VIRTUAL INFORMATION IN DATA-BASE SYSTEMS

Jeffrey J. Folinus, Stuart E. Madnick, Howard B. Schutzman

Information Systems Group
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Mass. 02139

ABSTRACT

This paper examines the concept and implications of virtual information in data base systems. Virtual information is any fact which does not physically exist in the data base, but is nonetheless accessible through combinations of algorithms and other data. Physically recorded information is only one of a number of ways to obtain information from a data-base system. Viewing an information system as a collection of functions shows that pure data and pure algorithm from the endpoints of a spectrum of ways function values can be realized, with the middle range being various types of virtual information. Several classes of virtual information are identified, and their usefulness is examined to show the appropriateness of the concept in a data-base system. Finally, the model is evaluated in light of the implications of virtual information for inference and automatic restructuring within a data base.

CONTENTS

ACKNOWLEDGMENTS

The research reported in this paper is part of the continuing work of the Information Systems Group at M.I.T.'s Sloan School Of Management. The goal of the Group's research effort is to develop information systems that deal with issues such as optimizing performance, automatic restructuring, inference, and information security.

The specific advances in this paper derive in part from earlier research in information systems for buidling design carried out in M.I.T.'s Department of Civil Engineering. That department and the facilities of the Civil Engineering Systems Laboratory provided the environment within which this paper was written.

INTRODUCTION

One view of data-base systems is as a method of describing and mapping data structures into physical storage. An alternative view is that, given appropriate stored data, the problem is how we use it to meet requests for information. Requests for "answers", whether made to processing programs or a stored data base, are essentially requests for a value of a function, given various argument values. A model of an information system as a collection of such functions helps unify many of our notions about data and algorithms, and provides a convenient construct for resolving several problems in data-base systems. Such a model will be presented in this paper.

Much recent work in data-base systems has concentrated in two areas: deriving a suitably powerful logical structure for abstractly representing information, and formulating ways of declaring the policies used to map this structure into a stored form. Logical structures based on the mathematical concept of relations have been proposed by Codd (4) and Mealy (21), whereas the Data Base Task Group (6), Engles (9), and Senko, et al. (24) have used groupings of objects with similar properties, sometimes referred to as entity sets. Several others have proposed methods for mapping the resulting logical data structures to a physical storage medium: trees and other hierarchical organizations (as in GIS (12) and IMS (13)), chained list structures based on rings (Bachman (1), DBTG (6), and IDMS (25)), encoded strings (Senko, et al. (24)), and schemes using symbolic rather than physical pointers (Davies (8) and Raver (23)).

Data relationships in recent information structuring models, however, essentially group and categorize data in some static fashion in the data base. But relationships can also be defined in a procedural fashion. An example of such an item is age. For example, to maintain a completely accurate value of someone's age, it would have to be updated continuously. Therefore, rather than assigning a particular stored value to the data item age, it might be preferable to define it procedurally as current date minus date of birth. This leads directly to the idea that a model of information should allow not only static, grouping relationships, but procedural relationships as well.

This paper presents a model of information based on functional requests to an information system. This model includes not only the classical concepts of pure data and pure algorithm, but also important classes of virtual information based on procedural relationships. The usefulness of various types of virtual information are presented to show its appropriateness as a concept in a data-base system. Finally, we examine the implications of virtual information for automatic restructuring and inference within a data base.

The idea of virtual information by itself is not new, having previously been discussed by the DBTG (6) and Engles (9). Many systems, especially inquiry-oriented reporting systems, already virtualize information, although they often don't consider it as such. All of

this previous work, however, has treated virtual information as a special case, and dealt with it in a largely ad hoc fashion. The real value of this concept occurs only when considered within a larger structure for information which also includes data and algorithms. In such a context, work can occur on the relative suitability of each for solving problems in data-base systems.

