A GENERALIZED DATABASE SYSTEM SIMULATOR BASED ON THE DATA-INDEPENDENT ACCESSING MODEL I

Lowell S. Schneider and Thomas W. Connolly

Martin Marietta Corporation
Denver, Colorado

ABSTRACT

The Data-Independent Accessing Model I (DIAM I)[1] has demonstrated a multifaceted applicability to the study of database problems. The literature contains accounts of DIAM being applied to the description of existing database systems (SDDTTG)[2], to the description of relational implementations[3], and as a conceptual basis for new database systems[4]. Such broad applicability makes the DIAM particularly attractive as the basis for a simulation model. A DIAM simulator could be profitably applied to:

1) The comparison and selection of existing database systems;

2) The design and evaluation of new database systems and database management techniques.

Responding to these motivations, a DIAM-based simulator has been designed and implemented. A technique for specifying the representation-independent characteristics of such a simulation was developed and reported previously[5] but the other details of the simulation have not been published. Therefore the purpose of this paper is to describe these details. The emphasis is not on the DIAM concepts themselves, which are well reported, but on their utilization and implementation for simulation purposes.

The paper begins with a discussion of the purpose and objectives related to database simulation and how they affect simulator design. Next a conceptual overview of the simulation facility is presented to provide a perspective for the ensuing discussion of the more interesting functions and techniques as implemented. These are:

1) Discrete query generation from a quantitative profile;

2) Search path enumeration, scoring, and selection;

3) Simulation of path traversal by an access engine.

The paper concludes by discussing what information can be obtained from the simulation and to what decisions the information can be applied.

INTRODUCTION

The purpose of this paper is to describe a discrete-event simulation of the functions of the general class of database management systems (DMSs). To represent a broad class of such DMSs, the simulator is based on an underlying canonic model--the Data-Independent Accessing Model I (DIAM I)[1,6,7]. It was conceived as a tool to aid the study and application of DMSs and allows the simulation of:

1) A user's application, in terms of its information structure and traffic rates;

2) A candidate implementation of the application in a DMS, reflecting the proposed implementation of data relationships and recognizing any restrictions imposed by a specific DMS;

3) A candidate host system representing the pertinent aspects of the planned host computer and its operating system.

A user may employ the simulator to conduct studies relevant to the choice and use of DMSs. Typical studies of this nature might include:

1) Comparison of the operating performance of two competing DMSs for the same application;

2) Comparison of the operating performance of various implementations of an application using options available within a single DMS;

3) Comparison of operating results based on differing host system configurations being considered;

4) Studies to enhance the user's knowledge and familiarity with DMS techniques.

The simulator has been implemented using the DIAM concepts as a foundation. The design considerations necessary to apply these concepts to a functioning simulator are discussed and the nature of the output reports from the simulator are discussed and illustrated. Analytic studies utilizing the simulator are now in progress.

SIMULATOR ARCHITECTURE

The simulator consists of subsystems, each subdivided into a number of modules in a hierarchical fashion. Four of the principal subsystems correspond generally to the four levels of the DIAM. These are:

1) Information-based model - This subsystem generates queries representative of the application under study and maintains the data population statistics;

2) Structure-based model - This subsystem accepts the queries as Representation-Independent Accessing Language (RIAL) statements and uses definitions of the implemented access paths to produce Representation-Dependent Accessing Language (RDAL) statements;

3) Procedure-based model - This subsystem accepts the RDAL and produces a sequence of input/output accesses based on the definition of how and where the access paths and data are stored;

4) Host model - This subsystem represents the logic of the host computer system, including its peripherals and its operating system, as it pertains to the calculation of response time and resource usage in processing the I/O access requests.

Other modules carry out various control and specialty functions on behalf of the simulation. These are grouped for discussion in the executive subsystem, which accepts control from the operating system at execution time. It contains modules to read and store the simulation data, configure the simulation, control the experimental runs, and produce the required output.

The entire simulator has been programmed in the Fortran language with very few deviations from the ANSI Fortran standards. The source code has been implemented on CDC 6000 and Univac 1100 systems; with minor modification it can be implemented on other computer systems. It operates in a batch environment.

THE SIMULATOR FUNCTIONS

The principal functions of interest performed by the simulator include the generation of queries in RIAL, the translation of these to the RDAL, the translation of the RDAL statements to input/output requests, and the simulation of the input/output request execution. There are other supportive functions such as data logging, data analysis, and report generation but these are not discussed.

GENERATION OF QUERIES

The first step in the simulation is to generate the queries whose processing is to be simulated. These resemble queries that might be posed to a truly representation-independent generalized data management system. They include both updates and retrievals, both of which may be complex in form and content. The queries are generated in a fashion that is in statistical agreement with the input profile specification called Quantitative Data Description[5]. This operation can be performed either in a self-contained mode to produce a file of RIAL queries or in an interactive mode with the other blocks of the simulation wherein a new query is generated as needed for the simulation.

RDAL TRANSLATION

Each RIAL statement must be translated into a set of RDAL statements. The latter are the calls for data and pointers that lead to the specific data requested. This set of RDAL statements is the result of a process that uses the access path definitions to map the RIAL (which, as the name implies, is independent of the data structure established for the application) into a selected sequence of RDAL. Selecting the sequence for a specific data element involves identifying the most efficient path from an entry point via intermediate paths to the data desired. When a path has been selected, its traversal is expressed algorithmically in RDAL, which may include iterative loops. At this level of the simulation it is also necessary to determine (from the statistical profile) how many instances of each path must be traversed and how far each path must be traversed to find the data or desired pointer.

RDAL TO I/O REQUEST TRANSLATION

For each RDAL statement, the following steps must be performed:

1) Analyze the definitions of how the path is implemented (e.g., chains, contiguity, etc) and where the data are stored (e.g., which file);

2) Request (for timing purposes) the implied data retrievals;

3) Test for overflow and iterate through 1) and 2) for alternatives as necessary;

4) Test for inclusion of the desired data (or pointer) in the buffer load retrieved and iterate through 1), 2), and 3) if necessary;

5) Perform any additional reads or writes.

RETRIEVAL PROCESS SIMULATION

The processing of each I/O request is simulated according to the configuration and dynamics of the host computer system and its operating system. This function determines, for each request, the resulting response time durations and resource usage.

