1973 Turing Award Lecture

The Programmer as Navigator

CHARLES W. BACHMAN

The Turing Award citation read by Richard G. Canning, chairman of the 1973 Turing Award Committee, at the presentation of this lecture on August 28 at the ACM Annual Conference in Atlanta:

A significant change in the computer field in the last five to eight years has been made in the way we treat and handle data. In the early days of our field, data was intimately tied to the application programs that used it. Now we see that we want to break that tie. We want data that is independent of the application programs that use it—that is, data that is organized and structured to serve many applications and many users. What we seek is the database.

This movement toward the database is in its infancy. Even so, it appears that there are now between 1,000 and 2,000 true database management systems installed worldwide. In ten years very likely, there will be tens of thousands of such systems. Just from the quantities of installed systems, the impact of databases promises to be huge.

This year's recipient of the A. M. Turing Award is one of the real pioneers of database technology. No other individual has had the influence that he has had upon this aspect of our field. I single out three prime examples of what he has done. He was the creator and principal architect of the first

Author's present address: Bachman Information Systems, Inc., 4 Cambridge Center, Cambridge, MA 02142.

commercially available database management system — the Integrated Data Store — originally developed from 1961 to 1964.[1,2,3,4] *I-D-S is today one of the three most widely used database management systems. Also, he was one of the founding members of the CODASYL Database Task Group, and served on that task group from 1966 to 1968. The specifications of that task group are being implemented by many suppliers in various parts of the world.*[5,6] *Indeed, currently these specifications represent the only proposal of stature for a common architecture for database management systems. It is to his credit that these specifications, after extended debate and discussion, embody much of the original thinking of the Integrated Data Store. Thirdly, he was the creator of a powerful method for displaying data relationships — a tool for database designers as well as application system designers.*[7,8]

His contributions have thus represented the union of imagination and practicality. The richness of his work has already had, and will continue to have, a substantial influence upon our field.

I am very pleased to present the 1973 A. M. Turing Award to Charles W. Bachman.

> Copernicus completely reoriented our view of astronomical phenomena when he suggested that the earth revolves about the sun. There is a growing feeling that data processing people would benefit if they were to accept a radically new point of view, one that would liberate the application programmer's thinking from the centralism of core storage and allow him the freedom to act as a navigator within a database. To do this, he must first learn the various navigational skills; then he must learn the "rules of the road" to avoid conflict with other programmers as they jointly navigate the database information space.
>
> This orientation will cause as much anguish among programmers as the heliocentric theory did among ancient astronomers and theologians.

This year the whole world celebrates the five-hundredth birthday of Nicolaus Copernicus, the famous Polish astronomer and mathematician. In 1543, Copernicus published his book, *Concerning the Revolutions of Celestial Spheres*, which described a new theory about the relative physical movements of the earth, the planets, and the sun. It was in direct contradiction with the earth-centered theories which had been established by Ptolemy 1400 years earlier.

[1] A general purpose programming system for random access memories (with S.B. Williams). Proc. AFIPS 1964 FJCC, Vol. 26, AFIPS Press, Montvale, N.J., pp. 411–422.

[2] Integrated Data Store. *DPMA Quarterly* (Jan. 1965).

[3] Software for random access processing. *Datamation* (Apr. 1965), 36–41.

[4] Integrated Data Store — Case Study. Proc. Sec. Symp. on Computer-Centered Data Base Systems sponsored by ARPA, SDC, and ESD, 1966.

[5] Implementation techniques for data structure sets. Proc. of SHARE Working Conf. on Data Base Systems, Montreal, Canada, July 1973.

[6] The evolution of data structures. Proc. NordDATA Conf., Aug. 1973, Copenhagen, Denmark, pp. 1075–1093.

[7] Data structure diagrams. Data Base 1, 2 (1969), Quarterly Newsletter of ACM SIGBDP, pp. 4–10.

[8] Set concepts for data structures. In *Encyclopedia of Computer Science*, Amerback Corp. (to be published in 1974).

Copernicus proposed the heliocentric theory, that planets revolve in a circular orbit around the sun. This theory was subjected to tremendous and persistent criticism. Nearly 100 years later, Galileo was ordered to appear before the Inquisition in Rome and forced to state that he had given up his belief in the Copernican theory. Even this did not placate his inquisitors, and he was sentenced to an indefinite prison term, while Copernicus's book was placed upon the Index of Prohibited Books, where it remained for another 200 years.