A MODEL OF INFORMATION

Resolving issues in data-base systems has become easier in recent years as the computer community has developed a clearer set of notions about information. Of particular importance has been the distinction between the logical structuring of facts, and the physical structuring of stored data.

The essential characteristic of a data-base system is the sharing of data by multiple applications. This type of environment demands a clearly defined distinction between system internals and the external view of the application programs, or what has been called data independence. A significant degree of data independence means that access methods and data organization are transparent to application programs, and that the physical aspects of storage are considered apart from the logical aspects of information. This implies a logical data structure against which application programmers can define their files and specify their requests for information.

Many data base designs assume each fact is physically recorded in the data base. Actually, physically recorded data is only one point within a spectrum of ways to obtain information, such as by algorithm or even by derivation from physically recorded data. Although these other alternatives are occasionally desirable, they have been largely ignored. As a consequence, most models of information are not adequate for obtaining dynamic or procedural relationships in the data base.

Viewing an information system as a collection of functions avoids these inadequacies. All requests for information are in a sense requests for a value of a function, given various argument values. This functional model retains all of the power to describe information of the models of Codd (4), Engles (9), and Mealy (21). Basically, a function can establish relations (in the set theoretic sense) between argument values and function values; it can thus serve the same purpose as a data map. Although conventionally a function returns a single value, our consideration includes functions which can return a set or series of values (which itself could be considered a value) by calling more primitive functions repeatedly. We also allow for a function value of "null".

As Iverson (14) notes: "In classical applied mathematics, most functions of interest can be approximated by some algorithm which becomes, for practical purposes, the definition of the function. In other areas, however, many functions of practical, if not general, interest (such as the correspondence between employee name and salary) can be specified only by an exhaustive listing of each argument

value and its corresponding function value."

Most functions of interest in a data-base system are of this latter type. The basic algorithm applied to evaluate these functions is a search of the list of arguments, i.e., a comparison of the given argument with the list of arguments to determine the correspondent to be selected. It thus becomes efficient to physically record the lists of function values on storage media. Recorded facts which are independent of other information in the data base may be considered as pure data. These enumerated facts can be obtained merely by use of retrieval procedures.

Function specification can also be by means of algorithm. Functions requiring no reference to the stored data base are pure algorithms (such as SINE). In most conventional systems, whenever function values can be determined without an exhaustive listing of argument values and functions values, the algorithm is usually associated with the processing program and not with data management. Certain functions should be associated with data-base systems in order to guarantee data independence. The functions of concern pertain to attribute values, such as summaries, which can be realized either through a search of a stored representation of a data map or by other means. Function references to information may require more than simple retrieval of a stored grouping of bits. Intermediate algorithms in the data management system will put the stored pure data into the necessary form for processing programs. A model of information, then, must include more than pure data and pure algorithm -- it requires combinations of algorithms working against values that are either stored in the data base or derived by other algorithms. Information obtained in this way, rather than by retrieval procedures or pure algorithms, can be termed virtual information. In the most general sense, virtual information is any fact which is accessible through combinations of pure algorithm and pure data, but which is not physically stored in the data base.

Viewing an information system as a collection of functions shows that pure data and pure algorithm are merely two different ways to furnish function values in response to argument values. Pure data and pure algorithm form the endpoints of a spectrum of ways these values can be realized, with the middle range being various types of virtual information (Figure 1). More generally, function requests are always satisfied by combining data and algorithms. Pure data is merely the special case involving no program, just as pure algorithm is the special case involving no data.

