February 1977

Language Access to Distributed Data with Error Recovery

by

Earl D. Sacerdoti

Artificial Intelligence Center

TN 140

ABSTRACT

This paper discusses an effort in the application of artificial intelligence to the access of data from a large, distributed data base over a computer network. A running system is described that provides access to multiple instances of a data base management system over the ARPANET in real time. The system accepts a rather wide range of natural language questions about the data, plans a sequence of appropriate queries to the data base management system to answer the question, determines on which machine to carry out the queries, establishes links to those machines over the ARPANET, monitors the prosecution of the queries and recovers from certain errors in execution, and prepares a relevant answer to the original question. In addition to the functional components that make up the demonstration system, equivalent functional components with higher levels of sophistication are discussed and proposed.

ACKNOWLEDGMENTS

The work described in this paper represents the joint efforts of an integrated, energetic group at SRI. Members of this group include Rich Fikes (now at Xerox PARC), Koichi Furukawa (now at ETL), Gary Hendrix, Paul Morris (now at UC Irvine), Nils Nilsson, Bill Paxton, Rene Reboh, Jane Robinson, Daniel Sagalowicz, Jonathan Slocum, and Mike Wilber. The research reported herein, other than the development of the LIFER system, was supported by the Advanced Research Projects Agency of the Department of Defense under contract DAAG29-76-C-0012 with the U. S. Army Research Office.

Language Access to Distributed Data with Error Recovery

Earl D. Sacerdoti
Artificial Intelligence Center
Stanford Research Institute

A. Introduction

Man's use of tools shapes his environment. Man's use of tools also shapes his behavior. As technology evolves more complex tools, the impositions these tools make on their users become more stringent. Although it is difficult to reproduce strings of ten digits, we have all learned to do it well, because the interface to the telephone system demands it. Although it is difficult to type very fast (the standard keyboard was originally designed to allow enough time between keystrokes to keep early typewriters from jamming), we have trained ourselves to use a suboptimal --indeed, subaverage-- arrangement of keys, because the interface to keyboard systems demands it.

As the amount of information moving across the man-machine interface increases, the impositions of machines on our behavior also increase. Since computers are our fastest and most sophisticated tools for processing information, the greatest impositions we face from our tools occur in dealing with computers. A goal of research in Artifical Intelligence is to reduce the extent of these impositions, thus making the benefits of computer use more widely available.

One example of the imposition set by the computer arises in the area of management information systems. Imagine that a user in a decision-making role knows that his data base contains some information that pertains to a decision he must make. The user wishes to extract that information from the data base and restructure, summarize, or analyze it in some way. Ideally, the user would be able to interact with the computer in his own terminology and issue a request for the information he desired. But today's computer systems typically require following a very stilted, formal mode of interaction. Even then, the user will only be able to obtain certain preprogrammed reports, and this is hardly what is needed for the typical decision maker in his role of managing by exception.

If the decision maker wants a new perspective on the information in the data base, he must call in a programmer who works with the data base on a regular basis. The programmer carries in his head four kinds of knowledge that must be used in order to gather the desired information. First, he knows how to translate the request for information from the decision maker's terms into the terms of the data that is actually stored in the data base. Second, he is able to convert the request for data from the overall data base into a series of requests for particular items of data from particular files. Third, he knows how to translate the particular requests into programs or calls on the data base management system's primitives in order to actually initiate the appropriate computation. Fourth, he knows how to monitor the execution of his request to ensure that the expected data is being obtained.

For the past year, a group at SRI has been working on automating the activities carried out by our hypothetical data base expert. The following section presents an overview of a running system that performs at least some of the expert's functions both reliably and efficiently. Our current progress on representing and using each of the four kinds of knowledge described above will be detailed in the subsequent sections.

B. Overview of the LADDER system

Our running demonstration system, called LADDER (for Language Access to Distributed Data with Error Recovery) represents an application of state-of-the-art techniques from the field of artificial intelligence in a real-time performance system. Because it consists of a number of rather independent, modular components, new capabilities can be incorporated easily as we learn how to make them run efficiently.

LADDER has been developed as a management aid to Navy decision makers, so the examples presented throughout this paper are drawn from the domain of Navy command and control. Applications of this work to other decision making and data access problems should be obvious.

