Presented is the Data Independent Accessing Model (DIAM)—a complete model for the representing, storing, and retrieving of structured information.

DIAM is a hierarchy of models formed by the Entity Set Model and three lower modeling levels—the String Model, the Encoding Model, and the Physical Device Level Model.

Data structures and accessing in data-base systems III Data representations and the data independent accessing model

The previous two sections of this paper have reviewed the technical progress of information systems and the structuring of information. We now discuss computer-oriented representations for structured information and the Data Independent Accessing Model (DIAM) multilevel general description of structured information and representations.

The main problem in the field of data-base systems continues to be in the programming area. Software is difficult to design and implement. Despite some difficulties in understanding them, systems engineers are installing and maintaining data-base systems in a growing number of installations. Applications programmers are similarly making the extra effort to achieve understanding so that they can write and maintain application programs.

Much of this comprehension difficulty can be overcome by having a simple, fundamental way of describing data-base systems. Current terminology is representation oriented and incompatible from system to system, being either too gross (generation data groups, ISAM) or too microscopic (bit functions in a record descriptor). As a result, we tend to pyramid gross functions and then implement these functions in terms of microscopic instructions. This leads us to overlook common, intermediate-sized functions that can be implemented and described once for the system, rather than being coded several times in the internals of several gross functions and described in slightly differing detail each time. For example, decoders for the contents of fields tend to proliferate. Thus there are different decoders for field contents if they are file names, record names, field names, field values, record lengths, identifiers, and so forth, and they are implemented one or more times in each gross function.

Each time a reasonably sized basic function is encoded redundantly in a special form into a gross function, the size of the system in terms of number of instructions increases. We are attempting to overcome as many as possible of the difficulties of understanding, implementing, and debugging systems, difficulties that many believe are growing at an exponential rate with respect to the size of the systems. Instead of adding function on function to obtain generality, we are attempting to provide descriptions of systems that emphasize common functions to provide generality, upward compatibility, and ease of understanding.

In this environment, previous work on the descriptions of database systems has taken two general directions. One approach has been to improve descriptions at a gross-feature level, which has lead to some excellent comparative descriptions of systems. Publications of the CODASYL Systems Committee[1] typify results of this approach. Such descriptions, however, tell us only about the external appearance of the systems. They do not give sufficient detail in the right form for one to see how the systems work.

Other workers have constructed more detailed system-independent descriptions, which are presented in a scattering of significant early papers such as Mealy,[2] D'Imperio,[3] and Information Algebras.[4] More recently, with the recognition of the importance of data-base systems, a growing body of useful publications has appeared exemplified by Hsiao and Harrary,[5] Earley,[6] SHARE/GUIDE,[7] Data Base Task Group (DBTG),[8] McGee,[9] Engles,[10] Codd,[11] Childs,[12] Smith,[13] Taylor,[14] and Severance.[15] These papers, however, provide insight only into specialized aspects of data-base systems. For this reason, it is difficult to evaluate these systems descriptions with respect to meaningfulness of their description of the real world, independence of their programs from changes in physical representation, power of representation, or possible implementation.

Basic concepts of the Data Independent Accessing Model

design approach

The Data Independent Accessing Model (DIAM) also does not cover all aspects of system description, but it does appear to describe and provide defined, detailed interfaces to a broad range of components of data-base systems.

In creating this model, we have looked at diverse, seemingly unrelated concepts that have been used to describe existing systems—indexes, sequential files, direct files, records, fields, bytes, blocks, tracks, cylinders, packs, and so on—in search of common properties. We have then used these common properties to define new general type descriptions for the functions. Variations of the parameter values within the type descriptions allow us to describe detailed characteristics of older terminologies in a simpler, more consistent manner.

By defining these new type descriptions, we have arrived at an overall model that has the following characteristics: (1) multiple self-sufficient levels of abstraction for describing and solving problems concerned with information systems, and (2) one place where each basic function is performed. For example, a name decoder supplants a file name decoder, a record name decoder, and so forth. Also incorporated is an appropriate relative stature of the various model concepts from the most basic concepts to higher structures that are to be described in terms of the basic concepts. For example, in Part II, we determined that hierarchic logical record structures should be described by the user in terms of the more basic system interface—the Entity Set Model.

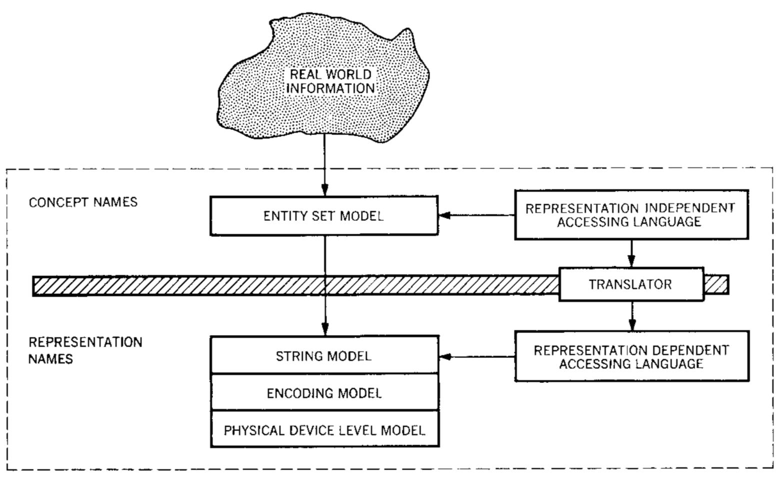
Such a general description has potential for application in describing and comparing the complex, monolithic functions of existing systems to discover their fundamental differences and commonalities.

It also can serve as a basis for implementing new data-base accessing systems that are easier to explain, understand, and use. By use of its primitive building blocks, such a system could provide any desired data representation structure. This could ease the problem of long-term evolution from existing systems. The general description is also a basis for the simplified calculation of relative file organization performance, and for a generalized data representation translator.

The DIAM structure consists of four successive levels of abstraction—an Entity Set Model (previously discussed), a String (or Access Path) Model, an Encoding Model, and a Physical Device Model. Each lower level combines with the higher—more abstract—levels to provide a self-sufficient, more simplified environment for solving some appropriate information system design problems. The model levels are illustrated in Figure 1.

The earlier work of Davies,[16] Engles,[10] and Meltzer,[17] with regard to Entity Sets provides the basic concepts in our informa-

Figure 1 Data independent accessing model


REAL WORLD INFORMATION
CONCEPT NAMES
ENTITY SET MODEL
REPRESENTATION INDEPENDENT ACCESSING LANGUAGE
TRANSLATOR
REPRESENTATION NAMES
STRING MODEL
REPRESENTATION DEPENDENT ACCESSING LANGUAGE
ENCODING MODEL
PHYSICAL DEVICE LEVEL MODEL

tion structuring (Entity Set Model) level. We have only modified these concepts slightly, and used a more precise terminology. At the lower levels that deal with the descriptions of storage representation, the following basic concepts are defined: three kinds of *Strings* for describing access paths, a *Basic Encoding Unit* for describing bit-level encoding of the strings, and a *Physical Subdivision Specification* for describing device properties.

modification method

These basic concepts significantly simplify earlier terminology, but they must be refined to offer a complete detailed description of data structures and data accessing. From this point on, we expect DIAM to be improved just as other models in science are improved. That is, we plan to describe more features of exisitng and proposed systems by varying the values of parameters that we define. If the model does not exactly describe some significant aspect of a system, then the definitions of the parameters will be modified to improve the description. We believe that most significant features of data structrures and data accessing can be described without major descriptive changes to the present model.

As previously indicated, observations are an important aspect of the scientific approach to data-base system models. Observations to define and test a general model for representations of information in data-base systems are relatively easy to obtain. The necessary observations are descriptions of the data structures and accessing languages of existing or proposed systems. To obtain test observations, we have studied descriptions with a wide range of characteristics. These descriptions include the Data Base Task Group Report (DBTG),[8] the Generalized Information System (GIS),[18] the Integrated Data Store (IDS),[19] the Information Management System (IMS),[20] the Index Sequential

Access Method (ISAM),[21] the Sequential Access Method (SAM),[22] and the Time Shared Data Management System (TDMS)[23] Detailed discussion of these systems is beyond the scope of this paper. Certain aspects of these systems are, however, given limited use as examples of the descriptive capabilities of DIAM. The references just given provide adequate introduction to the systems.

DIAM hierarchic structure

The Data Independent Accessing Model (DIAM) structure is based on the hierarchic set of models previously mentioned. This structure is illustrated in Figure 1. DIAM does not attempt to provide a complete model of real-world information. Rather, at the interface between real-world information and our overall descriptive model, the Entity Set Model is specified as a structured model of the real-world information. The Entity Set Model together with an accompanying Representation Independent Accessing Language (RIAL) for describing the accessing of the Entity Set Descriptions are tailored to be a self-sufficient level where the end user can discuss, specify, and solve the problems of information structuring independently of extraneous details of implementation. Using a set-like notation at this level, we wish only to discuss the information to be stored, its interrelations, and the questions to be asked by the end user.