When one asks for SINE(37.2), it is irrelevant whether the appropriate function value is obtained by table look-up, a Taylor series expansion, or possibly by interpolation between stored table entries. Which method is used to realize the value is properly a concern of data-base management. As Engles (9) notes, the important point in regard to data independence is that the intermediate algorithms necessary to map stored data into the logical structure (and vice versa) must be transparent to the processing programs. (If something is virtual, you can see it, but it isn't there; if something is transparent, it is there, but you can't see it). The opportunity

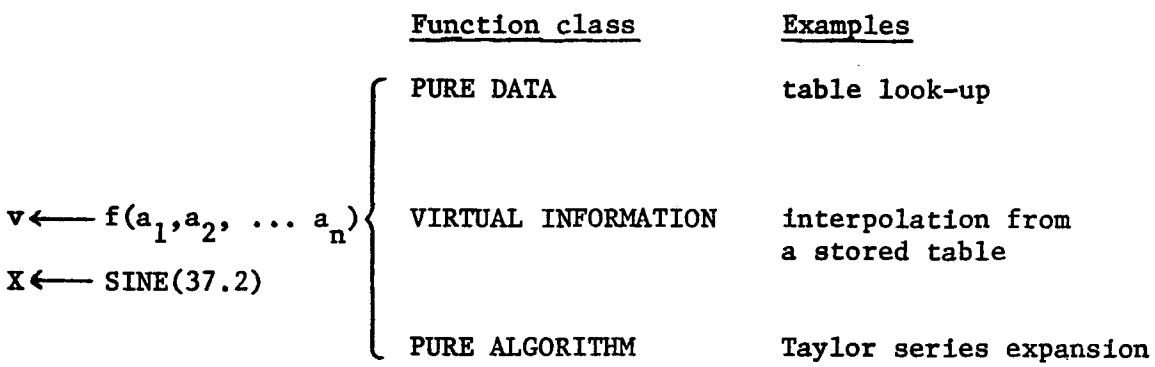


FIGURE 1

The Spectrum of Information Functions

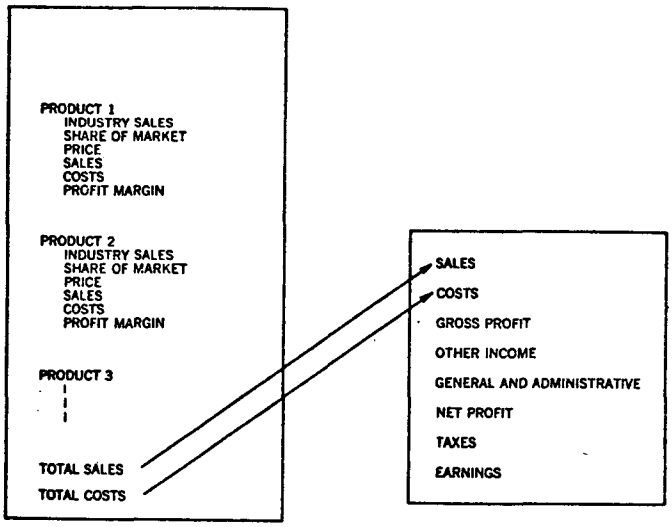


FIGURE 2

Transferred data

(from reference 15)

to realize the information in the logical structure by other tools than merely stored data should make it easier to achieve data independence.

In summary, a model of an information system as a collection of functions not only helps unify our view of algorithms and data, but also is consistent with other trends and needs in the computer field. Increasingly centralized control of information in data bases necessitates functions in data-base management to preserve data independence. Finally, the procedural definition of information resolves the constraints resulting from physical limitations in much the same way as procedural definition, or virtualization, of other system resources (e.g., virtual memory, virtual processors).

CLASSES AND USES OF VIRTUAL INFORMATION

Several categories of virtual information are of sufficient generality to merit their inclusion in data-base systems. In particular, various classes of virtual information can resolve the fundamental issues of representation and materialization in data-base systems identified by Engles (9).

Representation, or data type, is the relationship between data items and values. The same data item can be represented as different values; different data items can be interpreted as the same value. Representation is thus primarily a matter of form. The same fact can be represented in many forms. The form which is appropriate to application programs is not necessarily the best representation for storage in the data base. Numbers to be displayed to users are not in the same form required for computation. The form required for computation by a particular CPU or programming language is not necessarily the best form for storage.