The LADDER system consists of three major functional components, as displayed in Figure 1, that provide levels of buffering of the user from a data base management system (DBMS). LADDER employs the DBMS to retrieve specific field values from specific files just as a programmer might, so that the user of LADDER need not be aware of the names of

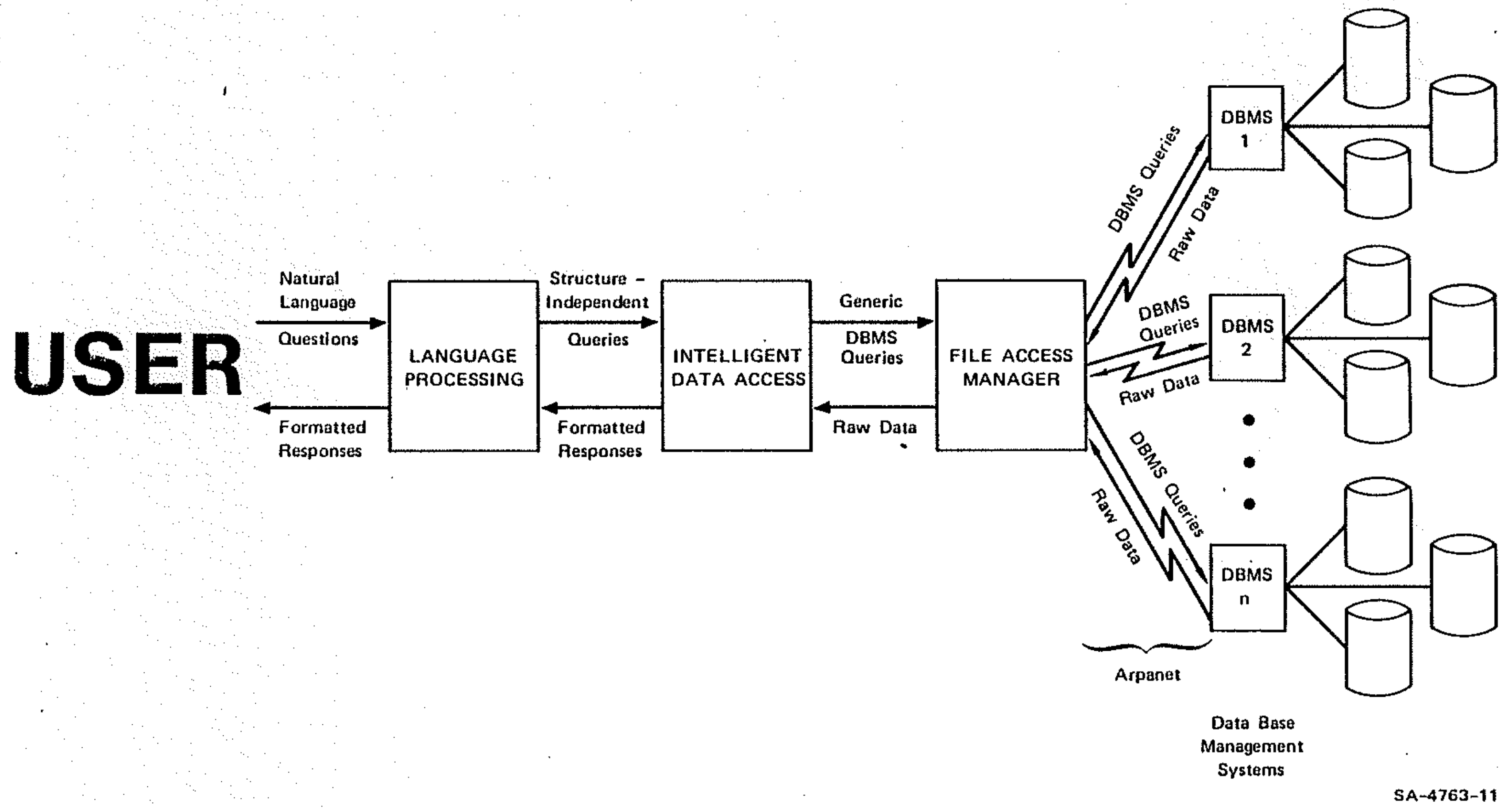

FIGURE 1 OVERVIEW OF THE LADDER SYSTEM

specific fields, how they are formatted, how they are structured into files, or even where the files are physically located. Thus the user can think he is retrieving information from a "general information base" rather than retrieving specific items of data from a highly formatted, traditional data base.

LADDER's first component accepts queries in a restricted subset of natural language. This component, called INLAND (for Informal Natural Language Access to Navy Data) produces a query or queries to the data base as a whole. The queries to the data base refer to specific fields, but make no mention of how the information in the data base is broken down into files.

For example, suppose a user types in "What is the length of the Kennedy?" (or "Give me the Kennedy's length," or even "Type length Kennedy"). INLAND would translate this into the query:

```
((? LGH) (NAM EQ 'JOHN#F.KENNEDY')),
```

where LGH is the name of the length field, NAM the name of the ship name field, and 'JOHN#F.KENNEDY' the value of the NAM field for the record concerned with the Kennedy. This query is then passed along to the second component of the system.

