DATA MANAGEMENT IN ENGINEERING

J.C. Browne
The University of Texas

SUMMARY

Engineering practice is heavily involved with the recording, organization and management of data. This paper is an introduction to computer based data management with an orientation toward the needs of engineering application. The characteristics and structure of data management systems are discussed. A link to familiar engineering applications of computing is established through a discussion of data structure and data access procedures. An example data management system for a hypothetical engineering application is presented.

NEED FOR DATA MANAGEMENT

Formal data management procedures become necessary for a body of information when the information

- has an extended useful lifetime,
- is shared among or used by a substantial group of workers,
- has established relationships among data items.

The use of computer based data management systems is justified by combinations of several circumstances.

- The volume of data outstrips convenient use through traditional media such as handbooks, microfilm, etc.
- The data is produced through computer processing and will perhaps be subjected to further computer processing.
- The data requires frequent revising and updating.
- There is a large and geographically compact group of users.

It is clear that many types of engineering projects meet both sets of criteria. The design of an aircraft or ship makes a cogent example. The design process depends heavily upon the use of computers. The design process may take several years and involve hundreds of engineers. The design data may involve millions of words of specifications and an immense volume of numeric data. 1% or 2% of the data change on a weekly or monthly basis over much of the design cycle.

Engineers have traditionally been heavily involved in the classical forms of data management such as data compilations, design handbooks and system maintenance manuals. Computer based data management has been relatively slow to penetrate standard engineering practice. This may be in part due to the fact that engineering education tends to stress the use of computers as numerical

problem solvers rather than as information managers. It is certainly in part due to the fact that most existing data management systems are oriented towards commercial and business data processing applications.

Engineers have now begun turning to computer based data management for assistance. Since available data management systems are not in general well-suited to engineering applications there is considerable activity in the engineering community towards designing and implementing data base systems which are useable in engineering environments. It is the purpose of this paper to give a perspective on the design and implementation of such data management systems.

Three recent texts, Martin (ref. 1), Date (ref. 2) and Katzan (ref. 3), cover data management systems in readable fashion.

DATA STRUCTURES, DATA REPRESENTATIONS AND STORAGE MAPPING FUNCTION

The basic concepts of data management, data structuring, data representation and storage mapping functions are presented in the familiar context of general purpose programming languages such as FORTRAN or PL/1. Data management systems present and utilize these concepts in more formal and complex forms.

A data structure consists of a conceptual object, i.e., a sparse array, a name or name set for referring to the object and a set of operations on the object.

A realization of a data structure consists of a storage mapping function which maps the name space of the data structure onto a memory structure and the definition of the operations on the structure in terms of primitive operations.

These definitions are completed by defining a storage or memory. A cell is a physical realization which holds a value. Memory consists of an ordered collection of cells. An address is the location in memory for a given cell. A value is an instantiation of a data object or data structure. A storage mapping function accepts a name as input and produces an address of a cell (or cells) in memory as an output.

The definition and realization of a data structure thus consist of a sequence of actions:

- A structure declaration which defines the data type.
- A name assignment which associates the name with a type or structure.
- The definition of the operations on the structure. The only required operations are of course storage and retrieval.
- An allocation of memory to the named instantiation of the data structure.
- The definition of the mapping function which maps the name space onto the allocated memory space.

This sequence of steps is seldom clearly delineated in traditional programming

languages. A FORTRAN DIMENSION or COMMON declaration of a rectangular array executes all of the above steps except the definition of operations upon the array. DIMENSION A (10,10) recognizes the square array as a data structure of the program, associates the name A with the array, assigns 100 contiguous cells of memory each of which will hold a floating point number and assigns the implicit familiar mapping function.

$$\text{Address } [A(I,J)] = A(1,1)+10(I-1) + (J-1) \qquad (1)$$

The definition of operations on an array (except for I/0 operations) must be defined by the programmer in terms of operations on the primitive data objects.

The most complicated data structure definable in FORTRAN is a multidimensional array of identical objects. PL/1 allows arrays whose elements are not identical. Data management systems may allow the definition of considerably more complex structures which include the stipulation of relationships between the data elements in a structure. The aspect of this problem not familiar to the scientist and engineer is the representation of the data structure in the computer memory system and the definition and implementation of storage mapping functions.