The key issue, then, becomes how to provide a fact, once it has been retrieved by our system of functions, in the form desired. In the DBTG proposal (6), this is accomplished by the mapping between sub-schema and schema data definitions. More generally, data-base systems need to contain a library of conversion procedures that enable any obtained fact to be translated into any appropriate standard data type. Where conversion is necessary, the resulting value is virtual information. Such converted forms enable a data-base system to provide many views of the same collection of facts.

A special case of conversion occurs when facts are not represented as standard data types, but are represented as encoded forms, such as may be required for security or storage compaction considerations. Compaction techniques can save considerable amounts of storage but require a transformation between the encoded and decoded forms. Many attributes with a limited number of possible values can be more efficiently stored as code to save space. Engles (9) offers the example of an application program which stores or retrieves a field which contains the name of a state. The data item as manipulated by the application program is a character string such as 'CALIFORNIA'.

In the data base, however, the value is represented by a numeric code and a function maps these state codes into state names and vice versa. Such mapping functions should be part of data-base management and their use should be transparent to application programs.

Materialization is primarily a matter of content. Specifically, it is the matter of obtaining facts from the information system, regardless of form. In the real world, facts are mostly derived rather than pure data. As an example, consider the chart of accounts for a firm. The only pure data needed are original journal entries; all other facts are derived by manipulating this data. Derived facts must exist in data-base systems as well as in the real world.

The key issues related to materialization are diverse. On a practical level, storing facts procedurally as virtual information will typically involve tradeoffs of storage and access time, and will obviate updating. More significantly, there is the matter of obtaining facts which are implicitly available given a collection of pure data and pure algorithms. As a corollary, there is the question of which facts should be represented in this collection to maximize the amount of implicit information. Three major classes of virtual information deal with these issues of extracting the factual content: factored facts, inferred facts, and computed facts.

Factored facts. As Senko, et al. (24) note, recognizing and taking advantage of the distinction between types (such as sets of entities) and instances (such as individual entity occurrences) offers great power in building data-base systems. To improve efficiency, information that is common to all instances of a particular type can be collected and placed in a catalog. The complete information about a particular instance is thus a combination of the information common to all instances of this type and information that is specific to it. Factoring, the process of looking for collections of instance information common to all instances of a collection and placing it into a type description, is a powerful method of organizing, simplifying, and condensing the information about a collection of instances. Recombining factored values requires procedures to produce the virtual information about each individual entity from the type description.

This task becomes more formidable if multi-level factoring is employed. For example, in considering information about U. S. cities, we might factor out information that pertains to all cities in the same state (e.g., name of governor), as well as information that pertains to all states (e.g., name of president). (This multilevel factoring is a major motivation for so-called "tree-structured" data-base systems). The user should be able to access information independent of the factoring employed.

Inferred facts. Data maps between different entity sets lead to the notion of related data maps. Consider the maps EMPLOYEE ---> POSITION, POSITION ---> SALARY, and EMPLOYEE ---> DEPARTMENT. Using these basic maps, we can infer the mapping of EMPLOYEE ---> SALARY. In addition, the mapping DEPARTMENT ---> EMPLOYEE can be inferred by

the inverse of the EMPLOYEE ---> DEPARTMENT map. Furthermore, the DEPARTMENT ---> NUMBER-OF-EMPLOYEES data map can be derived from the inferred inverse map DEPARTMENT ---> EMPLOYEE. This may be preferable to storing a representation of the DEPARTMENT ---> NUMBER-OF-EMPLOYEES data map, which has to be updated whenever a change is made to the EMPLOYEE ---> DEPARTMENT data map. The user should be able to access the data to ascertain the NUMBER-OF-EMPLOYEES in a DEPARTMENT, whether the desired fact is actually stored or inferred.

