREATE command. When CREATE is entered, questions are asked concerning the description of the relation. First, the relation name (TEST), is entered. Relation names must be unique and restricted to 8 characters or less. The number of attributes (6) is then entered for the example in Figure A3. A maximum of 50 attributes is allowed. For each attribute in the relation, the attribute name, dimension, type, inverted attribute/duplicates allowed and primary key selection must be entered. The attribute name must be unique within a

relation and is restricted to eight or less alphanumeric characters. Remember, once a relation has been created, the attribute definitions cannot be changed.

The dimension (an extension to the relational model) is the number of elements that can be placed in that attribute. For a dimension equal to 1, only one value can be placed in the attribute. An attribute dimensioned n, where n is greater than 1, has n values associated with it. The dimension can be specified as 0. For this case, both the dimension and the type must be added to the tuple when the data is being entered into the database (see the later sections on the Interactive Data Manipulation Language and ARIS FORTRAN Library). There are four data types allowed--character, real, integer, and file--and they are specified by the key words CHAR, REAL, INT, or FILE, respectively. For character data type, the alphanumeric character string length must be entered. This defines the string length for each attribute element. For file data type, the filename character string length must be entered. This data type option, upon input, will open a file (by the name declared in each data tuple), allow any type input, and then close the file. Real and integer types take one computer word per element and character and file types take the next largest integer of the defined alphanumeric character string length divided by 4 computer words per element. A string declared as 5 characters would require 2 words of storage. There is a maximum limit of 1000 words per tuple in the interactive mode of ARIS. In relation TEST, TEST# is defined as character data type with length 8; therefore, number of words per element is 2.

Each attribute can be inverted by entering Y for YES to the inversion question (Fig. A3). Inversion is a specification to be made if fast access is needed to a tuple based on the value of the inverted attribute.

Inverted attributes are used primarily for relations with a large number of tuples (cardinality greater than 10,000). Search time is reduced to a logarithmic search by the use of B*-trees (Ref. A4). For the case of a relation with 10,000 tuples, the difference in retrieval performance between inverted and non-inverted attributes is approximately 7:1. The increased access speed must be traded with increased disk storage (approximately double the storage requirement for 1 inversion, triple for 2 inversions, etc.) and increased storage time (approximately a factor of 4 increase). With inversion, the tuples are automatically sorted in ascending order on each inverted attribute. When an attribute is inverted (by entering Y), a specification can be made for duplicates allowed (by entering Y or N for YES or NO). If duplicates are not allowed (N specification), no tuple can be entered with the value of the inverted attribute the same as one previously stored. This specification can be used as a simple data protection scheme, e.g., only allow unique test numbers in the relation test.

Primary key is an integrity constraint option that can be used to ensure that each tuple is unique. This specification is similar to duplicates allowed but is more encompassing. In the TEST relation, entering Y (for yes) to the primary key question for attribute TEST# (Fig. A3) ensures that each tuple will have a unique test number. Another example would be a salary relation in which first name, middle name, last name, and salary are attributes. To ensure that a salary is unique to each employee, the three attributes first name, middle name, and last name would be the primary key. The extreme case would have all attributes participate as the primary key but the tuple input performance is degraded with each primary key attribute because each primary key attribute

in all tuples must be checked to ensure uniqueness. For best performance no attribute has to be declared as primary key. Caution must be exercised since attributes cannot be changed directly to primary key after the relation is created.

The other attributes are entered in Figure A3: model number (MODEL#), wind tunnel name used for the test (TUNNEL), the cognizant engineer (ENGINEER), a description of the test (COMMENTS), and the date (DATE). After a relation is entered, the relation description is printed along with the number of words required to store each attribute.

Figures A4-A6 illustrate the CREATE command for three other relations. Relation MODEL describes the wind tunnel models tested with the model number (M#), type of model (TYPE), description of the model (COMMENTS), and the model scale (SCALE) which is the model size divided by the real size of the aircraft. The relation TEST-RUN describes the tests. The attributes are test number (TEST#), run number (RUN#), description of the run (COMMENTS), the configuration buildup (CONFIG), the control settings of the aileron (C1) and rudder (C2) in degrees, the run attitude change (POLAR) and the speed parameter of the test (MACH). Finally, illustrating dimensioned attributes, the relation TDATA describes the aerodynamic data measured. The aerodynamic data (DATA) is presented as a function of test number (TEST#), run number (RUN#) and point number (POINT#). The aerodynamic data measured are angle of attack, side-slip angle, lift coefficient, and drag coefficient--DATA(1) through DATA(4) respectively.

CHANGE. The CHANGE command is used to change the relation description. To change an attribute name in a relation, the command

CHANGE attribute name-1 TO attribute-name-2 IN relation-name

is used. An example is given in Figure A7. To change a relation name

CHANGE relation-name-1 TO relation-name-2

is used. To change inverted attribute specifications, use the command

CHANGE attribute-name IN relation-name TO $\left\{\begin{matrix}\text{INV} \\ \text{NONINV}\end{matrix}\right\}$

where INV specifies attribute inversion (Fig. A7) and NONINV eliminates the inversion. Only duplicates allowed inversion is permitted since duplicate attribute values may already exist in the relation. The time required to execute this command will depend on the cardinality of the relation.

RELATION. All the relation names in the database can be listed with the RELATION command (Fig. A7). By appending a relation name to the entry RELATION as shown in Figure A7, the description and cardinality of a relation are displayed. The relation command syntax is

RELATION [relation-name]

End of Example. Once relations have been defined, data can be stored in these relations either interactively, by file, or by a FORTRAN application program. Although no tuple data has been stored in Figures A2-A7, the session can be ended and the database saved (relation descriptions) by using the QUIT command.

Interactive Data Manipulation Language

The interactive data manipulation language is based on the relational algebra constructs outlined in references A1 and A3. Additional commands have been added in ARIS for user convenience. To begin the interactive session (Fig. A8) the database is opened by entering the database name,

DBNAME.

INPUT. To enter tuples interactively, the command

INPUT[R] relation-name

is used. Each attribute is entered individually as shown in Figure A8. When the key word, $END, is entered for the first attribute value of a tuple instead of entering a value, control is returned to the interactive command level.

For variable length attributes (dimension specified as 0), both the dimension and type are entered before the attribute values.

The INPUTR option replaces old data with that just entered if a primary key conflict occurs. If a primary key conflict occurs with the INPUT option, an error message would be printed and the old data would not be changed.

DLOAD. To enter tuples from an external file the command

DLOAD relation-name file-name

is used. The external file data is structured as stacked attribute values similar to the INPUT command. An example of INPUT and the external file, AWTEST, is shown in figure A9. Tuples are entered into the other relations in figures A9-A11.

The command RELATION TEST in figure A9 outputs the relation description and the number of tuples entered. Five tuples have been entered--one from interactive input and four from the external file AWTEST.

PRINT. The command

PRINT relation-name

lists all tuples in the relation as shown in Figures A9-A11.

SELECT. The most comprehensive command for data retrieval is SELECT

because it can retrieve any or all attributes for a relation based on any combination of logical and boolean operations. The results can be sorted and returned to the user or placed in a temporary relation for further manipulation. The SELECT format is

```
SELECT { *                    } FROM relation-name
       {att-1  [,att-2]....   }

[ WHERE attw-1 { EQ } value-1 [ { TOL  }     tol-1 ] ]
               { NE }          { PTOL }
               { LT }          { MTOL }
               { LE }
               { GT }
               { GE }

                [ { AND } attw-2 { EQ } value-2 [ { TOL  }     tol-2 ] ] ...
                { OR  }          { NE }          { PTOL }
                                 { LT }          { MTOL }
                                 { LE }
                                 { GT }
                                 { GE }

[ { UP   } atts-1 ]  [ { UP   } atts-2 ]  ...
{ DOWN }             { DOWN }

[ GIVING { TTY              } ] [ RENAME { #attr-1 } [ { ,#attr-2 } ]... ] ]
         { new-relation-name }             {       }   {         }
```

where * represents all the attributes in the relation and att-n are selected attributes that are retrieved. The default command

SELECT * FROM relation-name

is identical to the

PRINT relation

command as shown in figure A12. The command

SELECT TEST# FROM TEST

retrieves all occurrences of just test number and prints the results to the terminal (TTY).

In the WHERE clause, attw-n are the attributes that are logically compared with values, value-n, to specify the conditions for tuple

retrieval. A tolerance, tol-n, on numeric data (TOL for ± tolerance, PTOL for positive tolerance, and MTOL for a negative tolerance) can be specified on any logical comparison. Figure A13a is an example of the WHERE clause that retrieves all tests conducted by Spencer or using the SH10 model. Figure A13b is a tolerance example.

The UP (DOWN) clause with the attribute atts-n is used to sort the tuples in ascending (descending) order. If more than one sort clause is used, the first sort is the major sort and the following sorts are minor sorts (fig. A13c).

If the new relation name is unique in the GIVING clause, the relation definition and retrieved tuples are placed in the temporary database. All data manipulation commands can be used with all relations, either permanent or temporary. If the same relation name exists in both the permanent and temporary databases, only the relation in the permanent database can be manipulated. But with the CHANGE command, relation names can be changed to be unique, thus eliminating the problem.

When a temporary relation is created, the attribute names can be changed with the RENAME clause. A one-to-one correspondence must exist with the retrieval attributes (att-n) and the renamed attributes (attr-n). The symbol # denotes that the temporary attribute name will not be changed from the permanent name. The RENAME clause must be used whenever an element of a dimensioned array is retrieved and stored in a temporary relation. The RENAME clause is illustrated in figure A13d where the point number and angle of attack for test LA70 and run 1 are placed in a temporary relation called LA701.

DELETE. To permanently delete a relation the command

DELETE RELATION relation-name

is used. To delete tuples from a relation the command

DELETE relation-name [WHERE ...]

is used. The WHERE clause (see SELECT command) specifies the tuples to be deleted. Figure A14a illustrates the use of the DELETE command.

ASSIGN. Once tuples have been entered into a relation, attribute values can be changed with the following command:

ASSIGN value TO attribute-name IN relation-name

[WHERE ...]

Without the WHERE clause, the value of attribute-name for every tuple will be equal to the value entered with this command. For variable length tuples where data type can change from one tuple to another, numeric values will be converted correctly. String values will not be stored in numeric (REAL or INT) data type attributes. An example of the ASSIGN command is given in Figure A14b.

JOIN. The JOIN operation combines two relations into a third relation that has the combined attributes of each of the two joined relations. For each tuple, the value of one attribute in the first relation is compared with the value of an attribute in the second according to the logical operator declared. If the comparison is true, then the tuples are combined. The JOIN syntax is

JOIN relation-name-1 AND relation-name-2 OVER att-1 [{ EQ / NE / LT / LE / GT / GE / CS } att-2]

[GIVING { TTY / relation-name-3 }]

where att-1 and att-2 must exist in the respective relations and have the same data type definition. If the att-2 logical option is not used, then the JOIN is an implied EQUAL JOIN where att-1 must exist in both relations and EQ is the implied logical operator. An example of the

JOIN command is shown in figure A15 which illustrates a group of queries to determine the engineers that have tested Shuttle models.

