SUMMARY OF CURRENT WORK

ANSI/X3/SPARC/STUDY GROUP - DATABASE SYSTEMS

Draft prepared by Charles W. Bachman

Honeywell Information Systems

ASTO MS 425C

200 Smith Street

Waltham, Massachusetts 02154

This document has been approved for distribution as a status report by the Study Group on January 25, 1974.

revised January 29, 1974

INTRODUCTION

In late 1972, SPARC (the Standards Planning and Requirements Committee of ANSI/X3) established the Study Group - Database Systems to review the current state of development in the database systems field with objective to determining whether standardization activities were appropriate. The Committee was charged with preparing "SPARC 90's" for each identified area. Loosely speaking, a SPARC 90 is a document which identifies a proposed area of standardization and develops for that area a recommendation as to what action should be taken. Part of that recommendation is technical, other parts deal with the social and economic justification for the action. For a detailed description of a SPARC 90, refer to document No. X3/SPARC/90, dated April 10, 1972, entitled "Guideline for Presenting a Proposal..."

The Committee has the following members:

Charles Bachman ..... HIS, Vice Chairman
Larry Cohn ..... IBM
William Florence ..... Kodak
Frank Kirshenbaum .... Equitable
Charles E. Mairet .... Deere & Co.
Bruce Puerling ..... Bell Labs
Ed Scott ..... NCR
Edgar H. Sibley ..... U. of Michigan
David M. Smith ..... Exxon (Chairman)
Tom Steel ..... Equitable Life (SPARC Chairman)
Jon A. Turner ..... Columbia University (Secretary)

This Committee has met approximately every two months for the last year in two day sessions with working papers being produced and distributed between meetings.

METHODOLOGY

After several approaches were examined, a general mechanism was accepted for the development of the required SPARC 90's. This mechanism was based upon the concept that the only things that can be standardized are interfaces: interfaces between man and machine, interfaces between machine and machine, and interfaces between man and man. Therefore, a general plan was established to try to identify, describe and organize the set of interfaces which would make up a generalized database management system that could maintain its validity for a period of at least ten years. In effect, this meant that a gross architecture had to be developed as to how things do and will fit together. The implication was that this activity, to be successful, would have to find an organization of parts which would in one sense offend no serious student of the subject while at the same time consider the major problems seen with existing products and known projected products. At the same time it was recognized that standardization of series of coordinated interfaces did not and could not prevent the development of products based upon other ideas. In the final analysis, the most basic issue before the study group was to determine whether that is enough now know about the subject of database management to construct a definitive architecture of modules and interfaces with well defined and accepted functionality. If this could not be done, then it would appear to be premature to consider standardization of any substantive portion of the field of database systems at this time. On the other hand, if a definitive architecture could be stated, and broadly accepted by manufacturers and users of database systems, then a systematic plan could be evolved to proceed with the investigation, specification, evaluation and ultimate standardization of the interfaces which would surround the functional parts of the database systems and ultimately provide the benefits which would flow from interchangeability of these parts. Furthermore, it would mean that each such interface could develop at its own pace toward standardization with a confidence that it would fit in with the balance of the components when it was completed.

Associated with this report are several attachments. One of these is the master architecture sheet which represents the current thinking of the committee with regard to the basis for describing database systems. The other attachments are examples of the work sheets which are being used to describe each of the interfaces. Work sheets like these will ultimately be created for each of the interfaces which has been identified on the master architecture sheet.

GROSS ARCHITECTURE

The gross architecture as outlined on Schematic #1 is based upon both historical grounds and projected extensions. It has been reviewed to see how well I-D-S, IMS, and COBOL DBTG implementations would relate to it. It has been more loosely reviewed to see how "access method" oriented software would relate. It has also been reviewed to see how some nagging problems could be solved in the areas of

end-user languages, data independence and restructuring. While the gross architecture has been reviewed in its relationship to these target systems, the converse investigation as to how well the interfaces of those systems would serve as candidates for standardization of particular SPARC interfaces has not been approached, except in a most superficial sense.

The gross architecture is based in part upon the concept of nested machines. These machines began at the outside with the most complete support of database functionality and as each layer of machine is stripped away we find less logical capability and more physical capability until we reach the actual secondary storage device and its capability to read and write. It is clear to the Study Group that somewhere along the line, as machine after machine is revealed, we are no longer really talking about "database machines" but rather are talking about "storage machines". At the moment, it has not seemed important to make a sharp cut off of interest as the desire to be able to transport databases from one system to another requires that a database be ultimately mappable onto removable media.

In addition to the multiple layers of machines, the gross architecture recognizes that there are both declarative as well as manipulative interfaces associated with database systems. This is most sharply recognized when it is considered that a database system is charged with certain security and integrity responsibilities and that these responsibilities require that the system itself understand some informational aspects of the database as well as the more obvious data aspects. At the present time, ther are eight interfaces identified which are basically data declaration and description interfaces. Subsequent detailed examination of these interfaces will show some of these to be people to machine interfaces, some of these to be machine to machine interfaces, and two of them to be a machine to people interface. The reason that people are either on the sending or receiving side of such an interface relates to their role in the enterprise.

A number of people roles were identified as part of an effort to understand what went on with regard to a database system. These are listed below. For the present, the descriptive name of the role will have to suffice for a description of their roles. Later, it is expected that these roles will be given more detailed descriptions. In the context of describing an interface where such a person interacts with the database system, his role will be in part clarified by what he can do at that interface.

It should clearly be recognized that the roles identified below may be played by several persons in an enterprise and frequently one person may wear different hats at different times.

