The Origin of the Integrated Data Store (IDS): The First Direct-Access DBMS

Charles W. Bachman

The Integrated Data Store (IDS), the first direct-access database management system, was developed at General Electric in the early 1960s. Revisiting the development challenges that lead to its first production version reveals the origins of DBMSs and their impact on software development and business management. IDS and its derivative systems are still in use today, supporting a thousand mainframe installations.

In the late 1950s and early 1960s, no independent software vendor industry existed. Software of all kinds was developed either by computer manufacturers or by one or a group of computer users. It was either bundled with the computer at no extra cost or given away free and shared by the users who developed it, much like today's open source movement.

This article will examine the business requirements that lead to the development of the Integrated Data Store (IDS), the first direct-access database management system (DBMS),[1] and will look at some of the development challenges that lead to its first production version. To this day, IDS leads a healthy, productive life, driving large transaction-oriented systems around the world, 50 years after its conception.

When General Electric engineers first began developing IDS in 1961, there were no general-purpose operating systems, no file systems, no DBMSs, and no communications systems to learn from or build on. There was no multiprogramming, time sharing, or online debugging tools. The machines were essentially naked. For business data processing, it was a batch-oriented, serial-file-processing, load and execute one-program-at-a-time world.

GE Integrated System Projects

In the late 1950s, GE was the largest commercial user of computers in the world. GE was also the biggest manufacturer of computers for demand deposit accounting.

From 1958 to 1965, the GE Integrated System Projects (ISPs) were driven by GE's Manufacturing Services, lead by Halbert Miller. The company's corporate services provided research, expertise, and consulting to its 100 product manufacturing departments. These product departments were in a range of product areas: atomic energy, electric energy generation and distribution, jet engines, electric motors, home appliances, light bulbs, x-ray machines, diesel electric locomotives, artificial diamonds, and general-purpose computers. At that time, GE's Manufacturing Services management was greatly concerned that all of GE's manufacturing businesses were investing heavily in the design and installation of computerized manufacturing systems and that the development process was slow, expensive, and error prone.

The first ISP (1958–1959) developed some interesting product ideas in the area of manufacturing simulation (later leading to SIMSCRIPT), generative engineering, and decision tables (TABSOL).

The second ISP (ISP 2) began late in 1960. Its target was to design and build a generic manufacturing control system. The project was managed by GE's Production Control Service and lead by Stanley B. Williams. Williams came from the GE Large Transformer Department in Pittsfield, Massachusetts, and had 12 years of engineering and engineering systems experience. I joined the project as its chief architect, with 10 years of experience in engineering, finance, manufacturing, and data processing with the Dow Chemical Company in Midland, Michigan. Homer Carney came from the GE Computer Department in Phoenix, Arizona, and was our only experienced

 Published by the IEEE Computer Society 1058-6180/09/$26.00

programmer. Staff members from other GE service organizations served a project audit function. The project had no unified financing because each team member was carried on his home organization's budget.

This project lead to the development of four significant products and tools:

- the Materials/Manufacturing/Management[2] Information and Control System (MIACS) application system,
- the Integrated Data Store (IDS) DBMS,
- the Problem Controller (OLTP) operating system, and
- Data Structure Diagrams.

Requirements

During the first six months of 1961, ISP 2 focused on a detailed examination of the manufacturing operations at the GE High Voltage Switchgear (HVSG) Department, located in Philadelphia, Pennsylvania. The plan was to review their current manufacturing system, determine what was and was not working, review their processing requirements, and then determine whether a broadly useful solution could be envisioned. We identified five requirements that had to be addressed successfully.

Requirement 1: Work-load constraints

In 1960, industry experience with large-scale engineering and construction projects indicated the need to explicitly recognize the network structure of the interconnected steps within large projects and to deal with the precedence of some steps as related to others. Both the Critical Path Method (CPM), developed by DuPont, and the Program Evaluation and Review Technique (PERT), developed for the US Navy's Polaris project, focused on project steps and the precedence relationships between these steps.

Many of GE's manufacturing departments had requirements similar to the large-scale engineering and construction projects that gave rise to CPM and PERT, but they had additional requirements that elevated their level of complexity. Each new customer order arriving at GE's HVSG department required the creation of a new process network identifying all the manufacturing steps and their precedence relationships. These new manufacturing orders had to compete with the existing orders for scarce manufacturing resources and for the common parts required for their assembly. New orders, with their manufacturing steps, precedence relationships, and resource and material requirements, were arriving daily. They had to be merged into the existing, business-wide network of planned manufacturing steps. Those steps had already been scheduled and their material requirements and resource requirements reserved. The process steps of the new orders had to be scheduled based on the use of resources and materials that were not already committed.

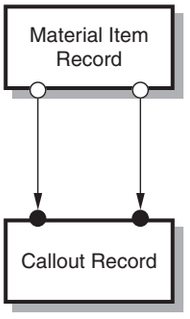

Figure 1. A Data Structure Diagram for a manufacturing bill of materials network structure. This diagram, and all those in this article, were created using the graphic design tool called the Bachman/DA (Bachman Data Analyst) product.[4]

Figure 1 is a data structure diagram that illustrates the classical manufacturing bill of materials network structure that is also the signature of the CPM and PERT network structures.[3] Such diagrams served as a graphic representation upon which we could trace the anticipated journey through the database to handle particular transactions. In the IDMS world, they are called Bachman Diagrams. Several variations of the graphic formats have appeared over time, such as ER Diagrams and Data Model Diagrams. Each **Material Item Record** may call out a list of subordinate **Material Item Records**. Conversely, each **Material Item Record** may be called out by a list of superior **Material Item Records**. The quantity called out is recorded as a data field of the **Callout Records**. The signature of a network data structure is the diagramming of entities (boxes) and owner/member (O/M) sets (arrows) such that one of the record types, the **Callout Record**, is a concurrent member in two O/M sets i.e., with the arrow heads pointing to it.

