INTRODUCTION

The Executive system of COMRADE I has provided the basic "living quarters" for all members of the computer-aided design team. Three distinct types of individuals have finally been united in this environment: the user, the person with little or no computer experience, possibly an aeronautical engineer or naval architect who received his training before electronic data processing was in vogue; the application programmer, not necessarily the recipient of any schooling regarding the design he is helping to implement; and the subsystem designer, the link between the other two, the person who must be part engineer, part computer scientist, and—most important—part nursemaid, soothing the inevitable wounds of the other team members, each of whom is a little too close to his own area of specialization to see, at all times, the design system in a completely objective manner.

But now an obvious problem arises with respect to the role of the subsystem designer. This subsystem designer is neither ever-present nor all-knowing. He can perhaps plan a smooth design as tasks proceed from the initial feasibility model to the finished product, but he can most assuredly not monitor the immense flow of data that is emerging from the design: data produced by one design module for use in several other design modules; data which is output from a design module and must be immediately available upon request by the engineer-user to aid in his decision-making process; data which serves as the only means of communication between designers in different disciplines within an integrated system; and finally, data which represents the finished product, the design of a nuclear submarine, a supersonic jet, or a multi-million dollar medical school and hospital.

There must be a data base management system to handle the vast number of data items considered to be the output of the preliminary design stage of a U.S. Navy ship. This data base management system must provide capabilities for the creation and management of complex file and data structures; the storing of enormous quantities of data; the selective, "item by item" update of this information; and the rapid retrieval of data base information either directly by name or by conditional expressions.

Moreover, these capabilities should not only be extremely easy to use by non-computer-oriented personnel, but should also be efficient (both in processing time and main memory utilization) when dealing with great quantities of data; such efficiency is expected by the application programmer and subsystem designer, both of whom are responsible for the overall use of computer resources within the design system.

The different backgrounds and needs of the people involved in computer-aided design cannot be emphasized too much, for it was the existence of such diversity that led to the development of the COMRADE Data Management System (CDMS).

The CDMS capabilities are built upon two major software components, these being the

- Data Storage Facility—Programs to build and randomly process user-named blocks of data on disk storage; and programs to build and process "inverted" element lists for retrieval by value; and the
- Block Type Definition Facility—A mechanism for naming, defining, and specifying individual data fields within a block, thereby enabling conversational use of the data base.

Building upon these components, the CDMS has attempted to satisfy the requirements of the developers and users of a computer-aided design system by providing various functional capabilities suited to different processing modes:

- Interactive commands
- Stand-alone batch programs
- Subroutine library

This introduction has tried to bring home the fact that it is no simple job to satisfy all members of the computer-aided design team at the same time. In particular, papers on computer-aided design are rarely equally valuable to
all involved in the design effort. For this reason, the remainder of this paper has been written primarily with only one type of user in mind, the ultimate user of the system, the design engineer. The focus is on capabilities—what are they, and how are they used? Explanations of programming techniques, for the most part, are omitted. Therefore, the Data Storage Facility, the mechanism for creating and maintaining data blocks (logical records) and “inverted” lists is not discussed in this paper. That facility is discussed, however, in Reference 3.

**BLOCK TYPE DEFINITION FACILITY**

While it is possible for one person, responsible for his own programs and data files, to employ a storage management capability such as the COMRADE Data Storage Facility, he then must be aware of the address and data structure of each block for use in data processing. The complexity of computer-aided design data bases, the multiplicity of users and design tasks, and the diverse roles of the members of the design team in an integrated system, however, render such an approach to data management impractical.

And so, enter the COMRADE Block Type Definition Facility (BTDF)! If the Data Storage Facility is the “thought,” then the BTDF is the “voice,” providing the means of communication between the engineer-user and the subsystem designer, the subsystem designer and the application programmer, the programmer and the programs, and, in general, between all of these data users and their data bases.

The BTDF is the mechanism by which the so-called data administrator defines the formats of all data blocks within the data base, thereby giving names by which each data item may be referenced. A CDMS data base may consist of a number of block type definitions. Some subset of the total number of data blocks within the data base will be thought of as belonging to each block type. For example, Figure 1 illustrates a “Presidents” Data Base,* in which five block types are defined and relationships structured. A printout of one of these types—the block type PRES—is shown in Figure 2. For block type PRES, which describes the personal information regarding each United States president, George Washington through Lyndon Johnson, there are 35 data blocks. There are 45 data blocks of type ELECTION, one for each presidential election between 1789 and 1964; 53 data blocks of type ADMIN; 90 of type CONGRESS; and 50 of type STATES. Thus five “block types” define the format of all 273 data blocks within the data base.