UNION. To add the tuples of one relation to another, the command

UNION[R] relation-name-1 AND relation-name-2 [GIVING {TTY / relation-name-3}]

is used. The relation-name-3 literal can be a new relation or one that already exists. The command

UNION A AND B GIVING A

simply adds the tuples of relation B to relation A. For a successful operation, the relations must be union compatible where the attribute names and attribute definitions for each relation must be identical. The UNION command can also be used for the union set operation where the tuples belonging to either relation A or relation B can be combined to form a third relation C. Using the results for the relation TEMP2 in Figure A15, the queries in Figure A16a determine the engineers that have tested Shuttle models (this is relation TEMP2) in either the LTPT or the Unitary wind tunnels.

Another use of the UNION command is to transfer relation tuple data from one database to another. A temporary relation can be created with the SELECT command and saved. A second database and this temporary database can be used in another session. By using a UNION command, the temporary tuple data can be added now to the second database.

The UNIONR option will replace old data with the new entry if a primary key conflict is encountered, whereas the UNION option will return an error message and keep the old data.

INTERSECT. The intersection of two (union-compatible) relations has the syntax

INTERSECT relation-name-1 WITH relation-name-2

GIVING { TTY / relation-name-3 }

which gives the set of all tuples belonging to both relations. The example presented in Figure A16b is to determine the engineers that have tested Shuttle models in both the LTPT and the Unitary wind tunnels.

MIN. The command

MIN { * / att-1 } [,att-2]... IN relation-name

prints the minimum value of each attribute listed (see SELECT command) for the relation (Fig. A17a).

MAX. The command

MAX { * / att-1 } [,att-2]... IN relation-name

prints the maximum value of each attribute listed (Fig. A17b).

Database dump and load. With the permanent and temporary database concepts, relation definition and tuple data can easily be copied from one database to another. In order to transfer database data across different computers, the data must be converted to ASCII (card image) form. Six commands are provided for this function. The commands

DUMP [relation-name] file-name

SDUMP [relation-name] file-name

DDUMP relation-name file-name

are used to dump both relation and tuple data, relation definition data only, and tuple data only, respectively. If no relation-name is specified in the DUMP or SDUMP commands, then all relations are dumped. The file-name specifies where the dump will be written.

Three inverse commands

LOAD file-name

SLOAD file-name

DLOAD relation-name file-name

read the file that was written with the dump commands.

PTCOPY. To copy the entire permanent database to the temporary database level (destroying the current temporary database), the command

PTCOPY

is used. If a duplicate relation name occurs, the command will abort with an error message.

TPCOPY. To copy the entire temporary database to the permanent database level (destroying the current permanent database), the command

TPCOPY

is used. Duplicate relation names are illegal, causing the command to abort and return an error message.

REPLACE. To replace data in a relation with data from another relation, the command

REPLACE relation-name-1 [.T / .P] WITH relation-name-2 [.T / .P]

is used. If a database has been loaded into the system at the permanent and temporary levels and the same relation name appears at both levels, then the extenders .P and .T refer to the permanent and temporary relations, respectively.

The REPLACE command deletes the data in relation 1 and inserts the data fom relation 2 into relation 1.

COPY. To copy an entire relation from one database level to another, the command

COPY relation-name-1 [$\begin{matrix}\cdot T \\ \cdot P\end{matrix}$] WITH relation-name-2 [$\begin{matrix}\cdot T \\ \cdot P\end{matrix}$]

is used. If relation-name-2 does not already exist, the copy command will create relation-name-2 and insert the data from relation-name-1 into it. Otherwise, the COPY command acts as a REPLACE command.

RECLAIM. With the present ARIS implementation, no garbage collection techniques have been implemented to automatically reclaim space on the disk occupied by tuple data or relation definitions that have been deleted (see Internal Structure). To reclaim this unused space, the

RECLAIM

command is used to make a new copy of the current database. When the new database is created with the RECLAIM command, the database performance should increase since all tuples in a relation will be stored physically together on the disk.

End of example. The end of this interactive session is shown in Figure A17c. In the interactive session four temporary relations were created. Since a temporary database name was not entered at the initiation of the session, the temporary relations are destroyed.

All data entered into the permanent relations are saved under the database name, DBNAME, entered at the initiation of the session in Figure A8.

ARIS FORTRAN Library

The ARIS system can be used directly by users in an interactive mode, as previously explained, or it can be used by FORTRAN application programs. A library of subroutines is provided so that the interactive

commands can be duplicated with a call to subroutines within a FORTRAN program. All ARIS subroutines and commands begin with the letter A to avoid conflicts with existing programs.

Initiation/Termination and Error Processing

AOPEN. The purpose of this subroutine is to open a database that has been created and saved with the interactive data manipulation language.

Syntax: CALL AOPEN (PERM, TEMP, ICODE)

Parameters: PERM - permanent database name, eight characters*
TEMP - temporary database name, eight characters*
ICODE - error return (see AERROR), integer

*note: the seventh and eighth characters must always be blanks

AQUIT. The purpose of this subroutine is to update all changed pointers and to close the database files.

Syntax: CALL AQUIT (PERM, TEMP, ICODE)

Parameters: PERM - permanent database name, eight characters*
TEMP - temporary database name, eight characters*
ICODE - error return (see AERROR), integer

*note: same as above

AERROR. Each subroutine in the ARIS library has a parameter, ICODE, that returns with a value of 0 if the subroutine was successful. If the subroutine is not successful, the description of the error can be printed with the AERROR subroutine.

Syntax: CALL AERROR (ICODE)

Parameters: ICODE - error input code, integer

For the program database system, it is the user's responsibility to make a copy of the database for backup. If an error destroys the database, it can be replaced with these backup files.

Data Manipulation Language

The data manipulation subroutines are used to input, delete, retrieve, and manipulate relations and tuple data.

APUTRL. The purpose of this subroutine is to identify the relation that will be used to input tuple data with the APUTTP subroutine.

Syntax: CALL APUTRL (RELNAM, ICODE)

Parameters: RELNAM - relation name, eight or less characters

ICODE - error return, integer

APUTTP. The purpose of this subroutine is to input a tuple into a relation defined by the APUTRL subroutine.

Syntax: CALL APUTTP (DATA, MDMDAT, ICODE)

Parameters: DATA - the data in the tuple

MDMDAT - the dimension of DATA, integer

ICODE - error return, integer

In the relation MODEL, there are four attributes, MODEL#, TYPE, COMMENTS, and SCALE. The number of words in this tuple is 11 as shown in Figure A4. In this case, the dimension of DATA would be at least 11 in the application program and MDMDAT would equal to 11.

For attributes that have been defined as variables, dimension and type (specified by dimension equal to 0 when the relation was created), the first two words for that dimension must be the number of elements and the data type. However, if the data type of a value being entered is character, then the dimension is not just the number of elements. The dimension is (1000 * number of characters per element) plus the number of elements. If the breakdown of the dimension information is needed at a later time, the following subroutine will decipher the information.

Syntax: CALL ALENDM (DATA(i), DATA (i+1), NOCHAR, LENWRD, IDIM)

Parameters: DATA(i) - input, dimension information, integer

DATA(i+1) - input, data type, 4 characters

NOCHAR - return, number of characters per element, integer

LENWRD - return, computer word length of each element, integer

IDIM - return, total number of elements, integer

AREPTP. This subroutine inputs a tuple into a relation defined by the APUTRL subroutine. However, unlike the APUTTP subroutine, AREPTP replaces an old data tuple with the new data if the two tuples have identical primary keys.

Syntax: CALL AREPTP (DATA, MDMDAT, ICODE)

Parameters: (see APUTTP subroutine)

ASELCT. Once data has been entered into the database interactively or by subroutines, the data can be found by using the ASELCT subroutine which is identical to the interactive SELECT command. The ASELCT subroutine finds the correct tuples. These tuples can be placed in a temporary relation or retrieved from the database into the application program.

Syntax: CALL ASELCT (RELNAM, NATT, ATT, ISATT,
NATTW, ATTW, ISATTW, VALUE, LOPT, BOPT,
TTOL, TVALUE,
NSORT, ATTS, ISATTS, IORDER,
RENAME,
NEWREL, ICNT, ICODE)

Parameters:

RELNAM - relation name, eight or less characters, dimension (2)

NATT - number of attributes retrieved, integer
= 4H* to retrieve all attributes, (ATT, ISATT parameters unnecessary in this case)

ATT - attribute names, eight characters
dimension (2,NATT)

ISATT - subscript of ATT, integer, dimension (NATT)
= integer for element retrieval from array
= 4H* to retrieve all array elements

NATTW - number of attributes in the WHERE clause (see interactive SELECT command), integer
= 4H* to retrieve all data from defined relation, (ATTW, ISATTW, VALUE, LOPT, BOPT, TTOL, TVALUE parameters unnecessary in this case)

ISATTW - subscript of ATTW (see ISATT), integer, dimension (NATTW)

VALUE - value of attribute, type is that defined for ATTW, dimension (NDIM), where

$$NDIM = \sum_{I=1}^{NATTW} \text{NUMBER OF WORDS IN ATTW (I)}$$

LOPT - logical operator (EQ, NE, LT, LE, GT, GE or CS), four characters, dimension (NATTW)

BOPT - boolean operator (OR or AND), four characters, dimension (NATTW-1)

TTOL - type of tolerance, dimensional (NATTW) if TTOL
= 4HNONE
= 4HTOL for ± tolerance

= 4HPTOL for + tolerance

= 4HMTOL for - tolerance

= 4H for no tolerance specification for a particular attribute in where clause

= 4HNONE for no tolerance specification involved in entire where clause

TVALUE - value of tolerance, dimension (NATTW) if TTOL =4HNONE

NSORT - number of sort attributes, integer

= 0 for no sorting

(ATTS, ISATTS, IORDER, parameters unnecessary in this case)

ATTS - see ATT (Sorting allowed only on names in ATT), dimension (2, NSORT)

ISATTS - subscript for ATTS (see ISATT), dimension (NSORT)

IORDER - ordering of relation (UP for ascending or DOWN for descending), four characters, dimension (NSORT)

RENAME - new attribute names of selected attribute(s) if a temporary relation is created, dimension (2,NATT)

= 0 if no temporary relation is created, integer

= 8H , if all attribute names are not to be changed, 8 characters

= 8H#, if specific attribute name is not to be changed, 8 characters or less

NEWREL - selection of display, create new relation, or hold for data retrieval GET subroutines,
= Eight characters, dimension (2)
= 8HTTY - display selected tuples at the terminal
= 8H.MYDATA. - hold tuple data for GET subroutines
= Eight character name - create and store relation with given name in temporary database

ICNT - return of the number of tuples found, integer

ICODE - error return, integer

ASELSH. This subroutine is a shortened version of the ASELCT subroutine. It retrieves all the attributes, limits the WHERE clause to a 1 limits the SORT clause to 1 restriction, and holds the data for GET subroutines.

Syntax: CALL ASELSH (RELNAM,
ATTW, ISATTW, VALUE, LOPT,
ATTS, ISATTS, IORDER,
ICNT, ICODE)

Parameters: (See ASELCT subroutine)

ASELTP. This subroutine is also a shortened version of the ASELCT subroutine. It disallows any tolerance testing, sorting, or renaming of attributes.

Syntax: CALL ASELTP (RELNAM, NATT, ATT, ISATT,
NATTW, ATTW, ISATTW, VALUE, LOPT, BOPT,
NEWREL, ICNT, ICODE)

Parameters: (see ASELCT subroutine)

AGETNX. After a relation has been retrieved with the ASELCT, or ATSELC subroutines with NEWREL equal to .MYDATA., this routine will single step through the tuple locations satisfying the command.

