(12) LEVEL III

IDA PAPER P-1489

DATA BASE ACCESS IN C^{3}I COMPUTER NETWORKS

T. C. Bartee
O. P. Buneman

June 1980

Prepared for

Assistant Secretary of Defense
(Communications, Command, Control and Intelligence)

INSTITUTE FOR DEFENSE ANALYSES
SCIENCE AND TECHNOLOGY DIVISION

IDA Log No. HQ 80-22364

The work reported in this document was conducted under contract MDA 903 79 C 0320 for the Department of Defense. The publication of this IDA Paper does not indicate endorsement by the Department of Defense, nor should the contents be construed as reflecting the official position of that agency.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A089 153 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DATA BASE ACCESS IN C^{3}I COMPUTER NETWORKS	5. TYPE OF REPORT & PERIOD COVERED FINAL February 1979- February 1980
	6. PERFORMING ORG. REPORT NUMBER IDA Paper P-1489
7. AUTHOR(s) T.C. Bartee O.P. Buneman	8. CONTRACT OR GRANT NUMBER(s) MDA 903-79 C 0320
9. PERFORMING ORGANIZATION NAME AND ADDRESS Institute For Defense Analyses 400 Army-Navy Drive Arlington, VA 22202	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Task D-23
11. CONTROLLING OFFICE NAME AND ADDRESS Director, Information Systems OUSDRE(C^{3}I), The Pentagon Washington, D.C. 20301	12. REPORT DATE June 1980
	13. NUMBER OF PAGES 90
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE --

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

None

18. SUPPLEMENTARY NOTES

N/A

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data base, protocols, packet systems, query languages, front ends, network languages

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Shared resources in C^{3}I systems require either standard query languages or translators. A standard for C^{3}I data base structures is needed to facilitate future system developments. The transmission control protocol is advancing and front ends are required for system usage.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

403108

IDA PAPER P-1489

DATA BASE ACCESS IN C^3I COMPUTER NETWORKS

T. C. Bartee
O. P. Buneman

June 1980

INSTITUTE FOR DEFENSE ANALYSES
SCIENCE AND TECHNOLOGY DIVISION
400 Army-Navy Drive, Arlington, Virginia 22202

Contract MDA 903 79 C 0320
Task D-23

ACKNOWLEDGMENTS

There were a number of people who contributed to this document through their helpful reviews, comments, and suggestions. Dr. Vinton G. Cerf contributed particularly to the protocol sections and we would like to acknowledge his assistance.

Others are:

Mitre Corporation - Peggy Craft, Steve Lipner, Frank Murphy, and John White

Defense Communications Agency - John Slattery

DIA - LTC N.C. Schwartz and Dennis Martin

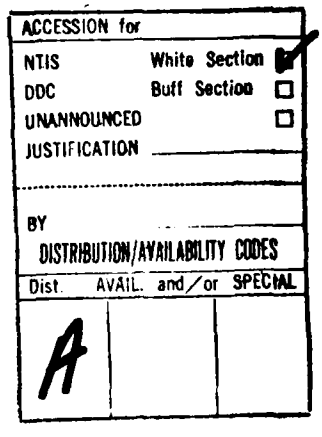
NSA - George Hickens

National Bureau of Standards - Steve Kimbleton and Pearl Wong

Massachusetts Institute of Technology - Mike Hammer

USC/Information Sciences Institute - Dennis McCleod and Dan Cohen

We would like to thank these people for their contributions.



CONTENTS

GLOSSARY

ACM	Association for Computing Machinery
Ada	A DoD Computer Language
ADAPT	ARPA Data Base Access and Presentation Terminal
ADCOM	Aerospace Defense Command
ARPANET	Defense Advanced Research Projects Agency Computer Network
COINS	Community On-Line Intelligence System
DAPLEX	A Programming Language
DARPA	Defense Advanced Research Projects Agency
DBMS	Data Base Management System
DBTG	Data Base Task Group
DCA	Defense Communications Agency
DDL	Data Definition Language
DIA	Defense Intelligence Agency
DIAOLS	Defense Intelligence Agency On-Line System
DTI	Data Transmission, Inc.
EUFID	End User Friendly Interface to Data Management Systems
FQL	Functional Query Language
IDS	Integrated Data Store
IP	Internet Protocol
INTERLISP	A Programming Language
LADDER	Language Access to Distributed Data with Error Recovery
NL	Network Language
NPIC	National Photographic Interpretation Center
NSA	National Security Agency

PASCAL	A Programming Language
PIRL	Photographic Information Retrieval System (NPIC)
PMO	Program Management Office
SOLIS	SIGINT On-Line Information System
TAS	Terminal Access System
TENEX	Operating System for DEC PDP-10 Computer
TCP	Transmission Control Protocol
TILE	TIPS Interrogation Language (NSA)
UDL	Uniform Data Language
UNIX	Operating System for DEC PDP-11 Computer
WWDMS	Worldwide Data Management System
WWMCCS	Worldwide Military Command and Control System
XNDM	Experimental Network Data Manager

More detailed descriptions of some of the terms frequently used in this document are provided below:

DATA BASE

A set of physically or logically related data. The physical relationships include the data structures needed for efficient access and storage; the logical relationships are those that are perceived by the user. With the present state of data base technology, the two relationships (physical and logical) are closely tied. However, this may change if some of the research efforts described in this report are successful, i.e., a user's logical data base may contain several physical data bases. A data base may range in complexity from a single file to an integrated collection of record structures in a "network" data base.

SCHEMA

A data structure that describes the (logical or physical) relationships between data elements. A schema describes facts about the data base that are independent of the data stored therein.

DATA DICTIONARY

A data base or document that both describes the physical structure of a data base and the effects of various applications programs that are used to maintain the data base and generate reports. Data dictionaries are normally used as an organizational aide to data base administrators and to applications programmers.

DATA MANIPULATION LANGUAGE (DML)

The language that enables the data base access routines to be called from a programming language. The term "language" is a nisnomer, for the DML is only the syntax of subroutine calls embedded in a programming language such as COBOL or FORTRAN, and is only of use in conjunction with such a language. This distinguishes it from a query language, which can be used without reference to any other language.

DATA BASE MANAGEMENT SYSTEM (DBMS)

A set of programs that maintains a physical data base. Such programs maintain the linked lists, index structures and other physical structures that are required for efficient access. They provide update and access programs as well as programs for creating the physical data base schema. They provide users with varying degrees of independence of the physical structures of the data base. System R, DBMS 20, IDMS, INGRES are all examples of data base management systems.

The terms "relational," "network," etc., refer to classes of DBMSs with similar property. ADABAS, CODASYL, SAFE, S2000, IMS are other DBMSs referred to.

QUERY LANGUAGE

A language that is specifically designed for data base queries. Unlike DML statements, a query language may be used on its own. Good query languages also maintain an interactive user environment that allows queries to be edited, filed and that provides informative prompts and error messages.

DATA DEFINITION LANGUAGE (DDL)

The formal, machine interpretable, language that is used to describe the physical structure (schema) of a data base.

DATA MODEL

A formal method of describing the logical structure of a data base.

MULTIBASE

A new DARPA project to further distributed data base systems research.

I. SUMMARY

In November 1978, IDA was requested by the Office of the Director, Information Systems, ASD(C^3I), to study user interfaces in forthcoming DoD computer communications networks and to continue an analysis of higher-level protocols that was undertaken in a previous study (Ref. 1). The work reported in Ref. 1, which emphasizes intelligence systems, was to be continued, but attention was also to be turned toward hardware and software developments within the Worldwide Military Command and Control System (WWMCCS) community and other DoD systems.

The development of AUTODIN II continues, and since this communication system will be utilized in many computer networks, our study takes this system as an important factor to be considered.

This study places its primary interest on the user interface with the computer networks in existence and now being developed by DoD. Particular attention has been given to what is called the multi-language retrieval problem. This problem arises because a considerable number of data bases with different data base management systems now exist in DoD. Using the computer networks in existence and being formed, it will be possible for intelligence analysts and other DoD users to access these data bases. However, in order to use the information available in a given data base, it is necessary to learn the query language of the DBMS and structure of the data base. As a result, to use several data bases at different locations, it is necessary to learn several different languages and systems. The problem goes even deeper. For instance, it is sometimes difficult even to determine where data exist and in what form.

The work on protocols has continued. In the report published by IDA in April 1979 (op. cit.), some recommendations concerning protocols were made along with a recommendation that an agency be given control over certain areas of protocol development for DoD computer communications systems. Since that time, DCA has been tasked in the protocol standards area and a users committee has been formed to give further guidance in this area. As a result, our work has concentrated on several aspects of the Transmission Control Protocol (TCP) and Internet Protocol (IP), which were recommended in our previous report. Because of information which has been gained in several tests, we have included some comments on front-end processors for DoD computer networks, a subject which was in our previous report and concerning which data are now available and reported herein.

A. OVERVIEW OF DATA BASE PROBLEMS IN A NETWORK ENVIRONMENT

Making information that is contained in a number of geographically dispersed file systems accessible is a problem on the frontier of present computer technology. The management of such systems must depend primarily upon technology that is presently available, and at the same time must sponsor and carefully monitor technology that is under development. This study attempts to point out areas where actions may now be taken that can aid in network development and also minimize the probability of certain problems arising as systems are developed. The management of systems must also be arranged so that it will be possible to incorporate new technological developments as they appear.

The multi-language retrieval problem results from the development of resource sharing computer networks in DoD that will provide access to both data base and processor capabilities

on computers at locations (and maintained by organizations) external to those of the user. Since these data bases have been developed independently, the data retrieval languages required to access the individual data bases have been chosen (and are sometimes custom made) with the particular applications of the data base and its local users in mind. As a result, a computer network user is faced in some cases with several data retrieval languages, and also computer systems resources that may be unfamiliar to the user. This represents a barrier to realization of the potential benefits of the computer network.

In C^3I systems, the computer networks that are being developed range from clusters of computers in a small area to geographically distributed networks such as the WWMCCS. Early computer-to-computer communications networks were developed for specific communities, and the sponsoring agencies carefully designed these networks to satisfy the particular requirements of the systems users. As time passed, it became apparent that interconnection of these networks would be desirable and that resources available on one network should be made available to users on another network. However, this internetworking can be utilized to the fullest extent only if the users on the various networks are able to access the data bases in a reasonable manner. The problems confronting the user range from different log-on and log-off procedures at the various facilities to learning the data base management system language in each data base in which information lies. In addition, there is a problem confronting the user concerning where needed data base information may reside. Managing C^3I networks in such as way as to develop a plan to deal with this problem is the primary motivation behind this report.

Since even local networks had already encountered this problem, several developments have already been initiated in this area. For example, the COINS network PMO recognized this problem fairly early. In this network there are four possible user languages that confront an analyst desiring to gain information from the files and there are indications this number might more than double in the reasonably near future. To alleviate this problem, the COINS Program Management Office (PMO) gave support to a project called ADAPT (the ARPA Data Base Access and Presentation Terminal), which had previously been funded by DARPA and which was in a developmental state. This project is discussed in this report. Also, because of DCA's early recognition of this problem, the EUFID project (the End-User Friendly Interface to Data management system) has been funded by DCA (among others). This system attempts to develop a natural language interface to data base management systems and our study includes some details and comments on this system and its applicability for future systems. Other systems studied included LADDER (Language Access to Distributed Data with Error Recovery), a more general system developed by DARPA, and MULTIBASE, a new project that DARPA is now undertaking at the Computer Corporation of America.

In considering any existing approach to the multi-language retrieval problem, it is necessary to evaluate the tradeoffs that occur when a specific approach is taken. A particular difficulty arises in making such an evaluation at this time. This is because the systems being studied are either in developmental or preliminary states (MULTIBASE, for example, has only recently been funded; ADAPT is only in the second of four developmental phases; and EUFID is primarily an exploratory project that is considered to be in the forefront of technology in that area). As a result, difficulties in making specific comments are often encountered because systems features that are not present in current versions (or even in initial plans) may

well be incorporated in later versions. We have tried to take this into consideration in our comments and conclusions.

When local networks are interconnected, there are bound to be tradeoffs concerning local network features and global requirements. In an attempt to reconcile these system problems, our previous study recommended that a Transmission Control Protocol called TCP 4, along with an Internet Protocol called IP, be adopted as C^3I standards. The protocols were singled out as the ideal place to start standardization and as the lowest point in the higher-level protocol hierarchy where standardization was necessary for successful host-to-host communication. Since that time both of these protocols have been tested by DCA and some preliminary data concerning performance of these protocols are now available.

Since these protocols are now DoD standards,* we have made some comments in this report concerning their implementation, and have included some details concerning implementation efficiency when these protocols are used. In particular, we emphasize material on TCPs implemented on 11/70s with UNIX operating systems, a widely-used configuration in DoD.

B. CONCLUSIONS

- The state-of-the-art in data base management systems and in translation programs is such that it is now possible to formulate a management strategy that will result in a substantial improvement in C^3I computer network data base access characteristics.

While translation programs that perform "true" wide-range natural language translations cannot be made, translators for the class of query languages that seem most desirable can be

*ASD (C^3I) letter dated 3 April 1980, subject: "Host-to-Host Data Communications Protocols.

made. Along with a highly desirable query language for the user, however, a system strategy must also be developed to reduce development costs and produce good operating characteristics.

In order to alleviate the multi-language problem for computer networks with several DBMSs, there are three broad general approaches. The first is to provide a translator for each user language into each and every DBMS. This requires NM translators, however, where N is the number of different DBMS systems and M is the number of query languages. Further, maintaining this large number of translators would be a continuing problem and many languages would not provide some of the desirable characteristics to be found in some of the DBMSs to be used. As a result, this strategy seems unrealistic and undesirable.

A second approach is to have a standard query language that can be used anywhere in the system. Translation will then be made from this language into each DBMS, requiring only N translators, one for each different DBMS.

A third approach is to have a network language that is always used to transport queries across the network. This language need not be a useful query language, but if it is, it can be used as in the second approach. If, however, the network language is chosen for its general system characteristics and ease of translation, then users can fashion individual languages and translators into the network language so that the individual languages have the characteristics each user community finds most desirable. N + M translators would be required, one for each DBMS, plus one for each query language used.

- If a language can be found to satisfy C^3I needs, then the second approach appears preferable since (1) it is less expensive, and (2) avoids another multi-language problem which might ultimately result from the third approach.

Because the computational power of simple-to-use query languages is usually limited, a combination of the second and

third approaches may be the most effective solution. Technical descriptions of the various types of languages are given in Chapter III.