A simple form of an inferred fact is a single data item referenced in several ways. Consider two entities with the same attribute value. For example, each MANAGER has a NAME, but also, each EMPLOYEE (which includes MANAGERS) has a NAME. If we define one attribute as having the same value as another (similar to the "ACTUAL/VIRTUAL SOURCE" clause in (6)), only one data item needs to be changed during updating. What has been called "transferred data" (15) is similar. Transferred data is summarized data based on a supporting subschedule and forwarded to a given portion of a line item (Figure 2). Thus, a data item in one summary table can be a virtual "copy" of more elementary data elsewhere in the data base (e.g., NUMBER-OF-EMPLOYEES in a DEPARTMENT may be derived from a summary of the EMPLOYEE ---> DEPARTMENT data).

More complex forms of inferred facts stem from the observation that, in a data base of any complexity, there will be several alternative combinations of related data maps that could be used to access a given fact. Selecting the best access path structure from a set of possible candidates becomes a crucial factor in achieving performance. Senko, et al. (24) propose that possible access paths ("strings") be explicitly specified. The access path catalog would record facts (such as length of this path and device characteristics) useful in access path selection.

In the most general sense, all inferred facts are instances where the appropriate fact exists in our collection of functions; the only problem is obtaining that fact. Explicitly specifying access paths (6, 24) is one solution, but such a solution seems to be more for the convenience of the system developer than of the end user. If implicit information is available, why should the user be allowed to get at it only if he had the foresight and knowledge of the data-base structure to specify it as an access path? Users may even attempt to specify unlikely access paths "just in case," leading to a data base cluttered with needless relational information. This leads to some of the same problems encountered in Codd's normalization strategies (4), which require a user to know which fields will be used as identifiers when the data base is defined. Frequently, this will be difficult and the result will be the designation of an identifier with many fields, some of which are completely unnecessary. A more appropriate solution is to let the information system itself develop the proper access strategy for a fact. An information system could use the explicit intermediate relationships necessary to define the data base to discover whether implicit relationships exist.

Computed facts. Whereas factored and inferred facts are developed merely by accessing facts available in the data-base system, computed facts are derived by processing algorithms. A major distinction to be noted here is that some computed facts are in terms of an individual entity occurrence, whereas others are in terms of an entire entity set, or more complex forms. An example of the first would be that, given an entity such as a ROOM whose attributes are LENGTH and WIDTH, its AREA could be defined in a form such as PRODUCT (LENGTH,WIDTH). The DBTG (6) "ACTUAL/VIRTUAL RESULT" clause is of this type, and allows user-defined procedures to be used. A data-base system should incorporate common functions, such as SUM and PRODUCT.

More significant facts can be developed by performing operations over an entire entity set. Classification requests such as "List the ROOMS whose COLOR is BEIGE" can employ simple comparison operators on a single fact, such as =, >, <, and combinations thereof. Boolean conditions such as &, |, and ¬ can be used to construct even more complicated types of requests. Finally, functions such as COUNT, MAX, MIN, and AVERAGE can be combined with any of these above types of requests. Lists of suggested operators to use in data-base systems are presented in (5), (7), and (9).

All three types of facts -- factored, inferred, and computed -- may be used either singly or in combination to extract information from our collection of functions. Any fact may also be subject to any representation conversions that may be necessary.

IMPLICATIONS FOR DATA-BASE SYSTEMS

Virtual information completes the spectrum between pure data and pure algorithm, and allows an information system to be modelled as a collection of functions. Such a model provides a clearer and more consistent framework for studying the concept of information and provides new insights into the design and implementation of data-base systems. This section explores some of the implications of our information model. Two areas are examined: technical issues of system efficiency and conceptual issues of system effectiveness.

Technical issues. Data-base systems increase the variety and flexibility of ways to store a given fact. Data independence implies that the methods used within a system for representation and materialization are irrelevant to the user concerned only with logical issues. In a system with many users, such internal decisions should be made on a global basis, using a set of criteria which optimizes system performance as a whole. It has been suggested that, because users and uses of data change over time, the system could monitor itself and perform data restructuring dynamically (17). Rules need to be developed to aid the system in choosing the appropriate materialization method.