A block type, and therefore all data blocks of this type, consists of named data elements, repeating groups, and sub-blocks. A data element—the smallest quantity that may be referenced within a CDMS data base—may be defined as one of four data types: alphanumeric, real, integer, or pointer. An element of any of these types may be defined to be a single value or an array. In addition, every single-valued element may or may not be inverted. Therefore, every element appearing in a block type definition will have one status out of a possible twelve.

There may be a need in some data blocks to store more than one occurrence of a data element or elements. To this end, a set of consecutively defined elements may be united by membership within a named “repeating group.” For example, a set of points \((X_i, Y_i, Z_i)\) on a curve may be defined as three single-valued real elements, all members of repeating group COORD. The user is now free not only to request the retrieval of all \(X\)'s, or of a specific \(Y\) or \(Z\), but also to retrieve, say, the third point on a curve by asking for the third “occurrence” of repeating group COORD. Although the name “repeating group” implies repetition, this is by no means necessary. It is sometimes very handy for a higher level name to be assigned to a group of data elements even if there can never be more than one occurrence for the group within the data block (e.g., elements STREET, CITY, STATE, ZIPCODE might be grouped under the repeating group ADDRESS).

The elements of a block type may be divided into sub-blocks. The use of sub-blocks is twofold. First, it provides still another level of data element grouping so that one name may be used to refer to perhaps 20 or 30 data elements, thereby greatly simplifying a retrieval request. In addition, if a retrieval or update request on an element indicates its sub-block membership, the transaction will be processed in a more efficient manner than if the block type were not partitioned in this way or if no sub-block information were provided.

The mechanics of block type definition are indeed trivial. The “define block type” program prompts the user for a block type name, number of sub-blocks and first sub-block name. The user then defines the elements of the first sub-block (i.e., element name, status). When all elements are defined, the user types END. The program then asks for repeating group definitions (i.e., name and

---

* A data base containing information regarding the U.S. Presidents has been used as the basis for all examples in this paper.

---

Figure 1—“Presidents” data base structure
From the collection of the Computer History Museum (www.computerhistory.org)

MODES OF USING CDMS

Batch programs

COMRADE is basically a software system for the interactive designer and CDMS is basically a data management system for the interactive user. However, certain data management tasks are more suited to stand-alone batch processing. The first such task that comes to mind is a massive data base loading procedure. This procedure might occupy the central processor for minutes and some other resources of the computer system for hours. Therefore, such a run would best be made at a low-priority rate during non-prime time for financial reasons. The COMRADE Bulk Data Loader is such a program. This program will accept card image input specifying data blocks to be defined and values to be assigned to elements within these data blocks. A new data base may be created by the Bulk Data Loader or data blocks may be added to an existing data base.

The input to this program consists of Block Type records (only when the next block to be defined is of a different type than the previous block). Block Name records, and, optionally, Sub-block records. The majority of the input, however, is found in the Data records. Each Data record may define a variable number of data fields. The input to this program consists of Block Type records (only when the next block to be defined is of a different type than the previous block). Block Name records, and, optionally, Sub-block records. The majority of the input, however, is found in the Data records. Each Data record may define a variable number of data fields. The input to this program consists of Block Type records (only when the next block to be defined is of a different type than the previous block). Block Name records, and, optionally, Sub-block records. The majority of the input, however, is found in the Data records. Each Data record may define a variable number of data fields.
Informally documented in two reports, the first by R. Stevens and T. Rhodes (Jan 1969); the second by T. Corin and T. Rhodes (Feb 1971).

The examination of these software packages is discussed in the Corin and Rhodes informal report referenced earlier.
• a non-repeating group element
• one or all of the occurrences of an element within a repeating group
• one or all of the existing occurrences of an entire repeating group
• a new repeating group occurrence to be added
• a sub-block
• an entire data block

The Basic Retrieval Program may also be found in the command procedure RETRIEVAL. Another program accessible through this procedure is the File Statistics Program. This program retrieves and displays information regarding the data base, such as

• name and size of all data blocks
• name and description of any or all block types
• name and value range of all inverted elements

While the CDMS Basic Retrieval Program provides a mechanism for data retrieval of a certain nature, namely by physical location, the COMRADE Query Processor satisfies another type of retrieval request, i.e., retrieval by condition. The goal of the Query Processor was to provide the user with a language, not totally unlike English, with which he could conditionally query the data base on both element values and file structure. This program is truly the union of all features within CDMS: the Data Storage Facility which contains the software necessary to process "inverted" element lists thereby, providing the rapid response necessary to an on-line query system; the Block Type Definition Facility, essential to the expression of a query; and file structure traversal via elements of the "pointer" data type.