Following are some desirable characteristics of a query language:

1. The user should have a clear notion of the logical structure of the data, and the query language should be consistent with this structure.
2. The syntax should be as natural as possible and appear intuitively obvious to the user.
3. Multiple variant forms to achieve the same function should be minimized. The syntax should be uniform and consistent.
4. Selectable levels of guidance and system assistance should be provided for users with different degrees of expertise. Expert users should not be forced to use conventions designed for occasional users. Occasional users should be able to use the system comfortably and achieve results comparable to experts, although with more effort.
5. The end-user query language should not appear unpredictable or overly long and unwieldly. At the same time, it should not appear terse or give an effect of being overly clever and obscure in fashioning queries.

- A first step C^3I should take in selecting such a language from those available or producing a new language with ideal characteristics is to obtain a profile of the system users and to determine what is really needed.

It should be noted here that fashioning a query language for a DBMS is not nearly as large a job as making a general-purpose programming language. The DBMS languages are much simpler and the translators are also simpler. The benefits

from a standard query language, however, might well be on the order of those for COBOL, another DoD-developed language.

- An attempt should be made to define a standard query language with full support for occasional users; if possible, this language should also serve as the network query language. In any case, a network language should be developed.

This can be done in parallel with the query language development, and there are existing models for such languages that somewhat simplify this work. Also, the question of whether this network language can be used as an "expert's" query language needs to be reconciled. Again, our management plan deals with this.

A system consideration that arises here concerns whether translation from the standard language or network language should be into the local query languages of the DBMS or into the subroutine calls for the DBMS. Generally, when the translation is into the query language, a new translator must be made each time a new data base is added; however, there is often more uniformity among the subroutines for the DBMS than among query languages. The Data Base Task Group (DBTG)* committee, for example, has standardized subroutine calls but not query languages. This is discussed in Chapters II and III.

An important conclusion concerning the development of a query language or network language for C^3I use concerns what is called the "logical view of data" or "logical model" of data in the system used. When a data base management system is developed there is always a logical or users' view of the data's organization in the data base. At present, there are four

*The Data Base Task Group was organized by the ACM to develop a standard DBMS. There is no standard query language for DBTG systems, but working groups have approached this problem.

data organizations that are in general usage or development. These are: (1) relational (or flat files), (2) hierarchical, (3) network and (4) functional. An introduction to these organizations is presented in Chapter II.

The important fact concerning the logical data organizations is that the query language will heavily reflect the organization used. For example, the network organization is used in what are called DBTG systems (Ref. 2), which include WWDMS (IDS) and a number of commercial systems. The query languages for each of these systems are quite similar. Similarly, the hierarchical systems include several commercial systems (IMS, for example) and the query languages for these systems are also quite similar. The same fact applies to relational systems, including the recently developed IBM System R (Ref. 3). The functional organization is reflected in CCA's language, FQL.

Now, in order to develop a standard query language , some view of the logical structure of the data needs to be taken. There is a problem here for C^3I, however, because, for example, many systems are hierarchical, but WWDMS is a network system. (There are also a number of flat file organizations, mostly in locally developed systems, but these can easily be integrated into either hierarchical or network systems.)

In the Community On-Line Intelligence System (COINS) development of ADAPT, hierarchical systems are all that need be accommodated, so there is no problem. However, attempts to extend ADAPT into WWMCCS and WWDMS could present certain problems.*

- Choosing the data organization model is the most important step for C^3I in developing a standard query language. (Some technical aspects of this area and some further considerations may be found in Chapter II.)

*The authors understand that some recently proposed extensions to ADAPT may circumvent this incompatibility. They are unaware of the details, and the comments in this document refer to the present implementation.

Four projects now underway that are sponsored by DoD and relate to the problem of accessing data in distributed systems are described in Chapter V. These projects are: (1) ADAPT, the COINS project, (2) EUFID, (3) LADDER, and (4) MULTIBASE. ADAPT shows that a standard language can be used in a distributed system with some success. The EUFID and LADDER Projects show that "natural language" query systems can be produced, but leave open some questions concerning their desirability, particularly for expert use. In any case, our observation is that they do not seem ready for adoption at this time. MULTIBASE is a new DARPA project to further distributed data base systems research.

In an attempt to help with the multi-language problem at a different level, a query language, FQL, was described in Ref. 1. This language is based upon a functional model of data, which is described later. The purpose of this was to produce an "intermediate" language that could operate as a network query language against all DBMSs.

The language has since been implemented in PASCAL, and is the interface language for two experimental Natural Language Systems. Unlike query translators, FQL operates through direct calls to the data base subroutines. As a result, the amount of processing involved in interpreting a query is substantially reduced. Also, for reasons described later in this report, transportability problems may be substantially reduced.

The language is mathematical in nature and not suitable for the occasional user. However, it may prove useful as an example of what we have termed a network query language, and should also prove useful as a method of building interfaces between new programming languages such as ADA and PASCAL, and existing DBMSs.

Security and integrity are important issues in C^3I networking and our studies in this area are in an early phase. Some comments concerning the security issue can be found in the sections on natural languages (Chapter IV). This deals

with a need for programmer interaction in constructing new interfaces. Since most of the files are updated locally, or by special purpose applications programs which are not generally available, we have deferred further consideration of integrity until some of the primary issues in networking have been resolved. For example, security measures such as access privileges can only be rigorously defined after the data model has been decided on.

The TCP and IP protocols are progressing through implementation and testing and many implementations are in regular use. The standards have reached a stable level and results are encouraging. Some performance problems* have been noted for UNIX-based TCPs but improvements in UNIX interprocess communication and context switching as well as development of front-end options are potential means for achieving necessary performance objectives.

- Users with smaller systems have need for a front end smaller than that for WWMCCS and, in some cases, an efficient TCP program that can be loaded on their present computers.

The primary front-end project at DCA is WWMCCS-oriented, however, and a relatively large 11/70 with 500 Kbytes of memory is now the primary vehicle for front-end development. While this large system may be necessary for future WWMCCS systems (this front end now supports 70-80 Kbits per second, but even more will be necessary for WWMCCS), it may not be appropriate for smaller systems. This problem could impact AUTODIN II users to a considerable extent unless it is dealt with promptly.

---

*The first tests of TCP indicated an 11/70 was 80 percent occupied at a 6-8 Kbit data rate. This has subsequently been improved, but is a sobering figure. See Chapter V for details.

- The need for a standard front-end-to-host protocol is equally pressing.

Failure to provide such a protocol could lead to the development of several different front ends and their host interface packages such that the host interface packages are not transportable across computer lines and this will lead to a proliferation of interface software packages. Substantiating material may be found in Chapter V, along with some status material in this area.

A topic of considerable interest concerns protocols for local cable bus networks. These are in the formative stages, although a number are already operating successfully. This is a very promising area for DoD and since these small networks will in many cases be internetworked they must be developed with care. A recommendation follows in the next section.

C. RECOMMENDATIONS

The primary recommendations concern a management strategy for developing a language and translation system for C^3I data base management system usage. Following this is a recommendation concerning front ends and one on protocol development for local cable bus networks. Amplification and justification are to be found in the chapters following.

1. Query Languages for DBMS in Networks

a. Three steps need to be taken to determine a desirable query language for C^3I use. These steps can be taken concurrently.

(1) A committee should be formed to determine requirements for a standard query language, and to evaluate existing candidates for this language.

(2) A study to determine a profile of C^3I users of DBMSs should be undertaken. This will facilitate the design or selection of a query language (or family of languages) for C^3I usage.

(3) The logical data model to be used in C^3I systems should be investigated by a study that will examine present and future systems to try and find a data model that can be used over a significant time period. A working group should be instituted for this purpose.

b. A network language suitable for C^3I usage should be decided on and a standard developed. This language should be based on a clean, logical data model that can be used to represent existing data base structures. The data model should be examined. In this step, logical data dictionaries, based on the model, must be defined for each existing DBMS, as this will form an integral part of the translation process.

c. The general structure of the translator system to be used in C^3I should be studied to determine the desirability of translating directly into DBMS subroutine calls versus translating into the query languages for the DBMSs. Since both classes of systems exist, the study should include the advantages and disadvantages of each (some of which are in Chapters II and III) so that a system structure can be determined.

d. Both ADAPT and EUFID constitute attempts to solve the multi-language problem. Funding for these should be continued. The progress of MULTIBASE and LADDER should also be monitored, as both will provide information on this problem.

e. All efforts to solve the multi-language problem through query translation technology must be examined with a view toward security, and this subject is described later in this report. Translation methods with respect to security problems should be carefully evaluated by a working group formed by C^3I.

2. Front-End Development

It is crucial that a standard front end be developed for C^3I usage in connection with AUTODIN II. This should be done as soon as possible so that computer users do not suffer unreasonable losses in throughput when connecting to AUTODIN II.

Chapter V enlarges upon this problem. Equally important is the adoption of a standard protocol for the front-end-to-host interface. Justification for this recommendation can be found in Chapter V.

3. Cable Bus Network Protocols

The development of cable bus networks will provide C^3I, and in particular WWMCCS, with a considerable potential for improved operations. The protocols used in these local networks should be monitored carefully and, if possible, should be developed around some standards framework. A leading question concerns the use of TCP as a standard here, because it is fairly complex and may provide local networks features not normally needed. However, internetting will require its usage. Some coordination of cable bus nets in WWMCCS and other developing intelligence networks is called for.

The use of TCP for local nets should be encouraged, first because it unifies communication for intra- and internet communication and second because overhead such as header size is often not important in a multi-megabit local net. Also, most of the TCP error recovery mechanisms are needed due to packet loss from errors and contention needs, sequencing, and flow control. It is possible to make a case against IP for intranet, but to present a uniform interface for security systems (regardless of traffic destination) it may be reasonable to include IP for intranet traffic, too.

The following sections contain considerable technical detail. Although we have included a glossary, the vocabulary and concepts presuppose some knowledge in data base and computer networks.

II. DATA MODELS

Any successful solution to the multi-language problem must be based upon a simple logical data model. In this chapter the well-known models are reviewed and considerations are presented toward their advisability as part of the solution of the multi-language problem.

When a data base is constructed, the designer defines structure among the data for two reasons. In the first place, the data must be represented on the storage media in a way that is economical in storage and that makes for efficient access and update. Secondly, the structure should reflect "real-world" relationships among the data. These two structures are called the physical and logical schemas. While there is no reason for these two schemas to be similar, in practice they are usually closely related. Most data base management systems provide the designer with a set of basic structures out of which the data base schema, a more complicated structure, may be built. At the logical level, these building blocks are called the logical model. A very thorough description of data base models is to be found in Date (Ref. 4); only the more important models and their relationship to query languages are reviewed here. Also, since we are concerned primarily with data base queries, we emphasize logical models.

A. THE RELATIONAL MODEL

The relational model (Ref. 5) was designed to simplify user access and transportation of data. It is strictly a logical model, and the underlying representation may involve linked

lists, hash tables or other methods of improving efficiency. Informally, a relation is a rectangular table whose entries are "printable" objects; no internal pointers are permitted.

DEPARTMENT

DEPT#	DNAME	CITY
23	Manuf.	Boston
7	Sales	Washington
:	:	:
:	:	:

EMPLOYEE

EMP#	DEPT#	ENAME
132	7	Jones
456	7	Smith
890	23	Doe
:	:	:
:	:	:

VEHICLE

V#	DEPT#	VTYPE
84	7	Ford
96	23	Mack
:	:	:
:	:	:

JOURNEY

EMP#	V#	TIME	DISTANCE
890	96	040779	47
132	74	050979	78
:	:	:	
:	:	:	

FIGURE 1. A relational data base

Figure 1 shows an idealized relational data base containing information about departments, employees, vehicles and journeys (that the employees take in company vehicles). Formally, a relational data base consists of a set of <u>domains</u> [D_1, D_2, ..] and a set of <u>relations</u> [R_1, R_2, ..] each of which is a subset of a cartesian product of domains. That is, $R \subset Di_1 \times Di_2 \times Di_3$ ... In this example, the domains are EMP#, V#, DNAME, etc., and the relations are DEPARTMENT, EMPLOYEE, VEHICLE, JOURNEY. The structure of this relational data base would normally be described by:

DEPARTMENT(DEPT#,DNAME,CITY)

EMPLOYEE(EMP#,DEPT#,ENAME)

VEHICLE(V#,DEPT#,VTYPE)

JOURNEY(EMP#,V#,TIME,DISTANCE)

Relational query languages provide a set of operations that manufacture new relations out of existing ones. They are relevant in this context because many of the existing C^3I user languages may be viewed as subsets of these operators. The three most important operators are the following:

1. Restriction. Select a subset of the rows of a relation according to some predicate.

   Restrict VEHICLE with VTYPE = 'Ford'

   creates a new relation containing just those tuples that contain 'Ford' as the VTYPE. This relation will have the same number of columns as the original.

2. Projection. Select a subset of the columns; for example:

   Project JOURNEY on V#, DEPT#

   which yields a relation containing two columns. Note that this relation will, in general, also contain fewer rows. A given V# and EMP# will appear at most once in the result, even if employee EMP# has made several journeys in vehicle V#.

3. Join. This allows the combination of two relations into a new relation. For example, if we require finding the name of the department in which an employee works, we would first construct the new relation:

   Join EMPLOYEE, DEPARTMENT on DEPT#

   to obtain a new relation with domains (DEPT#,DNAME, CITY, EMP#, EMPNAME). A suitable projection and restriction may then be used to provide the appropriate result.

The syntax used here is not representative of any particular language; it is just meant to convey the basic relational operations. The relational languages will be covered in more detail in a later section. It should be noted here though that many simple query languages, especially those that operate on flat files (ISS) provide a projection and restriction capability. The user of these must therefore have a view of the data base that approximates the relational model.

The relational model has been criticized (Ref. 6) for its semantic weakness: there are simple constraints that cannot be specified in the relational schema. For example, it would be desirable to require that any DEPT# appearing in the VEHICLE or EMPLOYEE relation should also appear in the DEPARTMENT relation. If this constraint is not enforced, there is a possibility of employees being in "undefined" departments. Another drawback lies in the implementation of most relational systems that do not treat domains properly. In relational implementations it is possible to join on any two columns which have the same type. For example, joining DEPARTMENT to VEHICLE with DNAME=VTYPE (both columns are character strings) is allowed, even though such a join does not have any meaning. However, the relational model is by far the simplest and has proved of great value in describing problems associated with distributed data.

B. THE NETWORK MODEL

The details of this model are described in the DBTG report (Ref. 2). The model is often called CODASYL, after the committee that developed it. The elements of the model are record classes, attributes (sometimes called items) and sets. A record class describes a set of records having the same set of attributes. Informally, a record class resembles a relation. However, records in different classes may be related through a set. A set is nothing more than a many-one relationship between the records of two distinct classes.