QUERY GENERATION

An important aspect of the simulation is producing sets of queries that typify the application being considered. There are both conceptual and implementation hurdles to overcome. The description

that follows illustrates the considerations for design and the resulting implementation approach.

It was recognized early that at least two views of traffic activity existed and that these two somewhat competed. The views were:

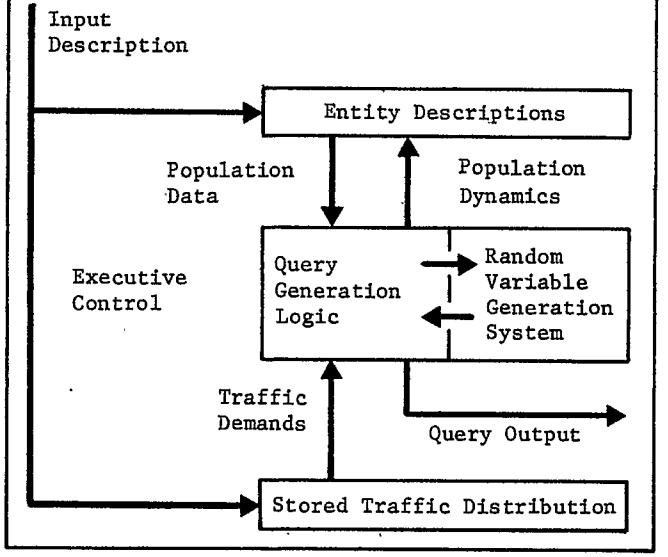
1) The generation of queries based on purely probabilistic rates associated with the various real-world entities that the database describes;

2) The generation of deterministic queries emanating from "sources" (such as computer terminals or batch programs) at prespecified rates.

The implemented Query Generation Subsystem (QGSS) accommodates queries attributable to both concepts; it permits the mixing of queries from both specifications and the transition from one to the other.

The interaction of these two viewpoints is depicted in Figure 1. The entity set description is the fundamental element of the information-based model. It includes the basic declaration of the entity sets and their interrelationships as appropriate to the application. This information is supplemented for simulation purposes by inclusion of statistics that reflect the entity populations and the anticipated rates of activity--both innate to a specific entity and associative among them.

FIGURE 1
Simplified Information Flow for Query Generation



The traffic distribution represents statistical demands for specific services. These demands are over and above those that may be specified for each entity via the entity description. The traffic distribution accommodates the modeling of certain types of demands that are best considered from an external viewpoint. These types of demands might include intermittent use of a terminal, scheduled batch operations, and special transactions that, while important, are unlikely to arise via the entity activity statistics.

The query generation logic produces a sequence of queries that reflects the activity implied by both the entity activity and the demands of the stored traffic distribution. As illustrated in Figure 2, an explosion of the query generation logic block in Figure 1 reveals the following major components:

1) The traffic time generator that determines the next time at which a prespecified query will arrive;

2) The entity time generator that determines the next time at which each entity will be affected by an event;

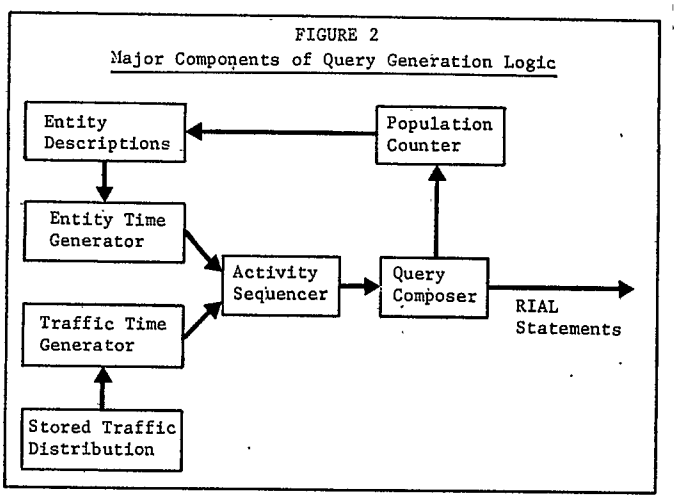
3) The event sequencer that merges the activities of 1) and 2);

4) The query composer that generates the RIAL statement to be processed;

5) The population counter that records the dynamic effects of a query on the entity populations and interrelationships.

Each of these is now discussed in more detail.

FIGURE 2
Major Components of Query Generation Logic



TRAFFIC TIME GENERATOR

This generator (Fig. 3) determines the next time at which a query will occur and what query it will be. This is done by inspecting every traffic source defined, examining its interquery rate, and predicting the next time at which a query will occur. When the time is determined, a description of the query is selected by sampling from among all the possible query descriptions from that source. The description is then placed in the traffic queue.

ENTITY TIME GENERATOR

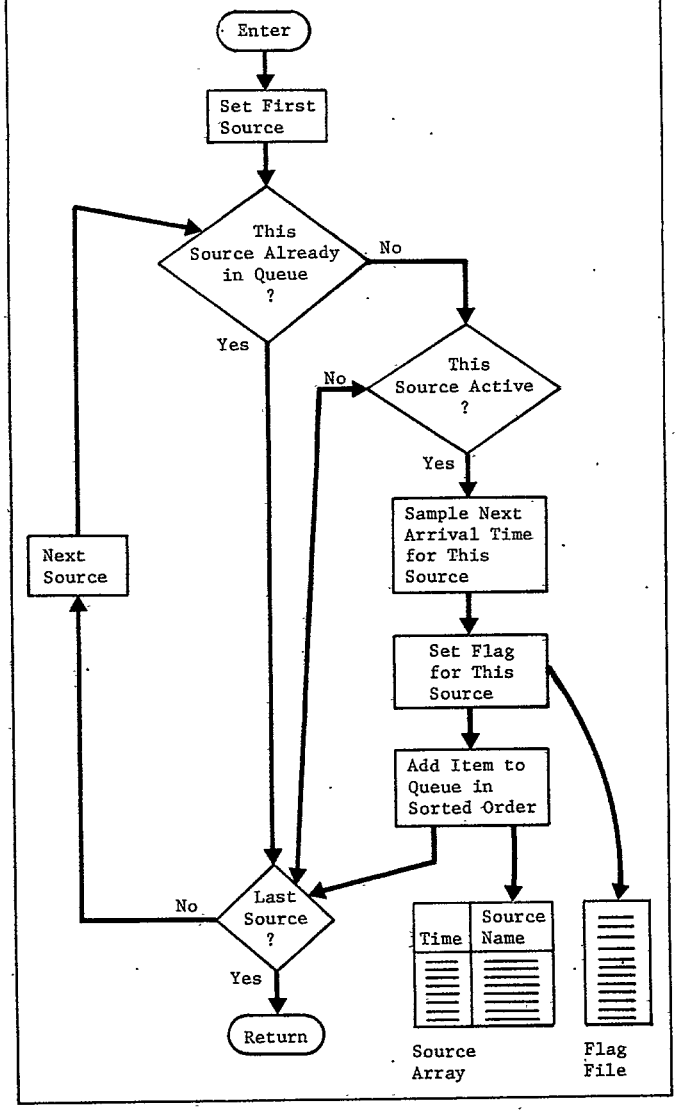
This generator (Fig. 4) operates analogously by inspecting all of the entity descriptions and examining their rates of arrival, departure, and change. The next time at which each event will occur for each entity is predicted and the predictions are placed in the entity queue.