Syntax: CALL AGETNX (ICODE)

Parameters:

ICODE - return code, integer

= 0 tuple data location found

= 0 no (more) data

AGETTP. Tuple data can be retrieved into the FORTRAN program using this subroutine preceded by a call to the AGETNX subroutine.

Syntax: CALL AGETTP (DATA, MDMDAT, ICODE)

Parameters:

DATA - tuple data

MDMDAT - dimension of DATA, integer

The dimension, MDMDAT, is the sum of the dimensions (or word length multiplied by the dimension if attribute type is character) of all the attributes selected (ATT).

ASAVED. The current processed position of the tuple locations list can be saved by a call to ASAVED. This allows for other testing without losing tuple locations already retrieved.

Syntax: CALL ASAVED (DATA, ICODE)

Parameters:

DATA - DATA(1) = dimension of DATA (260)

DATA(2) = file name, four characters

ICODE - Error return, integer

ARSTOR. This subroutine restores the pointer to the list of tuple locations previously saved by a call to ASAVED. A call to AGETNX will

begin the processing of the list again.

Syntax: CALL ARSTOR (DATA, ICODE)

Parameters: (See ASAVED subroutine.)

ADELET. The purpose of this subroutine is to delete a relation or tuple(s) in a relation based on a WHERE clause.

Syntax: CALL ADELET (RELNAM, NATTW, ATTW, ISATTW, VALUE, LOPT, BOPT, ICODE)

Parameters: (See ASELCT subroutine)

NATTW = 0 for relation deletion

ADELTP. The purpose of this subroutine is to delete a single tuple from a relation. It must be preceded by a call to the AGETNX subroutine.

Syntax: CALL ADELTP (ICODE)

Parameters:

ICODE - error return, integer

ASSIGN. The ASSIGN subroutine is used to change an attribute value in tuples based on the WHERE clause restrictions.

Syntax: CALL ASSIGN (ATT, ISATT, NEWVAL, TYPE, NWRDS, RELNAM, NATTW, ATTW, ISATTW, VALUE, LOPT, BOPT, ICODE)

Parameters: (see ASELCT for parameter description)

NEWVAL - value being assigned to ATT, must be character type if ATT is variable dimension and type

TYPE - type of NEWVAL (CHAR, INT, or REAL), 4 characters

NWRDS - number of words in NEWVAL, integer

= 1 for integer and real type

= n where n is the next largest integer of the length of the character string divided by 4

ATTMOD. This subroutine changes the value of an attribute in a single tuple. It must be preceded by a call to the AGETNX subroutine.

Syntax: CALL ATTMOD (ATT, ISATT, NEWVAL, TYPE, ICODE)

Parameters: (see ASSIGN subroutine)

AJOIN. The purpose of this subroutine is to join two relations to form a third relation based on a logical comparison of one attribute in each relation.

Syntax: CALL AJOIN (RELNM1, RELNM2, NEWREL, ATT1, ISATT1, ATT2, ISATT2, LOPT, ICODE)

Parameters:

RELNM1 - first relation name, 8 characters, dimension (2)

RELNM2 - second relation name, 8 characters,dimension (2)

NEWREL - new relation name, 8 characters, dimension (2)

ATT1 - attribute in first relation, for logical comparison, 8 characters, dimension (2)

ISATT1 - subscript of ATT1, integer

ATT2 - attribute name in second relation for logical comparison, 8 characters, dimension (2)

ISATT2 - subscript of ATT2, integer

LOPT - logical operator (EQ, NE, LT, LE, GT, GE, CS), four characters

ICODE - error return, integer

AUNION. The purpose of this routine is to combine two union-compatible relations (see interactive UNION). The subroutine can be used to append tuple data from one relation to another.

Syntax: CALL AUNION (RELNM1, RELNM2, NEWREL, ICODSR, ICODE)

Parameters: RELNM1 - first relation name, eight characters, dimension (2)

RELNM2 - second relation name, eight characters, dimension (2)

NEWREL - can be RELNM1, RELNM2, or a new relation name, eight characters, dimension (2)

ICODSR - integer

= 0 insert data obeying all relation definition rules

= 1 replace old data if primary key conflict occurs

ICODE - error return, integer

≠ 0 if relations are not union compatible relations

AINTSC. The intersection of two (union-compatible) relations is the set of all tuples belonging to both relations.

Syntax: CALL AINTSC (RELNM1, RELNM2, NEWREL, ICODE)

Parameters: (see AUNION)

AMINIM. The purpose of this routine is to retrieve the minimum value from the relation for each attribute listed.

Syntax: CALL AMINIM (RELNAM, NATT, ATT, ISATT, VALUE, ICODE)

Parameters: RELNAM - see ASELCT

NATT - see ASELCT

ATT - see ASELCT

ISATT - see ASELCT

VALUE - minimum values of the attributes, dimension (sum of the dimensions, or word length per

element times the dimension for character type, of ATT)

AMAXIM. The purpose of this subroutine is to retrieve the maximum value from the relation for each attribute listed.

Syntax: CALL AMAXIM (RELNAM, NATT, ATT, ISATT, VALUE, ICODE)

Parameters: VALUE - maximum values of the attributes, dimension (see AMINIM)

ACOPPT. This subroutine copies the entire permanent database to the temporary database level, replacing all current information at the temporary level.

Syntax: CALL ACOPPT (ICODE)

Parameters: ICODE - error return, integer

ACOPTP: This subroutine copies the entire temporary database to the permanent database level, replacing all current information at the permanent level.

Syntax: CALL ACOPTP (ICODE)

Parameters: ICODE - error return, integer

ACOPDB. The purpose of this subroutine is to allow one relation to be copied to another. If both relations involved exist, then the data tuples from one relation replace the data tuples of the second relation. If the second relation does not exist, then it is created with a schema identical to the first relation, and then tuple replacement proceeds as above.

Syntax: CALL ACOPDB (RELNM1, RELNM2, ICODE)

Parameters: RELNM1 - first relation name, eight characters, dimension (2) (see COPY command)

RELNM2 - Second relation name, eight characters, dimension (2) (see COPY command)

ICODE - error return, integer

Program Architecture

The architecture of ARIS is illustrated conceptually in figure A18. The architecture is structured in several levels for two reasons. First, the levels provide two separate packages in which one package can be used as a library of FORTRAN subroutines that can be called from application programs. The other package, the interactive system, consists of a controller and a user command translator and the library. For each interactive command, there exists a subroutine to perform the identical function. Thus, application programs can have as much or more flexibility than the interactive system.

A second reason for the levels is to separate the host computer subroutines from the rest of the system to ease the conversion process from one computer to another. This separation limits the number of ARIS subroutines that use host subroutines to only a few. The host subroutines required are for random access files, sorting, and the date and time. Conversion to systems with word lengths other than 32 bits is more cumbersome.

The MAIN level simply interprets the type of user command in order to select the correct parsing routine. The parsing routine translates the command and constructs the calling parameters for the command. The command routines actually perform the desired data management function, such as SELECT, UNION, JOIN, etc. The command routines interact with the internal storage of the information (see INTERNAL STRUCTURE). The internal storage routines consist of various data structures to store and retrieve tuple data. These internal storage routines in turn use the random access procedures that are provided by the host computers to

physically store and retrieve data from the disk. At the command level, the SELECT routine uses the sort package provided by the host computer.

Internal Structure

The B*-tree data structure is used to store and retrieve data because it maintains logarithmic performance for random access of large databases and can also be used to retrieve data sequentially. Reference A5 provides a detailed discussion of B*-trees.

All locations of data (both relation and tuple) are determined with the B*-tree. For each key (i.e., relation name) in the B*-tree, there exists the disk location on which the data is stored. Basically the B*-tree consists of three levels: B-tree (index), sequence list, and replicate list (Fig. A19). The B-tree provides a road map to the keys (relation name or inverted attribute values) in the linked lists. Once the key is found in the linked list with the B-tree, the location of the relation or tuple that is associated with that key can be determined. If duplicates of the key exist (duplicates-allowed specification on an inverted attribute), the replicate list provides a list of the other disk locations of tuples associated with that key value.

Figure A20 illustrates the insertion of keys into a B*-tree. The maximum number of keys per block is three for this example. Insertion starts at the linked list level where the block consists of the keys, the disk location associated with each key, a pointer to the block previous to the current block and a pointer to the next block. The first key to be inserted is B (Fig. A20a). Since there are no other blocks, there are no previous or next pointers. The next two keys are inserted in Figure A20b. As shown in Figure 20b, the keys in a block are placed in ascending order. Because there is no more room to insert the next key, F, in the

list block, the list block is split and the middle key (actually left of middle since no middle key exists) is promoted to the B-tree as shown in Figure A20c. The format of the B-tree block is just the key itself (no disk location) and pointers to the leaf blocks. Note that the next/previous pointers in the list block now exist. Figure A20d shows the split after the keys Z and X are inserted. Another split is shown in Figure A20e. The final two insertions not only split the linked list block but also the B-tree block. Note that the key F is not needed in both levels of the B-tree but is needed in both the B-tree and the list (Fig. A20f).

To find the disk location of the tuple where the key is equal to N, the key N is compared with F at the root of the tree in the B-tree. Because N is greater than F, the right pointer is followed. At the next level, N is compared with L. The key N is greater than L, so it is compared with the next key P. Because N is less than P, the appropriate path is followed and comparisons are made at the list level. The key N is found and the tuple disk location of 8 is returned to be used to position the disk so that the tuple can be read.

To find all tuples greater than N, the list is simply followed to the right. The process of searching for the key with the B-tree and processing the keys sequentially is called an index-sequential access. Other logical operations are similarly processed. Thus the B*-tree is a powerful data structure for relational queries.

In reality the list block also has duplicate pointers added for each key, in order to store disk locations of tuples of attributes that have the same inverted attribute value. Figure A19 illustrates the replicate list where the replicate block consists of the disk locations for 3 keys

and next/previous pointers.

In the previous discussion, a block could only hold a maximum of 3 keys. For the present implementation on the PRIME computer, the node size was optimized for relations with a large number of entries (greater than 5,000). As shown in Figure A21, the block size was selected to be 120 keys. In Reference A4, the minimization is a trade of disk access time (large for small number of keys) versus disk character transmission time and key search time (large for large number of keys). To minimize key search time in a block, a binary search (Ref. A2) is used instead of the linear search in the current ARIS configuration.

For deletion of keys in the B-tree, sequence list, and replicate list, a deletion flag is used instead of collapsing the blocks to accommodate the empty space (suggested in Ref. A4). The delete flag in the B-tree is very useful since the keys in the B-tree do not have to be redistributed to restore balance, and separate blocks do not have to be concatenated. Deletion is faster with the space left after a deletion is reclaimed if possible. Problems occur with delete flags since only the flag is set when deletion occurs. On insertion, the space left after a deletion is reclaimed if possible. Problems with delete flags can occur after many deletions because searches for keys include the deleted keys. Also, disk space is occupied by deleted keys and associated information. To correct the problem, the RECLAIM command can be executed. This command reads the old database and creates another with the exact same information. This new database not only has all the delete data, keys, and flags removed, but it also stores all the tuple data together physically on the disk. This contiguousness of data can increase retrieval performance significantly because disk seek time is reduced.