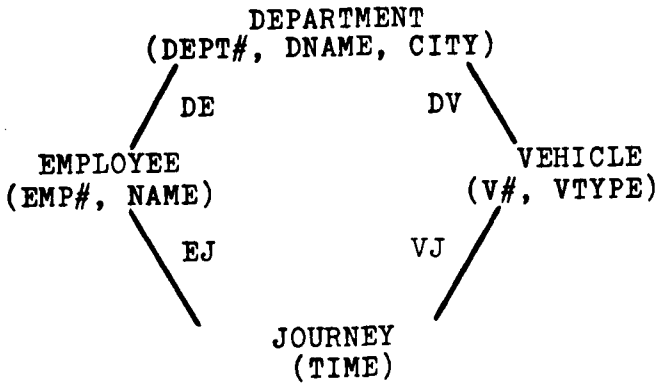
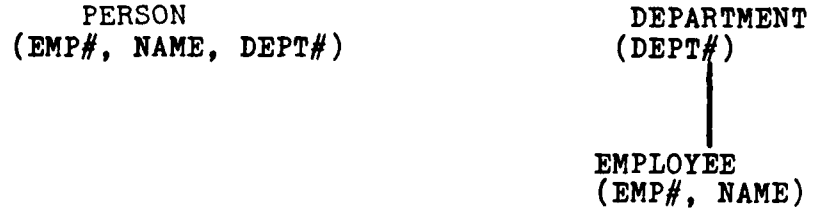


FIGURE 2. A network schema

The example in Fig. 2 describes the same data base that was used in the relational example. At the top of this diagram, there is a DEPARTMENT record class, with attributes DEPT#, DNAME, CITY. Each record in this class "owns" (through the set DE) a set of records in the EMPLOYEE class, and, through DV, a set of vehicle records. The rules of set ownership specify that no two records in the "owning class" may have a common record in the "owned" class. Thus, since no two departments can own the same employee, an employee may be in at most one department. Set ownership therefore specifies a many-one relationship between record classes. Note that, in contrast to the relational model, many-record classes have fewer attributes than their relational counterpart. The other fields may be through owning records. Thus, the DEPT# of an employee is determined by finding the owning record in the DE link.

It is also possible, in the CODASYL model, to ensure that <u>every</u> record in an owned class has an owner in the owning class. Thus, it is possible to enforce the constraint (lacking in the relational model) that each employee must be in some department.

A criticism of CODASYL as a logical model is that logical relationships in the data may be represented in more than one way. Consider the two schemas



Both schemas represent the same information; however, in one case the department number of a field is represented as a field, while in the other case, a set is used. The two representations will make a crucial difference to the efficiency with which a given query can be executed, but it is unlikely that an end user will wish to be acquainted with such details unless absolutely necessary.

CODASYL systems require highly-trained personnel to design and manage them. Even the degree of technical competence needed to query the data base is quite high. There is a common practice of keeping "stripped files," a flat-file representation of part of the data base, which is periodically updated (FORSTAT is an example). Such files simplify the problem of querying a selected subset of the data base, and may also improve the efficiency. However, the practice is sometimes unfortunate because it means that the user can be provided with data that are out of date* and because queries written against the flat file generally will not run against the full file and vice versa.

C. OTHER MODELS

In this section we shall briefly review other data base management systems and the logical models associated with them.

1. Hierarchical Systems

A hierarchical system may be viewed in the same terms as a CODASYL system. There is a restriction that no class may participate as an owned set in more than one set relationship.

---

*In some cases, the full file is updated continuously (as changes come in) but the stripped file is only updated at set times.

This means that the schema of a hierarchical system is tree-like and in the CODASYL example above, the JOURNEY record class would not be allowed to participate in both the EJ and VJ sets. To convert to a hierarchical schema, one of these links would have to be broken, or the JOURNEY record class would have to be duplicated, and a key added to the two resulting record classes in order to denote the correspondence.

IBM's IMS is a well-known example of a hierarchical DBMS. The NSA-supported ADAPT system (see Chapter V) uses a hierarchical model. The criticisms that hold for the CODASYL data model also hold for the hierarchical model.

2. Fixed Schema Models

A number of C^3I data bases are designed for very specific uses. In particular, the document retrieval systems such as SOLIS, PIRL and DSRS are designed for the efficient storage and retrieval of documents by name and subject matter. Such systems provide the user with a fixed schema. Although a general-purpose DBMS could in principle be used as a storage medium, the resulting data base would be more cumbersome to manipulate than that provided by the special purpose systems.

3. "Flat File" Systems

It was claimed in the discussion of the relational model that a number of file manipulation programs could be viewed as relational operators. This is not quite true. A number of flat file systems (ISS for example) lie somewhere between the relational and hierarchical view. Variable length records are allowed and some of the fields may be repeating: this means that the field may contain a sequence of similarly formatted values. Such a file may be viewed as a two-level hierarchy. The presence of repeating fields considerably complicates the query languages that are commonly used against such files.

4. Inverted Systems

Inversion is an implementation technique and has nothing to do with logical models of the data base. However, several

DBMSs are advertised as "inverted" or "fully-inverted" (ADABAS and a WWDMS supported sub-system). In practice, knowing which fields are inverted is extremely important in the efficient execution of queries.

D. THE FUNCTIONAL MODEL

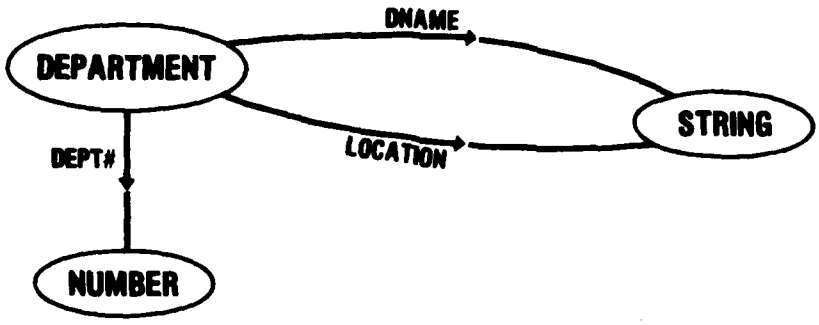
The functional model was first discussed by Kerschberg and Sibley (Ref. 7), and is important here because it has been proposed independently as a method of producing a unified view of all the data base management systems and data models that are currently in use. One of these proposals was briefly described in a previous report (Ref. 1). Another is contained in the MULTIBASE project, which will be reviewed in this report. While the query languages differ greatly in syntax, the underlying primitives are similar, and the data models, with minor differences, are the same.

According to the functional model, a data base contains a collection of types and functions. (In DAPLEX, the language of MULTIBASE, the word "entity" is used instead of "type".) Each function maps one type into another, and it is assumed that there are a few primitive types that are common to all systems. These are types such as STRING (character string), NUMBER, BOOLEAN, etc. The types that the data base provide usually correspond to our traditional notion of record class.

In this model, a department would be represented by a DEPARTMENT entity, with the following functions:

```
        DEPT#:  DEPARTMENT → NUMBER
         DNAME:  DEPARTMENT → STRING
      LOCATION:  DEPARTMENT → STRING
```

or diagrammatically,

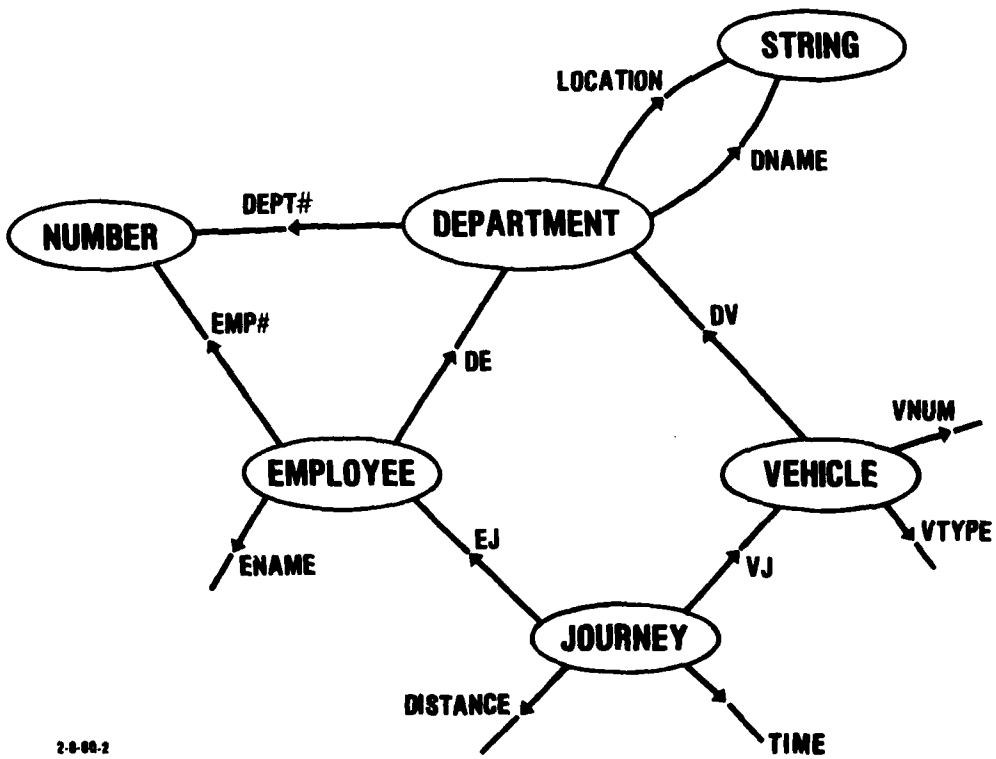


2-8-86-1

Moreover, at least in the CODASYL representation of the data base, we may view the relationship between EMPLOYEE and DEPARTMENT as another function: DE, so that

DE: EMPLOYEE → DEPARTMENT

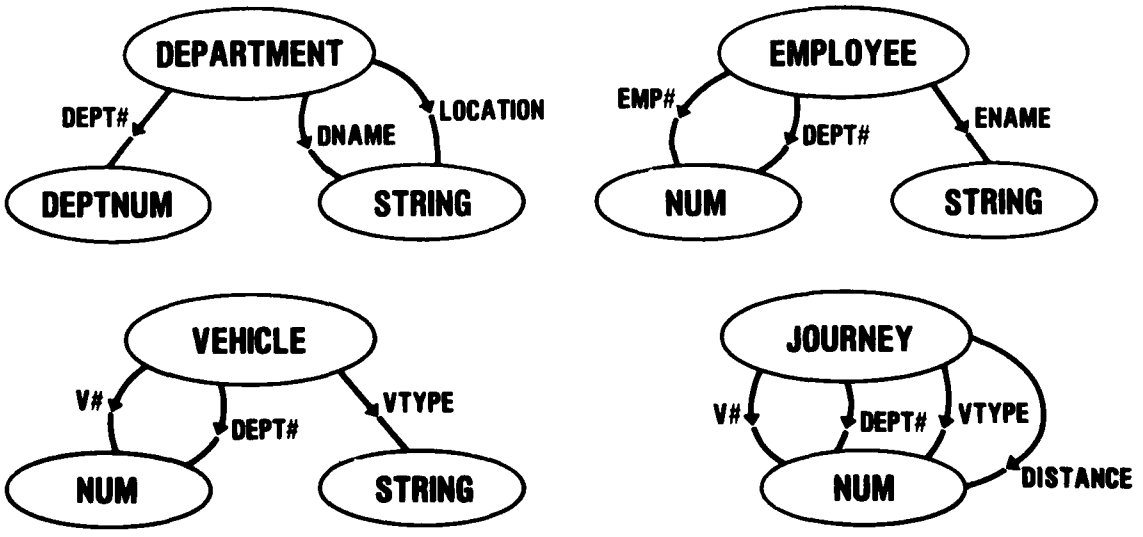
and if we represent other CODASYL sets in the same way, we may construct a diagram of the whole data base as:



2-8-86-2

where functions have been, for simplicity, left with unspecified ranges. Such diagrams can become unwieldy, and it may be that the type of data definition language supplied by CODASYL (see Chapter IIID) would be simpler to follow. Associated with each function there may be an inverse. The inverse of a function may be treated as another function whose range is a set or sequence. Thus, DE^{-1} is a function on DEPARTMENT which produces a set of EMPLOYEEs. (The set of employees in that department.) Whether or not all the functions in the schema have inverses depends upon the query language. In a low-level implementation, the user would be aware that only some functions have inverses, and would have to write code to determine the inverse himself. In a higher-level system, the most efficient method of computing the inverse could be determined automatically. Note that sets, hash tables, etc., are all methods of representing the inverse of a function.

The functional representation of the relational model would be the following



2-8-80-3

Note that the two representations of the same collection of data are different. This is as it should be, for the two models reflect differing constraints on the data. In this sense, the functional model is rather low-level, but for the purpose of interfaces to heterogeneous data base systems it has two important advantages.

1. It may be used to represent the logical models of all commonly available DBMSs.
2. The translation from the schema of the data base to the functional model may be performed automatically. Therefore, a query language based on the functional model should require no additional maintenance or pose the threat to security mentioned in the introduction.

Although, in our example, the functional models for the CODASYL and relational data bases are different, it is possible to turn one model into the other by a process of function definition. In the CODASYL model, there is no direct function DEPT# on EMPLOYEE, but it may be defined by the composition of the functions DEPT# and D.

E. MAPPING BETWEEN MODELS

This subject has been extensively researched (Ref. 8). The simplest models upon which a uniform query language can be based, in our opinion, are the relational and functional models. We have shown how the relational model may be mapped into the functional model, and a general technique is described in (Refs. 9 and 10). Can a mapping in the other direction be constructed? By answering this question, we would also provide an answer to mapping CODASYL and hierarchical structures into relational structures, since there is a mapping from CODASYL, etc., into functional.

The answer to the question is not simple. Roughly speaking, it is true that a relational model can always be constructed from a functional model, but that the result is not

unique. Practically, this means that the mapping cannot be performed automatically: some intelligent intervention is required in order to decide which is the "best" set of relations to represent the functional schema. The problem of producing relations from a functional schema is the same as choosing a set of "stripped" files that will fully represent the contents of a CODASYL system.

III. QUERY LANGUAGES

A. INTRODUCTION

Data base query languages are more varied than at first seems reasonable: terms such as "high-level," "low-level," "procedural," "non-procedural" are often used to describe a query language, but give very little idea of its power, convenience or efficiency. Some languages bear a close resemblance to conventional programming languages and allow direct access to the physical data manipulation procedures of the DBMS. Such languages usually provide for arbitrary computations, i.e., they can be used as conventional programming languages, but require a considerable investment of time and expertise to be at all useful. Other languages provide a much simpler format for queries, but almost invariably lack computational power. In this chapter we review a representative selection of query languages and attempt to draw some broad comparisons.