At the next lower level of the hierarchy in Figure 1 we specify a *String Model* for describing and tracing through all interesting representations of the Entity Sets in terms of access paths. (By interesting, we mean all representations that have reasonably general applicability for which there is no obviously more efficient representation.) The String Model and its associated *Representation Dependent Accessing Language* (RDAL) use graph-like notions, and its parameters are specifications for the composition, interconnection, and ordering of subsets. The String Model is used to discuss, specify, and solve problems of efficiency of representation, search, and maintenance to the first order. Typical problems deal with the possible use of indexes, hierarchic structured physical records, and serial vs. direct search.

The *Encoding Level* of the hierarchy is concerned with the bit-pattern encoding of the string paths. Specified at this level is an atomic unit called the *Basic Encoding Unit* (BEU). There is one BEU for each element of the access path structure. It is our goal to describe all interesting encodings of data from generalized list structures to fixed-record-length sequential files by varying the parameters for the BEUs. At the Encoding Level, we can discuss questions of efficiency of storage and search to the second order.

At the final—most detailed hierarchical level—the encoded structures are allocated to physical devices. This is the *Physical Device Level* where the most detailed efficiencies are to be gained from appropriately understanding and using device characteristics. The main considerations deal with space and overflow-handling rules for physical subdivisions of the devices. Here, again, the goal is to describe all possible interesting variations at this level.

Relationship between the Entity Set Model and the String Model

There has been considerable discussion of data independence in the literature, i.e., the separation of logical (information structure) aspects of data-base systems and physical (representational) aspects. The goal of data independence—that programs address stored information by a name structure does not change over time—is a continuing objective. The Information Management System (IMS) has taken a first step in this direction with its separation of logical structure names from names for its physical structure. DIAM is a further generalization of this property.

Earlier discussions in this paper are intended to make it clear that name structures for representations of information in a database system are relatively unstable. That is to say, each time a new Role Name is added to an Entity Description a new representation (i.e., new record type) must be created and the new representation must have a new name so as to distinguish it from older representations for encoding and decoding purposes. (Relational theory has a naming convention similar to that for representations, wherein an n-tuple has a different name from an $(n + 1)$-tuple. For example, each projection has a different name. Thus relational naming structures are similarly unstable.) We can, of course, override this instability by providing a mapping scheme from one representation to another. Such a mapping scheme, however, requires either a pairwise mapping among all related representations in the system, or a mapping chain from one related representation to another. Both of these alternatives seem unnecessarily complex.

For economy and stability, the DIAM information interface provides the end user with one relatively stable name structure (or a progression of a small number of stable name structures) to which all representation name mappings are referred. This name structure is based on names by which the user refers to entities in the real world. When a new role name is added to the description of an entity in the real world, the name of the description need not change because it is not used directly in the encoding or decoding of particular representations. In other words relationships between Description Set Names and their Role Names need not be changed. Old programs that exploit these relationships thus remain valid. Mappings need only be pairwise between the standard name structure of the Entity Set Model and each representation.

Thus, as in the IBM Information Management System (IMS), we have two different sets of names—one set for logical structures and a second set for physical representations for these logical structures. The set of Entity Set Model Names stands for concepts in the real world, and programs written in terms of these names are independent of data structure representation. The other is a set of String Model Names for representations, and programs written in terms of these names are data structure representation dependent. The system catalog provides for mapping from one name set to the other. Thus data-independent programs written in terms of Entity Set Model Names can be mapped into appropriate programs in terms of the String Model Names, which, in turn, can access existing representations.

The String Model

The String Model level is the first step toward efficient representation in computers—considering both space and time—of the Entity Set Model.

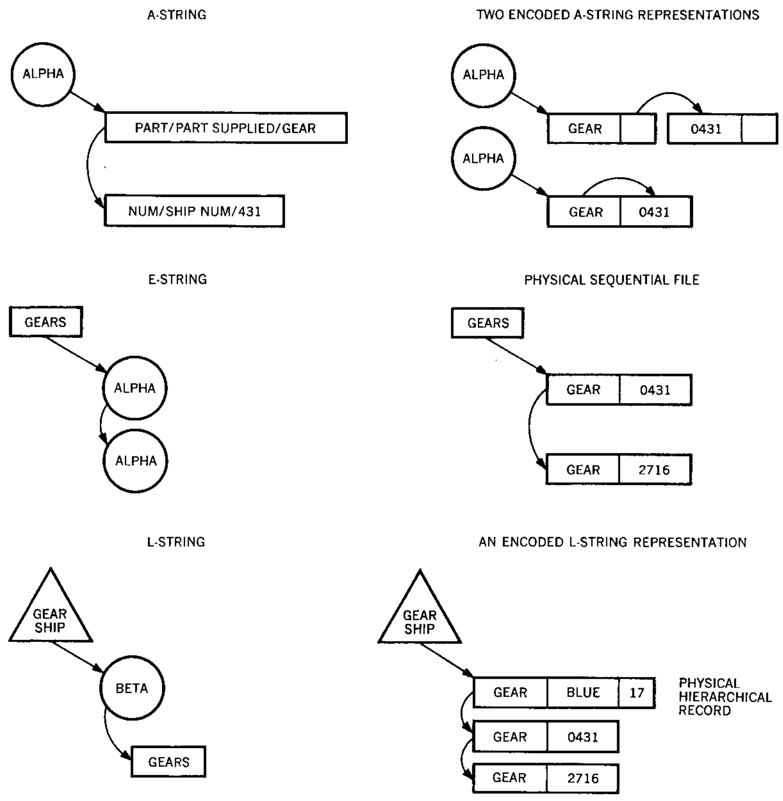
It results from the realization that a simple, self-sufficient, general model of file organization can be based on descriptions of unidirectional paths that provide direct access between representations of information elements drawn from the Entity Set Model. These descriptions are specifed in terms of three kinds of Strings that correspond to the three kinds of associations of the Entity Set Model. The strings describe access paths in terms of connected, ordered subsets of the information elements.

This paper is primarily concerned with the general form of the parameters for Strings rather than the values that the parameters might take for a specific set of data.

The String Model level has been found to be useful in describing the file organizations of existing and proposed systems at a level of abstraction that eliminates the confusion generated by the many possible encodings and physical device implementations of identical basic structures. This capability allows us to see many common elements in what appear to be completely unique functions. Another capability is that of determining the relative efficiency of various possible access-path structures with a very simple yet general performance model.

The String Model level has some parallels in the V-graphs of Earley,[6] though it differs in two principal ways. We make the

Figure 2 String types



type-instance distinction, which in information systems provides a much greater power and conciseness in the description of the data structures and data structure searches. Also Earley's semantics refer to descriptions of data structures at our String Level, whereas our semantics deal with the meaning of concepts in the real world.

specifications for strings

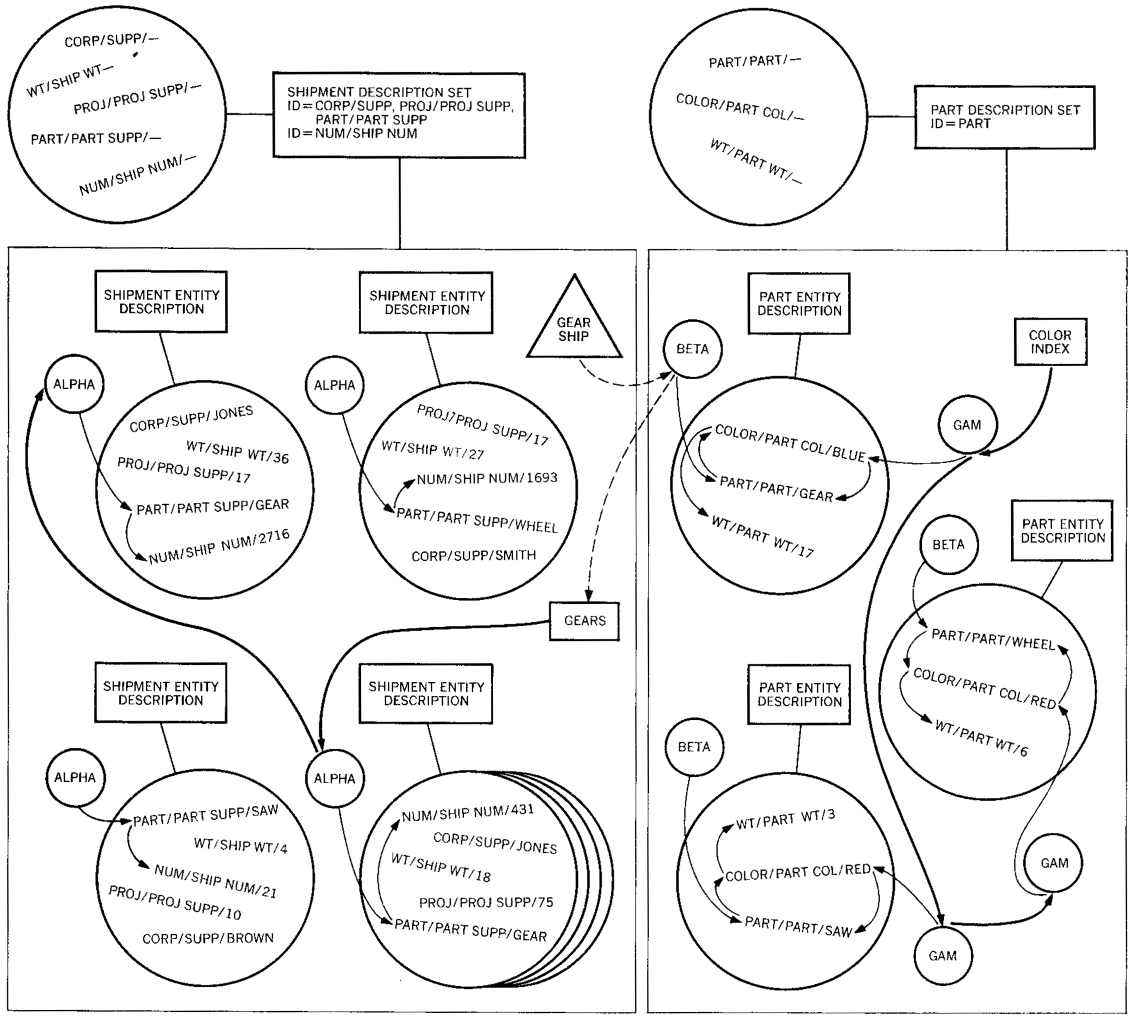
In general, only three things are required to specify a kind of string:

- Subset of the elements on the string
- Order of the elements on the string
- Strings to which a given string connects

Corresponding essentially to the three associations mentioned in connection with the Entity Set Model, are the following three kinds of strings.

Atomic Strings (*A-Strings*) connect specified subsets of Name Set Name/Role Name/Entity Names triplets from a single entity in a specified order. The subsets and order are specified by an ordered Exit List of the Role Names. This is shown in Figures 2

Figure 3 String model example



and 3 where an A-String named ALPHA connects PART/PART SUPPLIED to NUMBER/SHIPMENT NUMBER in that order.

Entity Strings (E-Strings) connect specified subsets of elements that have the same type description in a specified order. The subsets are specified by general Boolean conditional statements on triples stored in instances of the specified-type description. The order is specified in terms of sort orders of specified nested Role Names within the type description. An E-String may be used to connect subsets of A-Strings. In Figures 2 and 3 an E-String named GEARS connects only those A-Strings named ALPHA where PART/PART SUPPLIED = GEAR. It connects them in alphabetical order based on NUMBER/SHIPMENT NUMBER values.

Link Strings (L-Strings) connect elements based on a match between Entity Names for the same Entity that occurs in each of the elements (which may be an A-String, an E-String, or another L-String). For example, if we have a second Description Set called PART that is identified by a Name Set Name/Role Name, PART , then we can specify an L-String that connects an A-String BETA defined on PART (whose identifier is GEAR) with the E-String named GEARS because the E-String contains all the Entities in Shipment where PART/PART SUPPLIED = GEAR.

Figure 3 is a schematic diagram of a specified string structure for the Description Sets in Figure 2 of Part II. Table 1 of the present part provides a catalog for Figure 3.

generality of the string model

To create a generally useful description of access path structures, we could have defined one general kind of string with a large set of possible parameters that would allow us to create a path from any information element in the data base to any other element. In specific situations we would leave undefined those parameters that are not appropriate. Alternatively, we could define one or more kinds of strings, each having only the parameters appropriate for defining all those paths that have any likelihood of ever being used in an actual system.

We have chosen the second option because three kinds of strings with appropriate—though somewhat different—parameters arise naturally out of existing file organizations and the three types of associations mentioned in the Entity Set Model in Part II. These three kinds of strings also provide a useful conceptual interconnection to the Entity Set Level.

Another reason for not selecting one general kind of string is that we want to make it relatively simple to specify paths that normally make sense and to bias the model against the selection of feasible but impractical paths. This biasing parallels the situation in machine organization where meaningful registers and operations tend to be selected preferentially. The resulting machines can simulate, although with considerable overhead, the simple less meaningful operations of a Turing machine. We implement the more meaningful machine organizations, however, because they are more convenient to use and in most cases more efficient for the problems we wish to solve. Such a meaningful biasing of string definitions is a key distinction between DIAM and the models proposed in excellent papers by Smith,[13] Taylor,[14] and Severance.[15]

Of course, whenever one restricts the kinds of interconnection that can be easily specified, he must be alert not to lose any useful generality. The problem of generality has motivated studies of string descriptions of existing file organization concepts. Re-

Table 1 String catalog

Name	*Type*	*Identifiers*	*Name Set Name*	*String Name*	*String Type*	*String Parameters*
PART	Description Set Name	(PART)	–	BETA	ASG	[EXL = (PART, PART COLOR, PART WEIGHT); ON = GEAR SHIP]
			–	GAMMA	ASG	[EXL = (PART COLOR, PART); ON = COLOR INDEX]
			–	COLOR INDEX	ESG	[EXL = (GAMMA); OO = (PART COLOR) ON = ENTRY]
			–	GEAR SHIP	LSG	[EXL = (BETA, GEARS); MC = (BETA/PART = GEARS. ALPHA./PART SUPPLIED) ON = ENTRY]
PART/PART	RN	–	PART	–	–	[ON = BETA; ON = GAMMA]
PART/ PART COLOR	RN	–	COLOR	–	–	[ON = BETA; ON = GAMMA]
PART/ PART WEIGHT	RN	–	WEIGHT	–	–	[ON = BETA]
SHIPMENT	Description Set Name	(SUPPLIER, PROJECT SUPPLIED, PART SUPPLIED)	–	–	–	–
		(SHIPMENT NUMBER)	–	–	–	–
				ALPHA	ASG	[EXL = (PART SUPPLIED, SHIPMENT NUMBER); ON = GEARS]
				GEARS	ESG	[EXL = (ALPHA); SEL = (PART SUPPLIED = GEAR); OO = (SHIPMENT NUMBER) ON = (SEE PART-GEAR SHIP)]
SHIPMENT/ PART SUPPLIED	RN	–	PART	–	–	[ON = ALPHA]
SHIPMENT/ PROJECT SUPPLIED	RN	–	PROJECT	–	–	–
SHIPMENT/ SHIPMENT NUMBER	RN	–	NUMBER	–	–	[ON = ALPHA]
SHIPMENT/ SHIPMENT WEIGHT	RN	–	WEIGHT	–	–	–
SHIPMENT/ SUPPLIER	RN	–	CORPORATION	–	–	–

Abbreviations: EXL = Exit List; OO = Order On; SEL = Selection Parameter; MC = Match On Criteria; RN = Role Name ferring to Figures 2 and 3, recall that many data-base systems provide direct paths between fixed- or variable-length fields in a pointer-connected list, segment, or one-level record. The basis for these paths in the String Model are the A-Strings exemplified by the ALPHAS, BETAS, and GAMMAS. The String Model in this case helps us see that there are certain basic properties of these apparently unique things that are exactly the same. Based on this insight, we can now write one piece of program that is generally useful in processing all these apparently unique things rather than a separate program for each.

For transactions that request information on a particular Entity from more than one Description Set, direct access paths have been provided by data-base system physical structures called physical hierarchic records. The basis for these structures in the String Model is exemplified by the combination of strings: GEAR SHIP, BETA, GEARS, ALPHA. Here, the L-String GEAR SHIP connects the A-String BETA—the root (master) segment representation—to the E-String GEARS. The E-String GEARS connects a list of A-String ALPHAS, which are leaf (detail) segment representations. Hierarchies with more levels and more types of segments at the same level are easily representable. At the String Level, the hierarchic records of IMS, GIS, and IDS look essentially alike.

The strings COLOR INDEX, GAMMA provide the lowest level of what is often known as a secondary index. The essence of such an index is the selection of a pair of Role Names—a secondary key (COLOR/PART COLOR) and an identifier (PART/PART)—and ordering the Role Name pairs on the basis of the secondary key Entity Names. Higher levels of the index can be constructed by selecting subsets of the secondary key-identifier pairs.

Within the structure of the string model, it is possible to describe a wide variety of variations such as sequential files, direct hash-addressed files, primary and secondary indexes, generation data groups, key ranges, and their combinations in terms of ordered interconnected subsets of Description Sets and Role Names. We have not found any path structures whose precise description requires major modifications of existing parameterizations of the three kinds of strings, and we have found that having the three kinds of strings does simplify the study and solution of the search path resolution problem to be discussed later in this paper. Should there be a need for a path that is not covered by the present string parameters, it would be a path that is based on a relationship between information elements from different entities. Such paths could be provided by expanding the definition of the L-String.