The basic goal in formulating such rules is minimization of cost. Three types of costs need to be considered: space, update, and

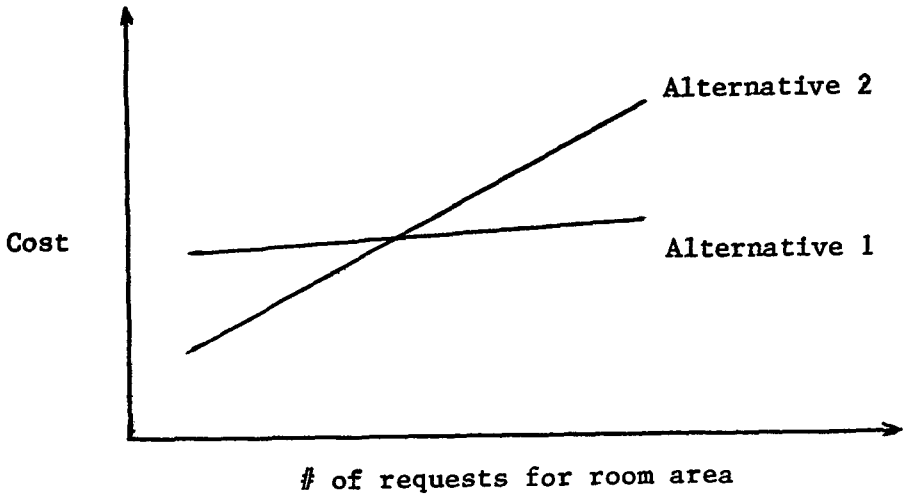
access. The space issue is concerned with how many bits a particular materialization method uses. Update deals with how static a particular data value is; certain data items, such as population of the United States, change value quite frequently while others, such as a particular person's social security number, have a constant value. The access problem involves the costs incurred in satisfying a request for a value. When considering the form a particular value should take, the tradeoffs between the various cost types must be weighed before a decision can be made.

For example, consider storing areas of rectangular rooms given a known length and width. The problem is whether to: 1)store each room's area with the information for that room, or 2)store the algorithm for computing the area in the logical dexcriptor for the general entity "room". An access time/space tradeoff is involved. Storing each room's area has the advantage of a decreased materialization time but the disadvantage of utilizing more space (assuming a large number of rooms). The essential characteristics of the tradeoff are illustrated in Figure 3. For a small number of requests for areas of rooms, alternative 2 has a smaller cost. However, as the number of such requests grow, the cost for alternative 2 increases faster than for alternative 1, and alternative 1 becomes attractive.

Conceptual Issues. Although a functional model for data provides the potential for improving technical performance and efficiency of the system, of more importance is the logical consistency provided by looking at information in this fashion. Rather than examining each type of information as a special case, the model supplies a uniform view of facts, as well as the capability of making information systems cleaner and more powerful.

An information system is, in some sense, a model of recorded facts about the real world. Unfortunately, the modelling process does not capture all of the knowledge about a real world system; certain characteristics are simplified and omitted. By improving the way in which an information system is conceived, that is, by making the model of information more accurately reflect the real world, the capabilities of the information system can be improved. This can be accomplished by giving the system more "knowledge" about itself. This is known as giving the system an inferential ability to use this knowledge. The result is what has been termed "information profit", or the ability to make intelligent assumptions about the manipulation of information (17). Such a system could develop answers to questions even though these answers had not been explicitly defined in the data base.