The familiar storage mapping function of equation (1) takes the name A(I,J) as input and evaluates the expression on the right hand side for output. This mapping function has the very useful property of mapping names onto addresses in a unique one-to-one fashion. There are other possible mapping functions which do not have this property even for the simple case of square arrays. Consider, for example, the mapping function

$$\text{Address } [A(I,J)] = (I \times J) \text{ MOD } N \qquad (2)$$

with N = 101. It is easily seen to generate identical addresses for many index pairs. It is the general case in data management applications that the magnitude of the name space is much larger than the potentially realizable address spaces. Thus, storage mapping functions which "fold" the name space into a smaller domain and thus lose the one-to-one property are required. Such storage mapping functions typically have several functional phases and are fairly complex. The example data management system which is specified in the last section of this article uses an inverted file or dictionary look-up storage mapping function to locate records and the mapping function defined succeeding to map data elements onto records.

Figure 1 defines a record structure for data relating to the design cycle of the wing section of an aircraft. Figure 2 displays the heirarchical relationships among the data elements. This structure defines the occurence of 40 data records on the design and evaluation of wing sections. The leftmost numbers in Figure 1 define the level in the definition hierarchy as shown in Figure 2. The components at any level with no immediately succeeding components at a lower level are terminal nodes of a tree. The bracketed numbers on the right hand side of the terminating nodes are the number of primitive data objects in each instance of the defined object. The bracketed numbers on the right hand side of the non-terminal nodes in the tree are the number of instances of the structure for which storage is to be allocated.

It is convenient to describe the structure in tabular form. (See Table 1). It is desired to allocate storage for each record in a contiguous block with each terminal node of the tree being stored contiguously for each instance of the structure or sub-structure. Let us define a reference expression (≡name) which orders the names of the structures from left to right by level.

$$A_1(I_1)A_2(I_2)A_3(I_3)$$

$$A_1(I_1)A_2(I_2)A_3$$

The reference expression $WS(I_1)SD(I_2)DH(I_3)$ refers to the I_3^{th} storage element within the I_2^{th} instance of SD within the I_1^{th} instance of WS. $WS(I_1)SD(I_2)$ refers to the I_2^{th} instance of SD within the I_1^{th} instance of WS while $WS(I_1)SD$ refers to all 10 instances of SD within the I_1^{th} instance of WS. A storage mapping function with the one-to-one property for these reference expressions can be derived (ref. 4):

$$\text{address}[A_1(I_1)A_2(I_2)\ldots A_k(I_k)] = \sum_{i=1}^{k} [Q(A_i) + M(A_i)(I_i-1)]$$

where $Q(A_i)$ and $M(A_i)$ are constants for each record element.

The constants Q and M can be defined recursively.

1. If A_i is a terminal node of the structure, then $M(A_i) = 1$.
2. If A_i is a structure or sub-structure with a typical instance $B_1\ldots B_n$
$$M(A_i) = \sum_{i=1}^{n} C(B_i)M(B_i)$$
3. If $B_1\ldots B_n$ is a sub-structure definition, then
$Q(B_1) = 0$
$Q(B_j) = Q(B_{j-1}) + C(B_{j-1})M(B_{j-1})$, $j>1$
$Q(A) = 0$, for root of tree
4. If B_n is the last item in a sub-structure $B_1\ldots B_n$ of A, then
$M(A) = Q(B_n) + C(B_n)M(B_n)$

The last two rows of Table 1 give the results of the calculations for the record structure of Figure 1.

Bertiss (ref.5) and Elson (ref.6) are good general references for further information on data structure. Knuth (ref. 7 and 8) is the most complete source for work prior to publication data in the areas of his coverage.

AN EXAMPLE DATA MANAGEMENT SYSTEM

This section will illustrate the structure of a prototype data management system for engineering data. The example system will be designed to store and retrieve design data on the wing sections of an aircraft.

The components of a data management system are:

1. A data structure definition capability: This includes a set of primitive data objects and a set of composition rules which will enable a user to create a structure which represents the objects of interest. The set of primitive objects and composition rules comprise what is often called the data definition model or data model in the data management system literature. A structure defined by the composition rules will be said to constitute a logical record.
2. A storage mapping function which enables access to the components of a logical data structure or logical record.
3. A representation which packs logical records onto physical records in executable memory.
4. A storage mapping function which determines the address of a logical block and the address of the physical block which contains it.
5. A block transfer function which transmits physical records to and from auxiliary memory.
6. A query language which allows the user to express his storage/retrieval requests in an application-oriented format. Commercial data management systems often have very highly developed query languages. It is often the interface which sells the system more than its internal performance. It is generally the case in scientific/engineering computing that simple or specialized query languages will be all that is required. The users of the system will often by familiar with programming and programming systems.

We proceed by defining for our example system each of the components previously described.