EVENT SEQUENCER

This sequencer then examines the queues each time a query is needed and determines the time and nature of the next event to be simulated.

FIGURE 3

Traffic Time Generator



QUERY COMPOSER

The process of query composition depends heavily on the concepts of RIAL Statement Description (RSD)[5]. Recounting briefly, RSD is a language for the partial specification of RIAL queries. Its syntax is an extension to the RIAL syntax to allow nondetermination of any or all terms in a query. It is a convenient technique for specifying patterns of similar queries that are known to occur at certain rates. The RIAL syntax and the RSD extension are illustrated in Figure 5.

When a query arrives at the composer, it is in the form of an RSD statement. Its content can range from completely specified

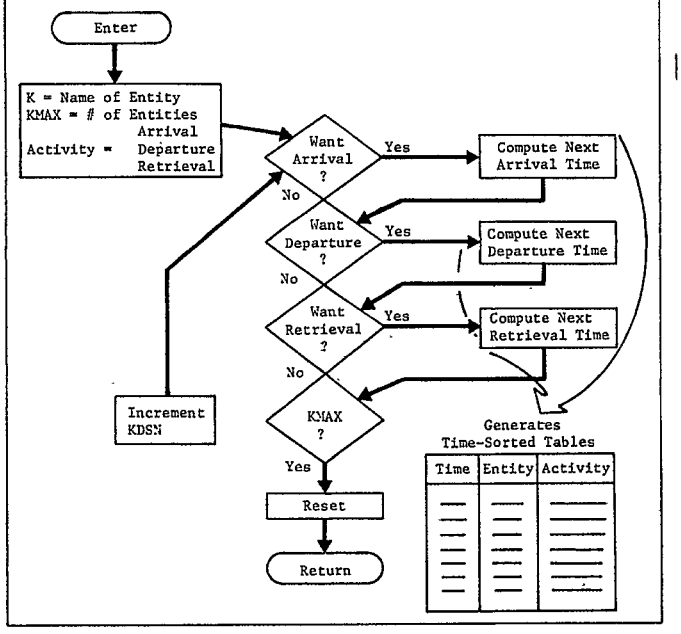
```
GET S1 OF CAR WHERE COLOR = 1 AND
MANUFACTURER = 1
```

to completely unspecified

```
F S1 OF ARG.
```

FIGURE 4

Entity Time Generator



The nesting of <argument's> within <value's> is also permitted.

For each argument, the <object> phrase and the <qualifier> phrase is constructed according to the RSD and the entity statistics that describe the likelihood of an entity being referenced by the query. The logic for <object> construction is diagramed in Figure 6; <qualifier> construction is analogous.

The resulting queries are fed to succeeding blocks of the simulation and are also logged for examination or reuse. The query output format is that of the RIAL syntax.

Each query has appended to it two information items for use at later stages of the simulation:

1) A query identification number (ID) - This is a sequential number (base 10) starting with the number 1 for the first query of a simulation run;

2) A query time tag - This represents the time at which the user or application program presented the query to the DMS.

POPULATION COUNTER

The population counter analyzes the resulting queries to determine what effect, if any, each has on the static populations. The types of effects recognized are:

1) Entity population increases (decreases) due to the presence of an add (delete) query;

2) Entity population increases (decreases) due to the addition of an attribute value that was previously nonexistent (the deletion of the last occurrence of an attribute value);

FIGURE 5

RIAL/RSD Syntax Diagrams

RIAL	RSD
<statement>:: = <function>{<set name> OF <argument>}	-
<argument>:: = <object> WHERE <qualifier>	ARG * <positive integer>
<object> :: = <entity name>{ / <attribute name>}	OBJ * <positive integer>
<qualifier>:: = <condition>{ OR <condition>}	QLF * <positive integer>
<condition>:: = <quality>{ AND <quality>}	CND * <positive integer>
<quality> :: = <attribute name> = <value>	QLT
<value> :: = <set name>/<attribute name>|<positive integer>	-
<entity name>:: = <name>	DSN
<attribute name>:: = <name>	RN
<set name> :: = S <positive integer>	-
<name> :: = {<character>}	-
<character>:: = <positive integer>|<alphabetic character>	-
<alphabetic character>:: = A|B|C|.....|X|Y|Z	-
<positive integer>:: = 1|2|3|.....	R
<functions>:: = A|D|R|C	F

3) Interrelationship changes as a result of any of the above or a change query.

These dynamics are accumulated in the entity description and accounted for as subsequent query generations.

TRANSLATION TO RDAL

As summarized earlier, the next subsystem of the simulator is required to accept the queries in RIAL language and translate them into RDAL statements. The latter gives instructions for traversing the internal data structure. A major conceptual problem arises if more than one traversal will yield the correct result. When this occurs in a real database, the programmer using the DMS decides which traversal to use. To give each DMS being simulated its "best shot," it is therefore necessary to simulate a "smart programmer" since choices can often differ in performance by several orders of magnitude. The net effect of this problem is that the simulator must also be an optimizing automatic programmer. This is an enormously complex operation that can only be cursorily discussed in the confines of this paper, but it basically involves two steps:

1) Search path enumeration - This activity determines and enumerates the set of traversals that allows all data required by the query to be found;

2) Search path selection - Of the set of workable traversals found in the preceding step, one must be selected. In the simulator this is accomplished by first assigning a "cost" to each path segment based on an expected value of the number of accesses required. Cost determination can and often does involve calling all remaining levels of the simulator for each candidate.