Application Programmer
Application System Administrator
Database Administrator
Enterprise Administrator
Inquiry Specifier
Operations Clerk
Report Specifier
Site Operator
System Programmer
Update Specifier

Several of the roles listed in prior paragraphs relate to what have been described as "end user" roles. These were the report specifier, inquiry specifier and update specifier roles. These are users who have largely been excluded from direct access to the database in the past but who have important assignments from the view of the enterprise and whose proficiency in those assignments would be enhanced by direct access.

One of the least precise terms now extant in the Database Management field is that of "data independence". Without attempting at this point to develop a clear definition of it, it should be said that the attempt to understand if the problems surrounding the issue have strongly motivated the members of the Study Group and have lead to some innovative results. It is in this area that this report will probably cause the greatest indigestion among its reviewers and critics. It is felt by the Study Group that it has developed a gross architecture which solves the most difficult problem which faces the growth of database systems. Simply stated, the problems of data independence revolve about the need for a feasible solution to the problem of the orderly evolution of an enterprise and the orderly evolution of the database systems which support the effective operation of the enterprise. If database systems were perfected, well understood, and easy to install, then the problem would simply be to track an enterprise as its database system requirements change. Unfortunately, the conditions stated above are not true, therefore there is a need within database systems to allow the enterprise, the database system and the know-how of the staff to evolve simultaneously. The target of the gross architecture set forth is to allow all the users of the system a maximum of stability within which to carry out their functional assignments without simultaneously stymieing either the progress of the enterprise or the database system itself.

It is recommended that the readers would look at the large chart labelled "system schematic #1",* while reading the following detailed explanation. This chart has a number of graphic symbols in use. They are set forth in the legend at the lower left hand corner. The first symbol which I would like you to consider is the hexagon. On this chart, each hexagon represents one of the people roles enumerated earlier. The name of the particular role is written into the hexagon as a title. All the hexagons tend to

*Revision 74-01-29.

be on the outside of the chart and their interfaces to the system are defined closer to the center of the chart. The next symbol to recognize is the rectangle. It is used to represent a computerized functional element or module. Within each rectangle is a descriptive title which gives a hint to the activity carried out at that location. Between the hexagons and rectangles are several different styles of arrows representing different types of data flow. The details of the data flow will be discussed later. However, the item to focus upon at the moment is the very heavily shaded bar which cuts across some of these data flow lines. These bars represent the interfaces which are currently under study by the Study Group. Associated with each such bar is a small circle with a numerical or alphabetic code which relates the interface to the legend in the upper right hand corner of the chart. This legend provides a descriptive name for each of the interfaces which will serve as a first hint as to the nature of the interface. The interfaces indicated by the letters, "l", "m", and "n" are interfaces which are not under study by the Study Group but whose usage is heavily interleaved with the usage of the database interfaces. For an example, Job-Control-Language is used to submit jobs to a computer system for execution. Therefore, submittal of jobs which interact with the database will require aspects of both. The triangle symbol on the chart represents a place where a program or schema can be catalogued or held until it is subsequently needed. The storage, maintenance, and retrieval of these programs or schemas are not considered to be a database activity, but rather a system support activity necessary to the computer and database operation.

Returning to the arrow as the symbol of data flow, the chart distinguishes four types. The first of these, "command" flow, is diagrammed with an arrow with a fish hook like arrow head. This is to represent the call by a person or functional module upon some other module for action. The arrow emanates from the caller of the function and the folding back of the arrow head represents the response which is typical of a command. Such command flow is indicated where job execution is being requested and also where one level functional module is calling upon a lower level module for support. Flow of data into and out of the database is diagrammed by a single or double headed arrow. The double head simply means that data may flow in either direction. The third type of data flow is diagrammed with a lightning bolt like double headed arrow. This represents the flow of "real i/o". The distinction is being made to separate the flow of data between programs and the outside world from data flow between a program and the database. In a Message Management or telecommunications sense this is clear. However, this is to include the input through card readers and output through high speed printers. The fourth type of data flow is the flow of programs and schemas. They are illustrated by an arrow with "X"'s breaking the shaft at intervals.

Probably the best place to dive into the gross architecture sheet is along the center line running from right to left. Best in the sense that the reader is most likely to encounter recognizable objects and thus more quickly be able to orient himself to the pattern of the architecture. As a starting point, we will start with the functional module labelled "Internal Data Transformation Functions" and the interface labelled "18" to its right. This is located almost at the exact center of the chart. This interface is where requests are made upon the functions within the rectangle. In a gross sense, this module carries out operations upon files, records, fields, and sets at the internal data level. The internal data level is essentially equal to the level we are familiar with in a COBOL program, where a record is a collection of physically adjacent fields, a field is a string of characters, and a file is a collection of records. The record level operations are the creation, retrieval, and destruction. The field level operations are extract and alter. The set level operations are insert, remove, find next, find owner. At the internal data structure level, all the fields, sets and records accessible are physically stored in the mass storage system through the interfaces (21,24,26 and 28) to the left. These objects may be either natural or derived, but in the sense of the DBTG Report, they are "actual" in their existence at the internal level. This distinguishes sharply with the external data structures which we will review next, where data may be either "actual" or "virtual". Programs which access the database at the internal data structure level have no data independence and therefore are vulnerable to the reformatting of data which would come about through a restructuring of the database. The interface labelled "18" is the user interface for those who would access the database at the internal data structure level. These users would include application programs, which have chosen to give up data independence to achieve greater performance, and utility programs such as copy, compare, load, reorganization, and restructuring. Most importantly to the continuation of this discussion, it is the interface through which the modules supporting data independence access the database.

There is one module directly to the right of the Internal Data Transformation Function Module. This is labelled as "External Data Transformation Functions". To the right of this rectangle is the interface labelled "#12". This is the "External Data Manipulation" interface. It is at this interface that users find data independence in their accessing of the database. This interface and the several schemas necessary for its support have all been constructed so that the application program, the report specification, the inquiry specification, and the update specification can be free from interference when the database is restructured. Restructuring which changes the internal data structure can make many kinds of changes in the name of efficiency of storage and processing. These changes include: changing the size of a data field or the recording mode, changing the sequence of fields within a record, deleting or adding new fields, changing the implementation mode of sets, changing the ordering of sets,

changing the implementation of primary key retrieval, adding derived fields, records and sets to the internal data structure, deleting old derived fields, records, and sets.