The planned execution dates for the manufacturing steps were always tentative and subject to reconsideration. The daily shop plan was based on the old foreman's rule of thumb: ''Dispatch, each day, sufficient job steps to keep all the available human and

shop resources busy.'' Such shop resources include various classes of skilled labor, machine tools, and specialized production areas.

Requirement 2: Expediting

In 1961, HVSG order-specific manufacturing plans were being created daily by a computer application as the orders arrived from customers. Their required manufacturing steps—machining, assembly, painting, and inspection—were planned and given a scheduled date, without any knowledge or consideration of the orders that had already been scheduled or of resource limitations. Common parts were planned and manufactured on an economic-order-size basis.

All too often, the first alert that customer orders were late came when the customers called to complain that their order had not arrived on schedule!

The root of the problem became clear after observing the many tote bins stacked under the windows and along the aisles of the various shops and assembly areas. Each tote bin contained a pile of parts, in the process of being manufactured, and had a small packet of punched cards wired to the box. Each card represented a planned manufacturing step. It displayed a date indicating when that step was supposed to be started. Many cards, for steps that should have been finished, were still attached to the tote bins. The original, computer-generated completion dates were frequently being missed, and the original schedule was seldom useful.

After some discussion with the manufacturing people at HVSG, it became clear that the back end of their existing manufacturing planning and control system consisted of a team of expediters (manufacturing control people). They were busy, running around with listings of the original, computer-generated schedules, trying to push parts and subassemblies through the machine shop and assembly areas so that the customer orders could be shipped on schedule. The existing manufacturing system had three key limitations; that is, (1) it did not actively manage the resource requirements of the manufacturing steps, (2) it did not adequately recognize the network structure of the interconnected steps between independent orders, and (3) it did not notice or manage unplanned events. Clearly, a new system was required.

Requirement 3: Planning for a moving target

The failures of the existing computerized planning at HVSG reminded me strongly of my World War II experience in the 165th AAA (Gun) Battalion in the Southwest Pacific Theater. The theory was that, ''our analog, mechanical, M1 fire control directors (computers) could make an 'accurate' plan to shoot down the enemy bombers.'' In practice, we almost always missed them. The planning was deemed accurate because we could always hit large, fabric wind socks being towed by a friendly target towing airplane that was flying straight and level. What was wrong? Once a gun was aimed, the fuse set, and the gun fired, there was a 15- to 25-second window when we depended on the enemy aircraft continuing to fly straight and level until the 90-mm shell could reach their predicted position and explode. The enemy pilots learned to not fly straight and level. This would sound like a failed system until you realized that the enemy bomber's bomb sights also assumed that they would fly straight and level. We both made a lot of noise but did not do much damage.

In a related manner, the HVSG manufacturing planning and control system assumed that the manufacturing process in the factory could and would ''fly straight and level'' from the time an order was received until that order was shipped. That also was unrealistic and almost never happened.

By 1960, when the ISP 2 project began, the world had become familiar with the US Air Force's air-to-air, Sidewinder missiles and their ''I can see you and I can fly twice as fast as you can'' targeting strategy. The new manufacturing planning and scheduling system would need to adopt the Sidewinder missile's adaptive planning strategy. It would need to initiate a replanning cycle whenever (1) new orders, (2) change orders, or (3) feedback made the old plan incomplete or unworkable. The goal would be to incrementally create a new plan, consistent with the new reality including resource availability, whenever the previous plan was no longer feasible.

Requirement 4: Record-at-a-time processing

The goal was to make the programming of the manufacturing control application programs as easy and foolproof as possible. Programming experience was limited in 1960 and nonexistent as far as programming applications that worked in and out of disk files. Most business application programs were written in an assembly language or a high-level programming language.

At GE, this meant in GECOM. For those not familiar with GECOM, it was one of a family of programming languages, including Comtran (IBM) and Fact (Honeywell), which were the computer-company-specific predecessors of Cobol.

These programming languages included logic statements, arithmetic statements, MOVE statements, and READ and WRITE statements, which controlled the serial data I/O on punched-card equipment, printers, and magnetic-tape devices. All these were serial-access devices, where streams of data were read in and written out. The READ and WRITE statements were designed to support databases, which existed in sequential files that were periodically updated and read to generate reports.

Even though these files were serially processed, from beginning to end, the programming languages were styled as ''record-at-a-time'' languages.[5] It was felt that preserving this record-at-a-time paradigm was important for the introduction of transaction data processing with data organized as networks of interrelated records. Thus, the database programmers could continue to work on the record-at-a-time basis, but the new network data structures would support ''next-record'' access, where the next record was programmatically selected from one of many, simultaneously accessible business-oriented owner/member (O/M) sets. The application programs could navigate the networks of interrelated manufacturing steps, resources, inventories, and precedence relationships that captured and held the manufacturing control information.

In other words, the requirement was to permit any record to be inserted and maintained in one or more logical sequences of records. Then the programmer could use the record-at-a-time file-processing metaphor to navigate through sets of records of any design or complexity. This was essential to support the complexity of the manufacturing business.

Requirement 5: A transaction-oriented OS

To achieve this, we needed to be able to quickly receive and process new information regarding new orders, order changes, changes in available resources and materials, and feedback from the factory, which made batch processing impossible. New information had to be made available to the manufacturing control system as soon as it became available.