The selection of the best access-path structure from a set of

relative efficiency of access paths

candidates is one of the crucial factors in achieving performance in data-base systems. Selection could be made by using existing simulations and calculating exact times for retrieval considering block access times, device queuing, etc. These simulations, however, are tedious to set up and often run much slower than the actual transaction program. In such a situation, the overhead to select a path becomes larger than the cost of using a less efficient path. Desired is a faster, simpler method of path selection, which we believe the string model provides.

In particular, the String Model Catalog provides a good structure for recording the statistics on the number of nodes for each type of string. Given a query and usually a small set of possible paths stored in the system, it is relatively simple to determine the number of nodes that must be touched along a particular access path to obtain the answer to the query. It is possible, therefore, to decide on the best path by selecting the one that requires the minimum number of nodes. If higher relative accuracy is required, these node counts can be crudely weighted as a function of the kind of string, the nature of the encoding (contiguity vs. pointers), and the device that the string is stored on. Since possible paths normally differ greatly in performance, these additional considerations should provide correct relative selection in most cases.

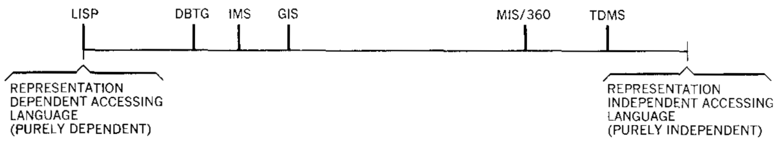
A similar process can be useful in selecting among possible file organizations to support a transaction load. Here, the relative sizes of two file organizations are factors that affect storage costs and to some extent the number of block accesses. All things considered, an appropriate model that is accurate to a factor of two is probably adequate. In essence, the String Model provides a basis for choosing among access path organizations, thereby allowing us to extract this problem out of the complex total problem of hardware and software selection and to solve it independently.

accessing language interactions

At this point, it is appropriate to consider further the interactions between the Entity Set Model and the String Model. The most obvious connection is that the String Catalog is simply the Entity Set Catalog with some of its entries augmented by additional string parameters. There are now two different sets of names—one set for concepts (Entity Set Model Names) and the other set for representations (String Names). The relationships between these two sets of names are described in the String Catalog.

As indicated in Figure 1, each model has its own accessing language. The purest form of Representation Independent Accessing Language (RIAL) allows requests for subsets of the information stored in the system to be made completely in terms of set

Figure 4 Spectrum of several data accessing languages


LISP
DBTG
IMS
GIS
MIS/360
TDMS
REPRESENTATION DEPENDENT ACCESSING LANGUAGE (PURELY DEPENDENT)
REPRESENTATION INDEPENDENT ACCESSING LANGUAGE (PURELY INDEPENDENT)

operations on Entity Set Model names without specifying search procedures and paths to be followed for accessing the desired subsets. For example, the language used for Representation Independent Accessing by DIAM contains statements such as the following:

Form set S1 *containing* (SHIPMENT NUMBER, PART SUPPLIED) *from* SHIPMENT *such that* S1/PART SUPPLIED *equals* GEAR.

This statement contains no procedures that mandate or describe how the search should be conducted.

The purest form of Representation Dependent Accessing Language (RDAL) would provide for a complete specification of the paths in the representation to be followed to obtain the required information. For example, a detailed RDAL might involve the following functions:

```
    GET        GEAR SHIP
    GET        BETA
    GET        GEARS
A   GET NEXT   ALPHA
    WRITE      S1 (SHIPMENT NUMBER, PART SUPPLIED)
    IF         LAST ALPHA, STOP
    ELSE       GO TO A
```

Assumed here as in most systems the decoding of an A-String ALPHA is so straightforward that it can be specified in a nonprocedural fashion. Most existing accessing languages are hybrids of the two pure forms. In Figure 4, we place various languages on a spectrum from representation dependent to representation independent accessing.

LISP[24] is the only language that gives microscopic path access to each element of information (although, for our purposes, it does not make the crucial type-instance distinction for describing access paths). With regard to systems that make the type-instance distinction, the Data Base Task Group Language (DBTG) accesses records as if they were sets of attributes. All other search operations, however, involve path-following instructions. The Information Management System (IMS)[20] and the Generalized Information System[18] have slightly more power to search hierarchic structures with set-oriented instructions. Since all the above systems involve some path following, the RDAL is a reasonable base for providing compatible language support for them. The System/360 Management Information System (MIS/360)[25] and the Time Shared Data Management System (TDMS)[23] are completely set oriented but they both lack the capability for accessing multiple sets (the multifile query).

search path resolution

The problem of a general translation between a pure RIAL and a pure RDAL is precisely equivalent to the problem of obtaining the most general form of data independence along with high accessing efficiency. Systems generally achieve high levels of data independence by restricting the kinds of file organizations they support, thereby reducing their data-accessing performance. Alternatively, they achieve high efficiency by sacrificing data independence. DIAM is a structure for achieving both efficiency (generality of file organization) and data independence.

In DIAM, the translation problem is called search path generation, and it has the following three phases.

Search-path enumeration involves finding all paths to the desired Role Names. The catalog illustrated in Table 1 provides this capability directly, as may be seen by going to a particular Role Name and following back along the strings that the Role Name is ON out to the entry point that would be stored in the catalog. As shown in Figure 3 and Table 1, with the RIAL query, there is one search path starting at SHIPMENT/SHIPMENT NUMBER that is formed by SHIPMENT/SHIPMENT NUMBER preceded by SHIPMENT/PART SUPPLIED preceded by ALPHA preceded by GEARS preceded by BETA preceded by GEAR SHIP (ENTRY). A second path follows the same set of strings to GEAR SHIP starting at SHIPMENT/PART SUPPLIED.

Search-path resolution involves the observation that the answer to the SHIPMENT/SHIPMENT NUMBER query can be completely obtained from the GEAR SHIP paths. Existing systems with restricted or single methods of representations have fixed algorithms for resolving set-oriented queries into search procedures. Little has yet appeared in the literature on resolving general set-oriented queries into graph-oriented search procedures for general representations.

Search-path scoring relates to efficiency calculations previously discussed. Essentially, search-path scoring involves determining which path in a set of candidates (say either GEAR SHIP or a sequential file defined over the complete SHIPMENT Entity Description Set) is the most efficient for answering a particular query. In the query presented, GEAR SHIP is most efficient.

We have developed some preliminary algorithms for the translation problem that are to be reported elsewhere in the literature.

As we can see, the RIAL is directed toward Entity Set Model names where there is only one name (neglecting synonyms) for each component of the conceptual structure no matter how many representations there are at the string structure level for that component. The RDAL provides a means for searching the existing representations in terms of string names.

security and integrity

The number of separate partial or complete representations for a particular entity set can be determined by counting the A-Strings defined on it. Although, as is shown later in this paper, an A-String can belong to several collections, there is a one-to-one relationship between A-String instances and their final encoding into bits in the data stream. This fact determines the number of copies (representations) of an Entity Name that must be changed when the user directs an Entity Name to be changed at the Entity Set Level. The procedure for locating and determining the number of copies is to look for the appropriate Role Name within the catalog and trace out all the paths to the A-Strings that may contain it. The subset selection criteria then determine whether the specific A-String actually exists on higher level E- and L-Strings. In the case of security, the catalog lists all physical copies of a fact under one Role Name. This means that the user can apply his security requirements to the fact rather than to all physical copies. The catalog, therefore, provides an excellent format for storing information required for data-base integrity and security with respect to updating, insertion, and query.

The Encoding Level Model

The encoding hierarchic level of DIAM describes the bit-level encoding of strings in a very general way. Considering the desirable uniform characteristics of bit or byte streams, it appears to be worthwhile to extract these uniform characteristics for another self-sufficient level and leave the complicated consideration of parameters for describing and handling physical devices to a separate level. This has been a useful separation because it has led us to the specification of a single form of *Basic Encoding Unit* (BEU). This one form is a sufficient primitive element for building most useful encodings of collections of information in information systems. The requirement of only one form to encode file organizations allows the specification of a small table-driven BEU encoder/decoder for handling all levels of file organization (files, records, etc.) in a uniform manner. Since the basis for the encoding level is an addressable bit or byte stream, this level is well-suited for implementation as the software interface to a virtual address machine. It also provides an excellent basis for a data translator.

basic concepts of encoding

In reviewing existing structures, one finds space set aside for actual field values and for the following three types of control information:

- Names for files, records, fields, etc. placed so that a program may determine the nature of the component to be decoded.
- Length indicators for files, records, fields, etc.
- Physical pointers to the next component to be decoded.