1. Data definition model: The primitive objects which we will need will be character strings, real numbers, integer numbers and real vectors. We will allow the composition of arbitrary tree structures utilizing these primitive data types. Figure 3 defines a logical record for a wing section similiar to the example in the previous section. The record consists of a character string for the aircraft designation, a set of design parameters including thickness, flexibility coefficient and strut spacing, each of which is a real number and a set of stress values which is a vector of length 25 of real numbers. Figure 4 is a tree structure for this data. The [15] following the record declaration declares that a physical record will contain 15 logical records. The primary purpose of this system is to be able to examine the stress values as a function of design parameters. It is anticipated that entire records will be added or deleted from the file but that records will seldom be altered or modified.
2. Storage mapping functions for logical records: We use the storage mapping function defined for hierarchical structures in the previous section.
3. Data representation in physical memory: The logical record will be organized and stored on physical record blocks (PRB) of 512 words in length. Each logical record will require 30 words. Fifteen logical records will be stored on each physical record block. Forty-five of the remaining sixty-two words will be used for the location in the PRB of logical record instantiations which contain a given design parameter value.
4. A storage mapping function for addressing logical records from physical records: The storage mapping function will utilize an inverted file structure (ref. 1). Each design parameter will be represented as an inverted file. An inverted file is a tabulation of record addresses associated with a given name or structure component whereas a normal file contains the values associated

with each name or structure component. Each entry in the inverted files for design parameters will consist of a design parameter, the number (= address) of each physical block which contains a logical record with that design parameter value and a pointer to the address on that physical block of the set of position numbers for logical records containing that particular design parameter value. Each entry in the inverted file on a given design parameter is sorted in ascending order on the design parameter values. The inverted files will also be stored on 512 word PRB's. It will be assumed for simplicity that the set of entries for a given design parameter will always fit on a single physical record block.

There will be a directory to each inverted file which is kept in executable memory. The directory entries for a given inverted file will consist of the largest and smallest value for a design parameter which is stored on a given inverted file PRB together with the number (= address) of the PRB holding those inverted file entries.

5. Physical record transmission: We will assume that the operating system provides a convenient capability for transmitting fixed length blocks to and from disk storage.
6. Query language: The query language consists of a knowledge of the table structures.

A summary of the relationships between the storage mapping function and a given physical record is illustrated in Figure 5.

This data management system structure will support queries for logical records which specify one, two, or three design parameters. To find all records which have a particular design parameter, say thickness = 0.002", the following process would ensue:

- A search would be made on the directory for thickness to locate the inverted file page (PRB containing 0.002" for the thickness design parameter. This PRB would be loaded into executable memory.
- A search of this page of the inverted file for thickness would return the set of physical record blocks containing the logical records with that thickness parameter and the pointer to the physical record block section which holds the positions on the PRB of the logical records containing the given design parameter.
- These physical records could then be read in from the disk. The logical records would be extracted from the PRB's and examined one by one using the hierarchical record addressing scheme.

To obtain all records which have two particular attributes, say a thickness of 0.002" and a strut separation of 0.8", one would carry out an identical search on the inverted files for both thickness and strut separation. The intersection of the two lists of physical record blocks will contain all of the logical records which have the specified value for both parameters.

A simple system such as the one described can be implemented with only a modest amount of effort in FORTRAN under a modern operating system. There are, of course, many other data representations and mapping functions which could be used.

REFERENCES

1. J. Martin, "Computer Data-Base Organization" Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1975.

2. C.J. Date, "An Introduction to Database Systems" Addison-Wesley Publ. Co., Reading, Mass., 1975.

3. H. Katzan, "Computer Data Management and Data Base Technology" Van Nostrand, Reinhold, New York, 1976.

4. P. Deud, "On a Storage Mapping Function for Data Structures", Communications of ACM, Vol.9, No.5, 1966, pp. 344-347.

5. A.T. Bertiss, "Data Structure: Theory and Practice", Academic Press, New York, 2nd Edition, 1975.

6. M. Elson, "Data Structures" Science Research Associates, Palo Alto, California, 1974.

7. D.E. Knuth "The Art of Computer Programming, Vol. 3, Sorting and Searching" Addison-Wesley, Reading, Mass., 1973.

8. D.E. Knuth "The Art of Computer Programming, Vol. 1, Fundamental Algorithms" Addison-Wesley, Reading, Mass., 1968.

Level	L	1	2	2	3	3	2	3	3
Name	N	WS	SD	DH	DD	PC	ED	TD	PR
Count	C	40	20	10	6	5	10	6	2
	Q	0	0	20	0	6	130	0	6
	M	210	1	11	1	1	8	1	1