Hence, a ''store and forward'' OS was required that could (1) collect and queue the various transactions and their input data, pending their execution; (2) schedule processing consistent with the priority of and precedence relationships between competing transactions; (3) load the appropriate program to handle each transaction; and (4) pass control of the input data associated with the transaction to the application program for processing. This rapid response transaction processing required that all the application programs be available on the disk file in ''load-and-go'' format.

Integrated Data Store (IDS)

The IDS DBMS was created by assembling many elements that had appeared in research papers and existing systems. We combined these with some new elements so that the whole would meet the envisioned manufacturing control system's requirements. This included the following elements:

- A direct access database was implemented on a virtual memory basis, with page turning, revitalized hash (calculated) addressing, data integrity control, clustered records, and database keys.
- The network data model, with logical records that mapped transparently onto physical records in the virtual memory and logical O/M sets, was mapped transparently onto linked lists of physical records.[6]
- A data description language, with Data Description Language (DDL) statements that defined the types of logical records with their data, relationships, and constraints that could appear within the database.
- A data storage and retrieval language, with Data Manipulation Language (DML) statements could be easily integrated into a record-at-a-time procedural language, such as GECOM, Comtran, Fact, Cobol, or PL/1, which were also available. The record-at-a-time data-manipulation statements included STORE, RETRIEVE, MODIFY, and DELETE.
- An exclusive ''working storage'' area for each record type, providing the computer memory locations where IDS and the application programs, could exchange data under tight integrity controls.

User interface: A DDL/DML primer

The IDS user interface is characterized by the DDL statements that define the records' logical structure, as established by the

database designer, the DML statements by which the application programmer can manipulate the defined data, and the working storage areas within the application programs, which serve as the point of data exchange between the application programs and the IDS DBMS.

The command names of these statements provide a basic understanding of their functions. Each DML statement was inserted into the application program at the point where the desired record-oriented action was required.

Prior to executing a STORE statement, the application programmer would load the required data for the new record into its designated working storage area. When the STORE statement was executed, IDS would validate the data and then copy that data and create the new record in one of the pages of the virtual memory.[7] Eventually the page would be written back to the disk. If the record had been defined as a mandatory member of an O/M set, it would be transparently associated with its appropriate O/M set instance(s).

At an appropriate time,[8] IDS would make sure that the updated page was written back to its assigned location on the disk device. When it was time to modify an existing record, working storage was used in the same way.

When a RETRIEVE statement was executed, IDS would retrieve the designated record and copy the retrieved record's data content from the virtual memory page where it had been stored and write the data into the retrieved record's working storage area, based on its record type.

The content-addressing version of the RETRIEVE statement, using data that had been preloaded by the application programmer into the working storage area's primary key data fields, would retrieve the virtual memory page of the requested IDS record and then copy the retrieved record's data content into working storage.

In addition to the working storage areas reserved for the user data of each record type, there were hidden, database key areas that maintained the database key value for the most recently processed record of each record type. Furthermore, there was a database key area reserved for the program's current record. This would be the record that was the subject of a STORE, RETRIEVE, or MODIFY statement, which was most recently executed. For DELETE statements, the current record of the program and the current record of each member record type deleted would be set to null, signifying that there was no current record. The current record working storage areas were transparently accessed and updated when the DML statements were executed, but they could not be directly accessed or updated by the application programs.

In addition to the content-addressing version of the RETRIEVE statement, there were context-oriented versions that were sensitive to the current context of the application program within the database address space. Retrieval could be based on (1) the current record of a designated record type or (2) the next, prior, or owner record of a designated O/M set type, as related to that current record of that O/M set type. This context-oriented information was updated as each record was manipulated during the program's execution. In many cases, this cached data could be immediately reused rather than having IDS re-access a physical record in order to execute the next RETRIEVE statement.

The data description controlled STORE statement optionally permitted the physical clustering of related records, within a specified database page, or as close as possible at the time that the subject record was being stored. This could reduce the number of physical pages accessed from the disk to successively access the members of an O/M set. If a record being stored had been described as a member of two or more O/M set types and if the clustering option was desired, then a choice would have to be declared with respect to which O/M set was to control that clustering.

These options included the ordering of records in an O/M set, established at storage or modify time, in predefined sequences, eliminating the need for most of the file sorts that are required in traditional batch systems. The traditional sequential file sorting was eliminated because an IDS logical record could be maintained in multiple, independent O/M sets, based on their O/M set definitions.

The O/M set options provided the choice between *optional* and *mandatory* membership to each member record type in an OM set. Mandatory members were inserted in their particular O/M sets when their record was stored. Mandatory membership leads to the automatic removal and deletion of member records when their owner record is deleted. Optional members were associated and disassociated with their O/M set by the use of the INSERT and REMOVE statements.

Optional membership leads to the automatic removal of member records from their O/M set, when their owner records are deleted.

IDS implementation plans

At the end of 1961, the High Voltage Switchgear (HVSG) Department general manager Don Beeman decided not to go forward into the MIACS implementation phase. He was using his prerogative and duty to evaluate the potential gains and risks of the proposed new information system. Although his decision was disappointing, his support during the first year of the project gave us a tremendous start.

In early 1962, Bob Lewis, the general manager of the GE Low Voltage Switchgear (LVSG) Department, located in the same building in Philadelphia, undertook the challenge and responsibility for the detailed development of the MIACS application software and the collection of the necessary engineering and manufacturing planning data. Ernie Messikomer, from production, and Herb Fullerton, from engineering, lead the LVSG team. Bob's decision to go ahead with MIACS meant that the work on IDS could proceed.