I raise the example of Copernicus today to illustrate a parallel that I believe exists in the computing or, more properly, the information systems world. We have spent the last 50 years with almost Ptolemaic information systems. These systems, and most of the thinking about systems, were based on a computer-centered concept. (I choose to speak of 50 years of history rather than 25, for I see today's information systems as dating from the beginning of effective punched card equipment rather than from the beginning of the stored program computer.)

Just as the ancients viewed the earth with the sun revolving around it, so have the ancients of our information systems viewed a tab machine or computer with a sequential file flowing through it. Each was an adequate model for its time and place. But after a while, each has been found to be incorrect and inadequate and has had to be replaced by another model that more accurately portrayed the real world and its behavior.

Copernicus presented us with a new point of view and laid the foundation for modern celestial mechanics. That view gave us the basis for understanding the formerly mysterious tracks of the sun and the planets through the heavens. A new basis for understanding is available in the area of information systems. It is achieved by a shift from a computer-centered to the database-centered point of view. This new understanding will lead to new solutions to our database problems and speed our conquest of the *n*-dimensional data structures which best model the complexities of the real world.

The earliest databases, initially implemented on punched cards with sequential file technology, were not significantly altered when they were moved, first from punched card to magnetic tape and then again to magnetic disk. About the only things that changed were the size of the files and the speed of processing them.

In sequential file technology, search techniques are well established. Start with the value of the primary data key, of the record of interest, and pass each record in the file through core memory until the desired record, or one with a higher key, is found. (A primary data key is a field within a record which makes that record unique within the file.) Social security numbers, purchase order numbers, insurance policy numbers, bank account numbers are all primary data keys. Almost without exception, they are synthetic attributes specifically designed and created for the purpose of uniqueness. Natural attributes, e.g., names of people

and places, dates, time, and quantities, are not assuredly unique and thus cannot be used.

The availability of direct access storage devices laid the foundation for the Copernican-like change in viewpoint. The directions of "in" and "out" were reversed. Where the input notion of the sequential file world meant "into the computer from tape," the new input notion became "into the database." This revolution in thinking is changing the programmer from a stationary viewer of objects passing before him in core into a mobile navigator who is able to probe and traverse a database at will.

Direct access storage devices also opened up new ways of record retrieval by primary data key. The first was called randomizing, calculated addressing, or hashing. It involved processing the primary data key with a specialized algorithm, the output of which identified a preferred storage location for that record. If the record sought was not found in the preferred location, then an overflow algorithm was used to search places where the record alternately would have been stored, if it existed at all. Overflow is created when the preferred location is full at the time the record was originally stored.

As an alternative to the randomizing technique, the index sequential access technique was developed. It also used the primary data key to control the storage and retrieval of records, and did so through the use of multilevel indices.

The programmer who has advanced from sequential file processing to either index sequential or randomized access processing has greatly reduced his access time because he can now probe for a record without sequentially passing all the intervening records in the file. However, he is still in a one-dimensional world as he is dealing with only one primary data key, which is his sole means of controlling access.

From this point, I want to begin the programmer's training as a full-fledged navigator in an n-dimensional data space. However, before I can successfully describe this process, I want to review what "database management" is.

It involves all aspects of storing, retrieving, modifying, and deleting data in the files on personnel and production, airline reservations, or laboratory experiments—data which is used repeatedly and updated as new information becomes available. These files are mapped through some storage structure onto magnetic tapes or disk packs and the drives that support them.

Database management has two main functions. First is the inquiry or retrieval activity that reaccesses previously stored data in order to determine the recorded status of some real world entity or relationship. This data has previously been stored by some other job, seconds, minutes, hours, or even days earlier, and has been held in trust by the database management system. A database management system has a continuing responsibility to maintain data between the time when it

was stored and the time it is subsequently required for retrieval. This retrieval activity is designed to produce the information necessary for decision making.

Part of the inquiry activity is report preparation. In the early years of sequential access storage devices and the resultant batch processing there was no viable alternative to the production of massive file dumps as formatted as reports. Spontaneous requirements to examine a particular checking account balance, an inventory balance, or a production plan could not be handled efficiently because the entire file had to be passed to extract any data. This form of inquiry is now diminishing in relative importance and will eventually disappear except for archival purposes or to satisfy the appetite of a parkinsonian bureaucracy.

The second activity of database management is to update, which includes the original storage of data, its repeated modification as things change, and ultimately, its deletion from the system when the data is no longer needed.