The determination of any one such traverse is based on the technique of searching the catalog of access path descriptions to determine which of those access paths each desired data element is on and then sequentially determining the paths those are on and iterating until one that is an entry point is found. The entire process is repeated for each case in which a data element or path is on more than one path. For each route found "outward" from the data element to an entry point, the reverse route from entry to the data is recorded as a candidate traversal.

The process of finding all available paths is applied to all data demands arising from a single RIAL statement.

It is quite possible that certain segments of a path to one data element may coincide with those to another data element. Therefore, the alternative paths resulting from one demand are compared with those from other demands in the same query and consolidated paths that utilize common segments only once are defined. The score attached at the consolidated path is reduced (improved) from the original total for the two independent paths. The consolidation is extended to include successive numbers of paths, which might include as many branches as there are demands in the RIAL statement.

When the preferred path has been identified and selected, it is expressed in the RDAL. This language, diagramed in Figure 7, expresses the detailed accessing steps necessary to retrieve and/or update the data required to satisfy the RIAL.

FIGURE 6

Object Phrase Construction
(Qualifier Phrase Is Similar Operation)

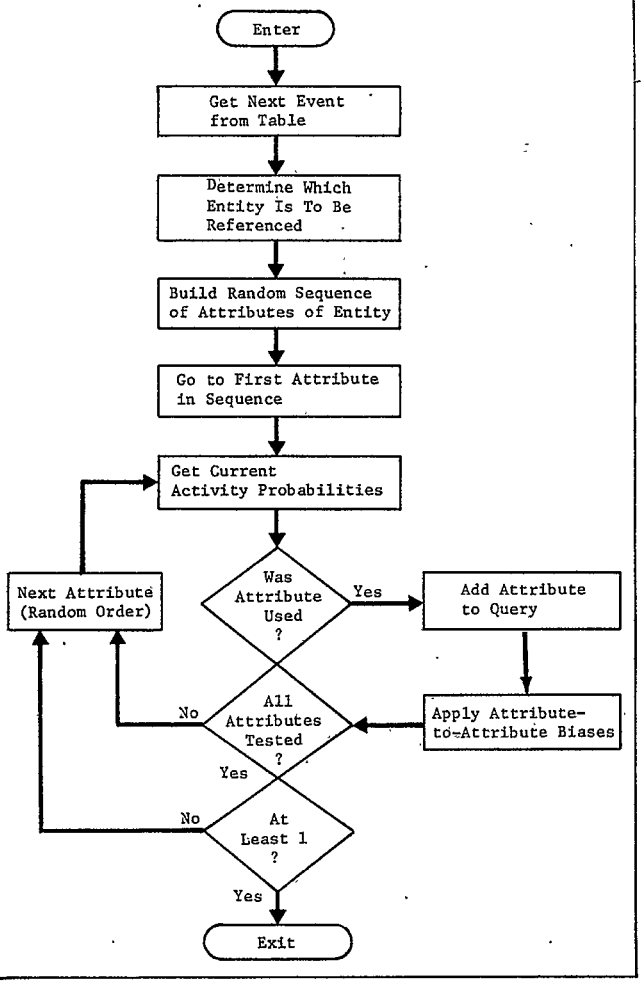


FIGURE 7

RDAL Syntax

```
<statement> :: = <function> <operand>
<function>  :: = GET|WRITE
<operand>   :: = <path name> [<modifier>]_0^1| <attribute name>
<path name> :: = X(30)
<modifier>  :: = SAME|ALL|FIRST|LAST
                 NEXT <positive integer>|
                 PREVIOUS <positive integer>
<positive integer> :: = 1|2|3|.....
```

The RDAL expresses not only the route to the data but also the number of accesses required to follow each path to the desired data or pointer and the number of data elements found at the desired location. In an actual database application, an inquiry such as "List all employees whose age is 33" will result in some number of names returned. The simulation, however, produces only a count of the number of instances found. This is done by inspecting the statistics for EMPLOYEES vs AGE to obtain a numeric sample value representing the number of instances of EMPLOYEE corresponding to an AGE qualification.

RDAL-TO-I/O TRANSLATION

The RDAL-to-I/O translation is a simulation of the logic of the DMS in responding to the data requests. When an RDAL instruction arrives, it references a specific path (or data element) name. For each path and data element, one or more definitions for how the path or data element has been encoded in storage (e.g., chains, contiguity, pointer array, etc) is given. Typically there are three:

1) Primary encoding - Describes the encoding of the path or data element in its primary area of storage;

2) Secondary encoding - Describes the encoding of the path or data element in its secondary (overflow) area of storage;

3) Transition encoding - Describes the encoding within the primary area of storage that indicates the path is continued in secondary storage and gives directions for getting there.

For the path or data element referenced by the instruction, the process routinely looks up the primary encoding definition. Among the information provided in this definition is the frame of storage (meaning file, area, partition, etc) in which the path or data element resides. Frames carry their own definitions, including their assignments to physical devices. These definitions are also used during this stage of the process.

OVERFLOW TEST AND PROCESSING

In this simulation overflow is considered with respect to each frame. The probability of overflow arising from an addition depends on the space management strategy employed by the frame. If a frame is composed of compact physical sequences, an attempted addition that would exceed 100% of the space in the frame results in an overflow. If the frame is composed of sparse data, overflow can occur even if the frame is not full. The strategy may call for linkage to a secondary frame in this event. For retrievals, the probability of having to extend the search to a secondary frame depends on the density of the frame.

The simulation maintains a counter of the number of instances of each encoding. These are totaled on a running basis for each frame (which may include several heterogeneous encodings). The current population of the frame divided by its total space establishes a density as a dynamic variable. A function of this density is taken to establish a probability of overflow in accordance with the pertinent cases above.

If probability dictates that a particular retrieval must look in the overflow region for

data, the secondary encoding definition is utilized. This in turn refers to a particular frame that has been assigned for overflow storage. If an addition is to go to overflow, an access to place a transition in the primary frame is simulated and the writing of data to the secondary frame is also simulated.