Stan Williams concentrated on developing the MIACS database design, the application programs, and the transaction control parameters, in cooperation with the LVSG people. Homer Carney concentrated on building the virtual memory and physical device control. I concentrated on developing the IDS data definition structures and data manipulation functionality for the IDS logical record processing and on programming the Problem Controller (OLTP) operating system.

IDS 1961–1962 version

The original 1961–1962 version of IDS was fully integrated with GECOM. This original IDS version used two IDS/GECOM implemented system programs as preprocessors. The first of these two programs had been written to capture the IDS DDL source language statements describing a target IDS database. It read the DDL statements, validated them, and stored the results as metadata in an auxiliary IDS database (data dictionary).

The second IDS system program preprocessed IDS application programs and expanded the program's embedded IDS DML statements. The auxiliary IDS database was accessed at this time to determine how to expand each of the individual DML statements.

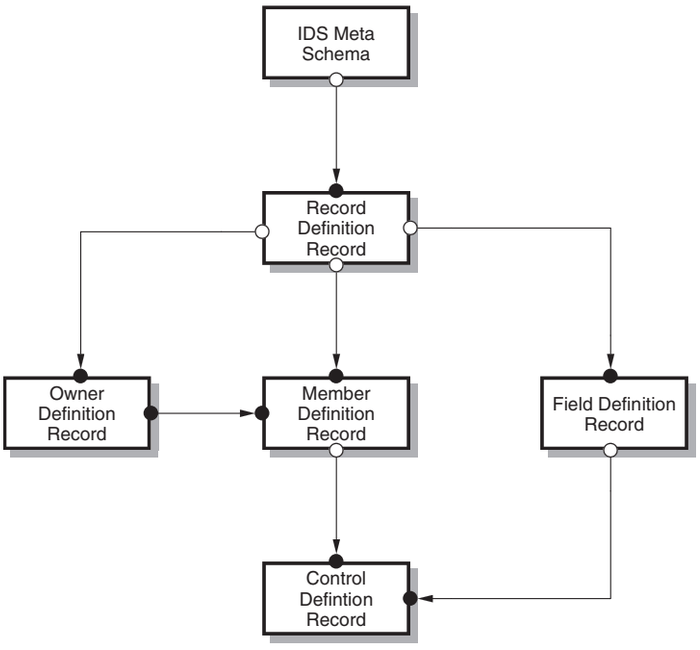

Figure 2. IDS meta data schema structure.

Each DML statement identified a specific DML command, possibly a command qualifier, and referenced the particular type of record (or type of O/M set) that had been defined in the DDL and previously stored in the auxiliary IDS database. Each of the IDS DML statements was translated and inserted back in line, as a sequence of GECOM and GMAP statements,[9] which could then be compiled, along with the balance of the application program, by the GECOM compiler and, finally, assembled by the GMAP assembler.

The IDS DML source language of the statement translated was preserved as the text of a GECOM `NOTE` statement and left in line so that both the original IDS statement and the generated GECOM and GMAP statements would show up together in the final program listing.

These two IDS system programs were the first IDS/GECOM programs[10] to be compiled, assembled, debugged,[11] and run successfully.[12]

Figure 2 is a data structure diagram that illustrates the metadata structure that was used to capture the IDS DDL statements and store them in the IDS meta database (data dictionary).

The **IDS Meta Schema** record type is a one-of-a-kind record with an O/M set of **Record Definition Records**. There is a **Record Definition Record** for every type

of record defined in the DDL. There is a **Field Definition Record** for every field defined in the DDL, and it is associated with the **Record Definition Record** that it characterizes. There is an **Owner Definition Record** for every O/M set defined in the DDL that is associated with the **Record Definition Record** for its owner record type. There is one or more **Member Definition Records** for every O/M set defined in the DDL, and each is associated with the **Record Definition Record** for its member record type.

The meta representation of the O/M set concept is embodied in the one-to-many relationship between each record of the **Owner Definition Record** type and one or more records of the **Member Definition Record** type. This O/M set provides the referential integrity that is inherent in the Network Data Model based DBMS.

The **Control Definition Records** served to record information about primary key fields and sort control fields for their respective **Member Definition Records**.

Figure 2 illustrates a more complicated example of a network structure than the first example shown in Figure 1. Records of two different record types, the **Member Definition Record** and the **Control Definition Record**, each independently serve as a member in two different O/M sets.

Test under fire

The first serious evaluation of IDS performance took place in the summer of 1963. GE's Internal Automation Operation (IAO) group was an in-house consulting organization that assisted various GE departments with designing and installing their computer-based information systems. IAO had been very negative about IDS from the time they first heard stories about it and its many generalized capabilities. They were just finishing up the installation of a dedicated, disk-based production control system for the GE Laminated Products Department (LPD) in Coshocton, Ohio, and it had been a long, painful slog. They liked the idea that IDS could give them shorter development times and a more reliable process for designing and installing new systems. Bill Helgeson of IAO said, ''If an IDS-based system was no worse than half as fast as their new system for LPD, IAO would use IDS in the future.''

So a test was planned, focusing on the parts explosion aspects of the LPD application. Actual LPD data was used in the test. Over two weeks, Stan Williams wrote the IDS/GECOM programs required to duplicate the LPD functions selected for the test. When the test was run, IDS ran twice as fast as IAO's custom-tailored application programs! The performance tests at LPD, sponsored by IAO, used the 1961–1962 version of IDS.

IDS had won, but Bill Helgeson had a second requirement. He said that the GE manufacturing departments could not afford to buy the 16-Kbyte word (20 bits) configuration of the GE 200 series computer, so ''IDS must be able to run on the cheaper, 8K word version.''