Table 1: Tabular Representation of Record Structure

```
1 Wing Section [40]                    1 WS [40]
  2 Surface Description [20]             2 SD [20]
  2 Design History [10]                  2 SD [20]
     3 Design Data [6]                     3 DD [6]
     3 Plant Code  [5]                     3 PC [5]
  2 Evaluation Data      [10]            2 ED [10]
     3 Test Data    [6]                   3 TD [6]
     3 Performance Rating[2]              3 PR [2]
```

Figure 1: Record Definition for Wing Section Data

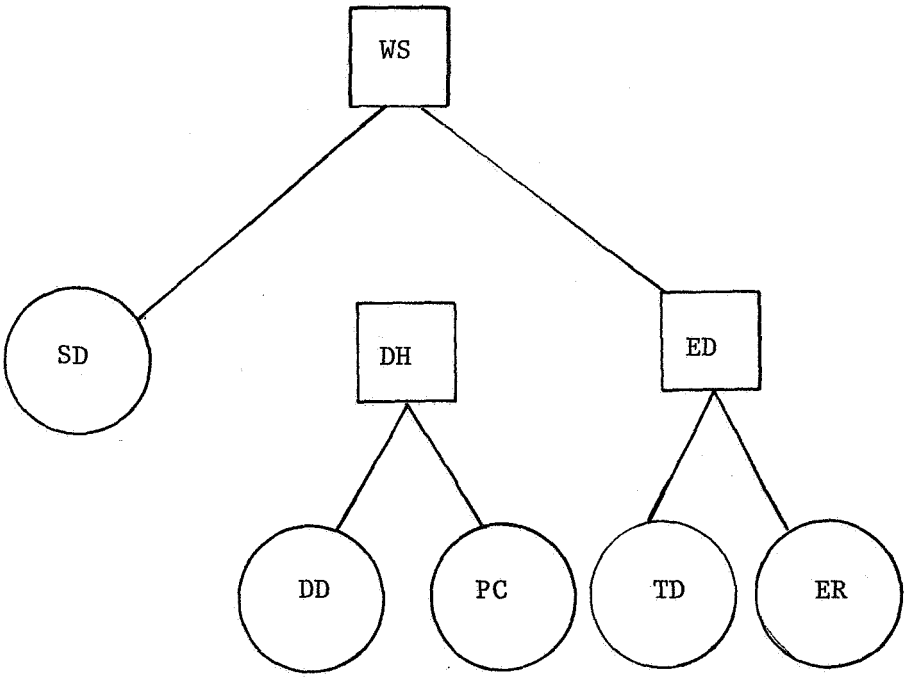

Figure 2: Tree Diagram of Wing Section Data Record

```
1 Wing Section [15]
    2 Aircraft Designation C10
    2 Design Parameters
        3 Thickness R1
        3 Flexibility R1
    2 Stress Values R [25]
```