Since each of the control information components looks like a field value, it seems possible to create a model that simply considers each control field as a pseudo-attribute and then defines its characteristics (length, etc.) in the same way as is done for actual field values. An encoding-decoding program could then use the control information pseudo-attributes to supply the decoding information that it requires. Excellent models of this kind are presented in the papers of Smith,[13] Taylor,[14] and Severance.[15] Such models, while quite general, give the user little systematic guidance on what pseudo-attributes should be defined and where they should be placed in the catalog. These models also require a decoding program that must be prepared to decode almost a random stream of intermixed control and field-value information. Preferable is a more systematic structure that gives general power with a few well-defined parameters. To accomplish this, we look again for regularities and similarities.

One generalization is that a length indicator is very much like a pointer that points to the end of a collection rather than to the beginning. Also, two fields related by contiguity are simply related by a special type of pointer whose origin is immediately after the present field and whose displacement is always equal to zero. A final and extremely useful observation derived from the String Model is that all file organizations are composed of named interconnected collections: A string has a name; it is a member of one or more higher level collections; it provides an entry to a collection of information defined by its parameters; and it requires a means for determining the end of its collection. These features appear at all levels of existing structures. Files, records, and fields are simply names for particular kinds of collections of smaller units that are connected into one or more larger collections.

These observations lead directly to the specification of the Basic Encoding Unit (BEU), one for each String instance (or Role Name instance on an A-String), and a corresponding BEU-type specification, one for each string type (or type occurrence of a Role Name on an A-String) in the catalog. Each BEU has the same general format and set of parameters no matter what kind of String or Role Name it represents. The recognition that attribute values, records, and fields are all special cases of collections leads to a simplification and systematization of the parameters required to describe these values. It also provides a basis for a simple table-driven BEU decoding program that decodes all levels of a given file organization in a uniform manner. That is, file, record, and field names are all decoded by the identical small set of instructions.

When the notion that contiguity of BEUs (fields, records) is obtained by a special case of pointer and that pointer value is the same for all instances of a BEU type, then the possibility of factoring the value information of the pointer into the catalog becomes apparent. The realization that the factoring tactic can be applied to all components of the BEU leads to a general capability for describing data structure encodings.

There are the following three basic concepts at the encoding level:

Named Address Spaces (AS), which provide reference addresses for the placement of encoded units and for pointers to encoded units.

Basic Encoding Units (BEU), which provide a single basic format for encoding all Strings and Name Set Name/Role Name/Entity Name triplets. For each named type (for example, the A-String Type ALPHA, the E-String type GEARS or the Role Name type PART SUPPLIED) there is a specified BEU-type format specification associated with its name (ALPHA, GEARS, etc.) in the catalog. For each instance of the named type (for example, PART/PART SUPPLIED/GEAR in the SHIPMENT Entity Description where NUMBER/SHIPMENT NUMBER/431 is the Identifier), there is a BEU in the data stream.

Factoring is a method analogous to algebraic factoring, by which information common to all BEUs of a BEU type can be placed in the type description in the catalog rather than in the BEUs in the data stream. In some cases, the method can be applied to all components of a BEU-type with the result that there will be no bits required in the data stream for the BEU representation.

address spaces

In this paper, we use the simplest form of Named Address Space, the Linear Address Space(s) (LAS), which are uniquely named, one dimensional, potentially infinite bit (byte) streams. Each bit (byte) has a unique integer address. Data and pointers are encoded in terms of these address spaces and the address spaces themselves are allocated to physical devices by procedures described in the later discussion of Physical Device Space Models. Each LAS can be considered to be a named Virtual Address Space.

basic encoding unit

There is a Basic Encoding Unit (BEU) for each instance of a string, and for each instance of a Role Name on an A-String. Four types of components appear in all BEUs in the following format:

LABEL	APTR 1	APTR 2	. . .	APTR N	VPTR	TERM

A LABEL component contains an encoded Name for the collection of information that the String or Role Name defines. The encoded Name may simply be the String (or Role) Name or some encoded synonym.

There is one ASSOCIATION POINTER (APTR) component for each distinct collection of which this BEU is a member. The APTR contains the information necessary to determine the Address Space location of the next BEU of the distinct collection. (For example, in Figure 3, BETA and GEARS are members of the collection GEAR SHIP. The access path from BETA to GEARS connects these two members of the collection and is represented by an APTR in the BETA BEU format.)

The VALUE POINTER (VPTR) contains the information necessary to determine the starting Address Space location of the collection defined by the parameters for a String corresponding to this BEU. In the case of a Role Name, the collection consists of the Entity Name. In the case of an A-, E-, or L-String, the collection consists of a set of BEUs. (For example, in the GEAR SHIP-BEU format, the VPTR provides the access path to its collection (BETA, GEARS) by pointing to the location of BETA, the first member.) The collection is then connected together by the use of an APTR component in the member BEUs that associates each BEU with its successor.

The TERM component contains the information necessary to determine the termination of the collection defined by the string corresponding to this BEU.

factoring

Factoring consists of placing in the BEU-type description the value of any component that has the same value in all instances of BEUs of a given type, thereby removing that BEU component from the data stream. This tactic systematically achieves any encoding combination from generalized list structures (unfactored BEUs) to serial fixed-length record files (completely factored BEUs). In the DIAM catalog, there are parameters for each BEU component that specify how its value is to be determined and how its value is to be used in the encoding and decoding processes. If the value of a BEU component is to be found in the data stream, then the parameters specify how the length of the byte string containing the value is determined. This is sufficient because, during decoding, the beginning of the string is known. If the value is factored into the catalog, then the catalog entry contains the component's value.

Figure 5A shows the stream of unfactored BEUs for the collection COLOR INDEX. The BEUs are so placed as to simplify the pointer representation, but it should be clear that any element connected by pointers may appear at any position on the data stream. Any element may even appear on a different Named Address Space if desired. The BEU components are presented in the order LABEL, APTR, VPTR, TERM.

Figure 5B is specified so that all Entity Names for the COLOR Role Name have the common length of four bytes and PART has a common length of six bytes. The terminator components can, therefore, be removed from the Address Space and placed in the Type description in the Catalog.

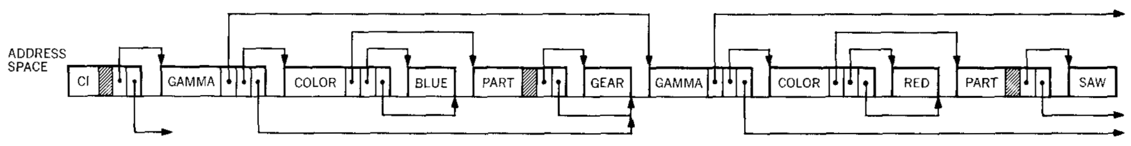
Figures 5C through 5F illustrate the factoring of other components into the catalog until there is a fixed-length record sequential file representation of the collection, COLOR INDEX, in the address space. The actual choice of elements to be factored depends on the possible commonalities between instances of a type and the insert load on the collection. It may, for instance, be useful to leave certain pointers in the representation to make certain types of inserts relatively easy even though the elements of the collection could be made contiguous.

Figures 6A-D present IDS, IMS, and GIS representations for the hierarchical collection GEAR SHIP. Figure 6A shows the GEARS collection terminated by a back pointer. In Figure 6B the catalog is essentially the same as 6A but with the redundant length field removed. In Figure 6C it may appear that pointers in the IMS Hierarchical Sequential organization have been omitted. However, the pointers for connecting pieces of records together are not really a property of the segments; they are a property of the physical record structure of IMS and would not appear at the encoding level. In Figure 6D the collection GEARS is terminated by count of its ALPHA instances. TERM = (COUNT = 2) in this case.

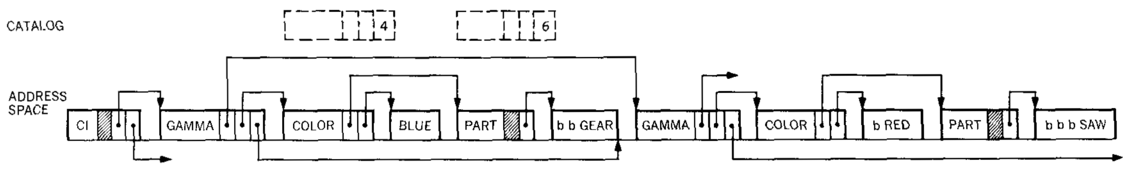
In this section we have presented another self-sufficient level, the Encoding Level and indicated the power that can be obtained from the simple BEU format. A more detailed presentation of the Encoding Level is given in References 26–28. We discuss next the Physical Device Level.