IDS 1963–1964 version

The 1961–1962 version of IDS, used in conjunction with GECOM, had to be scrapped and rewritten to operate in an 8-Kbyte word memory environment. The integration with GECOM and the IDS preprocessing strategy was abandoned because both required the larger, 16-Kbyte GE 200 series computers. Application programs were to be written in GMAP, the assembly language for the GE 200 series computers that could operate in the 8-Kbyte version.

This business requirement led to a complete reimplementation of the higher functional levels of the IDS DBMS. The auxiliary IDS database, previously created at precompile time, was replaced by a main memory copy containing the same DDL information (see Figure 2). The IDS users created them by hand, using GMAP statements and assembled them into each application program, to be interpreted at execution time. I created the new, interpretive implementation of the individual IDS DML statements as a set of IDS run-time subroutines. The working storage areas where the database information was exchanged between the application programs and the IDS functional code remained the same, although they too had to be declared in GMAP.

Homer Carney's virtual memory implementation, with its space management, page turning functions, buffer management, and all the physical device I/Os had previously been written in GMAP and was unaffected, thanks to the careful structuring of the functions in the initial design.

We were greatly concerned that the interpretive execution of the IDS statements would be slow compared to their inline coded counterparts. To our great relief, the application programs, interpreting their IDS statements, ran faster than the same

programs running the inline coded IDS statements. This was possible because the core memory space required to store the inline expanded IDS DML statements (1961–1962 version) was greater than the memory required to store the memory-resident IDS metadata plus the IDS functional subroutines (1963–1964 version), which interpreted that metadata. Thus the smaller, application program memory requirement allowed for the allocation of more page turning buffers. The larger number of page buffers allowed the application programs to build a larger, more efficient working sets of virtual memory pages (512 characters each) and thus required fewer disk reads and writes to process the same transactions.[13] This was an effect we had not anticipated.

I am told that all the current relational DBMSs have followed this paradigm of interpretive execution and software implemented virtual memories that IDS pioneered.

The 1963–1964 version traded off the ease of use of the 1961–1962 version to gain performance.

Problem controller (OLTP) OS

We had negotiated with the GE Missile & Space Vehicle (MSV) Department to write a transaction-oriented operating system (OLTP) to support the manufacturing control system's requirements. This was planned at the time when the 1961–1962 version of IDS was to be its companion, but then the IDS applications were required run on an 8-Kbyte GE 200. We were in real trouble when the special operating system written by the MSV arrived. It required more than 2-Kbytes of memory, which made its use impossible.

We quickly decided that we must use the IDS database to support the message store and forward requirements of our new transaction-oriented OS. We called it the Problem Controller.[14] It forwarded the next transaction to be processed, selected in business priority sequence.

Figure 3 is a data structure diagram that illustrates the data structure that supported the Problem Controller's message store and forward function. A simple hierarchical data structure containing four IDS record types was designed. From top to bottom, this structure contains the **System Record**, the **Problem Type Record**, the **Problem Record**, and the **Data Record** record types. The one-of-a-kind **System Record** has a set of **Problem Type Records** stored

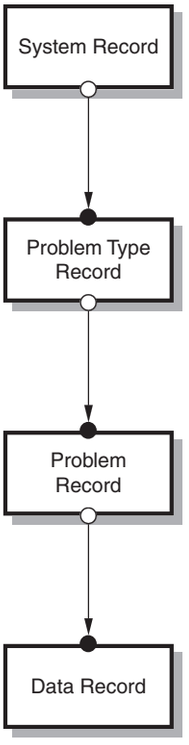

Figure 3. Problem Controller data structure.

in **Problem Type.number** (priority) sequence.[15] Each **Problem Type Record** has a set of **Problem Records** stored in FIFO sequence. They represent the transactions waiting to be processed. Each **Problem Record** has a set of **Data Records** stored in FIFO sequence. They contained the data required to support the transaction.

The Problem Controller's transaction-oriented records were stored in the same IDS database as the application data, competing for available record storage space. Most transactions, coming from the offices and the shops, were entered into the computer as card decks, each card deck with one **Problem Record** card and as many **Data Record** cards as necessary to provide the data input for the particular transaction. Other transactions were generated internally during the execution of manufacturing control application programs, to be selected and processed at some later time by the subsequent execution of the appropriate application program.

This transaction generation capability was valuable for several reasons. First, precedence relationships existed between some business transactions. The parts explosion problem is particularly sensitive to this issue because the explosion process must proceed level by

IDS Installation Horror Stories

The first few IDS installations created a few IDS horror stories that are hard to forget. One horror story regarding the International General Electric (IGE) was triggered by the unusual nature of one of GE's products. Among other things, GE manufactured and distributed diesel electric locomotives. For IGE, selling a locomotive and processing its order was not an every day event. The IGE order system was up and running, using IDS for several weeks before it encountered its first diesel locomotive order. The system had been handling orders at a good pace, but with this order, it slowed down and almost came to a halt.

What had happened? The difference was that each new locomotive came with a list of 2,000 spare parts, as additional line items on a locomotive order. Most new orders came in with a list of one to 20 line items, and occasionally an order came in with 100 line items.

When the input for the new orders and their line items arrived in the IDS database computer's card reader, it was already sorted in ascending, line item number sequence. The problem was created because the O/M set of an **Order** and its **Line Item** records had been declared in the DDL as ''order in ascending (**Line Item.number**) sequence.'' Therefore, the IDS subroutine supporting the STORE **Line Item** statement had to scan the O/M set from the **Order** record through all previously stored **Line Item** records in order to store a new **Line Item** record. To store the 2,000th **Line Item** record of an order, IDS had to scan from the first through the 1,999th **Line Item** record. The virtual memory pages were thrashing. The quick solution, after the situation was analyzed, was to change the declaration of the O/M set, for the **Order** record and its **Line Item** records to ''order last.'' Then, each new **Line Item** record was stored as the new, last **Line Item** record in the O/M set, after the previous, last **Line Item** record, which was noted in the context tables as the last member of the O/M set.