The discussion will concentrate on query languages. A data base query is, by definition, a procedure that does not modify the data base. Thus, in a query language, there is no need to provide update mechanisms for the data base. By generalization, it can be argued that there is no need to provide update mechanisms for any structure; that is, there is no need for an assignment operator in the language. The assignment operator is the operator that allows variables or structures to be modified: "X := 1;" and "A[3] := 5;" are examples of assignment statements. It is unfortunate that of the currently used query languages, the only languages that allow full computational power are those that contain an assignment statement, and these are the most difficult to learn. Languages without

an assignment statement are usually much cleaner, at least at a syntatic level, but lack some computational power. There seems to be no reason why this should be the case.

Before looking at individual languages, it will be useful to have some specific criteria to apply to query languages in order to establish useful comparisons among them.

B. FEATURES OF QUERY LANGUAGES

The purpose of studying query languages in this document is to try to answer two questions: Is there a query language, or could one be built, that will serve as a uniform interface method for all C^3I data bases? What languages should be considered for the variety of C^3I end users? The following subsections will cover a number of related criteria that can be applied to query languages, and the associated problems.

1. Computational Power

Many query languages, while providing a very simple formalism for data base access, are remarkably weak in other respects. For example, in most relational* query languages it is impossible to ask for a simple arithmetic expression to be evaluated. This is a serious drawback in a number of practical situations. For example, if an averaging function has not been defined, it is impossible for the user to give a formal definition of how an average is to be computed. The computation of simple statistics and of geographical distances present similar problems.

---

*The term "relational completeness" is often used (Refs. 11 and 12) to describe the power of a relational query language. This is a technical term that describes the power of operations on the data. It has little to do with its power as a general-purpose programming language.

Another weakness lies in the inability of languages to perform recursive traversals of the data (Ref. 13). Many problems associated with a bill-of-materials processor are recursive, and cannot be expressed in most simple languages. These difficulties could, in part, be overcome by a good interface to some other high-level, preferably interactive, programming language. However, embedding a high-level query language in an existing programming language is a difficult task, and the result may be more confusing to the user than having just the programming language and the low-level data base access routines.

Some of the languages reviewed below may be used as general-purpose programming languages. This is true of some of the CODASYL-based languages. However, the programming constructs (the ability to define new procedures and data types, etc.) are quite weak. As a result, the statement of some simple queries can be extremely complicated.

2. Transportability

Transportability is the ability of a data base query language to work against different data base management systems. None of the languages that work by directly calling the data manipulation subroutines have been designed to be transportable. However, some systems that work by query translation have been used against more than one DBMS. These include ADAPT and some of the Natural Language systems. Building an interface to a new DBMS requires technical expertise, but even once the interface for a DBMS is established, transporting the query language to a new data base is not always straightforward. In fact, none of the so-called "transportable" systems can be run against a new data base without some programmer intervention. Two questions are relevant to transportable systems:

1. What structures have to be created in order for the query language to run against a new data base? Invariably, a schema must be defined in the format required for the query language. In some cases, such as NL systems, a lexicon must also be created.

2. Who does the transporting? In the case of NL systems, it cannot at present be done by the data base administrator. An expert in the NL system itself is usually required to spend several weeks, perhaps months, understanding both the structure and contents of the data base, and the needs of the user community.

The need to redescribe the schema is in part a failure of the DBMS. Very few DBMSs have the power to describe their own structure. In other words, data base interface routines that answer questions about the schema are usually absent.

3. Querying the Schema

The ability to query the schema is important for several reasons. It is not uncommon for a technically competent user, i.e., one who understands the query language and the DBMS, to be confronted with a new data base about which he understands very little. Even if he knows that certain items are available (for example, VEHICLE and DEPARTMENT), he may not know how they are structurally related. To find this out, the usual strategy is to read the Data Definition Language (DDL) of the schema or some relevent subschema. This is a laborious task and often solved by making "hard" copies of the DDL available: this is not an attractive idea in a network environment.

Queries against the schema are also important for the reasons suggested above: if it were possible, by some means, to issue a query or call a subroutine, that returned information about the structure of the data base, many of the problems of translating between query languages could be solved automatically. Some DBMSs now provide on-line data dictionaries that describe both the physical structure of the data base and comments about the logical structure. While these are largely used to meet the organizational demands on data base administrators, it is possible that they could be adopted for interactive use by users unacquainted with the data base.

While "browsing" through the structure of a data base may be undesirable in a classified environment, a good security system should be able to control access to information about the structure in much the same way that information about the contents is presently controlled. The present situation, in which schema mappings have to be created "by hand" in order to create a new interface to a query language, may provide a greater threat to security.

4. User Convenience

What constitutes a "convenient" query language is obviously an extremely subjective matter. In particular, we reserve a discussion of NL systems until later in this section. It is worth briefly examining this issue in the more general area of programming languages. The following general points apply to programming languages and should apply equally to query languages.

1. Interactive systems are desirable. In particular, a system in which the user is always "talking to" the same command set (such as BASIC, LISP, APL) (Refs. 14 and 15) is preferable to one in which the user must jump between several systems, such as an editor, the operating system executive, and the program loader. This usually happens in a standard compile-load-and-go programming environment.
2. Interpreters (or incremental compilers) provide an ease of debugging that is not usually achieved with standard compilers. In particular, such systems allow a running program to be interrupted and the user to examine what its state is. This is an invaluable debugging aid. Although interpreters are usually slower than compilers, the reduced debugging time often compensates for the increased run time. Another observation relevant to query languages is that most elapsed time is due to the delays in i/o. As far as the user is concerned, delays in i/o will usually dominate all other processing.

3. A good "front end" is desirable. Languages such as BASIC, APL, and some versions of LISP all have a set of "system" or "work space" commands, which make the incremental development of programs extremely simple. As far as we know, data base query systems have done little to emulate this. Other interactive support, such as screen editors, speech understanding and touch interfaces are beyond the scope of this report.

5. Semantic (and Syntactic) Simplicity

Given that NL is not to be the language of communication, a simple syntax and semantics for the query language is desirable for many reasons. In the first place, a language that is simple in this sense is easy to learn. Secondly, since much data base querying is done by translation, the process of translation is greatly simplified if the syntax is reasonably simple. The relational languages without exception have a very clean structure because they are based upon the relational calculus, which itself is very simple.

Unfortunately, the commonly used CODASYL-based languages are extremely messy. Part of the problem lies with data currency, which is probably the most confusing issue in these languages. CODASYL, and a number of related DBMSs, communicate with secondary storage through a common area known as working storage. This storage contains room for precisely one instance of each record type in the data base. In some languages (DMS-20, for example) the user has to be aware of what record is current (in working storage) in order to program a query. This means that very simple queries that involve comparisons between two records of the same type are extremely difficult to program. The user must explicitly copy information from working storage into some other area. Other languages, such as the WWDMS language and ADAPT, manage to protect the user from worrying about data currency for simple queries, but in a complex traversal of the data base, it again creates problems.

6. Report Generation

A query language without a good method of producing a hard-copy report is usually deemed useless. Why are reports useful? Here are three reasons:

1. Reports provide a useful starting point for further analysis.
2. One can browse through reports easily.
3. Reports provide useful back-up. (They are also more transportable than terminals.

However, reports may be unnecessarily lengthy, or needlessly generated, because the query language, or applications program, is not sufficiently sophisticated to provide the required information. With the increased use of display terminals, there is likely to be a decrease in the demand for hard copy. However, in the short term a report generator, especially for display terminals, is essential.

C. RELATIONAL LANGUAGES

A good survey of relational languages is to be found in Ullman (Ref. 16). Although no relational systems are currently used within C^3I, it is possible that a relational system will be used at a later date.* Moreover, several of the available query languages may be better understood by comparing them to relational languages.

The relational model and the operators of the relational calculus were introduced in Chapter II. The operators: join, projection and restriction, do not constitute a complete set; set union and differences are also required. However, the majority of queries can usually be accomplished using the first three.

This set of operators, originally devised by Codd (Ref. 5), is called the relational calculus. Each operator takes one or more relations as argument and produces a relation as a result;

*System R may soon be available commercially.

a relational query is formed by composing these operators to produce a relation that (by default) is printed. The user of a relational system, in theory, need have no knowledge of what computations are being performed to create the required results. This is one example of a functional programming system, in which programs are written simply by composing a set of predefined functions (in this case, the relational operators).

1. SEQUEL

A number of relational languages have been implemented, these include SQUARE (Ref. 17), ISBL, SEQUEL (Ref. 18) and QBE. Of these, the most commonly used is probably the language SEQUEL, which is the query language for INGRES (Ref. 19), and the basis for the query language of System R (Ref. 3).

The basic form of a SEQUEL query is:

SELECT d FROM r WHERE p .

In this, d is a list of one or more domains; r is a relation, and p is a predicate which applies to each tuple (row) of the relation. If the "WHERE p" clause is missing, all tuples from r are selected.

For example, in the DEPARTMENT - EMPLOYEE - VEHICLE - JOURNEY data base of Chapter II,

SELECT VTYPE FROM VEHICLE WHERE DEPT# = 7

creates a one-column relation consisting of the vehicle types of vehicles owned by department #7. This type of statement gives the power to perform a projection (specified by d) and a restriction (specified by p) on a relation (r). Compare this with

SELECT V#, VTYPE FROM VEHICLE WHERE DEPT# = 7 .

This will produce a two-column relation of vehicle numbers and vehicle types. Since there are, in general, more vehicles than vehicle types, the second query should generate a relation with more rows. However, in SEQUEL, the two relations produced are

the same length; that is, duplicates are not removed from the result of a SELECT operation, and strictly speaking, the result of a SELECT operation is not a relation. In order to achieve elimination of duplicates the keyword UNIQUE is used:

```
SELECT UNIQUE VTYPE FROM VEHICLE WHERE DEPT# = 7 .
```

There are two reasons for not eliminating duplicates automatically: one is that it is computationally expensive to do so, and the other is that a pure set-based language makes certain useful operations awkward to incorporate into the language. For example, suppose we wish to find the average distance for trips that used vehicle with V# 96. We could create the sequence* of distances with

```
SELECT DISTANCE FROM JOURNEY WHERE V# = 96  .
```

Had the keyword UNIQUE been used, we would only create the set of distinct distances, which is not the appropriate argument for an averaging function.

Joins, that is, queries that span more than one relation, are achieved by suitably defining the predicate in the WHERE clause. Suppose, for example, that we wish to find the names of departments that own a Ford.

```
SELECT DNAME FROM DEPARTMENT
  WHERE DEPT# IN
    SELECT DEPT# FROM VEHICLE
      WHERE VTYPE = FORD  .
```

*Strictly speaking, a bag is the correct mathematical structure: it may contain duplicates, but does not have an ordering.

Again, note that this will produce a set of department names with duplicates. In order to remove the duplicates, we would write "SELECT UNIQUE DNAME..." If we do this, it makes no difference whether a "UNIQUE" is inserted after the second SELECT in this query. For reasons of (time) efficiency, one would omit this.

It is frequently desirable to introduce variables into a query. While this is not needed in the pure relational algebra, it is needed when extensions to the algebra (such as the aggregation operators sum and average) are introduced. Thus, to find the names of employees who have driven more than 1000 miles, we introduce a variable X, which ranges over tuples of the EMPLOYEE relation:

```
SELECT NAME FROM EMPLOYEE X WHERE
  1000 <
  SELECT SUM(DISTANCE) FROM JOURNEY
    WHERE EMP# = X.EMP#
```

The variable X ranges over tuples of the EMPLOYEE relation.

In addition to the operations described above, SEQUEL contains a number of other operators that give it the same power as the relational algebra.

2. Limitations of Relational Query Languages

We have described SEQUEL because it is a language that provides for a large range of data base queries. Also, its syntax is remarkably similar to a number of non-relational languages (query languages that work against data bases that are not in a relational form). Other relational languages provide similar extensions to the pure relational algebra. It is worth briefly measuring SEQUEL against the yardsticks described at the beginning of this chapter.

1. Computational Power. The pure relational algebra is, as we have noted, extremely limited. It does not contain aggregation primitives, nor can simple arithmetic expressions be defined or evaluated outside of a relational query. SEQUEL corrects this by introducing some aggregation operators, but in doing so, must deviate from the principles of relational algebra by operating on sequences, or bags, rather than sets. Moreover, the addition of these operators introduces a certain syntatic confusion: why do we say "SELECT SUM(DISTANCE) FROM..." rather than "SUM (SELECT DISTANCE FROM...)"? SEQUEL does not allow the user to define new functions. For example, in a nautical data base, if the function great-circle-distance is not defined, it would surely be desirable to define it, and to keep it as "data". In a similar fashion, it would be desirable to introduce MILEAGE (the total number of miles a vehicle has been driven) as a "virtual" domain of the VEHICLE relation. Some of these deficits are corrected in System R.
2. Transportability. SEQUEL, like other relational languages, is defined for a relational data base system. It is not designed to be transportable across DBMSs. However, some languages that are designed for this kind of transportability (ADAPT, for example) have a syntax that is not unlike that of SEQUEL. Thus, it is not unreasonable that a SEQUEL-like syntax could be used in a heterogeneous environment.
3. Querying the Schema. This is not considered part of the relational calculus. However, it is not hard to imagine the schema being represented in a relation whose definition is

   SCHEMA(RELATION-NAME,DOMAIN-NAME,DOMAIN-TYPE)

   or some equivalent construction. There are some more

sophisticated schema-querying proposals by Smith and Smith at CCA.

4. User Convenience and Simplicity. One of the advantages of the relational algebra is that it is extremely simple. This is reflected in the query languages that are based upon it. No one would claim that the CODASYL-based languages are "simpler" than the relational languages. As for user convenience, INGRES is provided with a good user interface, as is System R.
5. Report Generation. In one sense, a relation is a report: the rectangular display corresponding to the relation. However, reports are usually hierarchical in nature, and there are no facilities in relational languages for describing hierarchies, let alone displaying them. Again, System R offers extensions that allow the generation of more sophisticated reports.

In order to overcome these deficits in the relational languages there is a proposal (it may have been implemented) to embed relational languages, specifically SEQUEL and System R in a regular programming language, PL/1. Calls to the DBMS are made through character strings containing SEQUEL statements. The relation that is returned may be fed tuple-at-a-time into a user-defined PL/1 structure. The user of such a system will be faced with learning two languages, a task that is probably more difficult than learning the CODASYL systems of which we have been so critical. A user with a working knowledge of PL/1 and SEQUEL would probably be as happy or happier with direct access to the low-level primitives, hash tables, linked lists, etc., of the data base management system. Note that while SEQUEL is based upon a functional programming system, PL/1 is not. Any clean syntactic merge of the two languages is therefore unlikely.