The objects (fields, sets, records, and files) and the operations on those objects are identical at both the internal and external data interfaces. Their difference lies in that the internal to external transformation functions can dynamically realize (virtualize) derived items which are not actually stored at the internal data level, can reformat fields to the format expected by the external data user, can reorder sets to make them appear to be in the sequence specified in the external schema. Even for fields which have completely been removed from the internal data and for which there is no algorithm for reconstruction, they will deliver a "null" value which will let the external user continue. It is expected that all of the end-user processors would interface at the external data manipulation interface (#12). It is anticipated that most of the application programs would interface at the external data level with the internal data level interface being used only when extremely heavy processing demands make the greater efficiency of the data dependent mode at the internal data level desirable.

DATA DECLARATIONS

Three different kinds of data declarations have been specified as part of the mechanism to support data independence. One of these, the External Data Description Language, can easily be related to the Sub Schema of the DBTG Report. The other two can be related to the schema of the DBTG Report. However, there is enough difference in both cases to recommend a close readying of the following description. The Internal Data Description Interface (#13) is the place where the Database Administrator passes his description of the database to the database system. The Internal Data Schema Processor is the database module which receives that description, edits it for syntactical errors and internal and external consistency and, if all is well, places that internal data schema in "object" format (interface 14) into the library for subsequent usage. The restructuring of a database is the process of taking a database which exists on the basis of one internal data schema and reformating it to make it consistent with another internal data schema. These two internal data schemas must both be externally consistent with the Conceptual Schema. The objects described at the Internal Data Description interface are the files, records, fields and sets which may be manipulated through the Internal Data Manipulation Interface (#18). The operations are: create, destroy and modify declaration; and create, validate, and destroy schema. Where the internal data structures are to be directly processed by an application program, a host language format of the schema is provided (interface 15).

The Conceptual Schema is a formalization of an idea which could be seen in the SHARE/GUIDE Report and in work on restructuring. It proposes that the entities classes that are recognized in an enterprise and their attributes and relationships should be catalogued and used as the point of departure for the entire database system. The cataloguing of these entities, attributes, and relationships makes no requirement that any record, field, or set be implemented but does establish, if such a need arises, a basis in name and structure on which to build. Potentially, an entity class that exists within the enterprise and thus in the Conceptual Schema will be represented by one or more record classes in the Internal Schema. If data concerning the entity class is not currently in the database, then there would be no record class described in the Internal Schema for it. In some cases, there might actually be several record classes based upon the same entity class. For example, the entity class of employee might be represented by employee records, employee address records, and the employee pension plan records. The Conceptual Schema is completely disinterested in the details of the Internal Data Schema which is the Database Administrator's job to optimize and reoptimize over time. It is interested in the basic nature of the enterprise. As the result, it is anticipated that the Conceptual Schema will be very stable in nature. Its principal change over time would be that of addition as more and more parts of the enterprise are described to the database system. Actual modifications to the conceptual data structure would mainly be the result of discovering errors in the earlier Conceptual Schema or reflect major changes in the view of the enterprise. One anticipated result is that the Conceptual Schema of an enterprise would be substantially the same across an industry. All the Conceptual Schema in the Banking Industry would be substantially the same. All the Conceptual Schema in the Discrete Manufacturing Industry would be substantially the same. The same for Insurance, Universities, Local Government, Hospitals, etc. The differences between the enterprises within an industry being in the portion of the Conceptual Schema which is represented in their Internal Data Schema and the optimization decisions taken. All the Internal Schema would be validated for consistency with the Conceptual Schema when they are processed by the Internal Data Schema Processor. By definition, an Internal Data Description cannot describe any data element which does not exist in the Conceptual Schema as a natural element or as a function of natural elements.

External Data Descriptions exist for the purpose of presenting a particular view of the enterprise to a group of users and to define the operations that a user of that group is authorized to carry out against the database. In this sense it is a mask over the Conceptual Schema. In addition to its masking capability, it permits the viewer to define the size and recording mode in which he wants to receive the fields of a record, and to define the ordering rules for sets and files which he wants to access sequentially. The External Data Description exists in three different forms and thus there are three different interfaces.

The first of these interfaces (#4) is the interface through which the Application System Administrator transmits the External Data Structure Description to the database system. Through this interface, he would create, modify and destroy External Data Schema. The response of the external data schema processor would be to check the syntax of the schema and to validate the schema elements against more encompassing external schema or against the Conceptual Schema as the ultimate authority. In addition to the diagnostic comments to the application system administrator who authored the schema, if the schema were correct, an internal form of this schema would be created and stored for subsequent usage. This internal form is defined by interface number 5. This form will be used directly as input by the internal/external transformation functions and the external data manipulation functions. This form may be further processed by a host language formatter function which would produce a form acceptable to a particular host language compiler such as COBOL, FORTRAN, or PL/1. The host language formatter produces an output in the interface format (6). The subschema description language currently being worked on by the COBOL Programming Language Committee, is an example of a particular host language interface as envisioned here. That interface is intended both for the host language compiler and the application programmer who is writing the application program which will access the database at the external data structure level. The report generator, enquiry processor, and update processor may have their own host languages data descriptions formats.