The queries from INLAND to the data base are specified without any presumption about the way the data is broken up into files. The second functional component, called IDA (for Intelligent Data Access) breaks down the query against the entire data base into a sequence of queries against various files. IDA employs a model of the structure of the data base to perform this operation, preserving the linkages among the records retrieved so that an appropriate answer to the overall query may be returned to the user.

For example, suppose that the data base consists of a single file whose records contain the fields

(NAM CLASS LGH).

Then, to answer the data base query issued above, IDA can simply create one file retrieval query that says, in essence, "For the ship record with NAM equal 'JOHN#F.KENNEDY', return the value of the LGH field." Suppose, however, that the data base is structured in two files,* as follows:

```
SHIP: (NAM CLASS ...)
CLASS: (CLASSNAME LGH ...)
```

In this case the single query about the Kennedy's length must be broken into two file queries. These would say, first, "Obtain the value of the CLASS field for the SHIP record with NAM equal 'JOHN#F.KENNEDY'." Then, "Find the corresponding CLASS record, and return the value of the LGH field from that record." Finally, IDA would compose an answer that is relevant to the user's query (i.e. it will return NAM and LGH data, supressing the CLASS-to-CLASSNAME link).

In addition to planning the correct sequence of file queries, IDA

----

* This is how an actual Navy data base is likely to look. Naval ships are built in classes with similar physical characteristics just as automobiles are built in models.

must actually compose those queries in the language of the DBMS. Our current system accesses, on a number of different machines, a DBMS called the Datacomputer [1] [2], whose input language is called Datalanguage. IDA creates the relevant Datalanguage by inserting field and file names into pre-stored templates. However, since the data base in question is distributed over several different machines, the Datalanguage that IDA produces does not refer to specific files in specific directories on specific machines. It refers instead to <u>generic files</u>, files containing a specific kind of record. For example, the queries discussed above might refer to the SHIP file rather than file SHIP.ACTIVE in directory NAVY on machine DBMS-3. It is the function of the third major component of LADDER to find the location of the generic files and manage the access to them.*

To carry out this function, the third component, called FAM (for File Access Manager) relies on a locally stored model showing where files are located throughout the distributed data base. When it receives a query expressed in generic Datalanguage, it searches its model for the primary location of the file (or files) to which it refers. It then establishes connections over the ARPANET to the appropriate computers, logs in, opens the files, and transmits the Datalanguage query, amended to refer to the specific files that are being accessed. If, at any time, the remote computer crashes, the file

----

* In the introduction we described four activities that our system would carry out, and here we are describing only three functional components. This is because the third activity, translating particular queries into the primitives of particular DBMS's, is shared between IDA and FAM.

becomes inaccessible, or the network connection fails, FAM can recover, and, if a backup file is mentioned in FAM's model of file locations, it can establish a connection to a backup site and retransmit the query.

The existing system, written in INTERLISP [3], can process a fairly wide range of queries against a data base consisting of some 14 files containing about 100 fields. Processing a typical question takes a very few seconds of cpu time on a DEC KA-10 computer. An annotated transcript of a sample session with the system is provided in the Appendix.

Thus LADDER provides at least some of the functions of the hypothetical data base expert in each area of expertise mentioned in the previous section. The following sections will provide more detailed views of the demonstration programs and ongoing research efforts in each of these areas.

C. Natural Language Interface

The task of providing access to the data base in the decision maker's terms is served by a functional component that accepts typed English text as input and produces formal queries to the IDA component as output. In order to provide truly natural access, this component must allow each user to expand the language definition with his own idiosyncratic language use.

We are developing a family of language interface components with increasing degrees of generality and true "understanding" of the input. In this section we describe our initial performance system. In section F. below we present our plans for an integrated family of systems that will support the staged development of increasingly sophisticated language interface components that can be integrated into the running system.

Our initial system is built around a package of programs for language definition and parsing called Language Interface Facility with Elliptical and Recursive Features (LIFER) [4]. LIFER consists of a parser and a set of interactive functions for specifying a language fragment oriented towards access of an existing computer system. The language is defined by what may be viewed as a set of productions of the form

meta-symbol => _pattern_, _expression_,

where _meta-symbol_ is a meta-symbol in the language, _pattern_ is a list of meta-symbols and symbols in the language, and _expression_ is a LISP expression whose value, when computed, is assigned as the value of the meta-symbol.

The set of productions is used by LIFER to build internal structures, called _transition trees_, that represent the language defined.* The transition trees are then used to parse user inputs in a

----

* Transition trees are a simplification of Woods' augmented transition networks [5].

top-down, left-to-right order. The response of the system to a user's input is simply the evaluation of the response expression associated with the top-level pattern that matches the input, together with all the subsidiary response expressions associated with meta-symbols contained in the expansion of the top-level pattern or any expansion of a higher-level meta-symbol.