D. QUERY LANGUAGES FOR DBTG DATA BASES

1. DBTG Access Primitives

The DBTG data model was briefly described in the previous chapter. In order to understand how query languages for these systems may be defined, a certain amount of knowledge concerning the physical communication between the applications program (in this case the query language) and the physical data is required. The DBTG proposals do not specify a query language as such. Instead, they define a set of statements or subroutines which may be embedded in a "host" programming language such as COBOL or FORTRAN. A user who is familiar with these languages, and who understands the physical manipulation of DBTG structures, can therefore program any given query. However, in most cases, the resulting code is relatively cumbersome, and the query can, in most cases, be more concisely represented in "higher-level" languages. Some of these languages will be described in this section.

A DBTG data base has a fixed schema, which must be defined before a data base may be generated. To create the schema (here we are referring to the data structure that described the physical layout of the data), a user composes statements in a Data Definition Language (DDL). A preprocessor then compiles the DDL into the appropriate physical structure. The DDL for the VEHICLE data base of Chapter II would resemble the following:

```
RECORD DEPARTMENT
  DEPT# INTEGER
  DNAME CHAR(20)
  CITY CHAR(10)

RECORD EMPLOYEE
  EMP# INTEGER
  NAME CHAR(20)
  LOCATION MODE IS CALC USING EMP

RECORD VEHICLE
  V  INTEGER
  VTYPE CHAR(10)

RECORD JOURNEY
  TIME CHAR(6)
  DISTANCE INTEGER

SET DE
  OWNER DEPARTMENT
  MEMBER EMPLOYEE

SET DV
  OWNER DEPARTMENT
  MEMBER VEHICLE

SET EJ
  OWNER EMPLOYEE
  MEMBER JOURNEY

SET VJ
  OWNER VEHICLE
  MEMBER JOURNEY
```

FIGURE 3. The DDL for the DBTG data base of Chapter II. This example illustrates only a subset of the available structures. For example, it is possible to specify that records classes and sets should be ordered by a given field.

The DDL in Fig. 3 is a simplification of the actual DDLs used. Note that the "location mode" specifies an efficient (hashed) mechanism for locating a given EMPLOYEE record in the data base. If this statement were not present, sequential search would be the only method of finding an employee given an EMP#. It is also possible to specify that two records with the same key may not simultaneously reside in the data base.

Communication with the physical data base takes place through a fixed area of working storage, which has space for precisely one record of each record class. In the EMPLOYEE-VEHICLE data base, the working storage would be able to hold one record from each of the four classes. A record is located in the data base by a currency pointer. These are usually physical data base addresses. Once a currency pointer for a record has been established, the record is moved into working storage by invoking a procedure GET. Thus, the main purpose of the DML routines is to provide methods for creating currency pointers. In the DBTG proposals there is a fixed set of currency pointers: one for each record class, one for each data base set and a "current of the run unit," which is the currency pointer upon which the procedure GET operates.

The procedures that establish currency pointers are named by some variant of the word FIND. We shall adopt a slightly different method of describing the various FINDs. We shall view the FIND routines as functions that return currency pointers as results; the fact that in a true DBTG system, the FINDs may only modify one of the fixed set of currency pointers, is a complication that we may ignore for the purpose of describing the global structure of DBTG programs.

The following notation will be useful in describing the FIND routines.

CPT: Anything that is a currency pointer. If a function fails to establish a currency pointer, we shall assume that it has the value 0. In other words, 0 represents the value NIL in conventional programming languages.

RECID: An identifier; either a name, or an internal reference, to a record class.

SETID: An identifier for a set.

KEY: An identifier that is to be used as a key (i.e., for which some indexing structure is provided).

Strictly speaking, these terms refer to data types. When, for example, we describe the function FINDNC as

FINDNC: (RECID,CPT) → CPT

we mean that FINDNC is a function that takes a record class identifier, and a currency pointer, and returns a currency pointer. Using this notation, we can describe the various FINDs that may be used to traverse DBTG structures.

1. Traversing a record class. Two functions are used:

   FINDFC: (RECID) → CPT .

   This function takes a record class as argument, and returns the currency pointer for the first record in that class. If the class is empty, the result is 0.

   FINDNC: (RECID,CPT) → CPT .

   Given a currency pointer to a record, FINDNC returns the currency pointer for the next record in the class. An error should be generated if the given currency pointer does not reference a record in the given class. Again, 0 is returned if there is no next record, i.e., the class has been traversed.

2. Traversing a set requires three functions:

   FINDFS(SETID,CPT) → CPT .

   Given a set identifier, and a currency pointer to a record R, find the first record owned by R in that set.

   FINDNS(SETID,CPT) → CPT .

   Find the next member record in the set.

   FINDOS(SETID,CPT) → CPT .

   Given a set and a currency pointer for a member record, find the owner record.

3. Hashing directly to a record.

   FINDFK(RECID,KEY) → CPT .

   Find a record in the given class with the given key.

   FINDNK(RECID,KEY,CPT) → CPT .

   Find another record, if any, with that key.

In all cases, when one of these functions is given inconsistent arguments, an error should be generated.

Using this set of variants of the DBTG FIND commands, we may now give some sample queries in an ALGOL-like language. The simplest queries involve only one record class, and may be traversed with the functions FINDFC and FINDNC. For example, to print the names and cities of all departments, a program would have the following form.

```
1.    begin
2.      C:=FINDFC(DEPARTMENT);
3.      while C <> 0 do
40        begin
5.        GET(C);
6.        PRINT(NAME,CITY);
7.          C:=FINDNC(DEPARTMENT,C);
8.      end;
9.    end;
```

Line 2 of this program establishes C as a currency pointer for the first DEPARTMENT record. Lines 5 and 7 now specify an iteration over all the records in the DEPARTMENT class. C will take on successive values that will reference all records in this class. For each value, the procedure GET is called to bring the record into working storage, and the procedure PRINT (line 6) prints out the desired information.

As a more complicated example, we could ask for the names of departments and the vehicle types owned by each. This query involves a set traversal.

```
1.      begin
2.        C:=FINDFC(DEPARTMENT);
3.        while C <> 0 do
4.          begin
5.          GET(C);
6.            D:=FINDFS(DV,C);
7.            while D <> 0 do
8.              begin
9.                GET(D);
10.               PRINT(DNAME,VTYPE);
11.               D:=FINDNS(DV,D);
12.             end;
13.         C:=FINDNC(DEPARTMENT,C);
14        end;
15.     end;
```

This program should be contrasted with the corresponding relational query. In the first place, a line is printed for each vehicle. Thus, a (DNAME,VTYPE) pair may be repeated. In contrast to SEQUEL, it is not an easy matter to remove duplicates. In general, the way to remove duplicates is to perform a sort on the appropriate field. If, for example, the member records of the DV set were sorted by VTYPE, then it would be an easy matter to cause the PRINT to be performed only when the VTYPE changed.

For the generation of reports, this program structure provides considerable flexibility. For example, it may be required

for each DEPARTMENT to have the DNAME appear once only in the left margin, and the VTYPES for that department indented, and displayed in a column underneath.

As a final example, consider the problem of printing the names of all employees who have driven a vehicle that does not belong to their own department.

```
1.    begin
2.      E:=FINDFC(EMPLOYEE);
3.      while E <> 0 do
4.        begin
5.          D1:=FINDOS(DE,E);
6.          FLAG:=FALSE;
7.          J:=FINDFS(EJ,E);
8.          while FLAG = FALSE and J <> 0 do
9.            begin
10.             D1:=FINDOS(DV,FINDOS(VJ,J));
11.             if D1 = D2 then FLAG:=TRUE;
12.           end
13.           if FLAG = TRUE then
14.             begin
15.               GET(E);
16.               PRINT(ENAME);
17.             end
18.           E:=FINDNC(EMPLOYEE,E);
19.       end
20.   end
```

Again, it should be emphasized that we have represented the FIND commands in a somewhat different fashion to the DBTG specifications. In fact, in the DBTG proposals, a FIND involves an intermediate GET. That is, a record must be brought into working storage before the next record (in its set or class) can be found. This means that programs are often considerably more cumbersome than the examples given above. For example, the statement

```
FINDOS(DV,FINDOS,(VJ,J))
```

would expand to four statements in a conventional DBTG program.

2. Interactive Query Languages for DBTG Structures

A number of "interactive" query languages have been implemented that give the user direct access to the DML routines (FINDs and GETs). Roughly speaking, these languages resemble BASIC interpreters: the user may perform line-at-a-time editing, and has a number of useful debugging aids. The programs themselves use a syntax that resembles (more or less) that of BASIC. Thus, the control structures (while..do, begin..end) in the programs above, are replaced by GOTOs.

An example of such a language is the query language for Digital Equipment Corporation's DMS-20, a DBTG-based system. Programs in DMS-20 are approximately twice as long as the examples given above; this is in part due to the more primitive control structures (GOTOs), and in part because the examples above have used a more condensed notation for the FIND command.

3. WWDMS

The examples above indicate that a very common control structure for DBTG structures is:

```
C:=FINDFC(<recid>);
while C<> 0 do
  begin
  GET(C);
    ...
  end
```

There are similar programs for traversing sets and records with the same CALC key. A good query language should be able to provide this control structure as a primitive operator of the language. The query language of WWDMS (Worldwide Data Management System) is such a language. WWDMS, developed by Honeywell Information Systems Inc., is a comprehensive data management system that maintains not only DBTG-like structures, but a number of other structures, in particular:

1. Sequential Files
2. Indexed Sequential Processor (ISP) files. These are similar to the familiar ISAM files on IBM systems.
3. Inverted Files. Physically, this is a file with certain keys inverted for efficient access. This is similar to ADABAS structures.
4. Integrated Data Store (IDS). This gives a "network" structure. For most purposes, IDS structures can be viewed as DBTG structures.

The WWDMS query language (often also called WWDMS) provides access to all these file types. It has very sophisticated report definition, and provides a degree of "non-procedurality" for data base traversal. In expert hands, WWDMS is an extremely powerful tool. However, it is generally regarded as being difficult to use, and there is a lack of WWDMS experts. We shall try to suggest reasons for the unwieldiness of WWDMS, but first let us examine a simple WWDMS query. We shall express in WWDMS, the simplest of our DBTG queries, printing the names and cities of all departments:

```
1. RLINE.  LINE DNAME COL 1 PIC X(10),
2.         CITY COL 11 PIC X(10)
3. R1.     RETRIEVE DREC FROM DEPARTMENT
4. WHEN R1.
5.   PRINT RLINE
6.      END
```

Lines 1 and 2 of this query define (in detail) what the format of the output report is to be. Line 3 specifies that successive DEPARTMENT records are to be retrieved. The retrieval of a record "triggers" the event R1; and lines 4 and 5 specify an action to be taken when each such event occurs. Two important headers have been omitted from this query: the specification of the data base, and the device to which the report is to be routed. WWDMS queries operate in a batch processing environment, so these cannot be defaults set up at the start of a terminal session.

The second of our sample DBTG queries would be accomplished by a "nesting" of RETRIEVE statements:

```
R1.     RETRIEVE DREC FROM DEPARTMENT
WHEN R1.
R2. RETRIEVE VREC FROM VEHICLE {VIA DV}
     WHEN R2.
     ....
     END
```

(The VIA DV statement has been inserted by the authors to show the difficulty in specifying which DBTG set is to be used in a traversal when there is a choice.)*

These two queries illustrate that simple traversals can be simply expressed in WWDMS. More complicated programs are also possible, and WWDMS has a number of additional control structures to allow this. In particular, it is possible to define subroutines in WWDMS. More complicated WWDMS queries, however, do not seem to benefit. For example, the third of our sample queries is considerably more complicated to express in WWDMS than our original DBTG program. Since WWDMS includes the basic variable definition statements and control structures of a simple programming language, it is possible to write any program in WWDMS.

WWDMS is probably one of the most powerful languages ever designed specifically for data base manipulation, but it is only useful in the hands of a well-trained expert, and learning the intricacies of the language is not an easy task. Why is this? The following reasons are, in part, the authors' value judgments and are stated here not to denigrate WWDMS, but to provide some insight into the requirements for a successor to the WWDMS query language.

---

*It is possible that WWDMS data structures do not allow two sets to have both the same owning class and the same member class, in which case this ambiguity would not arise.

1. WWDMS programs, although they can be interactively edited and submitted, are designed primarily for a batch processing environment. The operating system (WWMCCS) under which WWDMS runs is not at all "friendly" when used from a terminal. Debugging a WWDMS program is therefore a difficult task, and unnecessary resources are often wasted in repeatedly resubmitting a WWDMS program, because incremental debugging of the program is almost impossible. This criticism is, however, not directed at the WWDMS language itself. It is more a criticism of the environment in which it operates.
2. Simple control structures for WWDMS queries are, as we have seen, extremely simple to program; but more complicated flow of control demands a very much more "procedural" programming style. For example, if a programmer wants to terminate prematurely a RETRIEVE iteration, the control statements are, if anything, more complicated than those of a conventional programming language. It is a pity that what starts out as a "non-procedural" language has to be explained through the use of procedural flow charts in the instruction manuals (Ref. 20). The scope rules of WWDMS are also extremely confusing: what lies inside the scope of a WHEN or RETRIEVE statement cannot be determined by any simple syntactic rule.
3. There is a lack of any abstract data model. This seems to be a flaw in a number of query languages. The notion of a sequence of records is fundamental to understanding the action of a RETRIEVE, but to our knowledge, there is no formal definition of sequences, or other relationships among records. It is never clear whether or not the user is expected to understand the physical structure of the data base.

E. QUERY LANGUAGES BASED UPON THE FUNCTIONAL MODEL

In a previous report (Ref. 1), a formalism was presented for a query language based upon the functional model. This has been implemented, and in a concurrent effort, DAPLEX, another language based upon the functional model, has also been defined but has not yet, to our knowledge, been implemented. We shall briefly review the operators required in a functional query language here, using the syntax of the Functional Query Language (FQL). However, FQL, like the relational algebra, is not intended as an ideal end-user query language. The queries given here could be formulated in the DAPLEX language, or in some system whose syntax matches that of SEQUEL. Our purpose here is to set out some FQL queries for comparison with the previous examples, and to define what we believe to be the correct internal semantics for a query language against the functional model. The following examples are not meant to be a complete description of the language itself. That will be found in (Refs. 1 and 9).

The first of our queries against the DBTG model was "The NAMEs and CITYs of all DEPARTMENTs". In FQL, this is represented by the query:

```
!DEPARTMENT.*[DNAME,CITY]
```

Informally, this expression is equivalent to the SEQUEL query:

```
SELECT DNAME, CITY FROM DEPARTMENT  .
```

However, its formal description in terms of functions is somewhat different than its expression as a relational query. DEPARTMENT is a data type. !DEPARTMENT is a function that generates all departments. Note that it is a function that takes no arguments. DNAME and CITY are both functions from the DEPARTMENT data type to the STRING data type. The expression [DNAME, CITY] is a function from DEPARTMENT into pairs of strings. The result of !DEPARTMENT is a sequence, or stream†, of

†The term stream is used because of the technique used in its implementation. This is not described here.

objects of type DEPARTMENT, but [DNAME, CITY] only applies to a single DEPARTMENT. The function *[DNAME,CITY] extends this function to operate over a stream of departments to produce a stream of pairs of character strings. Finally, these two functions are "composed" by the "." operator to produce the final query.