I/O REQUEST GENERATION

This section creates the I/O commands to the host simulation. The I/O command sequence expresses the execution of the selected traversal. Each I/O command contains the following information:

1) Operation to be performed (CONNECT, STATUS, SEEK, READ, WRITE);

2) Device type and number (e.g., DISK01=3330, TAPE02=3405, CELL03=3850);

3) Device address (either a constant displacement from current position or RANDOM);

4) Amount of data to be transferred (in bits);

5) Time tag as appended by the query generator.

EFFECT OF BUFFERING/PAGING

The number of I/O accesses will obviously be strongly affected by the amount of central memory buffer space available. It will also be affected by the amount of data transferred for each I/O. Another effect is introduced by choice of buffer management strategy--what criterion is used in making a choice of which portion ("page") of the buffer to be overlaid. For example, when the statement is GET NEXT, it implies that the next item in path order is desired. If the items are sequenced by concatenation, there is a reasonable likelihood that the next item desired is already contained in the buffer load previously obtained. If the operand of the instruction is SAME, this implies that the data element referred to is identical to a previously retrieved item, and the request is submitted to the buffering model to determine if the data item is still in fast memory.

OTHER CONSIDERATIONS

Several other factors that affect database system performance were intentionally not discussed. Among the more obvious of these are journaling/checkpoints, space reclamation (other than intra-frame collection), authorization, integrity checking, etc. Somewhat less related but certainly important are the DBMS program library, the data dictionary, and any supplemental system record-keeping activities (audit trails, satistical reports, etc). The reason these were not considered specifically is that they are all, to a degree, representation-independent functions; i.e., there are as many different ways to accomplish these functions as there are ways to implement a database. For example, if we choose to keep track of previously occupied space, we have at least the following choices:

1) A chain through spaces in a file;

2) An inverted index of spaces in a file;

3) A chain through related spaces in different files;

4) Combinations of 1), 2) and 3).

Analogously, if we choose to journal transactions, the resulting collection can be implemented under a variety of schemes, depending on how we wish to eventually access/update the collection. To accommodate the full spectrum of alternatives in the simulator, we had two choices:

1) Reinvent unique analogies to each of the already developed simulator components for each of the functions mentioned;

2) Describe the functions as representation-independent queries and then use the existing simulator components to describe their implementation.

Our choice of the latter approach only required generation of the appropriate supplemental queries at the right time. This was easily accomplished utilizing the transaction-to-transaction dependencies described in detail[5]. For example, to simulate journaling we simply define the appropriate transaction and state that the probability of the journal transaction is unity following any update to the user's database. The technique is similar for the other functions discussed.

SIMULATION OUTPUTS

Since the final purpose of a simulator is to help an investigator to draw conclusions, the information output by the simulator is of paramount importance. Furthermore, since there is no well-accepted figure-of-merit for database systems that is universally applicable to all experimental objectives, this information must include microscopic parameters as well as summary measures to allow derivation of a variety of figures-of-merit. Consequently, the simulator provides a total of 21 different output reports, any or all of which may be requested at the beginning of a simulation run.

To begin with there are a number of housekeeping reports to inform the experimenter about the nature of the run. These include a *Run Header* that identifies the run and reports summary execution data and a *Configuration Report* that tabulates and summarizes the input data. There is also a *Snapshot Dump* and a *Trace Report* to aid both the experimenter and programmer in debugging and planning future runs.

The *Response Time Summary* provides an ensemble picture of the query response time expected for the subject DMS in the hypothesized environment. It consists of a histogram of response times for each query. The beginning and ending ranges of each bar are reported in milliseconds and the height of the bar indicates how many of the queries processed had a response time falling in the indicated bracket. An example report page is shown in Figure 8. The *Detailed Response Time* data present the individual response times associated

with each query. Their production is optional since they can be voluminous for runs of long simulated time duration. Their principal use is in locating particular queries that are producing response times significantly differing from the mean.

FIGURE 8

```
DATA INDEPENDENT ACCESSING MODEL SIMULATOR
  PERFORMANCE FIGURE OF MERIT

MIN RESPONSE TIME =          0.000 MILLISECONDS
MEAN RESPONSE TIME =      9113.243 MILLISECONDS

MAX RESPONSE TIME =     14513.030 MILLISECONDS
STANDARD DEVIATION =     5568.197

REG RANGE    END RANGE        RANK (OCCURENCES)

      0.0        200.0                *
    200.0        400.0              *
    400.0        600.0              *
    600.0        800.0              *
    800.0       1000.0              *
   1000.0       1200.0              *
   1200.0       1400.0              *
   1400.0       1600.0              *
   1600.0       1800.0               *
   1800.0       2000.0              *
   2000.0       2200.0              *
   2200.0       2400.0              *
   2400.0       2600.0              *
   2600.0       2800.0              *
   2800.0       3000.0               *
   3000.0       3200.0              *
   3200.0       3400.0              *
   3400.0       3600.0              *
   3600.0       3800.0              *
   3800.0       4000.0              *
   4000.0       4200.0              *
   4200.0       4400.0              *
   4400.0       4600.0              *
   4600.0       4800.0              *
   4800.0       5000.0              *
   5000.0       5200.0              *
   5200.0       5400.0              *
   5400.0       5600.0              *
   5600.0       5800.0               *
   5800.0       6000.0              *
   6000.0       6200.0              *
   6200.0       6400.0              *
   6400.0       6600.0              *
   6600.0       6800.0              *
   6800.0       7000.0              *
   7000.0       7200.0              *
   7200.0       7400.0              *
   7400.0       7600.0              *
   7600.0       7800.0              *
   7800.0       8000.0                                        *

REG RANGE    END RANGE        0    5    10    15    20
```

The *Summary Resource Utilization* provides a capsule view of the resources consumed by the DMS under study during the simulated run. These data include total and average response time, total and average I/O time, total and average number of bits transferred, and total and average central processor time. A *Detailed Resource Utilization* report is also provided that lists, for each physical device (disks, controllers, tapes, etc), the total time utilized and the total amount of information transferred for each query processed. A sample of this report is shown in Figure 9.