Problem solved!

The problem at the GE Wiring Devices department came about in another way. They had their order-entry system up and running, happily, but it seemed to run slower and slower over time. They could not understand the problem. They watched as it became slower and slower. Then panic set in and they called us in New York City for help.

After a brief discussion, it came out that they had written the programs to process new orders and store the requisite IDS records. However, they had not yet written the programs to delete orders that had already been shipped and invoiced. The computer got slower and slower as IDS searched harder and harder for empty space in the database to store new orders, until finally there was none left. Then it stopped processing new orders.

Delete the completed orders! Problem solved!

All the IDS horror stories that I heard had a happy ending. They all involved asking IDS to do something, without carefully considering what was being asked and the circumstances in which it was to be executed. Unreasoned requests almost always ended up consuming unreasonable time to execute. Experience was soon accumulated, and the same errors were rarely repeated.

level, from the top level to the bottom level, within the network of interrelated parts and assemblies where all the requirements at the first level must be considered before going to the second level. The second level must be processed completely with its direct requirements as well as requirements generated by the first level. Only then can level three parts and assemblies be considered and so on.

Transactions generating transactions was also valuable for breaking up complex procedures where the complete procedure required for processing some transactions was too big to be loaded and executed as a single transaction processing program. The complete procedure would have to be organized as a sequence of programs. The first program, when completed, would generate a new transaction, which started the second program. When the second program was completed, it would generate a third transaction, which would start a third program, and so on. A variant of this usage was to initiate a long running, batch program with a natural transaction and instruct that batch program to periodically shut itself down after first generating a follow-on transaction telling the batch program's next iteration where to take up and continue the processing, This permits higher priority transactions the opportunity to be processed without a long delay.

Given the limited nature of the I/O devices available to us in the 1961 to 1964 timeframe,[16] the Problem Controller could be called the first, or one of the first, general-purpose online transaction processing (OLTP) OSs.

IDS and Problem Controller software began being delivered within GE during 1964.[17] The initial delivery was made to International General Electric (IGE) in New York City in early 1964. The GE Wiring Devices department installation, in Providence, Rhode Island, came later in the spring of 1964. Both were order-entry systems, and both sites contributed to the collection of IDS horror stories (see the related sidebar).

The first non-GE installation of IDS and the Problem Controller (OLTP) operating system was at Weyerhaeuser in 1965–1966. It was an order-processing system that covered order entry, inventory control, shipping, and invoicing, where the ''online'' part was very real. A GE 235, located in the Tacoma, Washington computer center, was directly connected to a GE Datanet 30, a communications-oriented computer. The Datanet 30 was controlling a store and forward message system for more than 100 teletype machines, which were installed at lumber mills, warehouses, and sales offices located across the US and at the corporate headquarters. In addition, it was passing computer-oriented transactions on to the GE 235 computer for queuing, dispatching, and processing.

At the same time, the Problem Controller was passing outbound messages to the Datanet 30 for forwarding. Now the Problem Controller was unambiguously an OLTP system.

The GE 235/Datanet 30 computer site had no local card readers, card punches, or online printers. All the input and output devices (teletypes) were remote to the computer site.[18]

High-priority transactions at Weyerhaeuser, such as assigning a customer code to a new customer, were processed and responded to within a few seconds. Low-priority transactions might take overnight to be processed and have their response(s) sent on to their final destination(s). At one point, new customer orders were arriving at such a rate that the queue of unprocessed customer order transactions started to build up on Monday and they could not all be processed during that day or night. Some were still unprocessed when Tuesday transactions began to arrive. The length of the **Problem Record** queues grew each day during the week and only on Saturday or Sunday, with no new orders arriving, were all the problem queues finally emptied, to start afresh on Monday.

The Weyerhaeuser GE 235/Datanet 30 configuration and application system was designed and implemented by a joint team of Weyerhaeuser and GE IAO people.

The Problem Controller (OLPT) operating system is an important story from two points of view. First of all, it was a success as an OLTP operating system. Second, it showed the breath of applicability of IDS beyond application-oriented databases into operating system support.

Enhancements to IDS

In 1964, Clark Karcher and Bob Blose at IGE made a couple of significant additions to IDS. They added a recovery and restart (R&R) capability, which made IDS much more robust. They assigned a magnetic tape drive to the IDS configuration and began writing before-change and after-change images of virtual memory pages to the R&R tape. A special mark was written to the R&R tape prior to the beginning of each transaction. Periodically, when no transactions were being processed, complete copies of all the IDS database pages would be copied from the disk to a second magnetic tape file.

If a transaction was about to update a virtual memory page, that page's before-change image was first copied to the R&R tape before being released for access by the transaction processing application program. If there was a program or computer failure during the processing of that transaction, the database state would be restored to the condition existing prior to the beginning of the aborted transaction by recovering the before-change images from the R&R tape and writing them back to the disk.

If a virtual memory page had been updated on the behalf of a transaction and had been written back to the disk drive, it's after-change image would also be written to the same R&R tape. If there was a disk hardware failure, the database's content could be reconstructed by restoring the most recent complete database dump on the disk drive and then using the most recent after-change image of each page on the R&R tape to completely reconstruct the database to the condition existing at the beginning of the transaction that was being processed at the time of the disk failure.