The complete B*-tree structure is used for relation definition retrieval and for tuple retrieval with relations that have at least one inverted attribute. To retrieve tuples from a relation with an inverted attribute based on a WHERE clause that does not include the inverted attribute, each tuple is retrieved by using only the sequence and replicate list of the inverted attribute. Thus every tuple in the relation must be compared to the condition set by the WHERE clause. For relations with no attribute inversion, only the sequence list structure is used to store the tuple disk locations. The tuples are added to the list in chronological order of insertion.

Performance between B-tree and list (sequential) structures can be compared in Table A1. To retrieve a tuple based on an attribute value (assuming unique values), there is a 5 to 83,334 ratio of disk access for the the B-tree versus the list structure for large number of tuples. Even for a small database (tuple entries $\approx 10^3$), the performance comparison is favorable. There is a 2 to 1 disk size penalty for this performance. Also, insertion of a tuple into a B-tree can take a significant amount of time since the B-tree must be searched on each insertion, and there is an added overhead when a B-tree node must be split. For storage in the list, list location of the last tuple entry is stored in the relation table. Insertion is simple because only the tuple location is written at this specified list location and this list location is updated. For the B*-tree, the list location is determined by searching the tree, key and tuple location is written in the list, and the key is inserted in the tree. Thus the trades between inversion versus non-inversion are insertion time, disk storage, and retrieval time.

System Architecture

The complete system architecture is presented in Figure A22. Its concept is to separate data from the locators of the data so that techniques to retrieve the locations can easily be changed to meet growing requirements and to place the locations in blocks (see Internal Structure) to increase storage/retrieval/deletion performance over other architectures.

The data section consists of a header, a definition of each relation, and the data for each tuple for each relation. Figure A22 illustrates how the data would appear after a RECLAIM command.

The header (Fig. A23) is used for two purposes. First it locates the B*-tree structure that is used to locate the relation definition table. It not only locates the B-tree for fast relation retrieval, it also locates the beginning (and the end) of the linked list to determine all relations in the database. The second purpose of the header is to provide the next free location in each of the four database files. These location values must be updated whenever new information is stored (at the next free location in any of these files).

The relation definition table (Fig. A24) consists of the relation name attribute descriptions and the pointers to locations of inverted attribute(s), B-tree(s), and linked list(s). As shown in Figure A22, the relation table points to the B*-tree structure that in turn points to the tuple data. Two relations are shown, one that has one inverted attribute and another with an inversion (degenerate B*-tree).

To insert a relation, the relation definition table is developed and stored at the next free data location. The header pointer to the relation

B*-tree is followed, the relation name is stored in the B-tree, the relation name and location are stored in the linked list block split, the storage location is determined from the next free locations in the header and the new block is stored at this location. Finally, the header next free locations are updated.

To store the first tuple in a relation with an inverted attribute, the relation definition table must be retrieved by using the relation B*-tree. A pointer to the B-tree (also to the linked-list at the same location) is then stored in the relation definition table. The tuple is written to the disk at the next free data location. The B-tree is created with the attribute value and tuple location. For multiple inversions, this B-tree creation process must be repeated for each inversion. The relation definition table is rewritten to the disk and the next free locations are updated and rewritten with the rest of the header. To add the next tuple, the same procedure is followed except that the attribute value and location are simply added to the block in the B*-tree structure. To increase performance, buffers are employed for each type file to reduce the number of physical disk accesses. Disk accesses in the location files do not occur until a block is split or a new relation is stored or retrieved. Tuple and header data storage retrieval is always a physical disk access.

The system architecture for a temporary database is exactly the same as shown in Figure A22. To access the temporary database, the file unit numbers for the four permanent database files are changed to the file unit numbers used for the four temporary database files. The header is also changed for the temporary database. Once these values (4 unit numbers and 7 header values) are changed, the temporary database is

processed exactly the same as the permanent database. Thus, switching from one database to another has about the same performance as working with one single database.

Concluding Remarks

A relational information system has been designed and implemented for use as a foundation system for computer-aided design applications. Emphasis in the design was placed on performance considerations for tuple storage and retrieval based on simple queries using state-of-the-art data structures. No optimization for complex queries or performance requirements for many of the relational functions have been considered. The system was designed for a small group of uses (10 or less) and a relatively small amount of data (less than 10 million words) because security, backup, and recovery systems have not been considered.

A permanent and temporary database system was developed for data transfer from one database to another on the same computer system. Also a dump/load feature was developed to transfer information from one computer system to another.

The relational model is well suited for engineering applications with its tabular form. Additional features like the variable length and type attribute have been added to the model in order to ease the interface of the model to a number of engineering application programs.

Finally, the system architecture has been designed to accept change easily. As new requirements evolve from using the system in an engineering environment, new data structures and models, query capability, and data display can be added with minimal change in the system software.

APPENDIX REFERENCES

[A1] Date, C. J. An Introduction to Database Systems. Addison-Wesley Publishing Company, February 1982.

[A2] Martin, James. Computer Data-base Organization. Prentice-Hall, Inc., 1977.

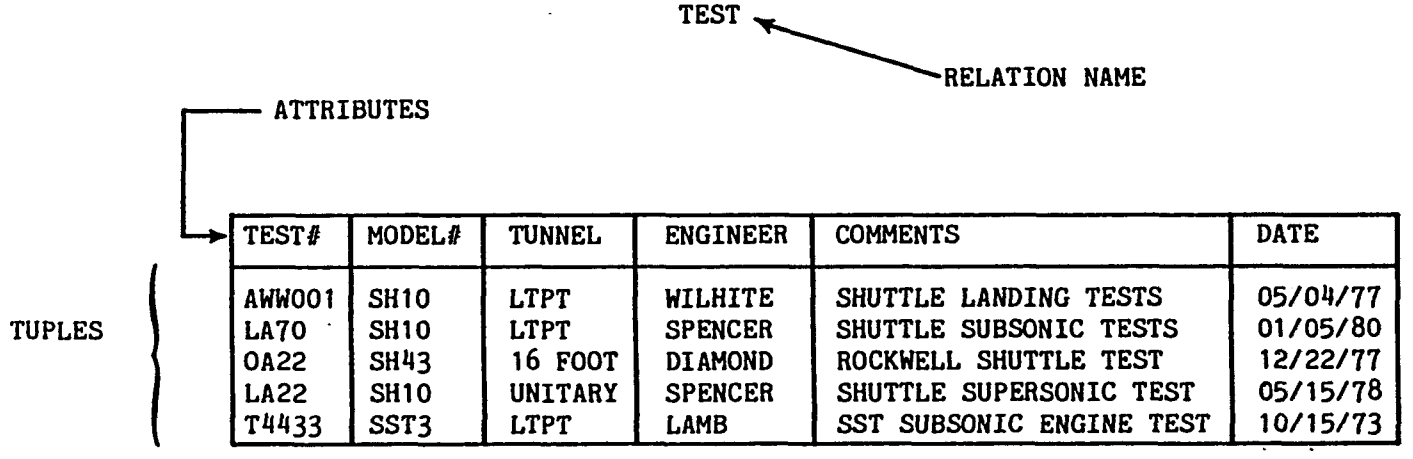
[A3] Codd, E. F. "A Relational Model of Data for Large Shared Data Banks." CACM, Vol. 13, No. 6, June 1970.

[A4] Horowitz, E. and Sahni, S. Fundamentals of Data Structures. Computer Science Press, Inc., 1976.

[A5] Comer, D. "The Ubiquitous B-tree." ACM Computer Surveys, Vol. 11, No. 2, June 1979.

NUMBER OF TUPLES		10^2	10^3	10^4	10^5	10^6	10^7
B-TREE	NUMBER OF BLOCKS (WORST CASE) NUMBER OF BLOCK ACCESSES	4 2	34 3	344 3	3447 4	34482 5	344826 5
LINKED LIST	NUMBER OF BLOCKS NUMBER OF BLOCK ACCESSES	2 1	17 9	167 84	1667 834	16667 8334	166667 83334

Table A1 - Comparison of the B-tree and linked list structures



TEST#	MODEL#	TUNNEL	ENGINEER	COMMENTS	DATE
AWW001	SH10	LTPT	WILHITE	SHUTTLE LANDING TESTS	05/04/77
LA70	SH10	LTPT	SPENCER	SHUTTLE SUBSONIC TESTS	01/05/80
OA22	SH43	16 FOOT	DIAMOND	ROCKWELL SHUTTLE TEST	12/22/77
LA22	SH10	UNITARY	SPENCER	SHUTTLE SUPERSONIC TEST	05/15/78
T4433	SST3	LTPT	LAMB	SST SUBSONIC ENGINE TEST	10/15/73

CARDINALITY = NUMBER OF TUPLES = 5

Fig A1 - Example of a relation

```
*****************************************
*                                       *
*   AVID RELATIONAL INFORMATION SYSTEM  *
*                                       *
*                (ARIS)                 *
*                                       *
*    WED, SEP 26 1984        15:23:26   *
*                                       *
*****************************************

INPUT DATABASE NAME(S)
>DBNAME

BEGIN INTERACTIVE SESSION
>HELP

  COMMAND          ABBREV     PHRASE
  ------------------------------------------------
  ASSIGN           - ASSI     value TO a1 IN rel WHERE...
  CHANGE           - CHAN     a1 TO a2 IN rel
  CHANGE                      rel1 TO rel2
  CHANGE                      a1 IN rel TO INV
  CHANGE                      a1 IN rel TO NONINV
  CREATE           - CREA
  DELETE           - DELE     rel [WHERE.... ]
  DELETE                      RELATION rel
  INPUT            - INPU     rel
  INPUTS           - INPU     rel
  INPUTR                      rel
  INTERSECT        - INTE     rel1 WITH rel2 [GIVING... ]
  JOIN             - JOIN     rel1 AND rel2 OVER a1
                                 [EQ,LT,GT,... ] [GIVING... ]
  PRINT            - PRIN     rel
  QUIT             - QUIT
  RECLAIM          - RECL
  RELATION         - RELA     [rel]
  SELECT           - SELE     * FROM rel [WHERE... ] [UP... ]
                                 [GIVING... ] [RENAME... ]
  TREE SEARCH      - TSEL     * FROM rel ROOT a1 EQ value
                                 THRU a2 [BY value LEVELS]
                                 [GIVING... ]
  UNION            - UNIO     rel1 AND rel2 [GIVING... ]
  UNIONS           -          rel1 AND rel2 [GIVING... ]
  UNIONR           -          rel1 AND rel2 [GIVING... ]
  DUMP DATA,       - DUMP     [[rel],filename]
       SCHEMA
  DUMP DATA        - DDUMP    [[rel],filename]
  DUMP SCHEMA      - SDUMP    [[rel],filename]
  READ IN DATA     - LOAD     [filename]
          SCHEMA
  READ IN DATA     - DLOAD  [rel,filename]
  READ IN SCHEMA   - SLOAD  [filename]

>QUIT
```