The updating activity is a response to the changes in the real world which must be recorded. The hiring of a new employee would cause a new record to be stored. Reducing available stock would cause an inventory record to be modified. Cancelling an airline reservation would cause a record to be deleted. All of these are recorded and updated in anticipation of future inquiries.

The sorting of files has been a big user of computer time. It was used in sorting transactions prior to batch sequential update and in the preparation of reports. The change to transaction-mode updating and on-demand inquiry and report preparation is diminishing the importance of sorting at the file level.

Let us now return to our story concerning the programmer as navigator. We left him using the randomizing or the index sequential technique to expedite either inquiry or update of a file based upon a primary data key.

In addition to a record's primary key, it is frequently desirable to be able to retrieve records on the basis of the value of some other fields. For example, it may be desirable, in planning ten-year awards, to select all the employee records with the "year-of-hire" field value equal to 1964. Such access is retrieval by secondary data key. The actual number of records to be retrieved by a secondary key is unpredictable and may vary from zero to possibly include the entire file. By contrast, a primary data key will retrieve a maximum of one record.

With the advent of retrieval on secondary data keys, the previously one-dimensional data space received additional dimensions equal to the number of fields in the record. With small or medium-sized files, it is feasible for a database system to index each record in the file on every field in the record. Such totally indexed files are classified as inverted files. In large active files, however, it is not economical to index every

field. Therefore, it is prudent to select the fields whose content will be frequently used as a retrieval criterion and to create secondary indices for those fields only.

The distinction between a file and a database is not clearly established. However, one difference is pertinent to our discussion at this time. In a database, it is common to have several or many different kinds of records. For an example, in a personnel database there might be employee records, department records, skill records, deduction records, work history records, and education records. Each type of record has its own unique primary data key, and all of its other fields are potential secondary data keys.

In such a database the primary and secondary keys take on an interesting relationship when the primary key of one type of record is the secondary key of another type of record. Returning to our personnel database as an example—the field named "department code" appears in both the employee record and the department record. It is one of several possible secondary data keys of the employee records and the single primary data key of the department records.

This equality of primary and secondary data key fields reflects real world relationships and provides a way to reestablish these relationships for computer processing purposes. The use of the same data value as a primary key for one record and as a secondary key for a set of records is the basic concept upon which data structure sets are declared and maintained. The Integrated Data Store (I-D-S) systems and all other systems based on its concepts consider their basic contribution to the programmer to be the capability to associate records into data structure sets and the capability to use these sets as retrieval paths. All the COBOL Database Task Group systems implementations fall into this class.

There are many benefits gained in the conversion from several files, each with a single type of record, to a database with several types of records and database sets. One such benefit results from the significant improvement in performance that accrues from using the database sets in lieu of both primary and secondary indices to gain access to all the records with a particular data key value. With database sets, all redundant data can be eliminated, reducing the storage space required. If redundant data is deliberately maintained to enhance retrieval performance at the cost of maintenance, then the redundant data can be controlled to ensure that the updating of a value in one record will be properly reflected in all other appropriate records. Performance is enhanced by the so-called "clustering" ability of databases where the owner and some or most of the members records of a set are physically stored and accessed together on the same block or page. These systems have been running in virtual memory since 1962.

Another significant functional and performance advantage is to be able to specify the order of retrieval of the records within a set based upon a declared sort field or the time of insertion.

In order to focus the role of programmer as navigator, let us enumerate his opportunities for record access. These represent the commands that he can give to the database system—singly, multiply or in combination with each other—as he picks his way through the data to resolve an inquiry or to complete an update.

1. He can start at the beginning of the database, or at any known record, and sequentially access the "next" record in the database until he reaches a record of interest or reaches the end.
2. He can enter the database with a database key that provides direct access to the physical location of a record. (A database key is the permanent virtual memory address assigned to a record at the time that it was created.)
3. He can enter the database in accordance with the value of a primary data key. (Either the indexed sequential or randomized access techniques will yield the same result.)
4. He can enter the database with a secondary data key value and sequentially access all records having that particular data value for the field.
5. He can start from the owner of a set and sequentially access all the member records. (This is equivalent to converting a primary data key into a secondary data key.)
6. He can start with any member record of a set and access either the next or prior member of that set.
7. He can start from any member of a set and access the owner of the set, thus converting a secondary data key into a primary data key.

Each of these access methods is interesting in itself, and all are very useful. However, it is the synergistic usage of the entire collection which gives the programmer great and expanded powers to come and go within a large database while accessing only those records of interest in responding to inquiries and updating the database in anticipation of future inquiries.