IGE also introduced ''inventory pages'' into the IDS disk storage files. Each inventory page represented a range of virtual memory pages. It recorded, with a single character (six bits) representing each virtual memory page, a compressed profile of the available space within that page. This made it possible to increase the utilization of space in each of the virtual memory pages without penalizing performance. Sequential page scans, to find a page with sufficient available space to store a new record, were no longer required. If the optimum placement page did not have enough space to hold the new record, then a search for available space was made in the inventory pages.

GE IAO also added performance-oriented updates in 1964. First, they reorganized the format of the virtual memory pages to include a 64-word pointer array, representing the maximum number of records that could be stored within a page.[19] Each pointer in the array, indexed by the line number of the desired record, pointed to the location of that record within the page. This let the IDS Block Controller access individual records within a page without having to scan, record by record, all the records in the page to find a particular record. This also reduced the garbage collection task within a page as records were deleted. Garbage collection could be delayed until the available space within the page needed to be consolidated in order to store a new record. Originally, all the records in the page were stored consecutively, in line-number-within-page sequence, and garbage was collected every time that a record was deleted.

Next, Irv Burch of IAO suggested a modification of the **Control Definition Record** used for content addressing based retrieval to allow record selection based on a range of values rather than a single value. This let the user define and create hierarchically structured index records for content-oriented record retrieval. These would be user defined and created as single level, or multiple level, indices based on user defined record types and O/M set types. This was useful in maintaining large O/M sets in a sorted sequence because it reduced the number of physical record accesses required to (1) to find the correct insertion point for a new record being inserted into the O/M set or (2) retrieve an existing record by its primary key value without parsing the entire O/M set looking for an equal value situation.

IDS' long-term contributions

The 1963–1964 interpretive IDS version was installed in a dozen GE departments between 1964 and 1966. IDS was distributed to interested GE 200 series computers customers at no cost.

In 1965, the ANSI/X3/SPARC List Processing Task Group (later renamed the Data Base Task Group) began an industry-wide effort to develop a specification for a DBMS that was modeled on the GE IDS product, with GE's permission to copy it as closely as they might like.

Later, after the GE 400 and GE 600 interpretive versions of IDS were developed (1966) at the GE Computer Department in Phoenix, Arizona, IDS installations continued to double almost every year. IDS was distributed to the users of the GE 400 and GE 600 series computers as part of the product package.

This growth continued with GE's customers and with Honeywell's customers after the GE Computer business was sold to Honeywell Information Systems in 1970.[20] IDS helped to sustained GE, and subsequently Honeywell, much longer than most industry experts had expected.

The GE 600 series IDS became the Honeywell 6000 series IDS and later became Bull IDS. Bull still maintains a strong position in the online banking business in Norway, based on IDS usage, and Internet search results indicate that the company is still offering courses on IDS.

In 1964, Dick Schubert of B.F. Goodrich Chemical received the source language program for IDS from Clark Karcher of IGE. It was delivered in two boxes of GMAP assembly language cards (4,000 cards) for the GE 200 series computer. Later, Goodrich Rubber translated it to run on the IBM 360 mainframe, resulting in the Integrated Data Management System (IDMS). Goodrich licensed IDMS rights to John Cullinane of Cullinane Database Systems (later renamed Cullinet). Cullinet was purchased in 1989 by Computer Associates (CA) for CA stock valued at $320 million.[21,22]

A 2006 CA report stated that there were currently 1,000 IBM mainframe sites worldwide running IDMS. The largest was the British Telecom (BT) site that was running an OLTP system to handle the business end of all of its UK telephone service. It is allegedly the second or third largest/busiest OLTP site in the world. I spent a half day with BT in London in October 2008, and the IDMS system received a strong bill of health. A brief summary of BT's IDMS OLTP system showed they had 12,000 gigabytes of direct access data, processed 295 million transactions per day, logged on 45,000 different users daily, and had 1,000 online programs consisting of 3 million lines of Cobol code.

MIACS roll out

The MIACS application, with IDS and the Problem Controller OS, became operational at GE LVSG in 1965 and later at the GE Insulator Department. MIACS initially ran on the GE 200 series of computers. Later, MIACS was reprogrammed at LVSG to run on the much larger, faster GE 600 series computer.

In 1966, Stan Williams took the MIACS product to the GE Computer Department in Phoenix and built a packaged version that was marketed as the GE Production Inventory Control System (GEPICS) for the GE 600 series computers. After the GE/Honeywell merger in 1970, GEPICS was marketed as the Honeywell Manufacturing Information System (HMIS).

Conclusion

This paragraph, published as a portion of my 1973 ACM Turing Award lecture, ''The Programmer as Navigator,'' seems as relevant in 2009 as it did 36 years ago.

> The Integrated Data Store (IDS) 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. A new basis for understanding had become available in the area of information systems. It was 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.[23]