Fig A2 - Initiation, HELP, and QUIT commands

```
     ****************************************
     *                                      *
     *   AVID RELATIONAL INFORMATION SYSTEM *
     *                                      *
     *              (ARIS)                  *
     *                                      *
     *                                      *
     *    WED. SEP 26 1984        13:11:59  *
     *                                      *
     ****************************************

INPUT DATABASE NAME(S)
>DBNAME

BEGIN INTERACTIVE SESSION
>CREATE
RELATION NAME
>TEST
NUMBER OF ATTRIBUTES
>6
ATTRIBUTE NAME ( 1)
>TEST#
DIMENSION OF ATTRIBUTE ( 1)
>1
DATA TYPE ( 1)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 1)   --  Y/N
>Y
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 2)
>MODEL#
DIMENSION OF ATTRIBUTE ( 2)
>1
DATA TYPE ( 2)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 2)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 3)
>TUNNEL
DIMENSION OF ATTRIBUTE ( 3)
>1
DATA TYPE ( 3)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 3)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 4)
>ENGINEER
DIMENSION OF ATTRIBUTE ( 4)
>1
DATA TYPE ( 4)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 4)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 5)
>COMMENTS
DIMENSION OF ATTRIBUTE ( 5)
>1
DATA TYPE ( 5)
>CHAR
LENGTH OF EACH CHARACTER STRING
>24
ATTRIBUTE NAME ( 6)
>DATE
DIMENSION OF ATTRIBUTE ( 6)
>1
DATA TYPE ( 6)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 6)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
     **********************
     *                    *
     * RELATION TEST      *
     *                    *
     **********************

ATTRIBUTE          TYPE      NWORDS PRIMARY KEY INVERSION
TEST#    (  1)  CHAR*  8       2         N          Y
MODEL#   (  1)  CHAR*  8       2         N          N
TUNNEL   (  1)  CHAR*  8       2         N          N
ENGINEER(  1)  CHAR*  8       2         N          N
COMMENTS(  1)  CHAR* 24       6         N          N
DATE     (  1)  CHAR*  8       2         N          N

0 ENTRIES PRESENTLY
```

Fig A3 - Initiate database and enter relation TEST definition

```
>CREATE

RELATION NAME
>MODEL
NUMBER OF ATTRIBUTES
>4
ATTRIBUTE NAME ( 1)
>M#
DIMENSION OF ATTRIBUTE ( 1)
>1
DATA TYPE ( 1)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 1)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 2)
>TYPE
DIMENSION OF ATTRIBUTE ( 2)
>1
DATA TYPE ( 2)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 2)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 3)
>COMMENTS
DIMENSION OF ATTRIBUTE ( 3)
>1
DATA TYPE ( 3)
>CHAR
LENGTH OF EACH CHARACTER STRING
>24
ATTRIBUTE NAME ( 4)
>SCALE
DIMENSION OF ATTRIBUTE ( 4)
>1
DATA TYPE ( 4)
>REAL
INVERSION ( 4)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
```

```
**********************
*                    *
* RELATION MODEL     *
*                    *
**********************

ATTRIBUTE          TYPE      NWORDS PRIMARY KEY INVERSION
M#       (   1)  CHAR*  8      2        N          N
TYPE     (   1)  CHAR*  8      2        N          N
COMMENTS(   1)  CHAR* 24      6        N          N
SCALE    (   1)  REAL          1        N          N

     0 ENTRIES PRESENTLY
```

Fig A4 - Enter relation MODEL definition

```
>CREATE

RELATION NAME
>TEST-RUN
NUMBER OF ATTRIBUTES
>8
ATTRIBUTE NAME ( 1)
>TEST#
DIMENSION OF ATTRIBUTE ( 1)
>1
DATA TYPE ( 1)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 1)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 2)
>RUN#
DIMENSION OF ATTRIBUTE ( 2)
>1
DATA TYPE ( 2)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 2)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 3)
>COMMENTS
DIMENSION OF ATTRIBUTE ( 3)
>1
DATA TYPE ( 3)
>CHAR
LENGTH OF EACH CHARACTER STRING
>16
ATTRIBUTE NAME ( 4)
>CONFIG
DIMENSION OF ATTRIBUTE ( 4)
>1
DATA TYPE ( 4)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 4)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 5)
>C1
DIMENSION OF ATTRIBUTE ( 5)
>1
DATA TYPE ( 5)
>REAL
INVERSION ( 5)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 6)
>C2
DIMENSION OF ATTRIBUTE ( 6)
>1
DATA TYPE ( 6)
>REAL
INVERSION ( 6)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 7)
>POLAR
DIMENSION OF ATTRIBUTE ( 7)
>1
DATA TYPE ( 7)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 7)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 8)
>MACH
DIMENSION OF ATTRIBUTE ( 8)
>1
DATA TYPE ( 8)
>REAL
INVERSION ( 8)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
```

Fig A5 - Enter relation TEST-RUN definition

```
**********************
*                    *
* RELATION TEST-RUN  *
*                    *
**********************

ATTRIBUTE          TYPE      NWORDS PRIMARY KEY INVERSION
TEST#    (   1)  CHAR*  8      2         N          N
RUN#     (   1)  CHAR*  8      2         N          N
COMMENTS(   1)  CHAR* 16      4         N          N
CONFIG   (   1)  CHAR*  8      2         N          N
C1       (   1)  REAL           1         N          N
C2       (   1)  REAL           1         N          N
POLAR    (   1)  CHAR*  8      2         N          N
MACH     (   1)  REAL           1         N          N

     0 ENTRIES PRESENTLY
```

Fig A5 (con't)

```
>CREATE

RELATION NAME
>TDATA
NUMBER OF ATTRIBUTES
>4
ATTRIBUTE NAME ( 1)
>TEST#
DIMENSION OF ATTRIBUTE ( 1)
>1
DATA TYPE ( 1)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 1)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 2)
>RUN#
DIMENSION OF ATTRIBUTE ( 2)
>1
DATA TYPE ( 2)
>CHAR
LENGTH OF EACH CHARACTER STRING
>8
INVERSION ( 2)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 3)
>POINT
DIMENSION OF ATTRIBUTE ( 3)
>1
DATA TYPE ( 3)
>INT
INVERSION ( 3)   --  Y/N
>N
PART OF PRIMARY KEY -- Y/N
>N
ATTRIBUTE NAME ( 4)
>DATA
DIMENSION OF ATTRIBUTE ( 4)
>4
DATA TYPE ( 4)
>REAL
```

```
**********************
*                    *
* RELATION TDATA     *
*                    *
**********************

ATTRIBUTE          TYPE      NWORDS PRIMARY KEY INVERSION
TEST#    (  1)  CHAR*  8       2         N          N
RUN#     (  1)  CHAR*  8       2         N          N
POINT    (  1)  INT            1         N          N
DATA     (  4)  REAL           4         N          N

      0 ENTRIES PRESENTLY
```

Fig A6 - Enter relation TDATA definition

```
>RELATION

*************************
*                       *
* RELATIONS IN DATABASE *
*                       *
*************************
    RELATION MODEL
    RELATION TDATA
    RELATION TEST
    RELATION TEST-RUN
*** NO RELATIONS IN TEMPORARY DATABASE ***

>RELATION MODEL

                    **********************
                    *                    *
                    * RELATION MODEL     *
                    *                    *
                    **********************

ATTRIBUTE          TYPE      NWORDS PRIMARY KEY INVERSION
M#       (    1) CHAR*  8      2         N          N
TYPE     (    1) CHAR*  8      2         N          N
COMMENTS(    1) CHAR* 24      6         N          N
SCALE    (    1) REAL          1         N          N

      0 ENTRIES PRESENTLY

>CHANGE M# TO MODEL# IN MODEL

>CHANGE MODEL# IN MODEL TO INV

>RELATION MODEL

                    **********************
                    *                    *
                    * RELATION MODEL     *
                    *                    *
                    **********************

ATTRIBUTE          TYPE      NWORDS PRIMARY KEY INVERSION
MODEL#   (    1) CHAR*  8      2         N          Y
TYPE     (    1) CHAR*  8      2         N          N
COMMENTS(    1) CHAR* 24      6         N          N
SCALE    (    1) REAL          1         N          N

      0 ENTRIES PRESENTLY

>QUIT
```

Fig A7 - CHANGE and RELATION commands and terminate example session

```
*****************************************
*                                       *
*   AVID RELATIONAL INFORMATION SYSTEM  *
*                                       *
*                (ARIS)                 *
*                                       *
*                                       *
*    WED, SEP 26 1984        13:22:32   *
*                                       *
*****************************************

INPUT DATABASE NAME(S)

>DBNAME


BEGIN INTERACTIVE SESSION

>INPUT TEST

 TYPE $END TO RETURN TO MAIN PROGRAM

 TEST#    (    1) -   8 CHARACTERS
>AW001
 MODEL#   (    1) -   8 CHARACTERS
>SH10
 TUNNEL   (    1) -   8 CHARACTERS
>LTPT
 ENGINEER(    1) -   8 CHARACTERS
>WILHITE
 COMMENTS(    1) -  24 CHARACTERS
>SHUTTLE LANDING TEST
 DATE     (    1) -   8 CHARACTERS
>05/04/77
 TEST#    (    1) -   8 CHARACTERS
>$END
```

Fig A8 - Initiate session and enter a tuple into relation TEST

```
>DLOAD TEST AWTEST

>PRINT TEST

                    **********************
                    *                    *
                    * RELATION TEST      *
                    *                    *
                    **********************

   TEST#          MODEL#          TUNNEL          ENGINEER        COMMENTS                        DATE
-------------------------------------------------------------------------------------------------------
   AW001          SH10            LTPT            WILHITE         SHUTTLE LANDING TEST            05/04/77
   LA22           SH10            UNITARY         SPENCER         SHUTTLE SUPERSONIC TEST         05/15/78
   LA70           SH10            LTPT            SPENCER         SHUTTLE SUBSONIC TEST           01/05/80
   0A22           SH43            16 FOOT         DIAMOND         ROCKWELL SHUTTLE STUDY          12/22/77
   T4433          SST3            LTPT            LAMB            SST SUBSONIC ENGINE TEST        10/15/73

>RELATION TEST

                    **********************
                    *                    *
                    * RELATION TEST      *
                    *                    *
                    **********************

 ATTRIBUTE          TYPE      NWORDS PRIMARY KEY INVERSION

 TEST#    (    1)  CHAR*  8      2         N          Y
 MODEL#   (    1)  CHAR*  8      2         N          N
 TUNNEL   (    1)  CHAR*  8      2         N          N
 ENGINEER(    1)  CHAR*  8      2         N          N
 COMMENTS(    1)  CHAR* 24      6         N          N
 DATE     (    1)  CHAR*  8      2         N          N

       5 ENTRIES PRESENTLY
```

Fig A9 - Load tuple data into relation TEST from an external file

```
>DLOAD MODEL AWMODEL

>PRINT MODEL

                    **********************
                    *                    *
                    * RELATION MODEL     *
                    *                    *
                    **********************
```

MODEL#	TYPE	COMMENTS	SCALE
SH10	SHUTTLE	ROCKWELL HIGH FIDELITY	1.00000E-2
SH43	SHUTTLE	ROCKWELL HIGH FIDELITY	3.00000E-2
SST3	SST	MIXED-MODE JET ENGINE MO	5.00000E-2

```
>DLOAD TEST-RUN AWT-R

>PRINT TEST-RUN

                    **********************
                    *                    *
                    * RELATION TEST-RUN  *
                    *                    *
                    **********************
```