The *Ensemble Query Statistics* report provides a statistical profile of the query stream processed. It reports the total number of queries and the total number of sample arguments processed, and breaks these down by function (ADD, DEL,..). Also, for each of the RSD specific query types, the number of queries resulting from that description is reported. In addition, a *Query* tabulation that actually lists each query and the time at which it was generated is provided. These reports are illustrated as Figures 10 and 11 respectively.

FIGURE 9

```
DATA INDEPENDENT ACCESSING MODEL SIMULATOR

  DETAILED RESOURCE USAGE

QUERY NO    NAS/HOS NAME   RESPONSE TIME   BITS TRANSFERED   NO. OF I/OS

     1          CPUFTN          24.40          4032.0            1.0
     1          CVALUS       13303.13       2132928.0          529.0
     1          CFINP           23.75          4032.0            1.0

     1          DISK01       13325.87       2136960.0          530.0
     1          CHAN1        13351.36       2140992.0          531.0
     1          CNTRL1       13351.36       2140992.0          531.0
     1          DISK02          24.45          4032.0            1.0

TOTAL RESPONSE TIME =     13351.363 MILLISECONDS
TOTAL BITS TRANSFERED     2140992.0
TOTAL I/OS GENERATED          531.0

QUERY NO    NAS/HOS NAME   RESPONSE TIME   BITS TRANSFERED   NO. OF I/OS

     2          CPUFTN          24.05          4032.0            1.0
     2          CVALUS       13166.91       2124864.0          527.0
     2          CFINP           27.45          4032.0            1.0
     2          CDATA           23.45          4032.0            1.0

     2          DISK01       13214.75       2132896.0          528.0
     2          CHAN1        13262.20       2136960.0          530.0
     2          CNTRL1       13262.20       2136960.0          530.0
     2          DISK02          47.80          8064.0            2.0

TOTAL RESPONSE TIME =     13262.297 MILLISECONDS
TOTAL BITS TRANSFERED     2136960.0
TOTAL I/OS GENERATED          530.0

QUERY NO    NAS/HOS NAME   RESPONSE TIME   BITS TRANSFERED   NO. OF I/OS

     3          CPUFTN          24.05          4032.0            1.0
     3          CVALUS       14363.73       2451456.0          608.0
     3          CFINP           50.75          8064.0            2.0
     3          CDATA           47.65          8064.0            2.0
```

Of course, the principal importance of the tabulation is the role it plays in the derivation of the other profiles, but it can also be used to insure that all possible query types have been anticipated in the application programs, and it can even be used as a test file for program checkout.

The ensemble statistics can provide insight to some effects that are not initially apparent:

1) Total number of queries in a time interval - This is an important parameter if the input statistics were assembled from piecemeal observations;

2) Percentage representation of each query class - This can be of particular importance for queries that occur as a result of other queries as is the case, for example, with system-oriented transactions such as journaling. It can also be useful for projecting the results of the partially specified queries.

FIGURE 10

```
DATA INDEPENDENT ACCESSING MODEL SIMULATOR

  QUERY STATISTICS .

                                          ALL TYPES          INTERNALLY GENERATED      SOURCE GENERATED

NUMBER OF TRANSACTIONS                      25                    8                        17
NUMBER OF QUERIES                           23                    8                        15

TOTAL COMPLEXITY                           113                   26                        87
AVERAGE COMPLEXITY/TRANS                  4.520                 3.250                     5.118
AVERAGE COMPLEXITY/QUERY                  4.913                 3.250                     5.800
FUNCTIONS REQUESTED PER QUERY
    PERCENT OF GETS (RET)                39.13                 25.00                     46.67
    PERCENT OF ADDS (ADD)                 0.00                  0.00                      0.00
    PERCENT OF DELS (DEL)                 0.00                  0.00                      0.00
    PERCENT OF CHGS (CHG)                60.87                 75.00                     53.33

NUMBER OF TIMES EACH TRANSACTION PROCESSED

TRANSACTION NAME = TOTAL REL        COUNT =     8
TRANSACTION NAME = BASIC CONTR      COUNT =     2
TRANSACTION NAME = MULT CONTR       COUNT =     0
TRANSACTION NAME = ALL CONTR-1      COUNT =     3
TRANSACTION NAME = ALL CONTR-2      COUNT =     0
TRANSACTION NAME = ALL CONTR-3      COUNT =     0
TRANSACTION NAME = ALL CONTR-4      COUNT =     0
TRANSACTION NAME = CONT AND RT      COUNT =     2
TRANSACTION NAME = BASIC REL T      COUNT =     2
TRANSACTION NAME = REL TASK 1       COUNT =     0
TRANSACTION NAME = REL TASK 2       COUNT =     1
TRANSACTION NAME = REL TASK 3       COUNT =     2
TRANSACTION NAME = REL TASK 4       COUNT =     0
TRANSACTION NAME = REL TASK 5       COUNT =     0
TRANSACTION NAME = ADD CONT         COUNT =     0
TRANSACTION NAME = ADD REL TASK     COUNT =     0
TRANSACTION NAME = CHG CONT-1       COUNT =     4
TRANSACTION NAME = CHG CONT-2       COUNT =     4
TRANSACTION NAME = CHG REL TASK     COUNT =     0
```

the number of instances of strings encountered. This report can also be used to design search logic for particular queries since, as previously discussed, it is the result of optimal automatic programming. This will also point out query classes that are not adequately supported in any way by the representation being considered. An example RDAL tabulation is shown in Figure 13.