Figure 5 Factoring of the Basic Encoding Unit (BEU)

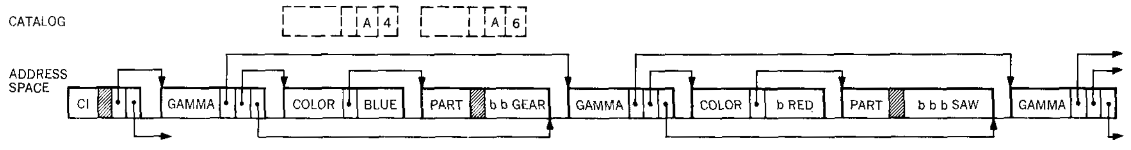
A ALL BEU INFORMATION ON THE DATA STREAM



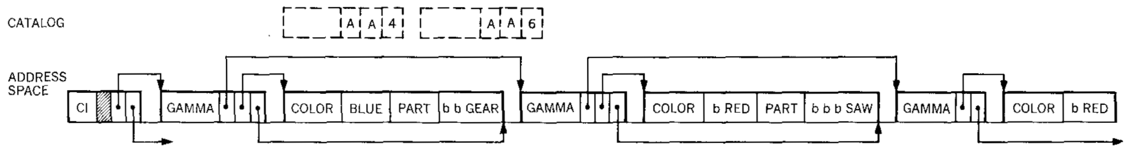
B FIXED-LENGTH ENTITY NAMES; ENTITY NAME TERMINATION IN CATALOG



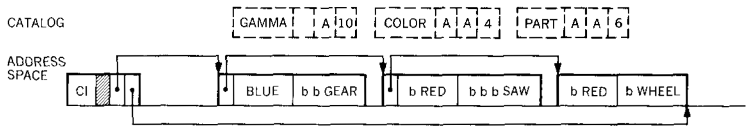
C ENTITY NAME ALWAYS AFTER ITS ROLE NAME BEU; ROLE NAME VPTR IN CATALOG



D PART ROLE NAME ALWAYS AFTER ITS ROLE NAME BEU; ROLE NAME APTR IN CATALOG



E ORDER OF STRING NAMES KNOWN FROM EXIT LIST; COLOR ROLE NAMES AFTER GAMMA BEU; FIXED RECORD LENGTH



F FIXED RECORD LENGTH SEQUENTIAL FILE; NO CONTROL INFORMATION IN DATA STREAM

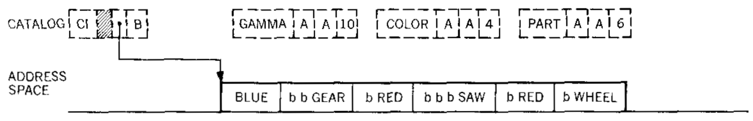
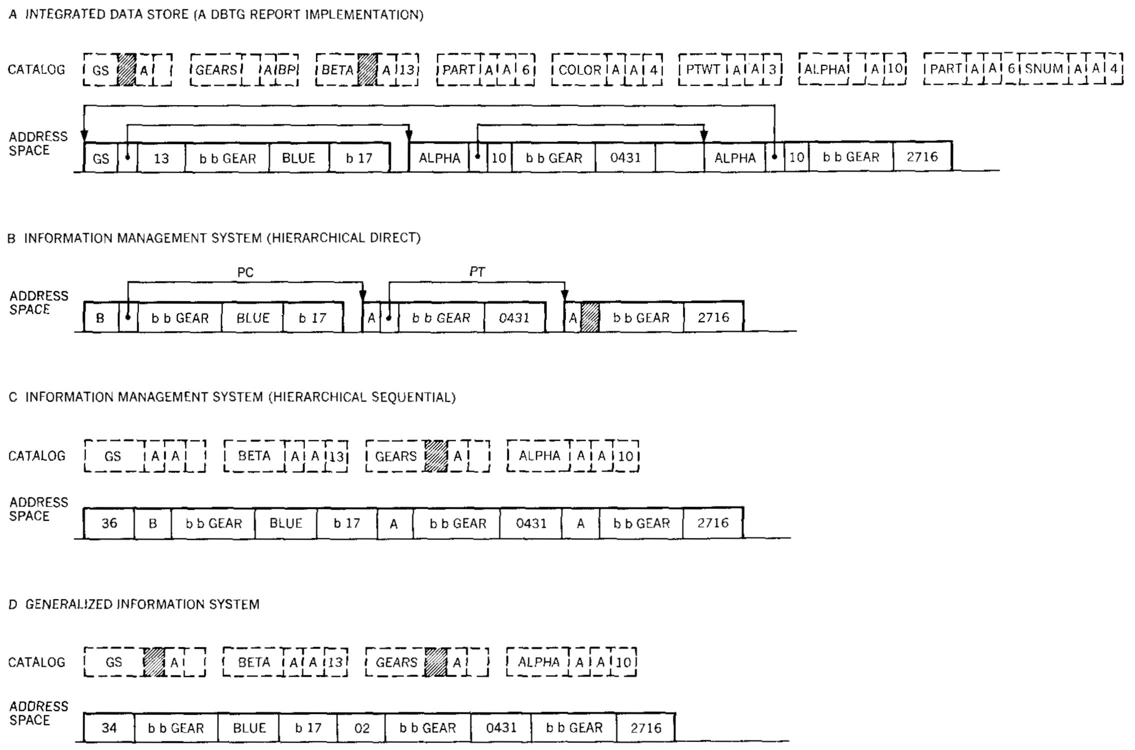


Figure 6 BEU encoding descriptions of four physical hierarchic records


A INTEGRATED DATA STORE (A DBTG REPORT IMPLEMENTATION)
CATALOG
GS A
GEARS A BP
BETA A 13
PART A A 6
COLOR A A 4
PTWT A A 3
ALPHA A 10
PART A A 6 SNUM A A 4
ADDRESS SPACE
GS 13 b b GEAR BLUE b 17 ALPHA 10 b b GEAR 0431 ALPHA 10 b b GEAR 2716
B INFORMATION MANAGEMENT SYSTEM (HIERARCHICAL DIRECT)
PC
PT
ADDRESS SPACE
B b b GEAR BLUE b 17 A b b GEAR 0431 A b b GEAR 2716
C INFORMATION MANAGEMENT SYSTEM (HIERARCHICAL SEQUENTIAL)
CATALOG
GS A A
BETA A A 13
GEARS A
ALPHA A A 10
ADDRESS SPACE
36 B b b GEAR BLUE b 17 A b b GEAR 0431 A b b GEAR 2716
D GENERALIZED INFORMATION SYSTEM
CATALOG
GS A
BETA A A 13
GEARS A
ALPHA A A 10
ADDRESS SPACE
34 b b GEAR BLUE b 17 02 b b GEAR 0431 b b GEAR 2716

Physical device level model

The Entity Set, String Structure, and Encoding specifications provide a basis for describing many features of data-base organizations, particularly data bases in high-speed storage and in virtual address system. These specifications do not, however, allow us either to take into consideration the periodic structures and inhomogeneous access times characteristic of large-capacity DASD or to tailor storage organizations to avoid the often serious penalties incurred in random accessing. Thus it is desirable to have a mechanism that automatically inserts new records near related old records so as to minimize long physical access times.

In investigating existing systems like ISAM, IMS, and IDS, we find that a rather complex set of parameters is required to describe the physical device level. In this paper, we present only the outline of the required parameterization. A substantial iterative refinement is required to describe the wide range of functions found at the device level in existing systems. Broadly speaking, the physical device level has been found to require the following four major provisions.

1. *Physical Subdivision Type Specification* is provided to define properties of named physical subdivision types such as named types of blocks, pages, or tracks.
2. *Physical Device Formatting* provides for formatting specific instances of physical devices in terms of the Physical Subdivision Type Specifications in 1.
3. *Address Space Allocation* provides a means for correlating named Address Spaces mentioned in the Encoding Level Model with the formatted device addresses. A named Address Space may be multidimensional and may span portions of one or more devices formatted by the specification in 2.
4. *Placement Specifications* define a record and describe where the system is to attempt to place it initially. Placement Specifications interface the Physical Device Level Model with the Encoding Level Catalog.

physical subdivision type specification

In existing data-base systems, the specifications of physical device formatting and the properties of the various physical subdivisions are deeply embedded in the system or access-method program. To provide a model with more general flexibility these aspects must be externalized in the form of separable, parameterizable components.

The first step in the parameterization process is to note that here, as with the earlier levels, it is useful to make the type-instance distinction. Hints of type description are found in the Indexed Sequential Access Method (ISAM) which has four types of physical subdivisions—prime, overflow, track index, and higher level index. Each physical subdivision type can be specified in the following fashion:

- *Composition* in terms of its component physical subdivisions
- *Contiguous Data Group* (physical record) insertion process
- *Available space* handling process
- *Overflow* handling of Contiguous Data Groups

These specifications are illustrated in the ISAM example that follows.

physical device formatting

Physical Device Formatting is simply the assigning of Physical Type Specifications to actual device instances. The correlation is made by indicating an origin in actual device space for a list of particular type specifications. From the example in Figure 8, the specification ORIG = PACK (17), CYLINDER (7), FORMAT = 9 (CYLC) defines cylinders 7–15 of pack 17 as ISAM-type cylinders with one index track, 17 prime tracks and two overflow tracks.

address space allocation

Given a formatting of devices, named address spaces must be allocated to them. A major characteristic of allocation is that it may be discontinuous with respect to the physical device address space. In the case of OS/360 extents for a data set, for example, the first extent might cover device cylinders 1–7 and the second, device cylinders 12–14. There is no reason why address allocation cannot be generalized to be discontinuous to the byte level, but it seems that the definitions of start and end points of extents in terms of blocks, tracks, etc. provide a reasonable level of detail. An example allocation might be ASPACEA = (PACK = 7 (CYLINDER = 1, 7 (TRACK = 5, 9 (BLOCK = 3, 6)))) which allocates address space ASPACEA to pack 7 where the allocation runs through blocks 3 to 6 of tracks 5 to 9 of cylinders 1 to 7. If we defined a linear address space, then the addresses exist in terms of bytes (bits), where byte 0 is the first byte of pack 7, cylinder 1, track 5, block 3. Higher dimension addresses may also be defined if desired, such as bytes within tracks where the first address on cylinder 3 becomes Track = 10, Byte = 0. In the ISAM example, the Address Space is similar to the Device Address Space and is, therefore, multidimensional.