Figure 3: Logical Record Definition for Wing Section Stress Data

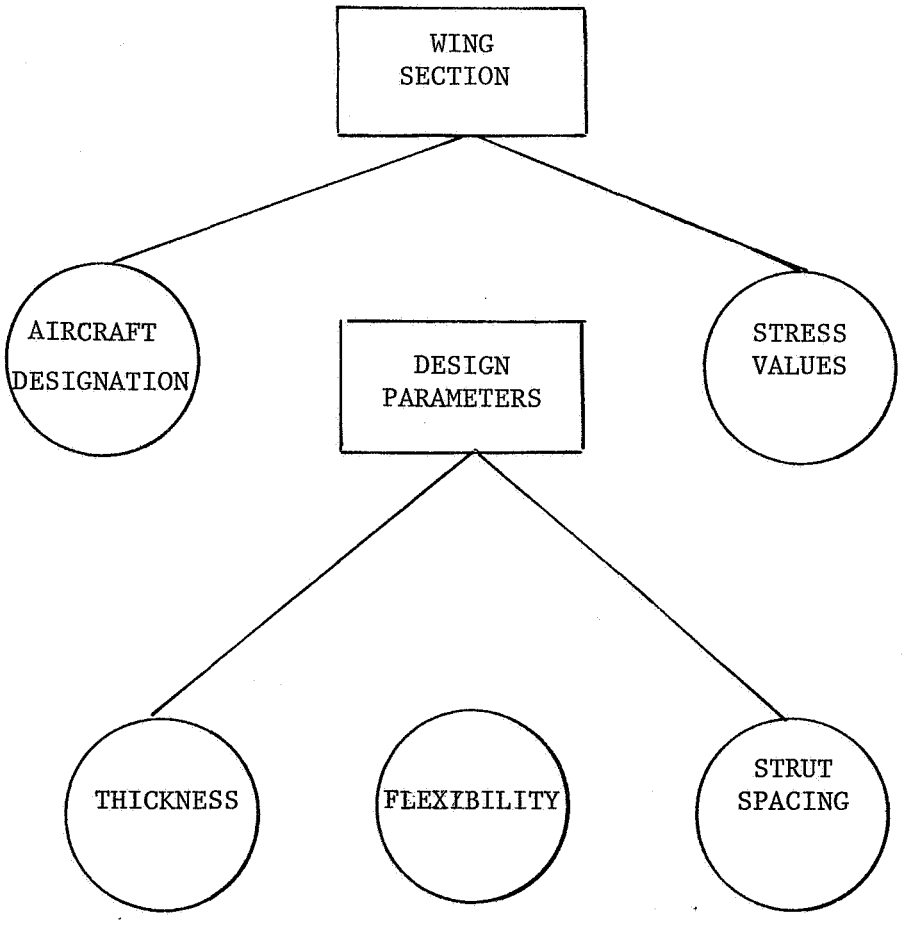

Figure 4: Tree Structure of Wing Section Logical Record

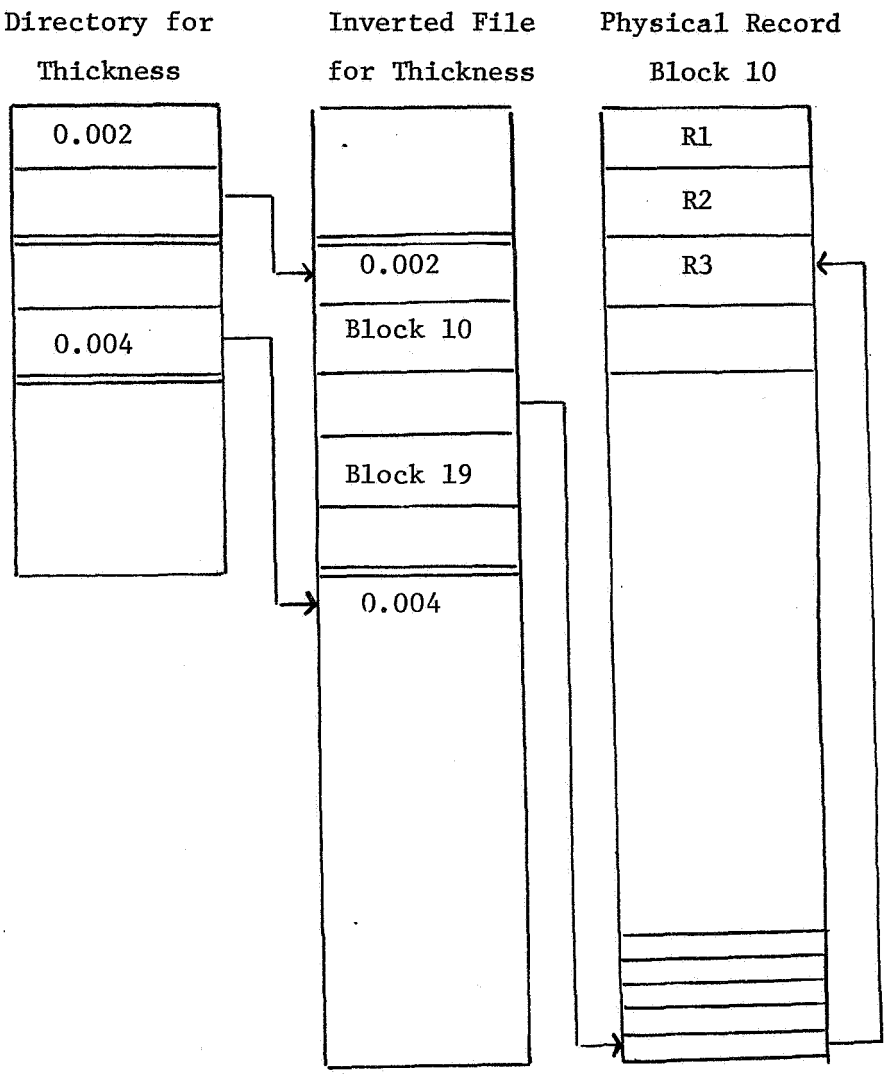

Figure 5: File Structures For Wing Section Data