The second of our DBTG examples was: "The NAMEs of DEPARTMENTs, and the VTYPES in each." In FQL, this is

```
!DEPARTMENT.*[DNAME, !DV.*VTYPE]
```

This query is not unlike the previous query except that the second term in the square bracket has been changed. DV is a function from VEHICLE to DEPARTMENT (refer to the functional diagram of the previous chapter). !DV is the inverse of DV, and given a DEPARTMENT, returns a sequence of all VEHICLEs in that department. The function !DV.*VTYPE, therefore takes a DEPARTMENT as argument and produces a stream of character strings, one for each VTYPE. The whole query therefore produces a simple "hierarchical" report consisting of a sequence of DNAMES and for each DNAME, a sequence of VTYPEs.

The last example was "The NAMEs of EMPLOYEEs who have driven a VEHICLE that did not belong to their own DEPARTMENT."

```
!EMPLOYEE.|P.*ENAME
  where P = [DE, !EJ.*(VJ.DV)].MEMBER.NOT;
```

The term |P is a restriction. As in the relational calculus, it restricts the stream of EMPLOYEEs to those that satisfy the predicate P. That is, |P is a function that takes a stream of employees as argument, and produces a stream of EMPLOYEEs as a result. *ENAME produces the names of those employees.

FQL allows function definition, and the second line of this query defines the predicate P. Within the brackets, the first term, DE, is simply the DEPARTMENT of the EMPLOYEE. The second term is the stream of all DEPARTMENTs that own vehicles

driven by that EMPLOYEE. The remainder of the definition is constructed out of a number of built-in functions, and tests whether the second term (the sequence of DEPARTMENTS) contains any member that is not equal to the first term. The details are not given here, but are to be found in FQL (Ref. 9).

The semantics of a functional language must be based upon a notion of data types. Our data base has seven types: DEPARTMENT, VEHICLE, JOURNEY, EMPLOYEE, CHAR, NUM and BOOL. Of these, the EMPLOYEE, VEHICLE, JOURNEY and DEPARTMENT types are specific to the data base, the others are standard (BOOL is also a standard data type denoting Boolean variables.) We shall use Greek letters, α, β, γ ... to denote arbitrary types. If α is a type, then $*\alpha$ denotes the type of a sequence or stream of objects of type α. Thus, the functions ENAME and !ENAME may be formally described by

ENAME: EMPLOYEE → CHAR
!ENAME: CHAR → *EMPLOYEE

Given types α_1, α_2, α_3 ... we may wish to discuss tuples of these types. This is denoted by $[\alpha_1, \alpha_2, \alpha_3 \ldots]$ so that, for example, [STRING, NUM] denotes the data type of character string-number pairs.

The purpose of an FQL query is to combine existing functions to create new functions. This is done through four operators, all of which have been explicitly used in the above examples.

1. Composition. If $f:\alpha \rightarrow \beta$ and $g:\beta \rightarrow \gamma$ then the composition of f and g, denoted by f.g maps α to γ. The result of f.g is the result of applying first f and then g. Note that this is "reverse Polish" order for function composition. It will be seen that this is a more natural order, as the left-to-right order of the functions describes a corresponding path through the data base schema. Example: DE.DNAME is a function from EMPLOYEE to STRING, and gives the DEPARTMENT DNAME of an EMPLOYEE.

2. Restriction. If P is a predicate on the type a, i.e, P: $\alpha \rightarrow$ BOOL then |P:*$\alpha \rightarrow$ *α. That is, | P takes a stream of objects of type and selects from it just those objects for which P is true. The result is another stream of objects of type.
3. Construction. In order to create functions from a given type into tuples of other types the notation $[f_1, f_2, \ldots]$ is used. Thus if $f_1:\alpha \rightarrow \beta_1$, $f_2:\alpha \rightarrow \beta_2$ ... then $[f_1, f_2, \ldots]:\alpha \rightarrow [\beta_1, \beta_2, \ldots]$. Example, [DNAME, CITY] maps a DEPARTMENT into a pair of STRINGS.
4. Extension. If f is a function from α to β, then *f maps *α to *β. Example *ENAME maps any stream of EMPLOYEES (*EMPLOYEE) into a stream of STRINGs (*STRING).

Finally, it should be noted that 'JONES' is, like 10000, a constant function. Also !EMPLOYEE denotes another "constant" function. It is a function of no arguments that returns a stream of EMPLOYEEs. This function, unlike 10000 and 'JONES' can, and probably will, change over time.

IV. NATURAL LANGUAGE INTERFACES FOR DATA BASE SYSTEMS

The prospect of using natural language to interrogate a date base is always appealing initially. A number of NL systems are under development, and there is one, ROBOT (Ref. 21), that has recently been made available commercially. The arguments in favor of NL systems are familiar: the user need know nothing about a query language nor is an understanding of the data base required. However, these "obvious" assumptions need to be examined in more detail to determine what impact the development of NL systems will have on the C^3I community within the next few years.

There are many research projects involving NL systems and data bases; however, three systems stand out as being robust enough to be in frequent use:

1. LADDER (Ref. 22) (Language Access to Distributed Data with Error Recovery). This system is funded by DARPA, and is currently interfaced to the "Blue File" data base on CCA's DATACOMPUTER system. For a comprehensive description of the LADDER system, see Ref. 22. LADDER has a sophisticated set of routines which are not part of the data base: for example, it can compute the shortest sea routes between given points, taking geographical considerations into account. A different DBMS system (DBMS-20) has been connected to LADDER, but this contains the same data. To our knowledge, no substantial interface to a data base representing a different domain has been built. The data base interface of LADDER (Ref. 1) has the ability to switch data bases in the event that one system should fail.

2. EUFID (Ref. 23) (End-User Friendly Interfaces to Data) is being developed for DCA. Versions have been constructed for two data base systems: one supported by INGRES, the other by WWDMS. It operates by translating into the query languages for those two systems (SEQUEL and the WWDMS language). EUFID is implemented in FORTRAN and is "table driven": to generate a new version of EUFID requires, in theory, only that a new set of tables be built. In practice, transportation of EUFID is a little more complicated, but it shows promise of becoming one of the easier systems to transport.
3. ROBOT (Ref. 21), developed by Artificial Intelligence Corporation, is the only commercially available system. At least ten ROBOT systems have been built for a variety of data base management systems, including ADABAS and IDMS. ROBOT is written in PL/1, but details of the system are not available.*

Since the last of these has had some commercial usage, it is worth asking what kinds of users have benefitted from this NL system. It is the authors' understanding that the ROBOT system is marketable when

1. The queries are simple.
2. The data base has a simple structure.
3. The users understand the data base.

Note that the last point does not support our initial contention that NL systems are useful when the users do not understand the structure of the data base. We shall give reasons for this below; however, it is our belief that, within the next few years, NL systems will only be effective if these three criteria are met.

---

*Among the purchasers of ROBOT is Cullinane, manufacturers of IDMS, a CODASYL system written in PL/1.

While NL systems may in the future play an important role in C^3I operations, they probably will not entirely supplant the more "formal" query languages. There are a number of reasons for this. In the first place, there are a number of processes that are difficult or impossible to specify in NL. The format of a data base report is not something that can readily be described in English (not surprisingly, since English did not "evolve" for that purpose). Complex arithmetic operations are also hard to specify, as are programs: for example, recursive functions with a complicated structure. The query of the previous chapter: "The names of employees who drove a vehicle that does not belong to their own department", is already rather clumsy and somewhat ambiguous. It may be that such queries could be more concisely formulated in some structured query language.

Another serious deficit in most query systems is a failure to give adequate responses, especially when queries fail, and a further problem arises when a response generated by one interpretation of the schema is misinterpreted by the user. These are complicated topics, and it is not appropriate to deal with them in any detail here; however, the following examples (taken from actual query systems) may illustrate some of the problems.

A data base containing information about ships and ports is asked, "Is there a doctor in Philadelphia?". The answer returned is, "No." The NL system has interpreted "Philadelphia" as the name of a ship, which it is. But the user has taken "Philadelphia" to refer to a port, which is reasonable, since Philadelphia is a port, and the date base contains information about (some) ports. Had the response been, "There is no doctor on the ship Philadelphia", the user, while not getting the required information, would not have been actively misinformed.

A university data base containing information about students is asked, "What are the names of undergraduates in the Computer Science Department?". The question is reasonable in the context

of a European university, but does not make sense for an American university, for students are not affiliated to departments. A NL system responds with a set of names. Although there is no direct link in the schema between DEPARTMENT and UNDERGRADUATE, the NL system has found an indirect link and has used that. The indirect link corresponds to "taking courses in" or "being supervised by a faculty member in". The user is unaware that this is how the query has been interpreted, and his misconceptions about American universities are confirmed. A more helpful response would have been: "I do not understand the question. It could mean 'What undergraduates are taking courses in Computer Science?' or it could mean...." Had this response been given, the user would have been enlightened about the structure of the data base.

A distributor keeps information on the stock levels of its customers in order to know when to suggest a new delivery. A user asks, "How many Xs does Y have in stock?" and the NL system responds with a single number. In fact, information on stock levels is notoriously unreliable, and the data base contains a set of reports on stock levels, containing information about when the information was obtained, and who made it. The single number returned to the user may be the most recently reported level, but is may nevertheless be hopelessly out of date and come from an unreliable source. The user, in this case, has misunderstood the whole information system used by the company. Had the response been: "The stock level reports on company X for item Y are: ...", the user might have been able to understand how the company's information system operates.

The second and third of these examples indicate that there is a danger in a NL system being too "smart". It is a danger that may be particularly serious in the C^3I community, where there is a great deal of unreliable information and where, once the C^3I network is established, there will be many users with an incomplete knowledge of the data bases they are using.

The developers of NL systems are well aware of these problems. Better response generation is a topic of active research (Ref. 24), and the problems described above will ultimately be corrected. Also, a great deal of the stimulus to develop better formal query languages comes from the desire to build better interfaces for NL systems. However, with the state-of-the-art NL systems, it would be very unwise to make natural language the only method of access to a data base.

It was indicated in the introduction that a drawback of NL systems is that they usually have to duplicate substantial portions of the data base. The lexicon, for example, contains the set of all names that could refer to some object (record, record class, etc.) in the data base. While this information is, in some way, encoded in the data base, it is often difficult or impossible to use it. Most NL systems, therefore, keep an internal lexicon, which may contain tens of thousands of entries--mostly information which is already in the data base. Not only may this duplication prove a problem for reasons of security, it is also difficult to keep the two sets of information consistent. For example, if a new record (a ship record, say) is added to the data base, it will require that a new name be inserted in the lexicon. Seldom, if ever, does this update happen automatically. At best, updates, when they are performed automatically, are only done periodically, and the NL system will "lag behind" the data base.

NL systems, and more generally, query translation systems, such as ADAPT (Refs. 25 and 26) require that a copy of the data base schema also be maintained. This again is a form of duplication; in this case of structural data. Again, these schemas cannot usually be generated automatically, either because of the incompatibility of data models, or, in the case of NL systems, because further semantic indicators must be added. Such schemas must be updated whenever the data base is restructured--an operation that is becoming increasingly frequent with modern,

modern, flexible, DBMS. Unfortunately, the task of building the duplicate schema is still the province of the implementators of the NL system. It cannot be performed automatically, nor can it be simplified to the point where it can be performed by the data base administrator, or someone else who is familiar with the structure of the data base. A figure of one man-month is sometimes quoted as a minimum time needed to interface the simple NL systems to an existing data base, once the general purpose translation routines have been built. This appears very optimistic when compared to actual bids for such work.

V. PROJECT SUMMARIES AND PROTOCOL IMPLEMENTATION

Several research and development projects are under way which directly address the problem of access to heterogeneous data base management systems (Refs. 25, 26, 27, and 28). Other projects, in particular the natural language systems (Refs. 22 and 23), address this problem indirectly and incorporate partial solutions to it. There is also an important data base project, SAFE, whose development should be closely tied to any effort to develop a uniform data base access system within C^3I. This chapter contains reviews of these projects in developmental stages.

A. MULTIBASE

1. Introduction

Of the projects reviewed in this report, there are two substantively-funded efforts that have as their stated goal a system that provides access to heterogeneous data base management systems. One of these is ADAPT, and the other is MULTIBASE, which, although it has only recently received funding from DARPA, will require serious consideration as it progresses. The MULTIBASE proposal comes from the Computer Corporation of America, a software company that has assumed a leading role in data base research, and which developed the distributed data base system, SDD-1 (Ref. 29), which was briefly described in a previous report (Ref. 1).

MULTIBASE attacks the problem of a multiple data base user interface at a significantly higher level than ADAPT in that it allows for user-defined updates and extensions to the data base

structures and exploits a number of recent developments in data base research. Of special note are the following advantages:

1. A global schema is maintained which represents the structure of all the data bases to which a user has access. The user need not, initially, be aware of which data base(s) are being used to answer a query.
2. The schema exploits a simple, but powerful, functional model of data similar to that described in Ref. 1. This permits a natural representation of other (relational, network, hierarchical) schemata.
3. A query language, DAPLEX, has been developed in order to allow more complex queries. It is proposed to embed DAPLEX in an existing programming language, such as PASCAL, thereby providing the capability for resolving any query.
4. Some sophisticated optimization techniques, especially for a CODASYL interface, are proposed.
5. The MULTIBASE proposal recognizes that data may be redundantly represented in more than one data base and that the data may be inconsistent. For this, a logic of "fuzzy queries" has been formulated. A user is informed when there is inadequate or inconsistent data in the resolution of a query and may, when there is a conflict, select the more reliable source.

The purpose here is to describe the MULTIBASE proposal in more detail and to provide some critical comments on its usefulness to the C^3I community.

2. The Structure of MULTIBASE

In order to provide information on how data are distributed among the various data bases, MULTIBASE provides a global schema. This is a data structure that describes all known entities and relationships in the data base system as a whole. Before any query may be interpreted, it must first be resolved against this schema. The global schema completely describes the structure

of the data base as it is seen by a user of MULTIBASE. Although the structure of this schema is described by a functional model, it may be thought of in conventional terms.