The *Named Address Space (NAS)* Usage Summary* and the *NAS I/O Summary* report the primary activity relative to each NAS used in the simulation. The data include total number of accesses of each type, the mean density and standard deviation, and the density at the conclusion of the run. Similar to these is the *Overflow Summary* that reports secondary (i.e., overflow) activity relative to each

FIGURE 11

```
DATA INDEPENDENT ACCESSING MODEL SIMULATOR

 QUERY TABULATION

QUERY NUMBER =      1     TRANSACTION NUMBERS =      1 THRU      1     AT TIME =    110.854
    CHG  SETN1  .OF. SLR CONT      /   CONT NUMBER   .WHERE.   CONT NUMBER    =     00001 .AND.SUB NO  =     00001

QUERY NUMBER =      2     TRANSACTION NUMBERS =      2 THRU      2     AT TIME =    297.281
    GET  S1     .OF. CONTRACT      /   CONTRACT NUMBER     /   CONTRACTOR    /   START DATE    /   COMPLETION DATE    /
         CLSATTCN      /   FEE      /   METHOD  /   TYPE    /   CONT AMOUNT   /   UPDATE  .WHERE.   CONTRACT NUMBER     =     00001

QUERY NUMBER =      3     TRANSACTION NUMBERS =      3 THRU      3     AT TIME =    831.871
    GET  S1     .OF. RELATED TASK  /   CONT NUMBER   /   D REQ   /   PR NO   /   T AMOUNT      /   PY      .WHERE.   CONT N
         UWEEP  =     00001 .AND.D REQ   =     00002

QUERY NUMBER =      4     TRANSACTION NUMBERS =      4 THRU      4     AT TIME =    861.532
    GET  S1     .OF. RELATED TASK  /   CONT NUMBER   /   D REQ   /   PR NO   /   T AMOUNT      /   PY      .WHERE.   CONT N
         UWEEP  =     00001 .AND.D REQ   =     00003
```

The ensemble query statistics thus help to characterize the aggregate workload of the system, as well as to identify certain queries or query classes to which the workload is most sensitive.

The *I/O Tabulation* lists, for each query, every input/output request processed, including the physical file name, the number of bits transferred, and the address at which the operation was performed. It is particularly useful for calibrating the simulation against instrumented benchmark runs using the real DMS being studied. An example is shown as Figure 12.

The *RDAL Summary* report shows the total and average number of accessing instructions over the simulation run. It is backed up by the *RDAL Tabulation* that shows, for each query, the resulting path traversed during its processing, including

NAS. Included in this report, shown as Figure 14, is the total number of overflow-related accesses, and the path on which the overflow occurred.

The path (called strings in the simulator's terminology) usage statistics describe each individual path and are useful for reaching conclusions about what to do about inadequate data structures. Path utilization frequency is helpful in identifying paths that are costing more to maintain than they are worth. Storage requirements can be adjusted by examining:

1) The number of instances of each path;

2) The ensemble average length of path instances.

*NASs are equivalent to physical files in other notations.

FIGURE 12

```
DATA INDEPENDENT ACCESSING MODEL SIMULATOR

QUERY NUMBER =        3
     READ  03936 BITS  OF   CELS    00001 TIME (DISP= R     )    READ  01440 BITS  OF   COEFTN 00001 TIME (DISP= R     )
     READ  00216 BITS  OF   CVALOR  00001 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00291 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00285 TIME (DISP= 00072)    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00513 TIME (DISP= 00072)    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00291 TIME (DISP= 00072)    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00301 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00365 TIME (DISP= 00072)
     READ  00072 BITS  OF   CVALUS  00001 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00365 TIME (DISP= 00072)
     READ  00072 BITS  OF   CVALUS  00001 TIME (DISP= R     )    READ  03168 BITS  OF   CFIND  00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CDATA   00001 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP=-00162)
     READ  00072 BITS  OF   CVALUS  00145 TIME (DISP= 00072)    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00301 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00366 TIME (DISP= 00072)
     READ  00072 BITS  OF   CVALUS  00301 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00510 TIME (DISP= 00072)
     READ  00072 BITS  OF   CVALUS  00001 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00510 TIME (DISP= 00072)
     READ  00072 BITS  OF   CVALUS  00301 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00513 TIME (DISP= 00072)    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00281 TIME (DISP= 00072)    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  03168 BITS  OF   CFIND   00001 TIME (DISP= R     )    READ  00028 BITS  OF   CDATA   00001 TIME (DISP= R     )

QUERY NUMBER =        4
     READ  03936 BITS  OF   CELS    00001 TIME (DISP= R     )    READ  01440 BITS  OF   COEFTN 00001 TIME (DISP= R     )
     READ  00216 BITS  OF   CVALOR  00001 TIME (DISP= R     )    READ  00072 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  00072 BITS  OF   CVALUS  00478 TIME (DISP= R     )    READ  00000 BITS  OF   CVALUS 00001 TIME (DISP= R     )
     READ  03168 BITS  OF   CFIND   00001 TIME (DISP= R     )    READ  00028 BITS  OF   CDATA   00001 TIME (DISP= R     )
```

FIGURE 13

```
DATA INDEPENDENT ACCESSING MODEL SIMULATOR

  RUN TABULATION

QUERY NUMBER =        1
     !GET KEY NAME     !GET O REC KEYNAME      !GET CONTRACT NO KEYNAME      !GET NEXT = 01212 .OF. CONTRACT NO VALUE  !GET
    CONTRACT NO VALUE INSTANCE     !GET CONTRACT ASG

QUERY NUMBER =        2
     !GET KEY NAME     !GET O REC KEYNAME      !GET CONTRACT NO KEYNAME      !GET NEXT = 01163 .OF. CONTRACT NO VALUE  !GET
    CONTRACT NO VALUE INSTANCE     !GET CONTRACT ASG !GET CONTRACT NUMBER     !GET CONTRACTOR   !GET START DATE   !GET COMPLE
    TION DATE     !GET DURATION     !GET REF     !GET METHOD !GET TYPE    !GET CONT AMOUNT  !GET UPDATE

QUERY NUMBER =        3
     !GET KEY NAME     !GET O REC KEYNAME      !GET NEXT = 02328 .OF. O REC VALUE  !GET O REC VALUE INSTANCE     !GET RELATE
    D TASK ASG    !GET CONT NUMBER  !GET O REC  !GET PR NO  !GET T AMOUNT     !GET QY     !GET  SAME  .OF. O REC VALUE  !GET
    NEXT = 02328 .OF. O REC VALUE  !GET O REC VALUE INSTANCE     !GET RELATED TASK ASG    !GET CONT NUMBER  !GET O REC  !GET
    PR NO. !GET T AMOUNT     !GET QY
```

FIGURE 14

```
DATA INDEPENDENT ACCESSING MODEL SIMULATOR

OVERFLOW SUMMARY

  NAS ACCESSED   TOTAL NO. RET     TOTAL NO. ADD
     TOTAL              0               0
     COEFTN             0               0
     CELS               0               0
     CVALOR             0               0
     CVALUS          40531              0
     CNAME              0               0
     CENTS              0               0
     CFIND              0               0
     CDATA              3               0

  REL CAUSING OVERFLOW      NAS ACCESSED  NO. RET  NO. ADD
  O REC VALUE REV NAME        CVALUS       16263      0
  CONTRACT NO VALUE REV NAME  CVALUS       22966      0
  REF VALUE REV NAME          CVALUS        1305      0
```

Overflow strategies are also easier to formulate when you know:

1) The minimum and maximum number of instances of a path type;

2) The ensemble average minimum and maximum length of instances of the path type.