References and notes

1. The IBM Bill of Material Process (BOMP) was available at approximately the same time. BOMP was an application package, designed for a specific purpose with predefined record types. It was not a general-purpose DBMS, capable of supporting a range of to-be-specified applications. Bill of material processing was one of the many manufacturing planning and control tasks implemented, using IDS, for the MIACS system. So was the Problem Controller (OLTP) OS.
2. The choice of the variable ''M'' word depended on which organizational level we were addressing.
3. C.W. Bachman, ''Data Structure Diagrams,'' *Data Base,* vol. 1, no. 2, Summer 1969.
4. The Bachman Data Analyst product was one of the CASE products created and marketed by Bachman Information Systems, which I founded in 1983 and helped to operate until 1998 when it was acquired by Sterling Software, which was subsequently acquired by Computer Associates (CA) in 2000.
5. Report generator languages and relational database languages like SQL are considered to be non-procedural languages.
6. An O/M set instance consists of one owner record and an ordered set of zero, one, or more member records. An O/M set type is defined by a set name, an owner record type, one or more member record types, and logical constraint declarations. A record type can be defined as the owner record of zero, one, or more O/M set types and, independently, as a member record of zero, one, or more O/M set types. It could not be declared as both an owner record and a member record of the same O/M set type.
7. IDS was the first user of virtual memory as an organizing principle for databases. The database was organized as a set of pages. Each page was organized as a set of lines (records). The Ferranti Atlas computer was the first user of virtual memory as an organizing principle for its computer programs. The control of virtual memory for program storage has been transferred to hardware control while control for database storage has remained to this day under database software control. The requirements are different.
8. IDS used a longest-inactive algorithm to age virtual memory pages and set the time of their being written back to the physical disk, or being overwritten if their content has not been changed.
9. GMAP assembly language statements were generated to access those computer machine functions not supported in the GECOM language.
10. In early 1963, only the two IDS system programs had been written to use and test IDS. The IDS functionality was tested, but the number of records stored and retrieved was too small to get any feeling for performance. Furthermore, these programs were written and tested at a time when we were waiting for the first GE disk file to be shipped to our test site in New York City. Homer Carney temporarily modified his physical I/O system to cause two magnetic tape drives to simulate the reading and writing of data pages to/from a disk file.
11. When one considers that each debugging run required a new GECOM compile and subsequent GMAP assembly, careful design and a lot of desktop debugging was essential. Considering the time to prepare, compile, make a test run, and then analyze the result, this test cycle was essentially a once a day operation. Furthermore, it required a trip from New York City to Schenectady, New York, to gain access to GE 225 computer time. My memory tells me that we got through the bootstrapping phase in a single trip to Schenectady.
12. The fact that the two IDS system programs were really just IDS/GECOM programs created a dilemma that was solved by a bootstrapping operation. In the first step, the IDS DML statements in both of the IDS system programs were manually translated into the equivalent GECOM and GMAP statements and then compiled and

assembled in the normal manner. The first versions of these two IDS system programs were debugged, making the necessary changes. When they were both operating correctly, the source language code for these two IDS system programs was reprocessed to machine translate their imbedded IDS DML statements into the appropriate GECOM and GMAP statements. The machine translated GECOM and GMAP statements were validated by comparing them with the now debugged, manually translated GECOM and GMAP statements to assure that nothing had gone wrong.

13. The 1963–1964 interpretive version of IDS reserved 4,000 words for the residence of IDS and the Problem Controller. The 4,000-word balance of the 8-Kbyte memory machine was shared by the application program and its page turning buffers. The larger 16-Kbyte memory machine had three times as much memory to be shared by application program and page turning data buffers. No one bought the 8-Kbyte configurations any more!
14. In the early 1960s, the GE Computer Department advertised themselves as ''Problem Solvers.'' We seized on the word ''problem'' and thus named our transaction-oriented OS the Problem Controller. Manufacturing control transactions were equated as being problems. Manufacturing transactions can legitimately be considered to be messages. The Problem Controller was a store-and-forward message system. It forwarded messages (transactions) for processing in a business-oriented priority sequence.
15. The subsequent Weyerhaeuser IDS and Problem Controller installation reminded us of an old system design rule that we had forgotten. ''Never use a data field with business meaningful values as a primary key. When the meaningful value changes, the primary key usage will be compromised.'' Weyerhaeuser working with IAO solved this problem by adding a second O/M set between the **System Record** and the **Problem Type Records**. The original O/M set was sorted by the **Problem Type Record.code** and controlled the access to existing **Problem Records**. The new O/M set was sorted by the **Problem Type Record.priority number** and controlled the problem type dispatching sequence. Some of these priority values were modified from day to night for business reasons and the second O/M set was reordered accordingly.
16. The standard configuration for a GE 200 series computer was a card reader, card punch, high-speed printer, and a console typewriter.
17. Two boxes of IBM punched cards, totaling 4,000 80-column cards, each holding one GMAP assembly language instruction.
18. The absence of any local peripherals drove the computer room operators crazy. They could see the console lights blinking on and off but could not tell whether the computer was in a loop or operating normally. Eventually, a console typewriter was connected. Hourly, the list of queued transactions was typed out so the operators could check the new list to see if that list was different from the list printed the prior hour.
19. The GE 200 series IDS allocated 20 bit word to the recording of a database key. The high order 14 bits (values from 0-16,183) recorded the page number of the virtual memory page where a record was stored. The low order 6 bits (values from 0-63) recorded the line number that was assigned to a record within a page. The database key value of zero was reserved to represent an invalid or null database key.
20. A good portion of the price Honeywell paid for the GE computer business was in Honeywell stock. That turned out to be a profitable deal for GE because the Honeywell stock appreciated considerably by the time GE was allowed to sell it.
21. Subsequently, the market value for that number of CA shares attained the value of $3 billion.
22. See J.M. Kruntorad's article ''History of the CA IDMS Database Management System'' in this special issue for the IDMS Goodrich/Cullinet/CA story.
23. C.W. Bachman, ''The Programmer As Navigator,'' *Comm. ACM,* vol. 16, no. 11, Nov. 1973.

Charlie Bachman is noted for his work on 9PAC, the Integrated Data Store (IDS), network data models, data structure diagrams (Bachman Diagrams), the Codasyl Data Base Task Group Report, the Three Schema approach to data independence, role data model, the ISO/TC97/SC16 Reference Model for Open Systems Interconnection, partnership set data model, and reverse and forward engineering (AD/Cycle). Bachman was awarded the ACM Turing Award and elected a distinguished fellow of the British Computer Society for his pioneering work in the development of DBMSs. Contact him at Charlie@Bachman.com.