Each data base in the collection understood by a multibase system has its own schema that will be defined in an existing Data Definition Language (DDL). This schema is termed the Local Host Schema and the mapping between the global schema and the local host schema is accomplished in two stages. First, a second MULTIBASE schema is constructed that is, in a logical sense, "close to" the local host schema. This is called the local schema. (In fact, the generation of the local schema could, for most existing data base systems, be accomplished automatically, but for other reasons this may not be desirable.) The translation of a query expressed against the local schema and the local host schema must be accomplished by a module that will be specific to the local data base system. The translation between the global schema and the local schema, which may be very different in structure, is specified in the MULTIBASE query language, DAPLEX. The advantage of this method is that the complex structural modifications that may be needed to make the local data base mesh with the global user view may be specified in a high-level language and do not require new application programs to be constructed within the confines of the local data base system.

3. The MULTIBASE Data Model and Data Definition Language

As stated earlier, MULTIBASE employs a functional data model not unlike that described in a previous report (Ref. 1). The MULTIBASE language, DAPLEX, has data definition facilities through which a schema may be defined. The two construct used in the MULTIBASE data model are entities and functions.

```
DECLARE Employee() ==>> ENTITY
```

defines a niladic function Employee that returns a set of entities. Any niladic function defines an entity type and the

term "Employee" may be thought of as ambiguously referring both to a data type and as a function which, when applied, returns all objects of that type.

```
DECLARE Name (Employee) => STRING
```

creates a function Name which maps the type Employee into the (predefined) type STRING.

```
DECLARE Driver ==>> Employee
```

again defines both a function and data type Driver that is a sub-type of Employee. This means that every function that may be applied to Employee may also be applied to Driver. However,

```
DECLARE License#(Driver) => NUMBER
```

defines a function that applies only to the data type DRIVER.

The possibility of defining types as sub-types of other types is an extension to data base semantics that captures, in part, the Aggregation-Generalization model of Smith & Smith (Ref. 30). This will surely prove a convenience in user interaction. No other commonly used data base system has this capability although the idea has existed in programming languages (Ref. 31) for many years. A formal translation method from existing data models to the functional model has been described (Ref. 9) and the automatic translation of existing DDLs into the DAPLEX DDL should pose no problems. However, since data base systems cannot directly represent the concept of sub-type, it is often found necessary to resort to special data structures or special constraints on applications programs to represent it indirectly. In such cases, the automatic translation to the DAPLEX model will not produce a semantically correct schema and, as described in the previous chapter, a further translation will have to be specified by someone who is an expert in the semantics of the particular data base.

4. The DAPLEX Query Language

Although update facilities have been proposed (Ref. 10) for DAPLEX, this report will examine only the uses of DAPLEX as a query language. In form, DAPLEX resembles an applicative language (Refs. 14 and 32), and while it fails, in any sense, to be a full programming language, it offers substantially more power than most commonly used query languages. Simple DAPLEX queries consist of an iteration over the set of elements of some type.

```
FOR EACH Employee
  SUCH THAT Salary(Employee) < 10,000
  PRINT Name(Employee)
```

This provides a simple iteration and selection mechanism common to most query languages. Note, however, that the functional notation may be used to great advantage in queries such as Salary(Manager(Employee)). An expression such as this is usually extremely cumbersome to manipulate in most query languages designed for CODASYL-like systems, while the data itself is trivial to represent in such systems. Predicates and other functions over sets are provided by the constructs FOR ALL, FOR SOME, etc., and functions such as AVERAGE, MIN, MAX, etc. Thus:

```
FOR EACH Department SUCH THAT
  FOR SOME Employee(Department)
      Salary(Employee) < 10,000
  PRINT Name(Manager(Department))
```

However,

```
FOR EACH Department SUCH THAT
  AVERAGE(Salary(Employee(Department)) < 10,000
  PRINT...
```

will not give the intended result as a function of a given type, when applied to a set of that type produces the set (not sequence) of results. Thus, Salary(Employee(Department)) produces the set of distinct salaries of employees in the given department. To correct for this, an AVERAGE... OVER.. construct is introduced.

In addition to the simple query types illustrated here, DAPLEX also allows the definition of new functions in terms of existing functions. This is not only of use in helping the user to decompose complicated queries, it plays a key role in providing a method of defining one DAPLEX schema in terms of another.

It is proposed that DAPLEX should be embedded in a general-purpose programming language and that one of the ALGOL-like languages would be suitable. At present, there are no details for this part of the proposal.

5. Optimization in MULTIBASE

In many data base systems, especially network systems, there are several methods of accessing a given set of data. For example, access to a given record class may be gained by direct addressing, sequential scan, or through links with other record classes. The problem is compounded when, for reasons of efficiency, data have been redundantly represented within a particular data base. The various access methods are represented in MULTIBASE by an access path family and some good heuristics have been designed to find the appropriate paths needed to resolve a particular query. It should also be possible to perform a certain amount of optimization on high-level queries within DAPLEX itself. Since some of the set processing operations used by DAPLEX are known to be intrinsically costly, it may be possible to eliminate some from the given query.

6. The Treatment of Redundant Data

MULTIBASE is designed for access to multiple data bases and it may be the case that the same data are represented in two distinct data bases. Methods have been proposed for dealing with the various kinds of conflicts that could occur as a result. For example, using DAPLEX notation, an object x and function f may be represented in each of two data bases, but the data bases may be inconsistent because (a) f(x) is defined in one and not in the other or (b) f(x) is defined in both but yields different values in the two data bases. In such an event there are various possibilities.

1. Report all conflicts to the user and allow the user to assess the reliability of each data base.
2. Report whatever is defined in case (a) and use a "preferred" data base in case (b) according to some predefined criterion (this may be a procedure defined on the data type of x).
3. Use a general heuristic for determining which is correct in case (b).

Provisions have been made for both cases (1) and (2) above. Also a formalism akin to modal logic has been exploited to develop more general heuristics that cope with these and other kinds of conflicts.

B. XNDM

XNDM (Ref. 28), Experimental Network Data Manager, is a research project at the National Bureau of Standards (NBS). It bears some relation to MULTIBASE, though details are not available at this time. Unlike MULTIBASE, the query language is based upon SEQUEL, giving the end user a relational view of the data. This information is based upon a project summary report and meetings with the system developers.

The Network Data Manager maintains a global schema and translation programs for a variety of data base management systems. Some common basis for the translation is obtained by using techniques in transformational grammar to transform an initial tree structure representing the user (SEQUEL) query into the target language. It appears that the target language is assumed powerful enough to represent the source query, although this may not be required if the processing power of the data manager is increased to allow some local processing.

XNDM maintains a global schema and mappings into a relational system, Multics Relational Data Store, and Honeywell's Integrated Data Store (a component of WWDMS) have been constructed. The latter is a network system based upon the DBTG proposals. Like ADAPT, the schema translation does not appear to support non-hierarchical structures. It is not clear whether or not this is a fundamental limitation.

The implementation to date will support only projection and restriction. In terms of SEQUEL, this means that SELECT statements may not be nested. Also, it means that queries across data bases are, at present, impossible.

XNDM appears somewhat similar to ADAPT in its overall power. However, the simplicity of the relational model may confer certain advantages upon it. It is, to our knowledge, the only attempt to build a network user interface based upon the relational model. Provided a local (to the user) data management system is incorporated, it should be possible to issue any SEQUEL query. A major problem remains in defining schema mappings from network systems into a relational model. It is often undesirable, and difficult, to do this by hand and there is no canonical translation method. Some automatic aids appear to be desirable.

C. PROJECT SAFE

SAFE is a substantial development project funded by DIA and CIA, and currently contracted to TRW with a view to producing a prototype within three years. The purpose is to integrate a number of data bases containing both structured and textural data. The system will contain redundantly distributed data, and will replace some ten of DIA's existing data bases. The data base system to support this has yet to be chosen, and the user interface has yet to be designed, although we have received preliminary specifications to date to be analyzed for this report.

D. EUFID

The EUFID (Ref. 23) system provides natural language access to data associated with a single applications area on a single data management system. EUFID is being developed by Systems Development Corporation (SDC), DCA funding and interoperates with INGRES and WWDMS. The basic motivation of EUFID is to provide a natural language interface powerful enough to accept English language expression of queries against specific data bases without requiring the user to understand the full range of syntactic constructions. The natural language it uses is therefore limited to the language that is natural to a specific application area as opposed to the most general idea of natural language. The original EUFID implementation was with a relational data management system called INGRES and the WWDMS data management system followed immediately thereafter. (The WWDMS system is an example of a DBTG system and is not relational.) The translator module for each version of EUFID interfaces with only a single data management system (the parser is common in all systems); however, EUFID might be usable with specific data bases, for which an English language query would be natural to the analyst or other users. EUFID has been sponsored by two different agencies. There are presently two

versions of EUFID; however, these are essentially the same in operating characteristics.

At present, construction of the user tables in a EUFID implementation requires the skills of a trained system designer. In the future, there is hope that much of this process could be automated. User access is controlled by use of a profile table that identifies the legal users and the data base accessible to each user. This table is constructed by the data base administrator. Each new table area describing both the data base and the user terminology must be established. The process of setting up a new data base requires an expert in both the EUFID language representation technique as well as someone who knows and understands the application area.

EUFID provides a facility that allows users to define private synonyms for terms already in the terminology tables. The synonym editor also provides for altering or changing these private definitions and removing terms from each application dictionary. The synonym editor sometimes finds possible ambiguities and problems introduced by such changes and announces these problems to the user. If EUFID cannot process a question, it attempts to correct the query.

EUFID deserves special mention in that it is one of the few NL systems that has been successfully transported both across data bases and across data base management systems. Prototypes have been built that interface with two different WWDMS data bases, and with an INGRES data base. One of the reasons for this transportability is that it uses a table driven parser, and while these tables may be redefined only by an expert in the EUFID system itself, there is some hope that the task could be partly automated, and partly performed by the data base user. The reference mappings are also represented by tables, and the system operates by translation into the existing query language (WWDMS) or SEQUEL.

EUFID is written in FORTRAN and C, which makes it transportable across computer systems.

E. ADAPT

The COINS is a digital communications network that provides Intelligence Community access to data bases on COINS computers operated by the Defense Intelligence Agency (DIA), the National Photographic Interpretation Center (NPIC), the National Security Agency (NSA), and the Aerospace Defense Command (ADCOM). At present, four different data retrieval languages are required to use the systems on the COINS II network and it is possible in the near future that up to ten different retrieval languages may be required to utilize completely the COINS system computers.

To deal with this problem, the COINS PMO is funding a project called ADAPT, and Logicon is the prime contractor. The ADAPT approach to the multi-language retrieval problem is to provide a single network retrieval query language and then to translate from this language to each of the other query languages associated with data bases in the network. The ADAPT I system demonstrated the feasibility of this approach by performing the necessary translation functions for the COINS network. It is possible to approach the system from two viewpoints: (1) how does the ADAPT system fulfill the requirements of system users and (2) is the general approach of ADAPT adequate for other systems and for more widespread use.

With regard to the first question, ADAPT provides a considerable facility to the COIN system users and should eventually provide an improvement to the system. At the same time, ADAPT will require future enhancement in order to make it powerful enough for more sophisticated users by providing several facilities that are not presently in the system. Nevertheless, it should be stated that ADAPT is an excellent first step forward for the COINS system.

Among the recommended enhancements for ADAPT are some additional capabilities that are needed for more sophisticated users

in making complex searches of data maintained on the various systems. Another general improvement is that the user must now be overly conscious of the syntactic rules in ADAPT and a specific query preparation capability should be added to enhance the language. Further, routines should be usable without knowing the retrieval language of the particular DBMS in which they reside.

The ADAPT (Refs. 25, 26) system has a language called the Uniform Data Language (UDL), and ADAPT provides a translator from this language into each of the other query languages for the DBMSs in the system. The original system provided for translation into the initial languages and gives a demonstration of the feasibility of this approach. The ADAPT II and ADAPT III systems are intended to expand on the initial facility, including other languages and DBMSs and also enhancing the basic ADAPT capabilities by providing text search capability and documentation adequate for day-to-day usage. The ADAPT IV system will provide for local file creation and manipulation. The ADAPT approach provides a transactional data base interface with both batch or interactive data retrieval systems. It is not possible to interact with the DBMS during query development or refinement in the initial systems.

The languages with which ADAPT interfaces are varied in basic construction and include NSA's SOLIS (SIGINT On-Line Information System), which is reasonably syntax-free, and NPIC's PIRL (Photographic Information Retrieval System) and the NSA TILE (TIPS Interrogation Language), which are syntactically rigid. The DIAOLS system will be a future addition. PIRL and TILE are scheduled to be replaced during 1980.

An interactive ADAPT interface is provided for the user. However, each transaction or query must be sent as a complete unit to the target system so that the user does not interact with the target system during preparation of a query but only with the local ADAPT programs. As is customary, ADAPT includes

a uniform data language, a data definition language and a transformation definition language that is available only to the ADAPT system personnel for building and maintaining the ADAPT system itself.

ADAPT software checks the TAS authorization list in order to see if a user is cleared to access a file before opening that file. The effort required to make a translation into a new data base is a function of the number of fields in the file and the adequacy of the documentation in the file.

The ADAPT UDL is syntactically rigid, having a single set of query entry forms. At present, the UDL does not allow for abbreviations for shortened input command forms nor does it allow buzz words or the omission of redundant constructions. The UDL also does not allow for DON'T CARE characters in formatting queries, which can be a problem since several of the systems in COINS already contain this facility. The structure of a file and the status of a query can be displayed by ADAPT. Entire queries cannot be edited at present and an entire new query must be entered. However, files of commands constructed using the UNIX edit packets can be utilized. This requires interaction between UNIX and the user who must switch into the UNIX system and then back into ADAPT to make use of the facility. It should be noted that some of ADAPT facilities are usable against one data base and not against another. We have described ADAPT in some detail in Ref. 1.

F. LADDER

The LADDER (Ref. 22) system is implemented on the DARPA computer network. It is, like EUFID, an implementation of the results of natural language research and is oriented toward development of a natural language system. The idea here is to use natural language and thereby eliminate the requirement that the system user know the representation of the data in the computer environment and to have learned a data base management

system user language. The LADDER system originally operated on the DARPA computer network and accessed only the relational data base management system data computer, which was developed by the Computer Corporation of America. LADDER is programmed in INTER-LISP and operates under the TENEX and TOPS-20 operating systems on the DEC 10 computers. If the first file interrogated does not contain a specific data needed for a query, the system will address another file and will continue looking for alternate sources until all possibilities are exhausted (in the ideal system). The input system utilizes a set of generalized called language interface facilities with ellipses and recursion, which includes a parser and a set of capabilities for defining an application.

The LADDER system has been very popular from a demonstration viewpoint and many users of the ARPANET are familiar with the LADDER system, having fashioned queries in this system against the Naval data base, which is currently the primary demonstration vehicle.