Imagine the following scenario to illustrate how processing a single transaction could involve a path through the database. The transaction carries with it the primary data key value or database key of the record that is to be used to gain an entry point into the database. That record would be used to gain access to other records (either owner or members) of a set. Each of these records is used in turn as a point of departure to examine another set.

For example, consider a request to list the employees of a particular department when given its departmental code. This request could be supported by a database containing only two different types of records: personnel records and department records. For simplicity purposes, the department record can be envisioned as having only two fields: the department code, which is the primary data key; and the department name, which is descriptive. The personnel record can be envisioned

as having only three fields: the employee number, which is the primary data key for the record; the employee name, which is descriptive; and the employees department code, which is a secondary key which controls set selection and the records placement in a set. The joint usage of the department code by both records and the declaration of a set based upon this data key provide the basis for the creation and maintenance of the set relationship between a department record and all the records representing the employees of that department. Thus the usage of the set of employee records provides the mechanism to readily list all the employees of a particular department following the primary data key retrieval of the appropriate department record. No other record for index need be accessed.

The addition of the department manager's employee number to the department record greatly extends the navigational opportunities, and provides the basis for a second class of sets. Each occurrence of this new class includes the department records for all the departments managed by a particular employee. A single employee number or department code now provides an entry point into an integrated data structure of an enterprise. Given an employee number, and the set of records of departments managed, all the departments which he manages can be listed. The personnel of each such department can be further listed. The question of departments managed by each of these employees can be asked repeatedly until all the subordinate employees and departments have been displayed. Inversely, the same data structure can easily identify the employee's manager, the manager's manager, and the manager's manager's manager, and so on, until the company president is reached.

There are additional risks and adventures ahead for the programmer who has mastered operation in the *n*-dimensional data space. As navigator he must brave dimly perceived shoals and reefs in his sea, which are created because he has to navigate in a shared database environment. There is no other obvious way for him to achieve the required performance.

Shared access is a new and complex variation of multiprogramming or time sharing, which were invented to permit shared, but independent, use of the computer resources. In multiprogramming, the programmer of one job doesn't know or care that his job might be sharing the computer, as long as he is sure that his address space is independent of that of any other programs. It is left to the operating system to assure each program's integrity and to make the best use of the memory, processor, and other physical resources. Shared access is a specialized version of multiprogramming where the critical, shared resources are the records of the database. The database records are fundamentally different than either main storage or the processor because their data fields change value through update and do not return to their original condition afterward. Therefore, a job that repeatedly

uses a database record may find that record's content or set membership has changed since the last time it was accessed. As a result, an algorithm attempting a complex calculation may get a somewhat unstable picture. Imagine attempting to converge on an iterative solution while the variables are being randomly changed! Imagine attempting to carry out a trial balance while someone is still posting transactions to the accounts! Imagine two concurrent jobs in an airline reservations system trying to sell the last seat on a flight!

One's first reaction is that this shared access is nonsense and should be forgotten. However, the pressures to use shared access are tremendous. The processors available today and in the foreseeable future are expected to be much faster than are the available direct access storage devices. Furthermore, even if the speed of storage devices were to catch up with that of the processors, two more problems would maintain the pressure for successful shared access. The first is the trend toward the integration of many single purpose files into a few integrated databases; the second is the trend toward interactive processing where the processor can only advance a job as fast as the manually created input messages allow. Without shared access, the entire database would be locked up until a batch program or transaction and its human interaction had terminated.

The performance of today's direct access storage devices is greatly affected by patterns of usage. Performance is quite slow if the usage is an alternating pattern of: access, process, access, process,..., where each access depends upon the interpretation of the prior one. When many independent accesses are generated through multiprogramming, they can often be executed in parallel because they are directed toward different storage devices. Furthermore, when there is a queue of requests for access to the same device, the transfer capacity for that device can actually be increased through seek and latency reduction techniques. This potential for enhancing throughput is the ultimate pressure for shared access.

Of the two main functions of database management, inquiry and update, only update creates a potential problem in shared access. An unlimited number of jobs can extract data simultaneously from a database without trouble. However, once a single job begins to update the database, a potential for trouble exists. The processing of a transaction may require the updating of only a few records out of the thousands or possibly millions of records within a database. On that basis, hundreds of jobs could be processing transactions concurrently and actually have no collisions. However, the time will come when two jobs will want to process the same record simultaneously.