INTERFACES

The balance of this document is a series of brief descriptions of the interfaces pictured on the gross architecture chart. The description will tell something about the intent of the interface, the objects which are the subject of conversation across the interface and the nature of the operations on those objects. The working papers of the Study Group include a standardized manner by which each interface is ultimately to be described. That method is too detailed for the purpose of this document. However, the Internal Data Structure Manipulation interface (#18), is given in brief form and also in the more detailed form to illustrate the Study Group's method of operation. The final report of the Study Group must be sufficiently precise that another group working in detail on a particular interface will have a clear understanding of the gross architecture so that their ultimate product will fit into the whole in the manner anticipated by the Study Group.

INTERFACE DESCRIPTION WORK SHEETS

The Study Group is experimenting with a technique which will define each interface in the same manner. At the heart of this technique are three work sheets. The first work sheet (Figure 1) attempts to establish the overview of the interface. It has space to state who is initiating transfer over the interface and who is responding to the transfer. Who, in these cases, is defined either as people

who have roles with regard to the database system or as database system modules. There may be several requesters or several responders for a single interface, but always at least one. Secondly, it attempts to establish in a few sentences the purpose of the dialog between the requester and the responder. The second work sheet (Figure 2) provides space for the drawing of a data structure diagram. The purpose of the diagram is to provide the Conceptual Schema of the database entities that the requester and responder both see and which are subject to discussion over the interface. In many of the interfaces of the database system, the principal entities being discussed are files, records, fields, and sets as declarations or as occurrences. The data structure diagram provides a compact form by which the entities and their relationships can be presented. The third work sheet (Figure 3) provides a place to enumerate the operations upon the entities from Figure 2 which may be requested by the requester.

It is the intent that these three work sheets would collectively give a clear picture of the interface such that their reader would clearly understand the nature of the interface. It is understood that a considerable amount of narrative may have to go with the description of each interface, particularly in that some of the entities or the operations upon the entities are not well known from other systems. Figures 4, 5, and 6 are completed work sheets which describe the internal data description language (object format) interface.

SUMMARY OF INTERFACES

#1 CONCEPTUAL DATA DESCRIPTION - Source Format

The source format of the Conceptual Data Description is the interface by which the Enterprise Administrator makes known to the database system his declarations concerning the nature of his enterprise. He describes the enterprise in terms of the entities with which the enterprise is involved, the attributes by which they are described, and the relationships which exist between them. In order to accomplish compactness, these entities are described in terms of entity classes. In other words, the description is not in terms of the description of individual entities (each employee, each customer, each plan) rather in terms of what in general is known about employees, customers, and plans. The source form is transmitted by the Enterprise Administrator to the Conceptual Data Schema Processor module of the database system where it is edited for syntax errors and internal consistency. The description of both natural elements and derived elements is permitted. An example of a derived element is a summary field in an owner entity which contains the sum of all the quantity attributes found in the members of its relationship. The quantity attributes are natural each representing aspects of real transaction. Another example is a summary entity which represents the unique intersection between a product and a customer for that product. It might contain summary attributes concerned with the sum of the dollars sold to the customer and the sum of the pounds sold to the customer over a period of time.

#2 CONCEPTUAL DATA DESCRIPTION - Object Format

The object format of the Conceptual Data Description is the interface by which an edited and correct Conceptual Schema is made known to the rest of the database system. It is this object format that is read by both the external data schema processor and internal data schema process modules so that they can validate the external and internal data schema as they arrive in source form. It is also used by the internal/external transformation function module.

#3 CONCEPTUAL DATA DESCRIPTION - Display Format

This is the display format of the conceptual schema prepared for distribution to and reference by a wide number of personnel both within and without the data processing organization. It is the most comprehensive statement of the enterprise and its operations seen as information.

#4 EXTERNAL DATA DESCRIPTION INTERFACE - Source Format

The external data description in source format is the interface through which the application system administrator submits a new external data schema and gets it validated. This operation includes syntax checking, checking for internal consistency, and checking that the items are mappable upon either more encompassing external data schema or against the Conceptual Schema. It will include the name of items to be referenced, their relationships one to another, and the operations which are to be allowed against such items by authorized users of the schema. Data security is based upon a particular person or process having access rights to use an external schema. That external schema itself declares all the objects visible and all the allowed operations on the projects. The database itself is accessible by all authorized internal and external data schemas.

#5 EXTERNAL DATA DESCRIPTION INTERFACE - Object Format

The object format of the external data description is the interface through which an edited external data schema is first made known to the rest of the database system. It includes both the description of the elements and the mapping of the elements onto the elements of the Conceptual Schema. It includes both logical and physical descriptions such that the source format could be regenerated for display purposes. It is also used as input to the host language formatter modules which would create a version according to the specifications of a particular host language.

#6 EXTERNAL DATA DESCRIPTION INTERFACE - Host Language Format

The host language format of the external data description is the interface made known to a host language compiler (COBOL, FORTRAN, PL/1, etc.) when it is to compile external data manipulation statements against the external data descriptions. It is created by the host language formatter module and placed into the library for future reference. It is displayable so that the application

programmer can get a copy to study. It is accessible from the library by the host language compiler or presumably punchable or copyable by a source program text editor. There are potentially as many host language interfaces as there are host languages or freestanding languages such as a report specification language.

#7 EXTERNAL DATA MANIPULATION - Host Language Format

The source format of the external data manipulation language is the interface by which an application programmer would describe his access and manipulation of external data structure objects within his application program. There would be one such host language for each host language (COBOL, FORTRAN, PL/1, etc.).

#8 REPORT SPECIFICATION LANGUAGE

The report specification language is the interface through which the end-user as a report specifier describes a particular report that he would like to have prepared. The report specifier is concerned with record and field selection, ordering of output lines, the internal formatting of output lines, the summarization of certain quantity fields and report header and page header formatting. It is assumed that certain of the record selection criteria could be handled interactively so that a standard report specification could be dynamically varied to permit it to select different records for printing or display at each execution. It is anticipated that there will be a mechanism for constructing, editing, and modifying the report specification as part of the interface. Language complexity should be designed toward a one day training course with a week's experience to gain proficiency.