Up to this point, the Physical Device Model has been independent of the Encoding model. The Physical Device Model catalog (which may be merged with the other catalogs of the system) includes the following items:

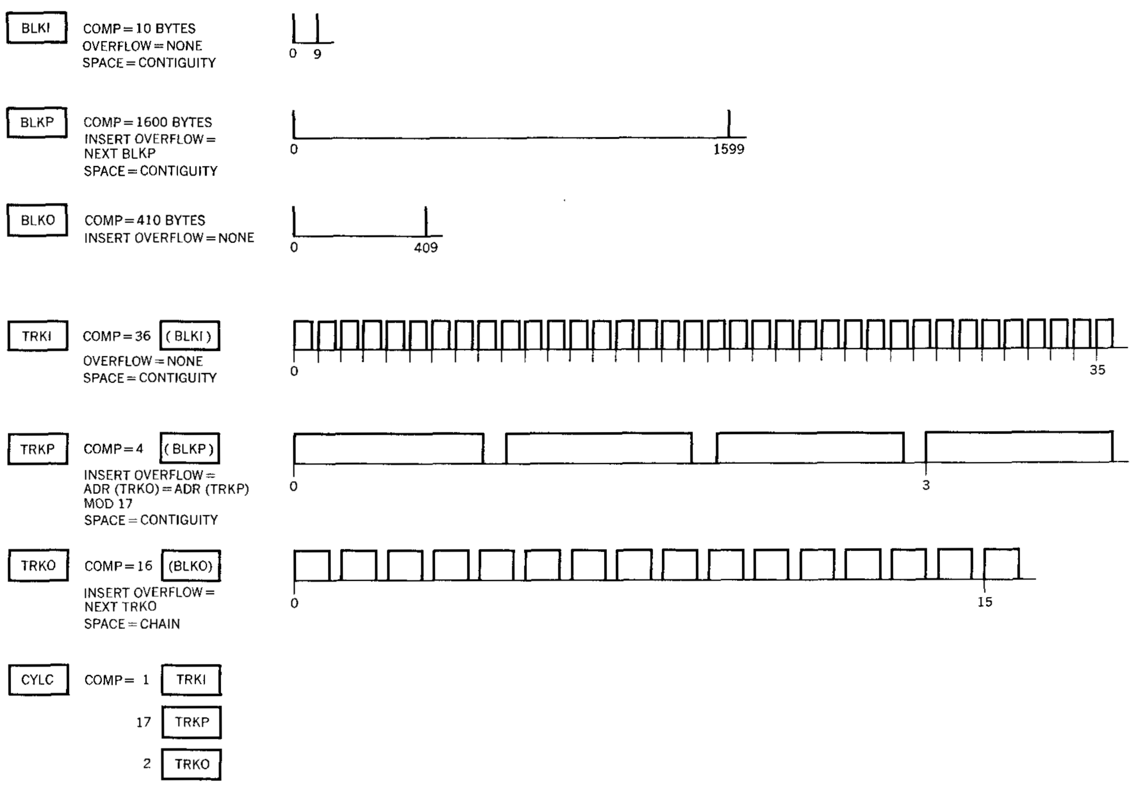
- Physical Type Specification names
- Physical device names with their format specifications
- Address space names with their allocation specifications

This catalog specifies an independent mechanism for storing and retrieving records with relatively comprehensive space and overflow handling.

placement specification

Required by the Physical Device Level Model is a more generalized definition of a record and its placement. In existing systems, the unit of placement is usually called a record with an ad hoc definition that varies from system to system—hierarchic records, master records, detail records, and so on. In general descriptive terms, a record is a reasonable-sized set of Entity Model Names and control information values related in a fixed fashion by physical contiguity. To avoid confusion, DIAM uses the term *Contiguous Data Group* (CDG) instead of record. A CDG is a set of BEUs that are related by contiguity and placed as a unit. Under this definition, the records of ISAM, GIS, and DBTG are CDGs as are the segments of IMS. A placement rule, directly associated with the highest level BEU catalog description of each CDG, states that the CDG is placed in the first appropriate available space in the address space after the prescribed address. (The term "appropriate" is related to E-String ordering, CDG insertion, and space handling rules.)

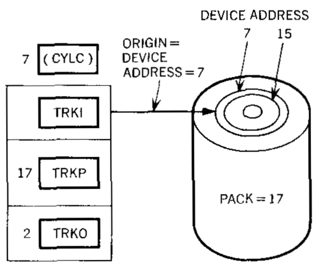
Figure 7 Physical type specification for ISAM cylinder overflow



example physical device level model

The Indexed Sequential Access Method (ISAM) makes use of a wide variety of Physical Device Level Model (PDLM) parameter variations. The creation and maintenance of the indexes, in particular, requires a relatively complex set of interactions. For simplicity, therefore, the following example, taken from Figures 7 and 8, is restricted to a PDLM description of the data cylinder areas.

Figure 8 Physical device formatting



Composition

Blocks are composed of bytes		
Index Block	BLKI =	10 BYTES
Prime Block	BLKP =	1600 BYTES
Overflow Block	BLKO =	410 BYTES
Tracks are composed of blocks		
Index Track	TRKI =	36 BLKI
Prime Track	TRKP =	4 BLKP
Overflow Track		15 BLKO
Cylinders are composed of tracks		
For cylinder overflow		
Cylinder	CYLIC =	1 TRKI + 17 TRKP + 2 TRKO

For separate overflow

TRKIS =	40 BLKI
CYLS =	1 TRKIS + 19 TRKP
CYLSO =	20 TRKO

Here the control fields such as the key field for a block are not specified, but such specifications are a natural extension of the Composition structure.

Insertion

CDG *identification and length.* Fixed-length ISAM requires neither identification nor length for its CDGs because it is assumed that all CDGs originate at a single source that provides its own identification. The fixed position and size of one Entity Description Identifier is provided to the access method.

Record ordering . The system program is instructed to maintain order based on the Entity Description Identifier.

Record placement. At load time, the CDGs are presented in order and each CDG is loaded in order by the Entity Description Identifier into the prime area (BLKPs) only. At run time, CDGs are loaded into a block immediately before the CDG with the next higher Entity Description Identifier. Contiguity is used to maintain ordering in the prime area. Chains are used to maintain ordering in the overflow area.

Available space

At run time, available space is maintained in the prime areas by contiguity and in the overflow areas by chain. CDGs may be deleted by the use of a control character, but CDGs are never part of the available space.

Overflow handling

At load time, overflow moves from BLKP to BLKP until the last BLKP on a track is filled. At this point, the track (TRKI and TRKP) overflow rule directs the overflow to the next TRKP. Overflow from the last TRKP on a CYL is sent to the BLKPS on a TRKI on the next cylinder. At run time, overflow moves from BLKP to BLKP, and from TRKI and TRKP, onto TRKO on the same CYLC if there is a cylinder overflow, or to TRKO on CYLSO if there is a separate overflow.

From this example, it is possible to say that ISAM is the result if a particular selection of model parameters is made. The model, however, is capable of generating other interesting file organizations for the experimenter who wishes to consider other kinds of transaction patterns. For example, one may wish to maintain order within a block by chain. This allows direct insertion of new CDGs into overflow without CDG movement in the block.

Implementation

Given a description of data structures and data accessing, it is natural to attempt to determine what the model characteristics would be if, in the long term, it becomes part of an evolutionary information system. Our present prototype effort is directed to proving out functions instead of performance, but the following observations seem realistic.

In the time-consuming accessing and decoding of data records, DIAM is expected to require less—and certainly no more—physical device accesses to obtain the requested data. To aid in upward compatible migration, DIAM is expected to provide any data structure presently provided by existing systems, and in addition provide special structures more directly tailored to the application. The resultant system should have fewer instructions and modules for data accessing than more generalized systems.