TEST#	RUN#	COMMENTS	CONFIG	C1	C2	POLAR	MACH
LA70	1	BODY ALONE	B	0.00000E+0	0.00000E+0	ALPHA	4.00000E-1
LA70	2	WING-BODY	BW	0.00000E+0	0.00000E+0	ALPHA	4.00000E-1
LA70	3	ELEVON AT 10 DEG	BW	0.00000E+0	0.00000E+0	ALPHA	4.00000E-1
LA70	4	RUDDER AT 2 DEGR	BWT	0.00000E+0	0.00000E+0	ALPHA	4.00000E-1
OA22	1	WING-BODY-TAIL	BWT	0.00000E+0	0.00000E+0	ALPHA	9.90000E-1
OA22	2	WING-BODY-TAIL	BWT	0.00000E+0	0.00000E+0	BETA	1.02000E+0
LA22	1	WING #1	BWT	0.00000E+0	0.00000E+0	ALPHA	2.45000E+0
LA22	2	WING #2	BWT	0.00000E+0	0.00000E+0	ALPHA	2.45000E+0
LA22	3	WING #1	BWT	0.00000E+0	0.00000E+0	ALPHA	2.45000E+0
LA22	4	WING #2	BWT	0.00000E+0	0.00000E+0	ALPHA	2.45000E+0
T4433	2	ENGINE AT 50% TH	BWT	0.00000E+0	0.00000E+0	ALPHA	3.00000E-1
T4433	3	ENGINE AT 100% T	BWT	0.00000E+0	0.00000E+0	ALPHA	3.00000E-1

Fig A10 - Load tuple data into relations MODEL and TEST-RUN from external files

```
>DLOAD TDATA AWDATA

>PRINT TDATA
```

****
* RELATION TDATA *
****

TEST#	RUN#	POINT	DATA(1)	DATA(2)	DATA(3)	DATA(4)
LA70	1	1	-2.20000E+0	0.00000E+0	-1.80000E-1	8.90000E-2
LA70	1	2	-6.00000E-2	0.00000E+0	-4.00000E-2	8.30000E-2
LA70	1	3	4.40000E+0	0.00000E+0	2.10000E-1	9.70000E-2
LA70	1	4	8.80000E+0	0.00000E+0	4.00000E-1	1.40000E-1
LA70	1	5	1.24000E+1	0.00000E+0	5.50000E-1	2.01000E-1
LA70	2	1	-2.30000E+0	0.00000E+0	-2.50000E-1	9.40000E-2
LA70	2	2	-5.00000E-2	0.00000E+0	-2.00000E-2	8.70000E-2
LA70	2	3	4.20000E+0	0.00000E+0	3.30000E-1	1.01000E-1
LA70	2	4	8.70000E+0	0.00000E+0	4.80000E-1	1.44000E-1
LA70	2	5	1.23000E+1	0.00000E+0	6.20000E-1	2.05000E-1
LA70	3	1	-2.70000E+0	0.00000E+0	-3.50000E-1	9.70000E-2
LA70	3	2	0.00000E+0	0.00000E+0	8.00000E-2	9.10000E-2
LA70	3	3	4.60000E+0	0.00000E+0	4.30000E-1	1.05000E-1
LA70	3	4	8.20000E+0	0.00000E+0	5.80000E-1	1.48000E-1
LA70	3	5	1.21000E+1	0.00000E+0	7.20000E-1	2.09000E-1
LA70	4	1	-2.70000E+0	0.00000E+0	-3.50000E-1	9.70000E-2
LA70	4	2	0.00000E+0	0.00000E+0	8.00000E-2	9.10000E-2
LA70	4	3	4.60000E+0	0.00000E+0	4.30000E-1	1.05000E-1
LA70	4	4	8.20000E+0	0.00000E+0	5.80000E-1	1.48000E-1
LA70	4	5	1.21000E+1	0.00000E+0	7.20000E-1	2.09000E-1
OA22	1	1	0.00000E+0	0.00000E+0	-4.20000E-2	8.30000E-2
OA22	1	2	2.00000E+0	0.00000E+0	9.70000E-2	8.56000E-2
OA22	1	3	4.00000E+0	0.00000E+0	2.07000E-1	9.66000E-2
OA22	1	4	8.00000E+0	0.00000E+0	2.94000E-1	1.13800E-1
OA22	2	1	0.00000E+0	0.00000E+0	-4.20000E-2	8.30000E-2
OA22	2	2	0.00000E+0	2.00000E+0	9.70000E-2	8.56000E-2
OA22	2	3	0.00000E+0	4.00000E+0	2.07000E-1	9.66000E-2
OA22	2	4	0.00000E+0	6.00000E+0	2.94000E-1	1.13800E-1
LA22	1	1	0.00000E+0	0.00000E+0	-2.30000E-2	6.40000E-2
LA22	1	2	1.00000E+1	0.00000E+0	4.89000E-1	1.10000E-1
LA22	2	1	0.00000E+0	0.00000E+0	-2.40000E-2	5.70000E-2
LA22	2	2	1.00000E+1	0.00000E+0	5.03000E-1	9.70000E-2
LA22	3	1	0.00000E+0	0.00000E+0	2.00000E-3	0.00000E+0
LA22	3	2	1.00000E+1	0.00000E+0	5.04000E-1	1.21000E-1
LA22	4	1	0.00000E+0	0.00000E+0	4.00000E-3	6.10000E-2
LA22	4	2	1.00000E+1	0.00000E+0	5.54000E-1	1.01000E-1
T4433	1	1	3.00000E+0	0.00000E+0	3.25000E-1	2.90000E-2
T4433	2	1	3.00000E+0	0.00000E+0	3.25000E-1	2.20000E-2

Fig A11 - Load tuple data into relation TDATA from the external file AWDATA

```
>SELECT * FROM TEST

                    **********************
                    *                    *
                    * RELATION TEST      *
                    *                    *
                    **********************
```

TEST#	MODEL#	TUNNEL	ENGINEER	COMMENTS	DATE
AW001	SH10	LTPT	WILHITE	SHUTTLE LANDING TEST	05/04/77
LA22	SH10	UNITARY	SPENCER	SHUTTLE SUPERSONIC TEST	05/15/78
LA70	SH10	LTPT	SPENCER	SHUTTLE SUBSONIC TEST	01/05/80
DA22	SH43	16 FOOT	DIAMOND	ROCKWELL SHUTTLE STUDY	12/22/77
T4433	SST3	LTPT	LAMB	SST SUBSONIC ENGINE TEST	10/15/73

```
>PRINT TEST

                    **********************
                    *                    *
                    * RELATION TEST      *
                    *                    *
                    **********************
```

TEST#	MODEL#	TUNNEL	ENGINEER	COMMENTS	DATE
AW001	SH10	LTPT	WILHITE	SHUTTLE LANDING TEST	05/04/77
LA22	SH10	UNITARY	SPENCER	SHUTTLE SUPERSONIC TEST	05/15/78
LA70	SH10	LTPT	SPENCER	SHUTTLE SUBSONIC TEST	01/05/80
DA22	SH43	16 FOOT	DIAMOND	ROCKWELL SHUTTLE STUDY	12/22/77
T4433	SST3	LTPT	LAMB	SST SUBSONIC ENGINE TEST	10/15/73

Fig A12 - Example of the default SELECT command

a) >SELECT TEST# FROM TEST WHERE ENGINEER EQ SPENCER &
> OR MODEL# EQ SH10

```
**********************
*                    *
* RELATION TEST      *
*                    *
**********************
```

TEST#
AW001
LA22
LA70

b) >SELECT TEST#,RUN#,MACH FROM TEST-RUN WHERE MACH EQ 1 TOL .5

```
**********************
*                    *
* RELATION TEST-RUN  *
*                    *
**********************
```

TEST#	RUN#	MACH
OA22	1	9.90000E-1
OA22	2	1.02000E+0

c) >SELECT * FROM TEST WHERE ENGINEER EQ SPENCER UP TEST#

```
**********************
*                    *
* RELATION TEST      *
*                    *
**********************
```

TEST#	MODEL#	TUNNEL	ENGINEER	COMMENTS	DATE
LA22	SH10	UNITARY	SPENCER	SHUTTLE SUPERSONIC TEST	05/15/78
LA70	SH10	LTPT	SPENCER	SHUTTLE SUBSONIC TEST	01/05/80

d) >SELECT POINT.DATA(1) FROM TDATA WHERE TEST# EQ LA70 AND RUN# EQ 1 GIVING LA701 RENAME #.ALPHA

>PRINT LA701

```
**********************
*                    *
* RELATION LA701     *
*                    *
**********,***********
```

POINT	ALPHA
1	-2.20000E+0
2	-6.00000E-2
3	4.40000E+0
4	8.80000E+0
5	1.24000E+1

Fig A13 - Examples of the SELECT command

a) >PRINT TEST

```
**********************
*                    *
* RELATION TEST      *
*                    *
**********************
```

TEST#	MODEL#	TUNNEL	ENGINEER	COMMENTS	DATE
AW001	SH10	LTPT	WILHITE	SHUTTLE LANDING TEST	05/04/77
LA22	SH10	UNITARY	SPENCER	SHUTTLE SUPERSONIC TEST	05/15/78
LA70	SH10	LTPT	SPENCER	SHUTTLE SUBSONIC TEST	01/05/80
OA22	SH43	16 FOOT	DIAMOND	ROCKWELL SHUTTLE STUDY	12/22/77
T4433	SST3	LTPT	LAMB	SST SUBSONIC ENGINE TEST	10/15/73

>DELETE TEST WHERE TEST# EQ T4433

>PRINT TEST

```
**********************
*                    *
* RELATION TEST      *
*                    *
**********************
```

TEST#	MODEL#	TUNNEL	ENGINEER	COMMENTS	DATE
AW001	SH10	LTPT	WILHITE	SHUTTLE LANDING TEST	05/04/77
LA22	SH10	UNITARY	SPENCER	SHUTTLE SUPERSONIC TEST	05/15/78
LA70	SH10	LTPT	SPENCER	SHUTTLE SUBSONIC TEST	01/05/80
OA22	SH43	16 FOOT	DIAMOND	ROCKWELL SHUTTLE STUDY	12/22/77

b) >ASSIGN WILHITE TO ENGINEER IN TEST WHERE TEST# EQ LA22
>PRINT TEST

```
**********************
*                    *
* RELATION TEST      *
*                    *
**********************
```

TEST#	MODEL#	TUNNEL	ENGINEER	COMMENTS	DATE
AW001	SH10	LTPT	WILHITE	SHUTTLE LANDING TEST	05/04/77
LA22	SH10	UNITARY	WILHITE	SHUTTLE SUPERSONIC TEST	05/15/78
LA70	SH10	LTPT	SPENCER	SHUTTLE SUBSONIC TEST	01/05/80
OA22	SH43	16 FOOT	DIAMOND	ROCKWELL SHUTTLE STUDY	12/22/77

Fig A14 - Examples of the DELETE and ASSIGN commands

```
>SELECT MODEL# FROM MODEL WHERE TYPE EQ SHUTTLE GIVING TEMP

>JOIN TEMP AND TEST OVER MODEL# GIVING TEMP2

>PRINT TEMP2

                    **********************
                    *                    *
                    * RELATION TEMP2     *
                    *                    *
                    **********************
```

TEST#	MODEL#	TUNNEL	ENGINEER	COMMENTS	DATE
AW001	SH10	LTPT	WILHITE	SHUTTLE LANDING TEST	05/04/77
0A22	SH43	16 FOOT	DIAMOND	ROCKWELL SHUTTLE STUDY	12/22/77

```
>SELECT ENGINEER FROM TEMP2 WHERE TUNNEL EQ '16 FOOT'
                    **********************
                    *                    *
                    * RELATION TEMP2     *
                    *                    *
                    **********************
```