#9 ENQUIRY SPECIFICATION LANGUAGE

The enquiry specification language interface is designed to permit the communication between the end-user as enquiry specifier and the Enquiry Processor module. It would be designed to be more easily learned than the report specification language. The language complexity should be designed toward a one hour training course with a day's experience to gain proficiency. Available operations would include record selection and field selection. Formatting would be left to the system's discretion.

#10 UPDATE SPECIFICATION

The update specification interface is intended to provide the end-user with the ability to cause simple updates to the database. It is envisioned that this would be largely limited to updating certain fields by replacement of the current value. Its record selection technique would be either based upon primary key retrieval where a particular record is to be updated ("horizontal changes") or by secondary key retrieval where an entire group of records is to be updated in the same manner ("vertical changes", in the vernacular of PAC). Target would be a training course of two to four hours with one to two days' experience to gain proficiency.

GENERAL NOTE ON END-USER FACILITIES

The end-user facilities will be an area where a great deal of imaginative development can be expected in the next decade. The interfaces (8, 9, and 10) probably represent families of interfaces which will be developed. The ease of learning to work through these interfaces will lower the pressure for standardization becasue an end-user can be readily retrained. Further, it is not expected that either the enquiry specification or the update specification interface would have facilities to save specifications for re-use, because of the ease in recreating them. Thus, there would not be great investments in specifications to be protected by standardizing the language. The report specification interface is sort of a half way house because of the more complex specification language and associated training and the inventory of report specifications which are to be preserved. Multiple styles of enquiry languages may develop, each "best" for some class of end-users.

In many cases the most used end-user facility will result from the interaction of operations clerks with interactive applications programs which are specifically oriented to that clerk's needs. In a sense these will be the easiest of the end-user facilities to use. It is in this sense that today's airline reservation systems support the airline reservation clerks and other operations people. This is interface #11.

#11 PARAMETRIC INTERFACE

This is the data interface as created by the application programmer. The format is potentially different for each such application program. It is illustrated here with the symbol of the lightning bolt to show that it is transported via message management (not database management) to the program. Standardization of this interface would be the result of application standardization rather than database standardization.

#12 EXTERNAL DATA MANIPULATION - Object Language Format

The object language format of the external data manipulation is the interface by which a compiled application program or one of the end-user language processors would access and manipulate database objects at the external data structure level. It can be envisioned as being in the form of subroutine calls which access the records to be processed and then "based variable" reference to the content of the records. Only a "move" mode is considered because the dynamic realization and reformatting, necessary to support data independence, make the "locate" mode impossible. This interface is independent of any programming language but is dependent upon the programming conventions and the procedure mechanism of a particular computer architecture.

#13 INTERNAL DATA DESCRIPTION - Source Format

The source format of the internal data description is the interface by which the Database Administrator makes known his internal data schema to the database system. This source format is edited

by the internal data schema processor module to check for syntax errors, internal and external consistency. The internal data description is described in detail or by reference to elements of other internal data schema. It includes references to the Conceptual Data Schema which are necessary to establish meaning for all data elements. The data description will include specifications as to the size and recording mode of each field; the set implementation technique and ordering rules for each set class; the primary key implementation technique, the clustering criteria, and the file residency of each record class; and other optimization information.

#14 INTERNAL DATA DESCRIPTION - Object Format

The object format of the internal description is the interface in which the internal data schema are cataloged and made known to the rest of the database system after they have been successfully edited by the internal data schema processor module. It is used in this format by the internal data manipulation function module, the internal data utilities, and the internal/external transformation function module. It is also used by the host language formatter modules to create the internal data schema in host language formats.

#15 INTERNAL DATA DESCRIPTION - Host Language Format

The host language format of the internal data description language is the interface used to tell the application programmer what the database at the internal data structure level looks like to him and the operations that it is authorized to execute. It also is used to communicate to his host language compiler the things that it needs to know in order to compile a program to manipulate the database at the internal data level. It is not known at this time whether or not the host language compiler will need to or be able to tell the difference between an internal data description or an external data description.

#16 INTERNAL DATA MANIPULATION - Host Language Format

The source format of the internal data manipulation language is the interface by which an application programmer would describe his access and manipulation of internal data structure objects within his application program. Each such host language (COBOL, FORTRAN, PL/1, etc.) would have its means for expressing the action to be carried out. For example, COBOL might have a "FIND" command as part of its language. On the other hand, the FORTRAN and PL/1 programmer might use the "CALL" command to call the find procedure which is part of the internal data manipulation function module.

#17 INTERNAL DATA UTILITIES - Control Language

This interface provides the site operator with the capability to describe and request operations on a database and its files, records, fields and sets. Generally the purpose would be to copy or compare files, or to reload, reorganize or restructure a data-

base. There would also be facilities for emergency usage to reconstruct damaged records, fields and sets when the normal recovery and restart facilities cannot be used.

#18 INTERNAL DATA MANIPULATION - Object Format

The object format of the internal data manipulation language is the interface by which a compiled program is able to access and manipulate the objects known to the internal data schema. This is envisioned as a subroutine call interface which access the records to be processed and then a "based variable" reference or a working area reference to the content of the record after it is retrieved. In other words, both a "move" mode and a "locate" mode of access are envisioned. This interface is to be independent of any programming language and would be specific to the programming conventions and instructions which implement procedure calls for a particular computer. Figures 4, 5, and 6 are the interface work sheets used by the Study Group as applied to the internal data manipulation interface.

#19 INTERNAL STORAGE DESCRIPTION

This interface describes the objects of internal structure: data sets, pages/blocks, available space inventory mechanisms. Historically the designer of the system specified these, and except for options, they were fixed within a system. They may be customer describable to a much larger extent in the future.