The two basic causes of trouble in shared access are interference and contamination. *Interference* is defined as the negative effect of the updating activity of one job upon the results of another. The example I have given of one job running an accounting trial balance while

another was posting transactions illustrates the interference problem. When a job has been interfered with, it must be aborted and restarted to give it another opportunity to develop the correct output. Any output of the prior execution must also be removed because new output will be created. *Contamination* is defined as the negative effect upon a job which results from a combination of two events: when another job has aborted and when its output (i.e., changes to the database or messages sent) has already been read by the first job. The aborted job and its output will be removed from the system. Moreover, the jobs contaminated by the output of the aborted job must also be aborted and restarted so that they can operate with correct input data.

A critical question in designing solutions to the shared access problem is the extent of visibility that the application programmer should have. The Weyerhaeuser Company's shared access version of I-D-S was designed on the premise that the programmer should not be aware of shared access problems. That system automatically blocks each record updated and every message sent by a job until that job terminates normally, thus eliminating the contamination problem entirely. One side effect of this dynamic blocking of records is that a deadlock situation can be created when two or more jobs each want to wait for the other to unblock a desired record. Upon detecting a deadlock situation, the I-D-S database system responds by aborting the job that created the deadlock situation, by restoring the records updated by that job, and by making those records available to the jobs waiting. The aborted job, itself, is subsequently restarted.

Do these deadlock situations really exist? The last I heard, about 10 percent of all jobs started in Weyerhaeuser's transaction-oriented system had to be aborted for deadlock. Approximately 100 jobs per hour were aborted and restarted. Is this terrible? Is this too inefficient? These questions are hard to answer because our standards of efficiency in this area are not clearly defined. Furthermore, the results are application-dependent. The Weyerhaeuser I-D-S system is 90 percent efficient in terms of jobs successfully completed. However, the real questions are:

—Would the avoidance of shared access have permitted more or fewer jobs to be completed each hour?
—Would some other strategy based upon the detecting rather than avoiding contamination have been more efficient?
—Would making the programmer aware of shared access permit him to program around the problem and thus raise the efficiency?

All these questions are beginning to impinge on the programmer as navigator and on the people who design and implement his navigational aids.

My proposition today is that it is time for the application programmer to abandon the memory-centered view, and to accept the challenge and opportunity of navigation within an n-dimensional data space. The

software systems needed to support such capabilities exist today and are becoming increasingly available.

Bertrand Russell, the noted English mathematician and philosopher, once stated that the theory of relativity demanded a change in our imaginative picture of the world. Comparable changes are required in our imaginative picture of the information system world.

The major problem is the reorientation of thinking of data processing people. This includes not only the programmer but includes the application system designers who lay out the basic application programming tasks and the product planners and the system programmers who will create tomorrow's operating system, message system, and database system products.

Copernicus laid the foundation for the science of celestial mechanics more than 400 years ago. It is this science which now makes possible the minimum energy solutions we use in navigating our way to the moon and the other planets. A similar science must be developed which will yield corresponding minimum energy solutions to database access. This subject is doubly interesting because it includes the problems of traversing an existing database, the problems of how to build one in the first place and how to restructure it later to best fit the changing access patterns. Can you imagine restructuring our solar system to minimize the travel time between the planets?

It is important that these mechanics of data structures be developed as an engineering discipline based upon sound design principles. It is important that it can be taught and is taught. The equipment costs of the database systems to be installed in the 1980's have been estimated at $100 billion (at 1970 basis of value). It has further been estimated that the absence of effective standardization could add 20 percent or $20 billion to the bill. Therefore, it is prudent to dispense with the conservatism, the emotionalism, and the theological arguments which are currently slowing progress. The universities have largely ignored the mechanics of data structures in favor of problems which more nearly fit a graduate student's thesis requirement. Big database systems are expensive projects which university budgets simply cannot afford. Therefore, it will require joint university/industry and university/government projects to provide the funding and staying power necessary to achieve progress. There is enough material for a half dozen doctoral theses buried in the Weyerhaeuser system waiting for someone to come and dig it out. By this I do not mean research on new randomizing algorithms. I mean research on the mechanics of nearly a billion characters of real live business data organized in the purest data structures now known.

The publication policies of the technical literature are also a problem. The ACM SIGBDP and SIGFIDET publications are the best available, and membership in these groups should grow. The refereeing rules and practices of Communications of the ACM result in delays of one year

to 18 months between submittal and publication. Add to that the time for the author to prepare his ideas for publication and you have at least a two-year delay between the detection of significant results and their earliest possible publication.