Current models of information lack this quality of providing the system with some intelligence. The relational model of Codd (4), for example, views information relationships in terms of the mathematical concept of relations. Returning to the example of the area of a rectangular room, this information can be expressed as the following relation:



cost = f(space) + f(# of requests)

FIXED VARIABLE

FIGURE 3

Access cost v. space tradeoff

Room Id.	Length	Width	Area
1	1	1	1
2	2	4	8
3	2	6	12

By expressing the information in this way, however, the system loses some knowledge about how the data was derived. Most models of information suffer from the same weakness. They only allow relationships to be defined by grouping and set type operations. It is up to the user of such systems to distinguish the underlying characteristics of these relationships.

The Automatic Programming Group at MIT's Project MAC have attacked this problem by classifying types of relationships which can exist. In their MAPL system (19,20), they allow such constructs as "a kind of" and " a characteristic of" to give some meaning to the defined relationships. The user, given several of these primitive types of relationships, can define his own more complex types. The major problem with this approach is that a complete set of primitives needs to be defined before all types of information relationships can be constructed -- a formidable task. In fact, one of the problems encountered in MAPL is that no procedural type of primitives are defined, so expressing a relation like area is not possible.

By taking the view that all facts are obtained through applying a function, the problem of giving a system increased inferential ability is reduced to defining the implicit concepts behind the functions which are applied (coming up with a good model for the functions, if you will). With some information, this is difficult to do. As an example, describing exactly what is meant by the concept of "color" is a non-trivial problem in artificial intelligence. With many functions, however, especially those near the pure algorithm end of the information spectrum, a good model is reasonably easy to do. Defining rectangular area as PRODUCT(LENGTH,WIDTH), for example, is a reasonably complete conceptual description.

One benefit from a system with inferential ability is ease of use. Because the system can infer things about its information structure, the user is saved from doing some work. Returning to the rectangular area example, the user only needs to input the length and width of a particular room, but he can query the system about the room's area even though that information was not explicitly given to the system.

Another benefit is a reduction in inconsistency of information in the data base. As the amount and complexity of information grows, the probability increases that contradictory facts exist in the data base. Mealy (21) uses the example data of a person whose date of death precedes her date of birth. If the information relationships are more explicitly defined, it is easier to build in consistency checks to help eliminate this problem.

An example of how a functional view of data can improve a system's inferential capability is in the area of handling units of measure

(22). The association of a unit of measure with an attribute can either be done with each occurrence of a data item or factored into a catalog describing common characteristics of all instances. In many systems, this distinction is important; viewing all facts as function values, it becomes irrelevant. Conversion between various units of measure is analagous to converting between data types except the conversion is based on the content of the information rather than the form of its representation. Representation of complex units of measure, such as pounds per square inch, is easily accomplished since procedural definitions are allowed. A more complicated units of measure problem is typified by the request for the cost in cents of five bolts if bolt cost is stored in units such as dollars per ton. The system must change the unit of cost so that it reflects cost per bolt rather than cost per unit weight. This can be accomplished by defining an "each" function which, given the weight of a bolt, performs the transformation.

The bolt problem illustrates another level to information systems besides the physical and the logical. This is the inferential level. Many inferential problems require that the system be given additional information. For example, solution of the bolts problem required the system to be able to ask for the weight of bolts. This implies the need for an inferential model of information as well. Virtual information, through procedural ways of relating data maps, provides a good starting point for such a model.

SUMMARY AND CONCLUSIONS

This paper presents a model of an information system as a collection of functions which produce values in response to arguments. Virtual information unifies data, algorithms, and their combinations into a range of alternative methods for meeting requests for information. Various classes of virtual information prove useful in resolving the issues of representation and materialization. Together, the functional model and virtual information have important implications in data-base systems in terms of improving system performance and by allowing a conceptually simple, yet consistent view of many formerly disparate issues. In particular, these concepts are useful in dealing with significant issues such as automatic restructuring and inference. Such issues will be critical in the "intelligent" data-base systems needed for the future.

BIBLIOGRAPHY

1. Bachman, C. W. "Data Structure Diagrams," Data Base (Quarterly Newsletter of ACM-SIGBDP), Vol. 1, No. 2 (1969), pp. 4-10.