#20 INTERNAL STORAGE UTILITIES - Control Language

This interface provides the site operator with the capability to describe and request operations on the objects of internal storage. Usage would be limited to emergencies where it is necessary to reconstruct damaged catalogs, file labels, page headers and other internal storage structures because the normal recovery and restart facilities cannot be used.

#21 INTERNAL STORAGE MANIPULATION LANGUAGE - Object Form

The internal storage manipulation language is concerned with the handling of records as they are transferred into main memory from the secondary storage devices. These are the blocking and deblocking operations and garbage collection operations necessary to the support of the access of records at the internal data structure level. In some systems, these would be operations within virtual memory. If pages or blocks were specified, then the buffer or page management operations is included. Records are known here by their main storage or virtual memory address. Indexing information would be visible as well as data records. Check sums, block numbers, and other software visible verification facilities would be present.

#22 EXTERNAL STORAGE DESCRIPTION

This interface describes the objects of external storage: volumes, cylinders, tracts, sectors and the labeling information about them. These are largely the descriptions of the hardware aspects of mass storage devices and their controllers.

#23 EXTERNAL STORAGE UTILITIES - Control Language

This interface provides the site operator with the capabilities to describe and request operations on the objects of external storage. Generally the purpose would be to label volumes, copy or compare volumes and reorganize volumes. There would also be facilities for emergency usage to reconstruct damaged volume labels and volume tables of contents when the normal restart and recovery facilities cannot be used.

#24 EXTERNAL STORAGE MANIPULATION LANGUAGE

The external storage manipulation language is concerned with the mechanical, electrical and electronic positioning of the secondary storage devices and the transfer of data between main store and the secondary storage device. This type of interface may come in several flavors depending upon the nature of the secondary storage device. Drums, moving head discs, bulk stores, bubble memories, have somewhat different logical structures, therefore, the interface will probably have to be specialized to the device class. Removable media will have mounting and dismounting operations.

#25 STORAGE DEVICE DESCRIPTION

At this time the author is rather vague about this interface and those following. However, its original intent was to be concerned with the actual physical recording technique, tracks per inch, bits per inch, special recording techniques, check sums, parity bits, cyclical checks, format tracks and the like. This type of thing is built into some of the hardware and is transparent. In other systems, it is variable and subject to either firmware or software control. In either case, if the media is removable then it is necessary to be published in such a way that the removable media may be processed by another system. Tape and disc labels and volume tables of contents also fall in here somewhere. In the end, the more physical or electronic, or magnetic of these items just discussed will probably be transferred to interface #27.

#26 STORAGE DEVICE MANIPULATION

This interface would control the actual hardware commands to read, write, format and position the moveable heads on mass storage devices. Channel programs are an example of this interface.

#27 MATERIALS DESCRIPTION

(See #25 STORAGE DEVICE DESCRIPTION)

#28 MATERIALS MANIPULATION

This interface is the actual electronic manipulation at the recording/reading level on the recording media.

"l". PROGRAMMING LANGUAGE INTERFACE - Host Language Format

This is intended to represent a host language such as COBOL, FORTRAN, or PL/1 which would be intermixed with Data Manipulation Language in the host language format to construct an application program. These languages are referenced to put the others into perspective. It is not the subject of study by the Study Group.

"m". PROGRAMMING LANGUAGE INTERFACE - Compiled Format

This is intended to represent the form of an application program after it has been compiled, linked and is ready to be loaded. This would be standard for a given computer architecture and is referenced here merely to make the other pieces fit together. It is not the subject of study by the Study Group.

"n". JOB CONTROL LANGUAGE

This interface is the mechanism by which an operations clerk can specify his instructions as regard to executing a compiled program as a job. It may in some cases include information relevant to selection files or devices. It is not the subject of study by the Study Group.

DATE: _____

Interface Identification: _____

Requestor Candidates;

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____

Responder Candidates:

1. _____
2. _____
3. _____

Interface Purpose:

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

FIGURE 1

DATE: _____

Interface Identification: _____

User Visible Entities:

DATA STRUCTURE DIAGRAM

FIGURE 2

DATE: _____

Operations on User Visible Entities:

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____
7. _____
8. _____
9. _____
10. _____
11. _____
12. _____
13. _____
14. _____
15. _____
16. _____
17. _____
18. _____
19. _____
20. _____
21. _____
22. _____
23. _____
24. _____

FIGURE 3

INTERFACE NUMBER: 8 DATE: 2/23/73

Interface Identification: Internal Data Manipulation

Requestor Candidates;

1. Database Management - internal/external transformation section
2. Run units (application process)
3. Database Management - internal data structure utilities
4. System procedures: (maintaining control structures)
5.
6.

Responder Candidates:

1. Database Management - internal data manipulation module *
2.
3.

Interface Purpose:

1. To request the storage, insertion, retrieval, modification, removal and destruction of internal records
2. To transfer data as part of the prior purpose
3. To receive notice of success or failure of operations
4. To request locking, unlocking, keeping, freeing of logical records
5. To receive notice of deadlock, or interference