ENGINEER
DIAMOND

```
>JOIN TEST AND TEMP OVER MODEL# GIVING TEMP3

>PRINT TEMP3

                    **********************
                    *                    *
                    * RELATION TEMP3     *
                    *                    *
                    **********************
```

TEST#	TUNNEL	ENGINEER	COMMENTS	DATE	MODEL#
AW001	LTPT	WILHITE	SHUTTLE LANDING TEST	05/04/77	SH10
LA22	UNITARY	WILHITE	SHUTTLE SUPERSONIC TEST	05/15/78	SH10
LA70	LTPT	SPENCER	SHUTTLE SUBSONIC TEST	01/05/80	SH10
0A22	16 FOOT	DIAMOND	ROCKWELL SHUTTLE STUDY	12/22/77	SH43

Fig A15 - Example of the JOIN command

a) >SELECT ENGINEER FROM TEMP3 WHERE TUNNEL EQ LTPT GIVING TEMP5

>SELECT ENGINEER FROM TEMP2 WHERE TUNNEL EQ LTPT GIVING TEMP4

>UNION TEMP5 AND TEMP4

ENGINEER
WILHITE
SPENCER
WILHITE

b) >INTERSECT TEMP5 WITH TEMP4

ENGINEER
WILHITE

Fig A16 - UNION and INTERSECT commands and an example of using subscripted attributes

a) >MIN MACH IN TEST-RUN

```
**********************
*                    *
* RELATION TEST-RUN  *
*                    *
**********************

        MACH
----------------
  3.00000E-1
```

b) >MAX MACH IN TEST-RUN

```
**********************
*                    *
* RELATION TEST-RUN  *
*                    *
**********************

        MACH
----------------
  2.45000E+0
```

c) >RELA

```
**************************
*                        *
* RELATIONS IN DATABASE  *
*                        *
**************************

    RELATION MODEL
    RELATION TDATA
    RELATION TEST
    RELATION TEST-RUN

**************************
*                        *
* RELATIONS IN TEMPORARY *
*        DATABASE        *
*                        *
**************************

    RELATION LA701
    RELATION TEMP
    RELATION TEMP2
    RELATION TEMP3
    RELATION TEMP4
    RELATION TEMP5

>QUIT
```