Also, *Path-to-Path Transition Frequencies* can help in making the critical decisions about which links to implement by "fast addressing" (e.g., contiguity) rather than "slow addressing" (e.g., pointers). They also provide insight to whether individual paths ought to be combined because they're almost always accessed together. Examples of these reports are shown as Figures 15 and 16.

Finally, it is possible to gain additional insight about the information itself. The *Entity Set Population Statistics and Description Set Population Statistics* report the cumulative effects of the population dynamics, showing the initial, minimum, maximum, and final populations. These can provide insight to the magnitude of the fluctuations, which ultimately determine headroom requirements.

And, by varying the query mix, it is possible to identify queries or query classes to which the fluctuations are most sensitive so some provisions can be made to control the problem. These reports are shown in Figure 17.

FIGURE 15

DATA INDEPENDENT ACCESSING MODEL SIMULATOR

STRING USAGE STATISTICS

STRING NAME	NUMBER	TYPE	INSTANCES	MAX	MIN	LENGTH	MAX	MIN	TIMES ACCESSED
RELATED TASK ASG	1	ASG	7500.00	7500.	7500.	5.00	5.	5.	71.
D REQ VALUE INSTANCE	2	ESG	7500.00	7500.	7500.	0.00	1.	1.	6.
D REQ VALUE	3	ESG	7500.00	7500.	7500.	1.00	1.	1.	16264.
D REQ KEYNAME	4	ESG	1.00	1.	1.	7500.00	7500.	7500.	23.

MINIMUM LENGTH FOR E-STRING 1. MAXIMUM LENGTH FOR E-STRING 7500. MEAN LENGTH FOR E-STRING 208.

FIGURE 16

DATA INDEPENDENT ACCESSING MODEL SIMULATOR

STRING USAGE STATISTICS

CORRELATION MATRIX

STRING NAME (PREVIOUS)	TYPE/NUMBER		(CURRENT STRING) 1	2	3	4	5	6	7	8	9	10
RELATED TASK ASG	ASG	1	65	0	0	0	0	0	0	0	0	0
D REQ VALUE INSTANCE	ESG	2	0	0	0	0	0	0	0	0	0	0
D REQ VALUE	ESG	3	1	1	16259	0	0	0	0	0	0	0
D REQ KEYNAME	ESG	4	0	0	0	0	0	0	0	0	0	0
REPEATING GROUP	ESG	5	0	0	0	0	0	0	0	0	0	0
CONTRACT ASG	ASG	6	0	0	0	0	0	0	0	0	0	0
CONTRACT NO VALUE INSTANCE	ESG	7	0	0	0	0	0	0	0	0	0	0
CONTRACTOR VALUE INSTANCE	ESG	8	0	0	0	0	0	2369	0	2714	0	0
REF VALUE INSTANCE	ESG	9	0	0	0	0	0	0	0	0	0	0
CONTRACT NO VALUE	ESG	10	0	0	0	0	0	0	0	0	0	22966

FIGURE 17

DATA INDEPENDENT ACCESSING MODEL SIMULATOR

ENTITY POPULATION STATISTICS

ENTITY NUMBER	ENTITY NAME	POPULATION DATA INITIAL	FINAL	MINIMUM	MAXIMUM	ADDITION	DELETION
1	CONTRACT NUMBERS	3000	3000	3000	3000	0	0
2	CONTRACTOR NAMES	500	500	500	500	0	0
3	CONTRACT DATES	400	402	400	403	3	1
4	NUMBER OF MONTHS	120	120	120	120	0	0
5	REF NUMBERS	3000	3000	3000	3000	0	0
6	AWARD METHODS	6	6	6	6	0	0
7	CONTRACT TYPES	10	10	10	10	0	0
8	DOLLARS	5000	5003	5000	5001	2	2
9	DIVISION REQ NUMBERS	6500	6500	6500	6500	3	0
10	PURCHASE REQ NUMBERS	6500	6500	6500	6500	0	0
11	PROGRAM YEARS	10	10	10	10	0	0
12	UPDATES	600	601	600	601	1	0
13	SUB NOS	5000	5000	5000	5000	0	0
14	MORE DATAS	10	10	10	10	0	0

DESCRP NUMBER	DESCRP NAME						
1	CONTRACT	3000	3000	3000	3000	0	0
2	RELATED TASK	7500	7500	7500	7500	0	0
3	SUB CONT	6500	6500	6500	6500	0	0

REFERENCES

1. Senko, M. E., Altman, E. B., Astrahan, M. M., Fehder, P. L., and Wang, C. P. A Data-Independent Architecture Model 1: Four Levels of Description from Logical Structures to Physical Search Structures. IBM Research Report RJ982, San Jose, California, February 1972.

2. CODASYL Stored Data Definition and Translation Task Group (SDDTTG). Stored-Data Description and Data Translation: A Model and a Language. Unpublished Technical Report, March 1976.

3. Schneider, L. S. "A Relational View of the Data-Independent Accessing Model." Proceedings of the 1976 ACM/SIGMOD International Conference on Management of Data, June 1976.

4. Senko, M. E. Specification of Stored Data Structures and Desired Output Results in DIAM II with FORAL. IBM Research Report, 1975. (T. J. Watson Research Center, Yorktown Heights, New York)

5. Schneider, L. and Spath, C. "Quantitative Data Description." Proceedings of the ACM SIGMOD International Conference on Management of Data, San Jose, California, May 1975.

6. Senko, M. E., Altman, E. B., Astrahan, M. M., and Fehder, P. L. "Data Structures and Accessing in Data Base Systems." IBM Systems Journal, No. 1, 1973, pp 30-93.

7. Senko, M. E. and Altman, E. B. DIAM Note 1 - A "Framework" Mode for Implementing a Record-Storing Facility. IBM Research Report RJ1365 (21150), San Jose, California, March 1974.