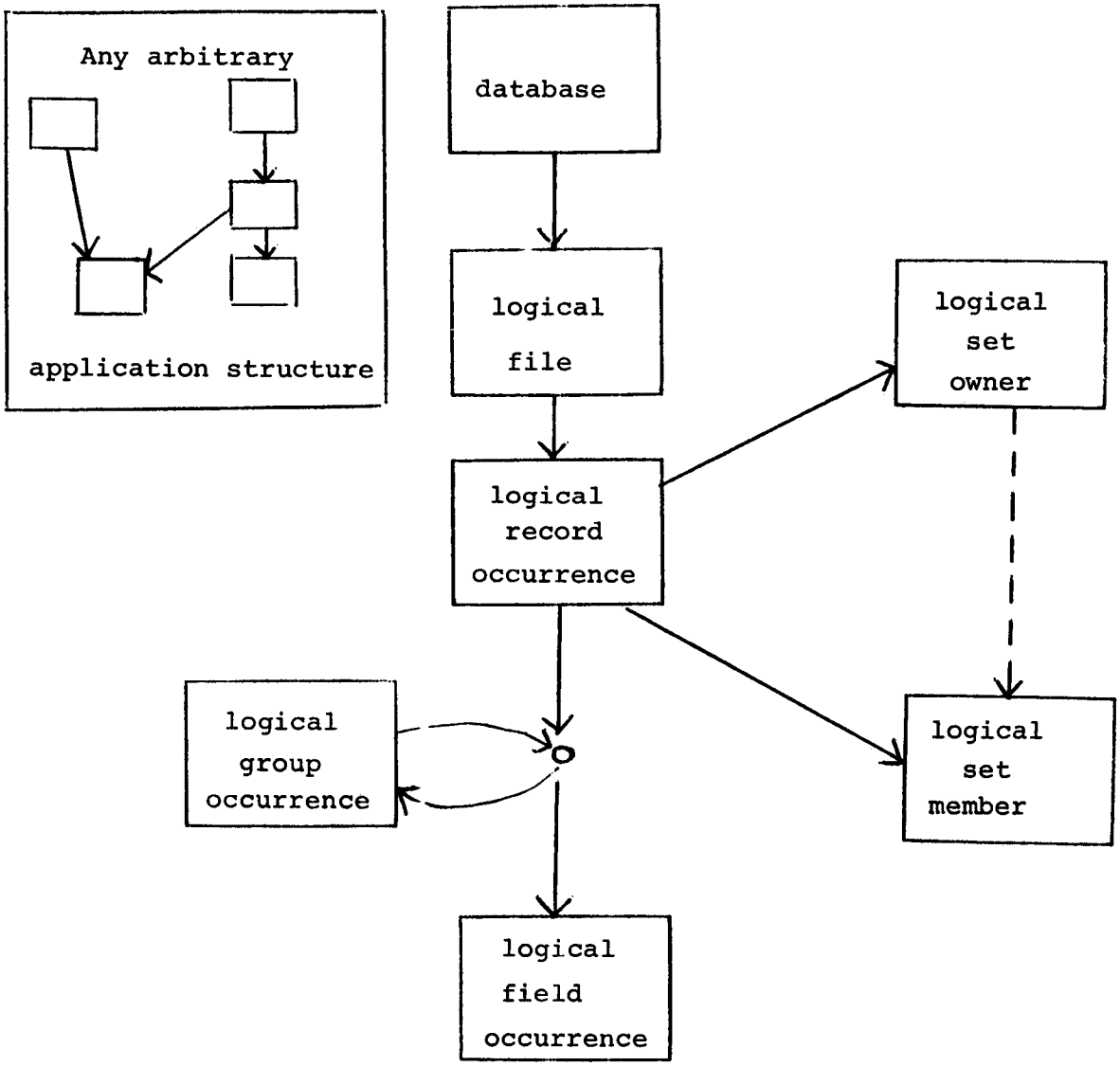
* This is basically a "call" interface which uses entity variables to reference records

FIGURE 4

DATE: 2/23/73

Interface Identification: Logical Data Manipulation Interface

User Visible Entities:



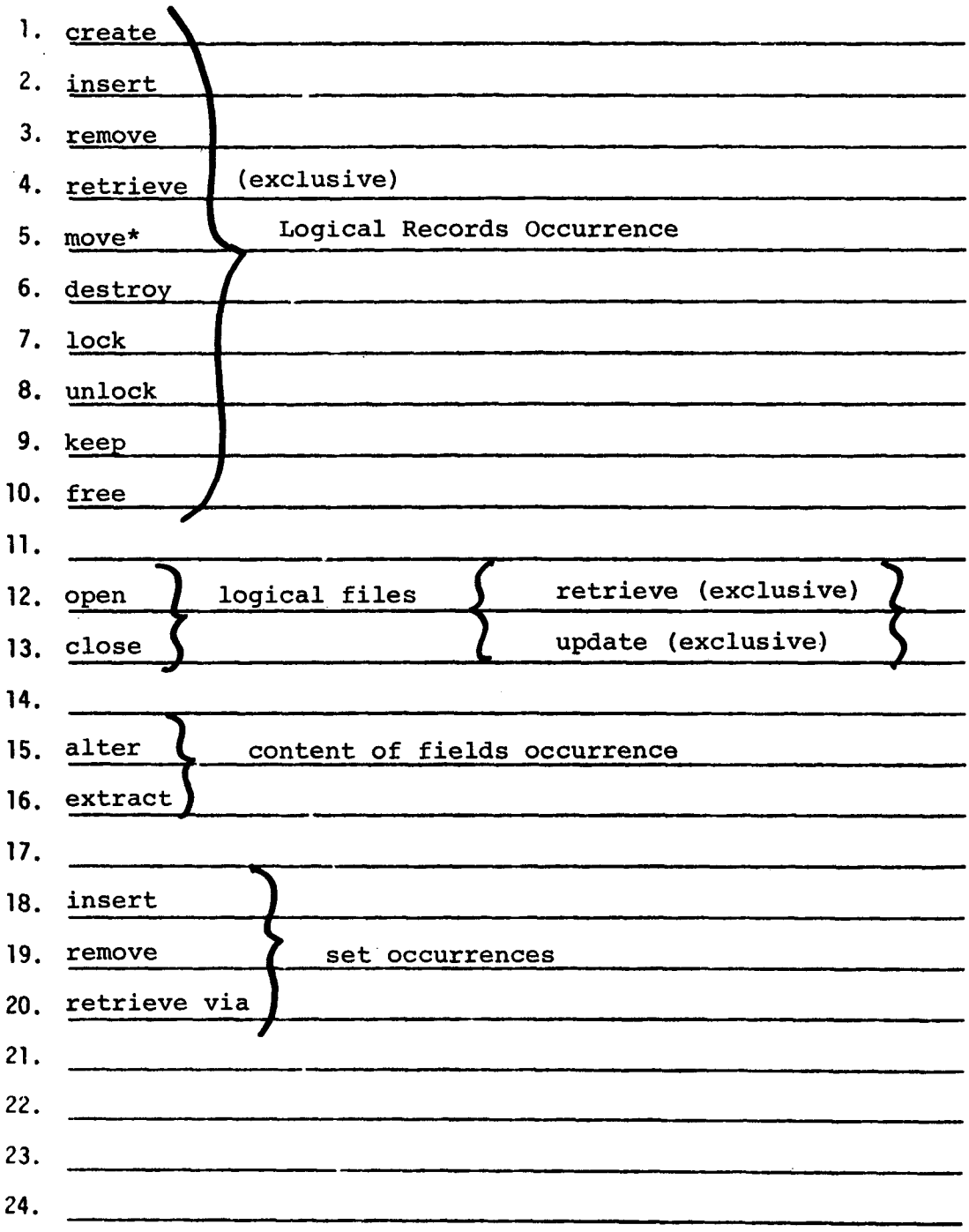
DATA STRUCTURE DIAGRAM

FIGURE 5

DATE: 2/23/73

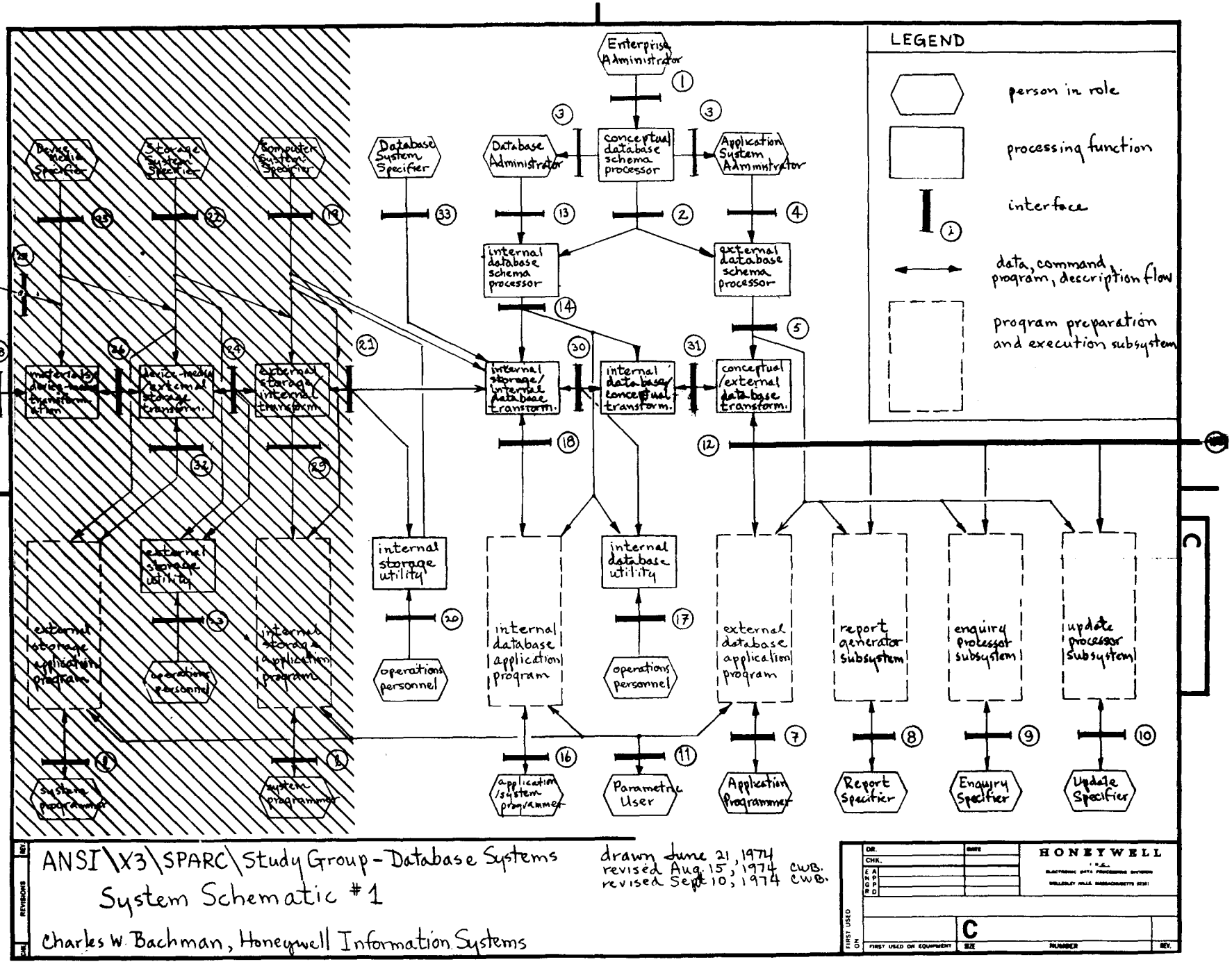
LOGICAL DATA MANIPULATION INTERFACE

Operations on User Visible Entities:



* field access is achieved by moving record to a processing area or through pointer references.

FIGURE 6



ANSI\X3\SPARC\Study Group - Database Systems
System Schematic #1

Charles W. Bachman, Honeywell Information Systems

drawn June 21, 1974
revised Aug. 15, 1974 CWB.
revised Sept 10, 1974 CWB.

HONEYWELL

C