Fig A17 - MIN, MAX, and RELATION commands and termination of the session

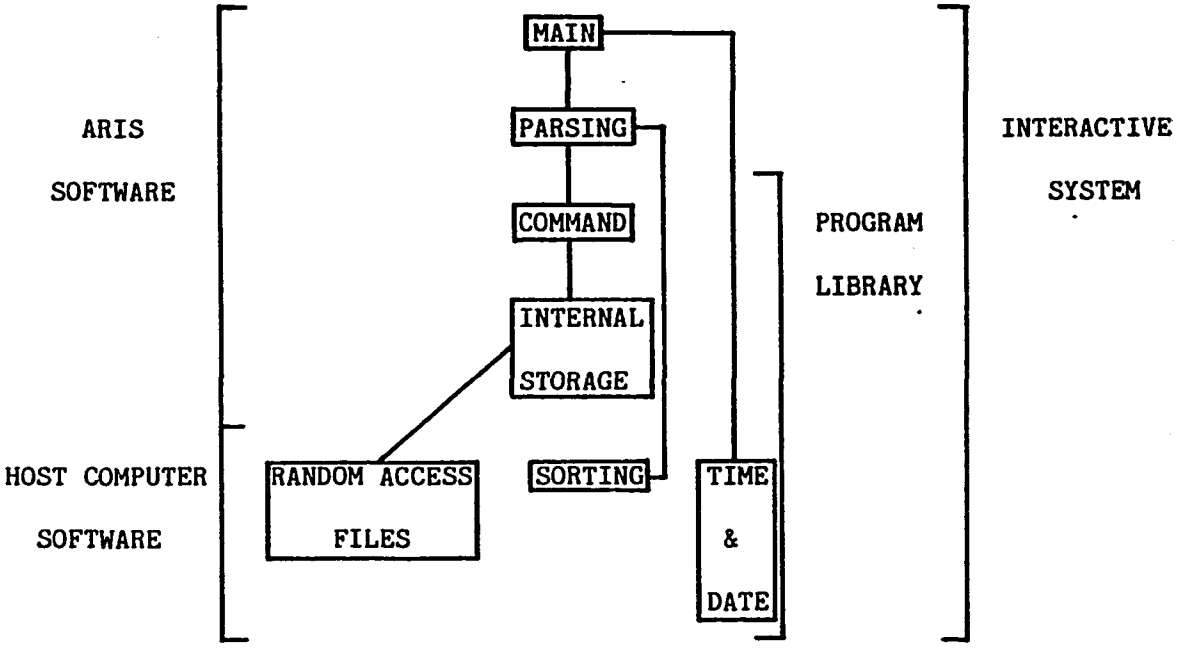


Fig A18 - ARIS program architecture

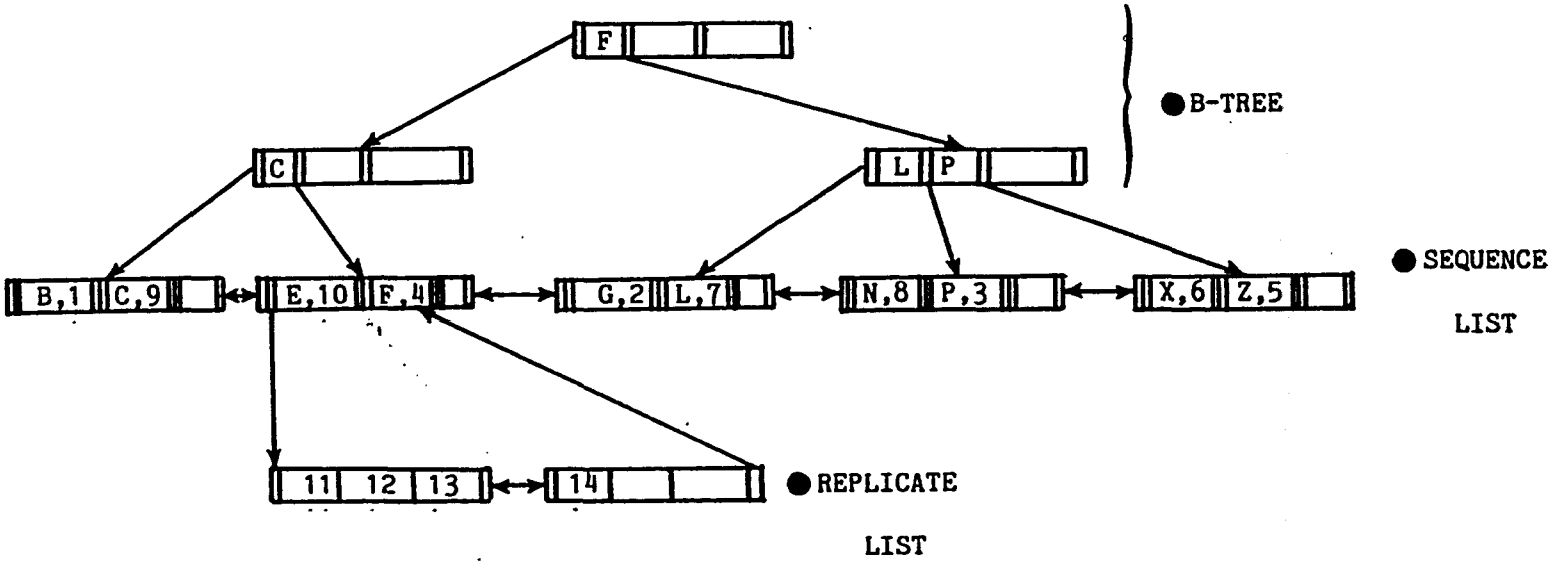


Fig A19 - Schematic of the internal structure

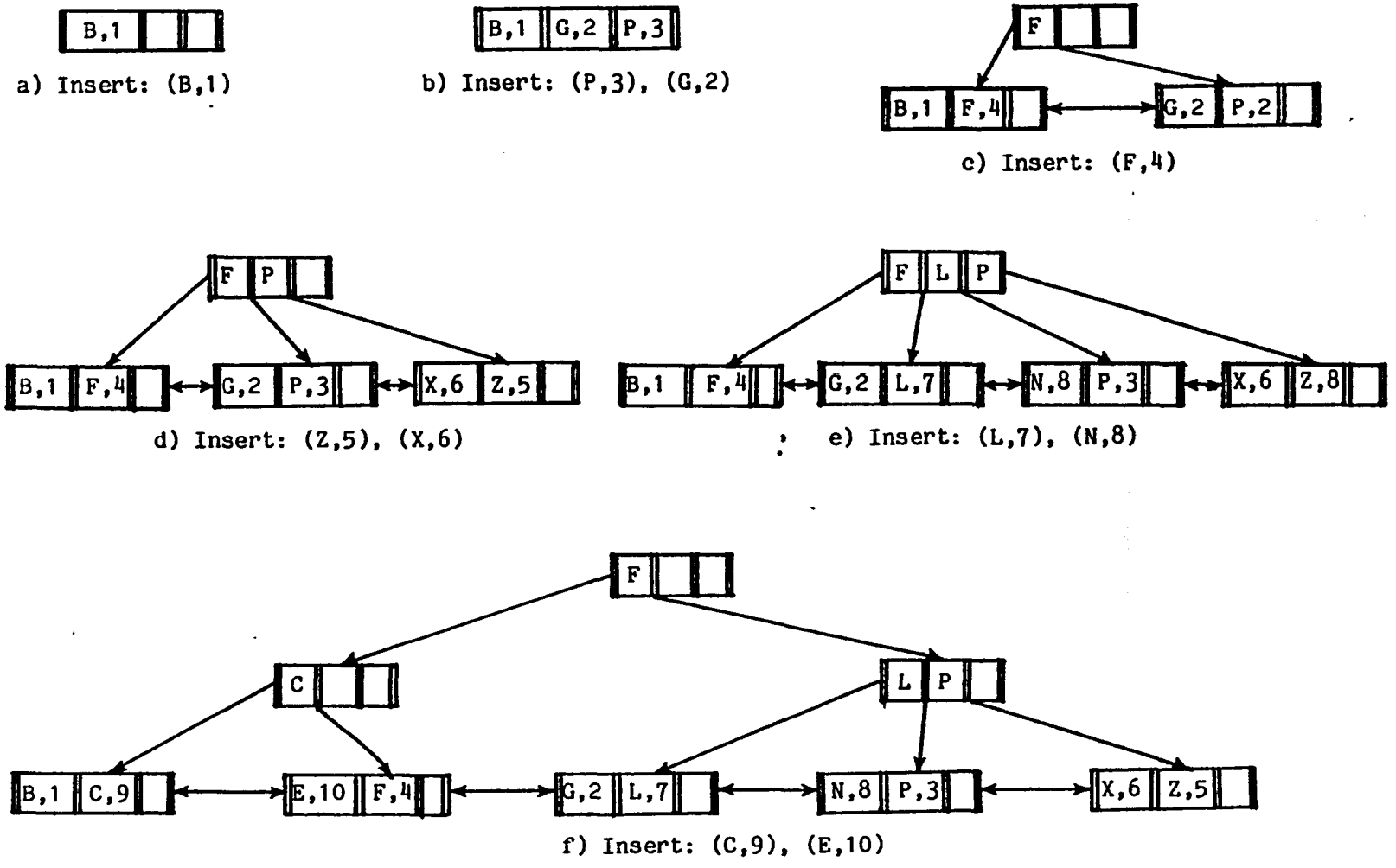


Fig A20 - Progressive growth of the B* -tree with insertions

TIMING IS RELATIVE TO A 120 KEY BLOCK SIZE

INSERTING 10,000 KEYS TOOK 283 SECONDS

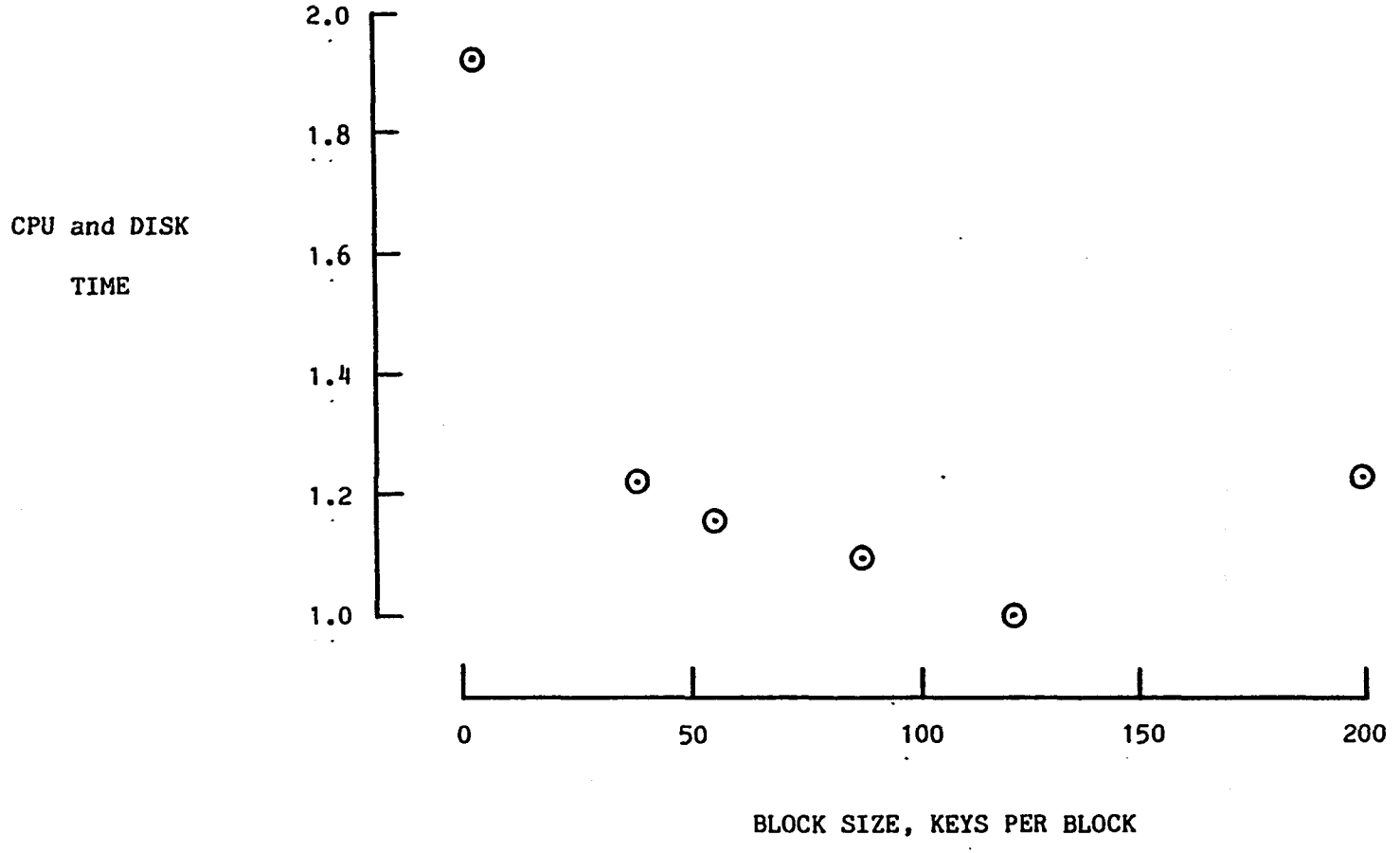


Fig A21 - Block size optimization

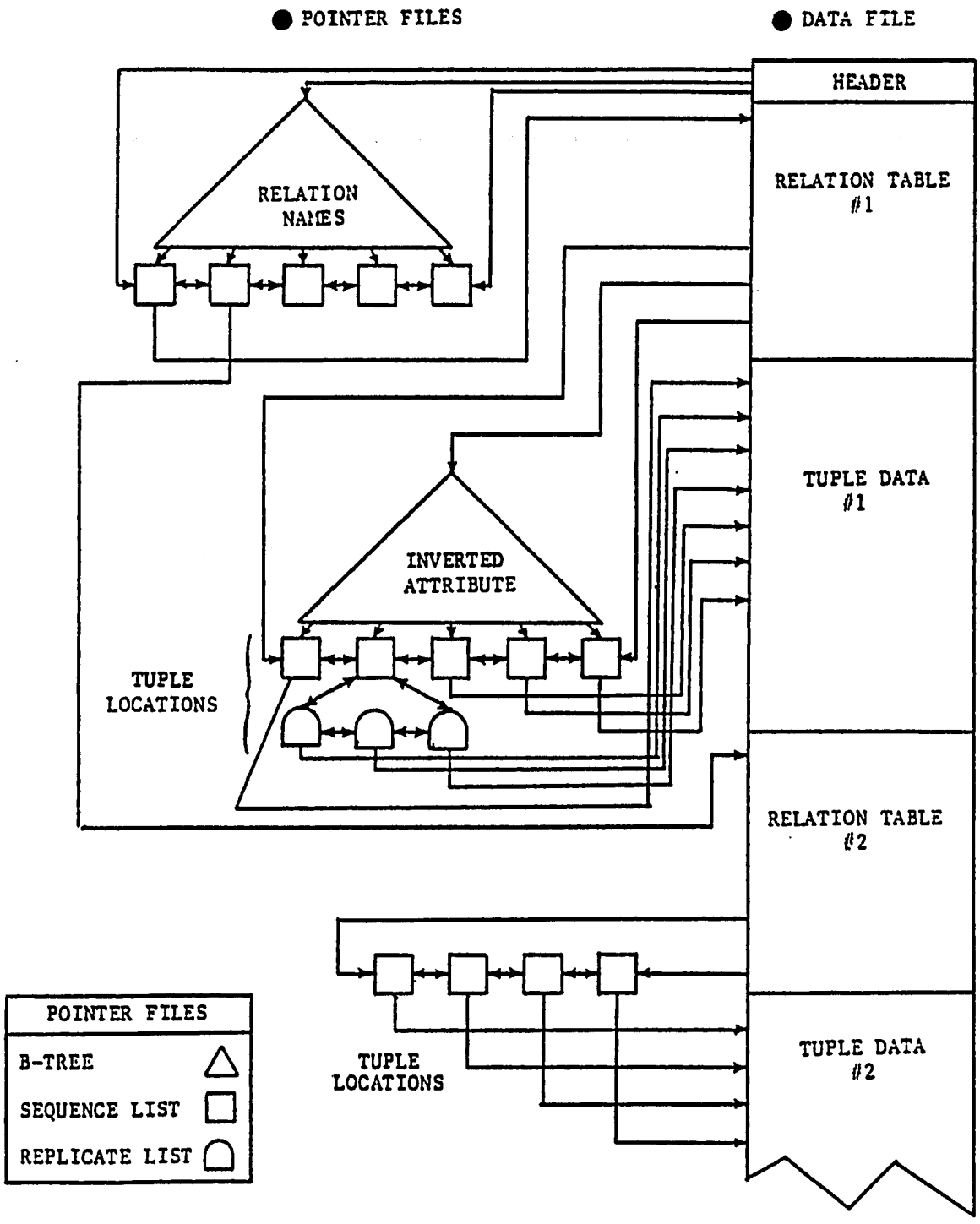


Fig A22 - Access structure architecture

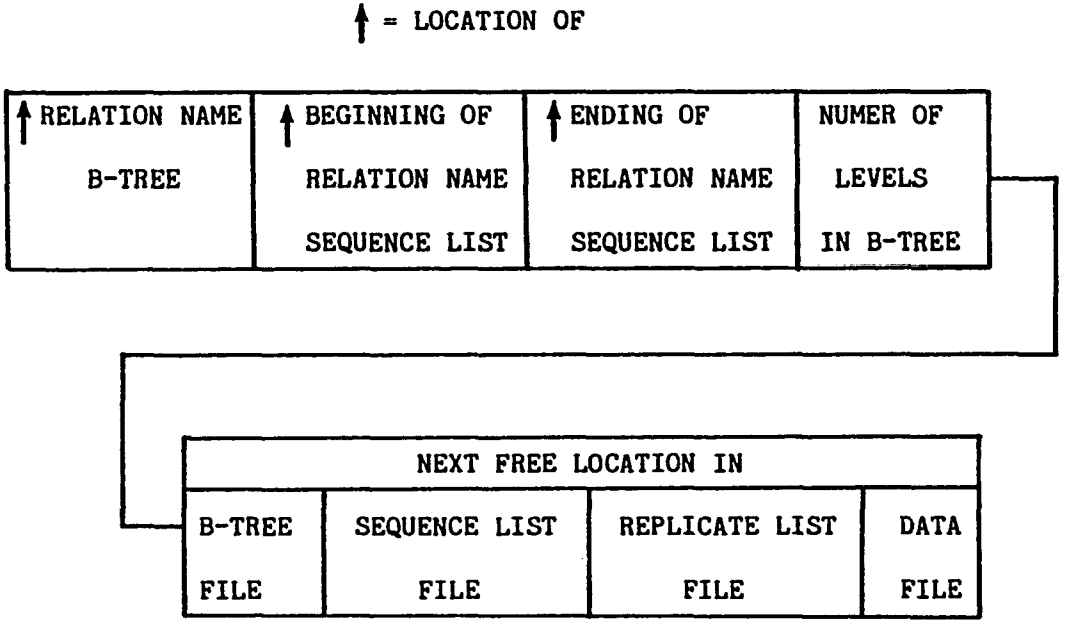


Fig A23 - Header record contents (8 words)

```
RELATION NAME

NUMBER OF ATTRIBUTES
     NAME
     TYPE
     DIMENSION
     NUMBER OF CHARACTERS

NUMBER OF ATTRIBUTE INVERSIONS (50 MAXIMUM)
     ATTRIBUTE NUMBER
     LOCATION OF B-TREE
     LOCATION OF BEGINNING OF SEQUENCE LIST
     LOCATION OF ENDING OF SEQUENCE LIST
     LEVELS IN B-TREE
     DELETION FLAG

NUMBER OF ATTRIBUTES IN PRIMARY KEY (50 MAXIMUM)
     ATTRIBUTE NUMBERS

NUMBER OF WORDS IN A TUPLE

NUMBER OF ENTRIES
```

Fig A24 - Relation table contents (558 words)