Major components of LADDER are capable of operating independently and could be used in other systems. At present, each time a new data base is placed on the LADDER system, substantial development is required, and this includes development of or addition to word categories, development of translation rules, development of protocol queries pamphlets and a number of other operations such that perhaps two man-years might be required for a typical data base. (This was the estimate for interfacing LADDER to one of the data bases used by EUFID.) We understand that research is being undertaken to reduce this time.

LADDER contains a number of features that make for considerable ease of use by the inexperienced user. Synonyms are liberally used in LADDER so that abbreviations for ships, etc., can be used in fashioning queries. Natural sentence limitations can be somewhat overcome by the paraphrase capability of

the system in processing ambiguities. The ability to use ellipses in LADDER is a well-known feature. For example, the results of one query can be used in the following query. If we ask, "Who is the captain of the Kennedy?", and follow this with a question, "Of the Missouri?", the system will assume we are asking, "Who is the captain on the Missouri?".

LADDER does some work on spelling, trying to correct spelling and find close words (LADDER informs the user when it has done this). There are few training aids in LADDER and the user is left pretty much on his own in fashioning queries and in writing alternate queries in case of a problem. LADDER in effect uses the full range of relational operators. Good computational capabilities for the kind of questions concerned with the Naval data base for which it is now used are provided, including circle searches. LADDER can also find the fastest, slowest, newest, etc.

G. SECURITY CONSIDERATIONS

It should be pointed out that all the above solutions to the multi-language problem should be viewed with some caution in the Intelligence Community, because in most cases they involve duplication of the data to an extent that may pose a threat to security. An example may be found in state-of-the-art implementations of natural language (NL) systems. The purpose of such systems is to translate the user's query, phrased in the user's natural language, into a query understood by the appropriate DBMS. In order to accomplish this, the translator needs:

1. A lexicon. An internal set of definitions of all terms associated with the data base. In most cases, every name appearing in the data base must be duplicated in the lexicon.

2. A schema. This is a structure that describes the structure of the data base to the NL system. In most cases, this schema closely resembles the data base's own schema.

Not only does this mean that there must be substantial duplication of the data base, possibly on remote machines; it also requires the intervention of expert personnel to perform this duplication. In the present state of these systems this can neither be performed automatically, nor by the existing data base administrators. In any case, there are also substantial problems involved in keeping the duplicated data consistent with that in the data base.

It is not our intention to say the NL interfaces are inappropriate for the Intelligence Community's needs. We simply wish to point out that there are problems associated with using "high-level" interfaces in an environment in which security is important, and that this should be taken into consideration in deciding where such interfaces be used.

With regard to the suitability of ADAPT for other systems, a fundamental problem with ADAPT (and with other systems such as EUFID and LADDER) is that each time a new data base (file in the case of ADAPT) is added to the DBMS that ADAPT is interfaced to, it is necessary for programmers to prepare a new translator. Of course, each new translator is made simpler to prepare because of previous translators that have been instituted. Nevertheless, new programs must be added to the system as a whole.

It is possible to make a translator that goes into the DBMS subroutines directly. In this case, a translator from the user language would be written, which translates directly into a data base management system such as WWDMS, and then data bases could be added without adding new translators. This approach is technically feasible and such systems have been made. This approach,

a general translator from the user language to data base management system, also appears to have security advantages. The preparer of the translator need not have knowledge of the particular data bases in the system but only of the DBMS and its rules.

H. PROTOCOL IMPLEMENTATION

This section is in two parts. The first discusses some of the implementation to date in the TCP-IP area and the second part discusses certain problems which have arisen and which are being dealt with in TCP-IP implementations. Communication system development is a treacherous area and great care must be taken in contracting and monitoring systems and parts of systems. The virtue of having well defined and understood protocols extends to the business of contracting and developing the implementations of these protocols. Also, the expertise of the developer is an important factor in accommodating system problems which arise. Fortunately, TCP-IP now appear to be progressing satisfactorily and DoD's standardization program appears to be moving forward in a reasonable manner.

As examples of TCP-IP usage, the TCP and IP protocols are now in operational use on ARPANET, packet radio net (PRNET) and the Atlantic Packet Satellite Net (SATNET). On ARPANET, Digital Equipment Corporation TOPS-20 machines supply services to the U.S. Army XVIIIth Airborne Corp at Fort Bragg and the UK Royal Signals and Radar Establishment (RSRE) through gateways to the Fort Bragg packet radio network and to the RSRE pilot packet switching network. These users depend on TCP and IP for reliable, daily access to network services.

TCP-IP is also in daily use to support development of packet radio software, monitoring of gateway and satellite IMP performance, and the development of a mixed media (text, fax, voice, graphics) electronic message handling system. Implementations exist for TOPS-20, TENEX, ELF, MOS, UNIX, OS/MVT and MULTICS. User and server TELNETs have been implemented

and file transfer protocol is in progress. Development of TELNET-TCP-IP for the ARPANET TIP is to be completed in spring 1981.

In performance testing of TCP-IP, it has become clear that operating system interprocess communication mechanisms and context switching capacity can have a significant impact on the performance of any particular TCP-IP implementation. For example, an early implementation of TCP in 1976 supported only 20 kb/s running under ELF (virtual memory version) but the same software supported 50 kb/s running under MOS (no virtual memory). These particular tests were conducted on a PDP-11/20 connected by Very Distant Host interface to the ARPANET.

More recently, tests have been conducted on TCP-IP for the UNIX operating system. The PDP-11/70 using UNIX is a popular support system in DoD, and it seems appealing to add a TCP-IP program package to this operating system to support host-host communication for packet network users. In performing a study of cable bus applications in command centers (Ref. 33) it was found that the TCP that had been developed by BBN, used in conjunction with UNIX on an 11/70, required 80 percent of total computer time to run at a 6-Kbit rate. The computer was essentially "bare," running only data from a buffer through TCP.

BBN, in performing tests on TCP for DCA, found the 11/70-UNIX combination would handle data at only a 6 to 10-Kbit rate.

In order to better assess the question of whether it is the complexity of TCP or the UNIX interprocess communications problem that causes these low rates, we note that an ARPANET NCP on an 11/70 with UNIX, and also with part of the NCP in the kernel, a definite advantage with regard to speed, gives a 23-Kbit rate and that already by means of some program improvements BBN has produced a TCP running at 20 Kbits on an 11/70-UNIX combination.

Further, in performance tests for DCA/CCTC, Digital Technology Incorporated (DTI) compared the performance of a UNIX-based Network Front End for WWMCCS against a HUB-based version. HUB is a proprietary development of DTI. All tests were conducted on PDP-11/70 computers. DTI reported bandwidths on the order of 80-90 kb/s with its (modified) UNIX version of TCP-IP and 200-240 kb/s using HUB (all these figures are process-to-process). The major differences between the two systems is memory-shared interprocess communication and very fast context switching in HUB compared to UNIX, even with the RAND and BBN port modifications. BBN obtained similar results with its TCP-IP version after improvement to the program modules, getting 50-60 kb/s throughput (process-to-process) using a modification of UNIX interprocess communication based on memory sharing. It is believed that additional improvements may be obtained through a speedup of UNIX context switching.

As a further comparison, an LSI-11 with byte-at-a-time interrupt driven I/O was able to achieve 25-30 kb/s through TCP-IP operating under MOS which does not support virtual memory and therefore uses shared memory for interprocess communication. Our evaluation of the situation is based on performance probes of the BBN-TCP-UNIX combination which indicate that 80-90 percent of the time the program is in the operating system. DTI's results are comparable. We also note that communications protocol programs typically execute in 100 microsecond "strips" while UNIX interprocess communication takes on the order of one millisecond. This makes it clear that interprocess communication and context switching in the 11/70-UNIX need to be improved to support higher TCP-IP throughput.

Concerning front-end development, in order for a front end to really help, the interface between the front end and the host needs to be more efficient ("cleaner," with less overhead) than would be the case if the network protocols resided in the mainframe. TCP performs many of the functions which are needed

in a host-to-front-end protocol, but for direct attachment of a front end to the mainframe, some of these functions can be made simpler. For example, an HDLC interface could handle local flow control, error control and sequencing between the front end and mainframe. A procedure for signaling the setting up and clearing of TCP connections or the transmission of pure datagrams would thus be needed in the host-to-front-end protocols above the HDLC layer, as well as error reporting, status reporting, and so on. Reduction in overhead from embedded TCP to front end TCP by perhaps 25 percent could be expected in such a configuration.

The DoD situation concerning front-end development appears as follows. There are some large system users such as the WWMCCS computers that will require throughput of 100 kb/s (or more) in the near future. There are also a number of smaller systems (Intelligence systems provide good examples) where throughput demands are lower. The requirements of WWMCCS appear to be driving the front-end developments at DTI, where the present INFE (UNIX-based) provides 80 kb/s throughput, but which uses an 11/70 with 500 Kbytes of memory. INFE also supports up to 20 terminal users which makes it a large front end for a small user. Future plans include a version based on a communications operating system (COS) using the DTI HUB secure operating system which can operate at several hundred kilobits throughput. This system will be comparable in size and cost to the existing INFE.

A small front-end system has been developed by SRI International based on LSI-11's using the MOS operating system. Roughly 3000 words of memory are devoted to TCP-IP, and throughput in excess of 25 kb/s is achievable. Higher speeds should be possible with DMA I/O devices now being added. This system implements full TCP-IP but does not incorporate a host-to-front-end protocol which would probably be needed to make a standard interface for use of the system as a general front end.

At present, it is used as a terminal interface unit for 4-16 terminals or as a special purpose front end for Army and Navy tactical computer systems operating on packet radio nets and the ARPANET.

When AUTODIN II is introduced, small computer users may be faced with the choices of obtaining a larger front end than necessary, developing a standard front end based on LSI-11/MOS technology, or implementing a TCP-IP individual system, possibly risking loss of throughput or difficulty in responding to peak loads. If front-end TCPs are developed independently by system users, then there is likely to be some duplication of effort. Furthermore, if the interfaces and host-to-front-end protocols are not the same, users will not be free to move to larger or better front ends or mainframes without new program investment.

Without a standard front-end protocol, bids for front-end development will be very hard to compare. Actual gains offered to the host in using a front end are a function of the host-to-front-end protocol. If this protocol were identical to TCP, only buffering improvement would be provided by the front end. Therefore, the "goodness" of the system is largely determined by the differences between TCP-IP and the front-end protocol. If several front-end protocols are offered on different systems at different prices and each requires software development on individual mainframes, a real procurement problem could arise. Consequently, we recommend that a host-to-front-end protocol standards effort be put into motion by C^3I. The DTE INFE front-end protocol used by the PLATFORM network at NSA is a possible starting point. This protocol standard development can proceed in parallel with front-end development program(s) intended to provide users with reasonable alternatives for future usage without severe system penalties.

REFERENCES

1. T.C. Bartee et al., "Computer Internetting: C^3I Data Communications Networks," IDA Report P-1402, April 1979.

2. CODASYL Data Base Task Group April 1971 Report, ACM, New York.

3. M.M. Astrahan et al., "System R: A Relational Approach to Data Management," ACM Trans. on Data Base Systems 1:12, pp. 97-137, 1976.

4. C.J. Date, "An Introduction to Data Base Systems," Addison Wesley, Reading, MA, 1977.

5. E.F. Codd, "A Relational Model for Large Shared Data Banks," Comm. ACM 13:6, pp. 377-387, 1970.

6. E.F. Codd, "Extending the Data Base Relational Model," TODS, June 1980.

7. E.H. Sibley and L. Kerschberg, "Data Architecture and Data Model Considerations," Proceedings of AFIPS National Computer Conference, Dallas, Texas, June 1977.

8. R. Gerritsen, "Understanding DBTG Structures," Comm. ACM, 1976.

9. O.P. Buneman and R.E. Frankel, "FQL--A Functional Query Language," Proc. ACM SIGMOD, May 1979.

10. P.A. Shipman, "The Functional Data Model and the Data Language Daplex," TODS (to appear).

11. E.F. Codd, "Further Normalization of the Data Base Relational Model," in Data Base Systems (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, N.J., pp. 33-64, 1972.

12. E.F. Codd, "Relational Completeness of Data Base Sub-Language," ibid, pp. 65-98.

13. ACM Computing Surveys. Issue devoted to Data Base Systems, 1977.

14. W. Teitelman et al., Interlisp Reference Manual, Xerox Corp.

15. A.D. Falkoff and K.E. Iverson, "The Evolution of APL," Proc. ACM SIGPLAN Conference of History of Programming Languages, 1978.

16. J.D. Ullman, "Principles of Data Base Systems," Computer Science Press, Potomac, MD, 1980.

17. R.F. Boyce et al., "Specifying Queries as Relational Expressions: the SQUARE Data Sublanguage," Comm. ACM 18:11, pp. 621-628, 1975.

18. D.D. Chamberlin et al., "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," IBM J. Res. 20:6, pp. 560-575, 1976.

19. M. Stonebreaker et al., "The Design and Implementation of INGRES," ACM Trans. on Data Base Systems 1:3, pp. 189-222, 1976.

20. R.A. Kogan et al., "Command and Control....(WWDMS) Self Instruction Manual," PRC Data Services, McLean, VA, 1976.

21. L.R. Harris, "User Oriented Data Base Query with the ROBOT Natural Language Query System," Proc. VLDB, Tokyo, 1977

22. G.G. Hendix et al., "Developing a Natural Language Interface to Complex Data," ACM TODS, June 1978.

23. I. Kamany et al., "EUFID: The End User Friendly Interface....," Proc. VLDB, Berlin, 1978.

24. D.G. Hendrix and H. Lewis, "Mechanical Intelligence Research and Applications," SRI Project Report, 1979.

25. L.E. Erickson, M.E. Soleglad, and S.L. Westmark, "ADAPT I Final Functional Specification and Design," Logicon, Inc., January 1978.

26. L.E. Erickson, M.E. Soleglad, and S.L. Westmark, "ADAPT I Uniform Data Language (UDL): A Final Specification," Logicon, Inc., January 1978.

27. J. Rothnie et al., "MULTIBASE," Computer Corporation of America, 1979.

28. S. Kimbleton and P. Wong, "XNDM--An Experimental Network Data Manager," NCC Conference, May 1980.

29. J. Rothnie et al., "SDD-1, A System for Distributed Data Bases," CCA Report, 1978.

30. J.M. Smith and D.C.P. Smith, "Aggregation and Generalization," ACM TODS 2, 1977.

31. C.A.R. Hoare, "Proof of Correctness of Data Representations," Acta Informatica 1, 1972.

32. J. Backus, "Can Programming be Liberated from the von Neumann Style?..," Comm. ACM, 21, 1978.

33. S.F. Holmgren, A.P. Skelton, and D.A. Gomberg, "Cable Bus Applications in Command Centers," The Mitre Corporation, MTR-79W00383, October 1979.