Possibly the greatest single barrier to progress is the lack of general database information within a very large portion of the computer users resulting from the domination of the market by a single supplier. If this group were to bring to bear its experience, requirements, and problem-solving capabilities in a completely open exchange of information, the rate of change would certainly increase. The recent action of SHARE to open its membership to all vendors and all users is a significant step forward. The SHARE-sponsored Working Conference on Database Systems held in Montreal in July (1973) provided a forum so that users of all kinds of equipment and database systems could describe their experiences and their requirements.

The widening dialog has started. I hope and trust that we can continue. If approached in this spirit, where no one organization attempts to dominate the thinking, then I am sure that we can provide the programmer with effective tools for navigation.

Related articles are:

The evolution of storage structures. *Comm. ACM 15,* 7 (July 1972), 628–634.

Architectural Definition Technique: its objectives, theory, process, facilities and practice (with J. Bouvard). Proc. 1972 ACM SIGFIDET Workshop on Data Description, Access and Control, pp. 257–280.

Data space mapped into three dimensions; a viable model for studying data structures. Data Base Management Rep., InfoTech Information Ltd., Berkshire, U.K., 1973.

A direct access system with procedurally generated data structuring capability (with S. Brewer). *Honeywell Comput. J.* (to appear).

Categories and Subject Descriptors:

H.2.2 [**Database Management**]: Physical Design—*access methods*; H.2.4 [**Database Management**]: Systems—*transaction processing*; H.3.2 [**Storage and Retrieval**]: Information Storage—*file organization*; H.3.3 **Information Storage and Retrieval**]: Information Search and Retrieval—*retrieval models*

General Terms:

Algorithms, Design, Performance

Additional Key Words and Phrases:

Contamination, interference

1973 Turing Award Lecture

Postscript

The Programmer as Navigator, Architect, Communicator, Modeler, Collaborator, and Supervisor

CHARLES W. BACHMAN
Bachman Information Systems, Inc.

Thirteen years have passed since the writing of the Turing Award paper entitled, "The Programmer as Navigator." Databases have become common, even popular. Some programmers navigate. Others join. I have spent considerable effort in arguing the merits of the network (CODASYL) data model and in extending it for greater modeling power.[1,2,3,4] Arguments and debates concerning data models waxed hot and heavy and have now pretty much simmered down. Today, the only reasonable consensus is that one can do useful work with DBMSs based upon any of the popular data models, even with those DBMSs that have no apparent affinity to any particular data model.

The Programmer as Architect

The study of the architecture of computer-based information systems has progressed well in this period. Two projects, important in their own right, were instrumental in bringing this subject to the forefront. The ANSI/X3/SPARC Study Group on Database Management (1972–1977) reported[5] its architecture of data storage and retrieval. This was one of the first attempts to clearly understand and document the layers of software and human activity involved in the process of data storage and retrieval. It went further and identified and described the interfaces between the various software modules and between them and their human counterparts (administrators, database designers, and programmers). It was significant that this report identified both administrative and run-time interfaces. This project was instrumental in establishing the concept of a *conceptual schema*[6] as a higher level abstraction of information structure definitions, which is independent of data representation.

[1] Bachman, C. W. Why restrict the modeling capability of the CODASYL data structure sets? In *Proceedings of the AFIPS National Computer Conference*, vol. 46. AFIPS Press, Reston, Va., 1977.

[2] Bachman, C. W., and Daya, M. The role concept in data models. In *Proceedings of the 3rd Very Large Database Conference*, 1977.

[3] Bachman, C. W. The structuring capabilities of the molecular data model (partnership data model). In *Entity-Relationship Approach to Software Engineering*. Elsevier Science, New York, 1983.

[4] Bachman, C. W. The partnership data model. Presented at the Fall 1983 IEEE Computer Conference (Washington, D.C.).

[5] ANSI/X3/SPARC/Study Group—Database Management Systems. *Framework Report on Database Management Systems*. AFIPS Press, Reston, Va., 1978.

[6] ISO/TC97/SC5/WG3. Concepts and terminology for the conceptual schema. January 15, 1981.

Author's address: Bachman Information Systems, Inc., 4 Cambridge Center, Cambridge, MA 02142.