In preparing to access data, a tailored system may be slower than existing systems, depending on the DIAM implementation options chosen, the generality of the data structures, and the language used. In particular, precompiled RDAL addressed to stable String Structures should be no slower than existing procedural languages. The RIAL is probably slower if catalog accesses are scattered and if the search-path selection problem must be solved for every transaction. Even in this case, however, the selection of the best search path may result in major economies. Here, again, there exists the possibility of precompiling RIAL transactions of a particular class if they occur quite often. The transactions could then be recompiled whenever a structure to which they refer changes.

The catalog for DIAM is probably larger than existing catalogs, but it probably does not differ greatly in size and may be smaller than catalogs required for the combined functions—essential to next generation systems—that DIAM supplies.

Concluding remarks

Features that are properties of the String, Encoding, and Physical Device Model Levels have been discussed. An implementation of a very generalized data-accessing mechanism that could provide, among other things, many of the desired functions specified by the Data Base Task Group Report might be based on

RDAL and these model levels without dealing with the questions of data independence and search path selection. DIAM does, however, provide an excellent basis for a generalized data independent system because of the stable Entity Set Model name structure and the wide variety of file organizations provided by the String Name Structure. The present catalog provides a good basis for the storage of the information needed to maintain data integrity and security. It also provides a good basis for the collection of statistics for performance enhancement.

In essence, a DIAM implementation is expected to be fast in decoding the data stream. It is not so clear what its performance would be with regard to preparation for accessing the data stream, but there are a number of tactics available to improve the DIAM performance over that of a straight interpreter.

Having considered this more general view of structured information, data structures, and data accessing in data-base systems, let us consider several applications. The Entity Set Model coupled with an RIAL description of transactions provides a consistent implementation-independent basis for describing a user application load. In this sense, RIAL has parallels in the Problem Statement Language of the ISDOS[29] project and the Time Automated Grid (TAG)[30] project in IBM. Given an implementation-independent description of a user's problem, regardless of the implementation system used, the next step is to determine from a gross efficiency standpoint the subset collections that are to be constructed. The String model provides a self-sufficient level for solving such a problem with a minimum of detail. Encoding and Physical Devices parameters are relatively constrained in existing data-base systems, but DIAM can be of assistance in defining the alternatives present in existing systems is a clear fashion.

With DIAM a better picture of common and differing primitive functions of alternative data-base systems may be obtained. We have seen earlier certain aspects of IDS, IMS, and GIS differ but slightly at the Encoding Level, and that these systems do not differ at all in their description of hierarchic record structures at the String Level.

With regard to designing new data-base system functions, a new function may be added in terms of high-level primitives instead of assembly language coding. The model also removes from the coding nearly all embedded knowledge of data structure, thereby making possible a completely table-directed accessing system.

These capabilities for providing and accessing general data representations using table-directed accessing plus the resemblance of the RDAL to existing access languages such as that provided for IMS should significantly ease the task of compatible migration.

For example, mapping from existing file organizations into DIAM's more general representation structure description should be relatively straightforward. The translation of a logically oriented "Get Next" language like that of IMS into RDAL in the case where the DIAM representation images the IMS physical structure again should be relatively straightforward. In the case where there is no direct imaging, translation seems slightly more difficult, but the loops and tests characteristic of a "Get Next" language should be transferable without requiring additional information from the user.

DIAM provides a clearcut separation of concept names in the real world at the Entity Set Level from representation names at the String Level, by use of a catalog structure for relating the two kinds of names. This results in a framework in which it is possible to achieve full accessing program (RIAL) independence of data structure representation modification.

Also provided is an initial basis for defining a system in which a fact need only be stored in one place and so characterized that the user can only perform meaningful operations on these facts. This is an area for considerable future investigation.

ACKNOWLEDGMENTS
The authors wish to acknowledge the contributions of their colleagues, Phyllis B. Baxendale, Sakti P. Ghosh, Armand D. Inselberg, Warwick D. Musson, and Mary C. Smyly.

CITED REFERENCES

1. CODASYL Systems Committee Technical report, *Feature Analysis of Generalized Data Base Management Systems*, ACM, New York, New York, (May 1971).
2. G. H. Mealy, "Another look at data," *AFIPS Conference Proceedings, Fall Joint Computer Conference* **31,** 525–534 (1967).
3. M. D'Imperio, "Data structures and their representation in storage," *Annual Review in Automatic Programming* **5,** 1–75, Pergamon Press, Inc., Elmsford, New York (1969).
4. Language Structure Group of the CODASYL Development Committee, "An Information Algebra, Phase 1 Report," *Communications of the ACM* **5,** No. 4, 190–205 (April 1962).
5. D. Hsiao and F. Harrary, "A formal system for information retrieval from files," *Communications of the ACM* **13,** No. 2, 67–73 (February 1970).
6. J. Earley, "Towards an understanding of data structures," *Communications of the ACM* **14,** No. 10, 617–628 (October 1971).
7. Joint GUIDE-SHARE, *Data Base Management System Requirements*, W. D. Stevens, Skelly Oil Co., Tulsa, Oklahoma 74102 (November 1970).
8. CODASYL Data Base Task Group, *Report to the CODASYL Programming Language Committee*, Report CR 11, 5(70)19,080; ACM, New York, New York (October 1969).
9. W. C. McGee, "Generalized file processing," *Annual Review in Automatic Processing* **5,** 77–149, Pergamon Press, Inc., Elmsford, New York (1969).
10. R. W. Engles, *A Tutorial on Data Base Organization*, Report TR 00.2004, International Business Machines Corporation, System Development Division, Poughkeepsie, New York (1970).
11. E. F. Codd, "A relational model for large shared data banks," *Communica-*

tions of the ACM **13,** No. 6, 377–387 (June 1970).

12. D. C. Childs, "Feasibility of a set theoretic data structure," *Proceedings of the IFIP Congress 1968* **1,** 420–430, North Holland Publishing Company, Amsterdam, Netherlands (1968).
13. D. P. Smith, *An Approach to Data Description and Conversion*, Moore School Report No. 72–20, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pennsylvania (December 1971).
14. R. W. Taylor, *Generalized Data Base Management System Data Structures and their Mapping to Physical Storage*, Ph.D. dissertation, University of Michigan, Ann Arbor, Michigan (1971).
15. D. G. Severance, *Some Generalized Modeling Structures for Use in the Design of File Organizations*, Ph.D. dissertation, University of Michigan, Ann Arbor, Michigan (1972).
16. C. T. Davies, *A Logical Concept for the Control and Management of Data*, Report AR-0803-00, International Business Machines Corporation, System Development Division, Poughkeepsie, New York (1967).
17. H. S. Meltzer, *Data Base Concepts and Architecture for Data Base Systems*, IBM Report to SHARE Information Systems Research Project (August 20, 1969).
18. *Generalized Information System GIS/360, Application Description Manual (Version 2)*, Form GH20-0892-0, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604 (1970).
19. *GE-600 Line Integrated Data Store*, Publication CPB-1565A, General Electric Information Systems Department, Phoenix, Arizona (September 1969).
20. *Information Management System IMS/360 Application Description Manual (Version 2)*, Form GH20-0765-1, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604 (1971).
21. *Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods*, Chapter 7, "Indexed sequential organization," Form GC20-1649-4, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604 (1969).
22. *Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods*, Chapter 5, "Sequential organization," Form GC20-1649-4, International Business Machines Corporation, Data Processing Division, White Plains, New York 10604 (1969).
23. R. E. Bleier, "Treating hierarchical data structures in the SDC Time Shared Data Management System TDMS," *Proceedings of the ACM 22nd National Conference*, 41–49, MDI Publications, Wayne, Pennsylvania (1967).
24. J. McCarthy et al., *The LISP 1.5 Programmer's Manual*, The M.I.T. Press, Cambridge, Massachusetts (1962).
25. G. F. Duffy and F. P. Gartner, "An on-line information system for management," *AFIPS Conference Proceedings, Spring Joint Computer Conference 1969* **34,** 339–350, AFIPS Press, Montvale, New Jersey 07645 (1969).
26. M. M. Astrahan, E. B. Altman, P. L. Fehder, and M. E. Senko, "Concepts of a data independent architectural model," *Proceedings of the ACM SIGFIDET Conference*, Denver, Colorado (1972).
27. E. B. Altman, M. M. Astrahan, P. L. Fehder, and M. E. Senko, "Specifications in a data independent architectural model," *Proceedings of the ACM SIGFIDET Conference*, Denver, Colorado (1972).
28. M. E. Senko, E. B. Altman, M. M. Astrahan, P. L. Fehder, and C. P. Wang, "A data independent architectural model 1: Four levels of description from logical structures to physical search structures," Report RJ 982, International Business Machines Corporation, Research Division, Monterey and Cottle Roads, San Jose, California (February 1972).
29. E. J. P. Tremblay, "A Problem Statement Language Definition and Its Syntactic Analysis" 30B, 1969–70, Abstract 4098-B, *Dissertation Abstracts International*, University Microfilms, Ann Arbor, Michigan 48106 (1969).
30. J. F. Kelly, *Computerized Management Information Systems*, Chapter 8, 533, The Macmillan Company, New York, New York (1970).