2. _____. "The Evolution of Storage Structures," Communications of the ACM, Vol. 15, No. 7 (July 1972), pp. 628-636.

3. CODASYL Systems Committee. Feature Analysis of Generalized Data Base Management Systems. New York: ACM, 1971.

4. Codd, E. F. "A relational model of data for a large shared data bank, "Communications of the ACM, Vol. 13, No. 6 (June 1970), pp. 377-387.

5. _____. "A Data Base Sublanguage Founded on the Relational Calculus, "1971 SIGFIDET Workshop Proceedings. New York: ACM, 1971. pp 35-68

6. Data Base Task Group. CODASYL Data Base Task Group Report. New York: ACM, 1971.

7. Date, C.J., and Hopewell, P. "File Definition and Logical Date Independence, "1971 SIGFIDET Workshop Proceedings. New York: ACM, 1971. pp 117-138.

8. Davies, C. T. A Logical Concept for the Control and Management of Data. Report AR-0803-00, International Business Machines Corp., System Development Division, Poughkeepsie, New York (1967).

9. Engles, R. W. A Tutorial on Data-Base Organization. Report TR-00.2004, International Business Machines Corp., System Development Division, Poughkeepsie, New York (1970).

10. Folinus, J. J. Design of a Data-Base Information System for Building Design. Research Report R73-52. Cambridge, Mass.: MIT Dept. of Civil Engineering, 1973.

11. Hsaio, D. K. and Harary, F. "A Formal system for information retrieval from files, "Communications of the ACM, Vol. 13, No. 2 (February 1970), pp 67-73.

12. International Business Machines Corp. Generalized Information System GIS/360: Application Description Manual (Version 2). Form GH20-0892-0, Data Processing Division, White Plains, New York 10604 (1970).

13. _____. Information Management System IMS/360: Application Description Manual. Form GH20-7765-1, Data Processing Division, White Plains, New York 10604 (1971).

14. Inverson, K. E. A Programming Language. New York: John Wiley and Sons, 1962.

15. Kingston, P. L. "Concepts of financial models," IBM Systems Journal, Vol. 12, No. 2 (1976), pp. 113-125.

16. Madnick, S. E. Design Strategies for File Systems. Project MAC Report TR-78. Cambridge, Mass.: MIT, 1970.

17. _____. "Automated Information Systems Generation Project, " Internal Memo, Sloan School Information Systems Group, MIT (October 1973).

18. Madnick, S. E., and Alsop, J. W. "A Modular Approach to File System Design," AFIPS Conference Proceedings, Vol. 34 (1969 SJCC).

19. Martin, W. A. and Krumland, R. "MAPL -- A Language for Describing Models of the World," Inernal Memo No. 6, Automatic Programming Group, Project MAC, MIT.

20. Martin, W. A., Krumland, R. B., and Sunguroff, A. "More MAPL Specifications and Basic Structures," Internal Memo No. 8, Automatic Programming Group, Project MAC, MIT.

21. Mealy, G. H. "Another look at data," AFIPS Conference Proceedings, Vol, 31 (1967 FJCC), pp 525-534.

22. Oshrin, A., and Schutzman, H. B. "The Units of Measure Problem," Internal Memo, Sloan School Information Systems Group, MIT (October 1973).

23. Raver, N. "File Organiztion in Management Information and Control Systems," File Organization - Selected Papers from File-68, IFIP Administrative Data Processing Group (IAG), Pulication No. 3, 1969.

24. Senko, M. E., Altman, E. B., Astrahan, M. M. and Fehder, P. L. "Data Structures and Accessing in Data Base Systems," IBM Systems Journal, Vol. 12, No. 1, pp. 30-93

25. Schubert, R. F. "Basic Concepts in Data-base Management," Datamation, July 1972, pp 42-47.