The Programmer as Communicator

The International Organization for Standardization, through its ISO/TC97/SC16, established (1979–1982) the Reference Model for Open Systems Interconnection. This Reference Model is an architectural master plan for data communications established as an international standard[7] with the intent that it be the controlling and integrating standard for a series of more detailed standards to follow. This architecture identified seven layers of processing involved in and supporting communication between application processes. Each layer was specified in terms of its "administrative entities,"[8] "processing entities," "services," and "protocols." For the processsing entities of each layer, there were four important interfaces to be established and standardized:

(1) the *services* that a processing entity offers to the processing entities in the layer immediately above;
(2) the communication *protocol* by which a processing entity communicates with other processing entities in the same layer;
(3) the *use*, by the processing entities of one layer, of the services provided by the processing entities of the layer immediately below;
(4) the administrative *protocol* by which a processing entity is controlled by the administrative entities within the same layer.

The detailed standards, developed subsequently for each layer, spell out the individual protocols, services, and service usage.

The vision and scope of this work can be seen in part by reviewing some of the discussions relating to addressability. How large should the address space be to identify all the processing entities that might wish to communicate with one another? One discussion followed this scenario:

> There will be close to 10 billion people in the world by the end of the year 2000 (10 billion addresses).
>
> Assume that, on the average, 100 robots will be working for each of these people (1 trillion addresses).
>
> Plan for unforeseen contingencies and a useful address space life of 25 years; so multiply by 10 (10 trillion addresses).
>
> Assume that the assignment of address is made through the political processes starting with the United Nations and that 99 percent of the addresses are effectively unavailable for applications level communications (1 quadrillion addresses).

Thus 1 quadrillion addresses is about the right order of magnitude for the address space being considered. This is a 1 followed by 15 zeros in the decimal system, or a 1 followed by approximately 50 zeros in the binary system.

This year the work on ISO standards for Open Systems Interconnection has received a great boost in support in the United States by the creation of COS (Corporation for Open Systems). COS is an industry-wide organization of users, carriers, and manufacturers formed to encourage the implementation of the ISO standards and to provide the testing environment so that a new or revised implementation can be validated for adherence to the ISO standards.

[7]ISO. Computers and Information Systems—Open Systems Interconnection Reference Model. Standard 7498. American National Standards Institute, New York, N.Y.

[8]The word "entity" is used in the ISO/TC97 world to mean an active element that plays some part in the communication process. I have used the adjectives "processing" and "administrative" to distinguish the communication-time entities from the set-up-time entities. This usage of the word entity contrasts with its use in the data modeling world where the word entity means something that exists and about which something is known.

The author, in his capacity as the chairman of ISO/TC97/SC16 reporting to ISO/TC97, recommended to TC97 that it develop a "reference model for computer-based information systems."[9,10] This extended reference model would be used to place all of ISO/TC97's work on computers and information systems into perspective and thus highlight the areas most critical for further standardization.

In 1984–1985, ISO/TC97 reorganized its committee structure creating a new subcommittee, ISO/TC97/SC21, which has assumed the former responsibilities of SC16 and has been given the additional responsibility of defining the architecture of data storage and retrieval. With time this responsibility should grow to include the aspects of data integrity and data security, since it is not possible to create a complete architecture for data storage and retrieval and data communication without their being integrated with the aspects of integrity and security.

The Programmer as Modeler

I have invested a good deal of my time in these 13 years in extending the conceptual schema work of ANSI/SPARC Study Group on DBMS, joining it with my work on data communications and formal description techniques. The scope of the original conceptual schema work was limited to the information that existed in the business and to its data formats as stored in files and databases (internal schema) and as viewed by programs (external schema). My goal was to extend this abstraction to include descriptions of all the *active agents* (people, computer programs, and physical processes) that were the users of the information, the *communication paths* that they use, and the *messages* that are exchanged.

I wanted to extend this abstraction further to include the rules that governed the behavior of the users of the information. These extended conceptual schemata have been called "enterprise models" or "business models".

Why build a business model? First, as a means of defining the information processing requirements for an organization in a manner that is equally clear to the user community and to the data processing community. Second, to provide the basis for automating the process of generating application software. I define the term *application software* to include database and file descriptions, the application programs, and the environmental control parameters required to install the required files and programs in the computers and to control their operation.

The step of translating a business model into the set of application software required to support that model is the step of translating the *what* of the business world into the *how* of the computer and communications world. This translation requires three additional elements over and above the business model as the formal specification:

1. It requires *information* about the quantities, rates, and response times that must be satisfied.

2. It requires *information* about the available processors, storage, and communication hardware and *information* about the available compilers, DBMSs, communication systems, transaction monitors, and operating systems.