The syntax of a query is

VERB OBJECT LIST OF CLAUSE WHERE CLAUSE;

VERB—The destination of the output of a particular query is governed by the user's choice of verb. PRINT will direct all output to the user's terminal. FILE will return only diagnostics directly to the user and write all query results on a scratch file. Prior to the completion of command procedure RETRIEVAL, the user will have the opportunity to dispose of this file as he sees fit, either to have it printed at one of several remote sites, or to keep this file for further use.

OBJECT LIST—The object list is simply a list of those items to be printed or filed for all "hit" blocks, i.e., for all data blocks satisfying the conditions of the query. Valid object list items are

• element names
• repeating group names
• BN (block name)
• BT (block type)

The appearance of a repeating group name in the object list will result in the output of all member elements for the "hit" data blocks. A maximum of twenty items may be requested, with a repeating group name counting as one, regardless of the number of member elements.

OF CLAUSE—OF. search path
where 'search path' consists of a starting data block name and a sequence of element names separated by slashes, each element being of "pointer" data type. This arbitrary path will be traversed, with all terminal data blocks considered to be "hits." The search path may consist of zero to eight pointer names, with zero signifying a basic retrieval operation.

Examples

(a) PRINT WIFE .OF. LINCOLN:
This is simply a basic retrieval. The element WIFE in data block LINCOLN will be retrieved and printed.

(b) PRINT WIFE .OF. E1960/ PRESPTR;
The data block E1960 is of block type ELECTION. One of the elements, PRESPTR, of the block type, is a pointer to the data block containing the personal information regarding the man who won that election, i.e., Kennedy. Therefore the "answer" is Jacqueline.

(c) PRINT CAPITAL .OF. E1960/
PRESPTR/STATEPTR;
In data block KENNEDY (i.e., E1960/PRESPTR), there is an element STATEPTR, a pointer to JFK's state of birth, Massachusetts. Hence the output of the query is Boston.

These simple examples do not illustrate the possibility of a rapidly expanding tree structure such as

.OF. BLOCK1/PTRA/PTRB

where BLOCK 1 has a 100 word pointer array PTRA, and each of the data blocks referenced by this array contains a 100 word pointer array, PTRB. Quickly this search path has produced a "hit" list of 10000 data blocks.

Now, a final word on the OF CLAUSE. If the user would like to specify the same search path in consecutive queries, he need only type .OF. SAME in the second query.

WHERE CLAUSE—WHERE. conditional expression
In this part of the query, the user establishes the conditions a data block must satisfy to be considered a "hit" via inverted list searching. The simplest conditional expression is a relational expression. A relational expression is defined as

element relational operator value

where the operator is one of the following: .EQ., .NE., .GE., .GT., .LE., .LT.
or

element .BET. value1, value2

Since both alphanumeric and pointer values are character strings, alphanumeric values must be surrounded by quotes to differentiate between these two data types.
Numeric values may be represented with or without the decimal point; however, exponential notation is not permitted.

Examples of legitimate relational expressions are

\[
\begin{align*}
&\text{WEIGHT .NE. 100} \\
&\text{AREA .BET. 18., 29.2} \\
&\text{NAME .EQ. "JOHN SMITH"}
\end{align*}
\]

More complex conditional expressions may be formed by joining up to five relational expressions with the Boolean operators .AND. and .OR. As in FORTRAN, the natural precedence of .AND. first, .OR. second, may be altered by parenthesizing parts of the expression. Finally the Boolean operator .NOT. may be used to form the complement of all or part of the conditional expression.

For example,

(a) \text{WEIGHT .LT. 20 .AND. .NOT.(NAME .EQ. "SMITH" .OR. NAME .EQ. "JONES")}
(b) .NOT.(A.EQ.10 .A)JD. XPTR .NE. BLOCK12

Figure 5—Sample queries

Just as .OF. SAME specifies a repeated search path, .WHERE. SAME specifies a repeat of the previous conditional expression. Not only does this feature save time on query input, but processing time is reduced in that the "hit" list from the last inverted list search has been saved.

Figure 5 presents a variety of queries on the Presidents Data Base thereby illustrating the range of capabilities of the COMRADE Query Processor.

**SUMMARY**

A successful computer-aided design system is a mixture of diverse design tasks, administrative tasks, and people. Every task and person possesses certain data management requirements. Application programmers require efficient dynamic data base access from their design programs. The data administrator requires a facility for describing an arbitrary data and file structure suited to his design. Project leaders and design engineers require a natural means of communication between themselves and their data bases.

The COMRADE Data Management System was developed with the idea that these requirements can in no way be compromised. Unless all personnel using the data base can do so comfortably and efficiently, the design effort will suffer. Therefore, COMRADE has attempted to provide a data management system which offers a wide variety of capabilities to be used in a wide variety of operational modes by a wide variety of users.
REFERENCES