[9]Bachman, C. W. The context of open systems interconnection within computer-based information systems. In *Proceedings of Gesellschaft für Informatik*, Jan. 1980.

[10]Bachman, C. W., and Ross, R. G. Toward a more complete reference model of computer-based information systems. *J. Comput. Standards 1* (1982); also published in *Comput. Networks 6* (1982).

3. It also requires the *expertise* to understand the operating and performance characteristics of the available software and hardware options and how to best use them to meet the functional and quantitative requirements in a cost-effective way.

This performance and optimization expertise has been embodied in the persons of real people, the database designers, application programmers, and system programmers. The best of them are very, very good, but the work of many has been disappointing. All these activities are expensive and more time consuming then any one would wish.

The Programmer as Collaborator

This shortage of good people has started us looking for a means of automating the work of database designers and systems and application programmers. This automation is difficult, as the process of translating the business model into efficient application software is not completely deterministic. There are frequently several alternative approaches with different dynamics and costs. Real expertise and judgment are involved. This difficulty has led to the examination of the tools and techniques coming out of the world of artificial intelligence, where there has been an emphasis on domains of imperfect knowledge.

The AI world, with its knowledge-based software system, has considerable experience developing interactive systems, where a resident human expert can collaborate with a "cloned" expert, which is built into the software to achieve some otherwise difficult task. Together they can carry out all the needed translations between the conceptual level of abstraction and the physical level taking into consideration the performance problems and opportunities.

Programmer as Supervisor

It is reasonable to think that these cloned experts, who are embodied in knowledge-based (expert) systems, will improve with time. As this happens, the role of the resident human expert (database designer, application programmer, or systems programmer) will progressively shift from that of a collaborator with the knowledge-based system to that of the supervisor. This supervisor will be responsible for checking the work of the knowledge-based system, to see that it has covered all modes of operation and all likely operating conditions. After checking and requesting any appropriate modifications, the human expert as supervisor will be required to countersign the final design, just as the engineering supervisor countersigns the work of the engineering staff. In business information systems, nothing goes into production without its being reviewed and someone's taking responsibility for it.

Summary

It is somewhat poetic to see the functional joining of database technology with AI technology. Poetic, because the early (1960) documentation of list processing in the artificial intelligence literature provided the basis for the linked lists used as the first and still most prevalent implementation mode for databases. The confusion between the concept and most prevalent implementation mode of the data structure set has been troublesome. There are a number of well-known techniques[11] for implementing data structure sets, each with its own

[11] Bachman, C. W. Implementation of techniques for data structure sets. In *Proceedings of SHARE Workshop on DataBase Systems* (Montreal, Canada, July, 1973).

performance characteristics, while maintaining the functional characteristics of the set.

It will be interesting to see whether the knowledge and implementation expertise of the database world will be able to make a significant contribution to the LISP and AI world as it reaches for commercial applications where the knowledge bases are large and concurrently shared among many distributed, cooperating AI workstations. Here performance and responsiveness are tied to the successful operation of shared virtual memories for knowledge-base purposes.

1975 Turing Award Lecture

Computer Science as Empirical Inquiry: Symbols and Search

ALLEN NEWELL and HERBERT A. SIMON

The 1975 ACM Turing Award was presented jointly to Allen Newell and Herbert A. Simon at the ACM Annual Conference in Minneapolis, October 20. In introducing the recipients, Bernard A. Galler, Chairman of the Turing Award Committee, read the following citation:

"It is a privilege to be able to present the ACM Turing Award to two friends of long standing, Professors Allen Newell and Herbert A. Simon, both of Carnegie-Mellon University.

"In joint scientific efforts extending over twenty years, initially in collaboration with J. C. Shaw at the RAND Corporation, and subsequently with numerous faculty and student colleagues at Carnegie-Mellon University, they have made basic contributions to artificial intelligence, the psychology of human cognition, and list processing.

"In artificial intelligence, they contributed to the establishment of the field as an area of scientific endeavor, to the development of heuristic programming generally, and of heuristic search, means-ends analysis, and methods of induction, in particular, providing demonstrations of the sufficiency of these mechanisms to solve interesting problems.

"In psychology, they were principal instigators of the idea that human cognition can be described in terms of a symbol system, and they have

Authors' present address: A. Newell, Department of Computer Science, and H. A. Simon, Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA 15213.