The most important feature of LIFER from the point of view of developing a rich and usable language definition is the ease with which the grammar can be updated and the consequent changes tested. The ease of altering the grammar is such that LIFER provides a facility for casual users to add paraphrases to the language definition, in English. For example, the user might type

```
DEFINE (? LENGTH KENNEDY) TO BE LIKE (WHAT IS THE LENGTH OF
THE KENNEDY).
```

Subsequently, the system will accept

```
? COMMANDER KITTY HAWK
```

and

```
? SPEED AND CURRENT POSITION SUBS WITHIN 400 MILES OF
GIBRALTAR
```

and interpret them correctly. Questions 9 through 13 in the Appendix provide examples of this capability.

The LIFER parser has a very powerful mechanism for processing elliptical inputs, as exemplified by questions 2, 3, and 15 in the Appendix. Simple kinds of anaphoric reference, such as that shown in question 5 in the Appendix, are handled within the language definition.

The nature of the LIFER parser imposes a discipline on the developer of the language definition. For parsing to operate efficiently, the grammar must fan out as broadly as possible, and the tests applied to words in the left-to-right scan must be as cheap as possible. These goals are best satisfied with a language definition that directly encodes into the syntax most of the restrictions imposed by the semantics of the domain. Rather than contain meta-symbols like "noun phrase," the INLAND grammar is composed of entities like "primitive ship specification," "carry-verb phrase," and "pair of positions." Questions 14 and 15 in the Appendix give examples of a small fragment of the INLAND grammar. This approach of producing a semantically-oriented syntax is similar to that used by Brown and Burton [6] [7] and Waltz [8].

Using LIFER's interactive language definition facilities we have developed a language definition that we believe is one of the most extensive ever incorporated into a computer system. It accepts a wide range of queries about the information in the data base as well as queries about the definitions of data base fields and about the grammar itself. Access to the paraphrase mechanism is also provided in natural language.

D. Intelligent Data Access

A casual user would like to be able to access a data base as if it were an unstructured mass of information. Unfortunately, a data base is in reality a collection of files, often with very complex linkages among them. Even worse, a distributed data base may consist of different files on different machines, possibly handled by different DBMSs. An operation amounting to automatic problem solving is required to decide how to link up the files in the data base to extract and aggregate the information requested in a given query. An example of this situation is presented in question 7 in the Appendix, where a single question from the user's point of view requires four queries of three files to develop an answer.

Our initial efforts in this area have concentrated on access planning for collections of data bases supporting a relational model of the data [9]. The knowledge necessary to decide how to link among relations is contained in what we call a <u>structural schema</u>. The structural schema contains information for each relation describing how it can be linked to other relations. In addition it contains information about each field's counterparts in other relations and certain special-case information.

We have taken two approaches to the process of intelligent data access. The first, embodied in a program called IDA [10], uses a heuristic approach to the problem of linking among files. The structural schema is embodied in a frame-like representation [11] with individual frames defined for each field and each file. The program generates a single query at a time, examines the results, and then determines the next query to be asked. This approach can lead to suboptimal sequences of file accesses or can even fail to answer an answerable question, but it trades these shortcomings for rapid execution and straightforward extensibility. The system, encoded in INTERLISP, processes queries at the rate of about 300 milliseconds per file accessed on a DEC KA-10 processor.

Our second approach, embodied in a design for a program called DBAP (for Data Base Access Planner) [12], uses a formal, theorem-proving approach. The structural schema is represented as a set of axioms about the elements in the query language, the fields, and the files. These axioms are encoded as QLISP [13] procedures. The program builds a complete sequence of queries to the data base before beginning the actual interactions with it. Thus, it can plan an optimal sequence of file accesses, given a sufficiently detailed model of the data base. A partial implementation indicates that this approach requires 15 to 30 seconds of CPU time to build an internal representation of the sequence of queries, which is essentially an order of magnitude slower than IDA. For very large files this expenditure of planning time would undoubtedly be repaid by faster data base retrieval.

E. File Access Management

The third major component of LADDER, called FAM (for File Access Manager) [14], locates particular files within the distributed data base, establishes connections to them, and transmits to and monitors the responses from the remote computers where the files are located. FAM can recover from a range of expected types of errors by establishing links to backup files and retransmitting the failed query.

FAM accepts as input Datalanguage commands that refer not to specific files on specific machines, but to generic file names, whose precise location is presumed to vary with time. We refer to this input language as generic datalanguage. Based on a locally stored model of the distributed file system, FAM selects the appropriate specific files for the generic files mentioned in the commands. If network links to the machines where the files reside do not yet exist, they are established. If the files in question are not yet open, they are opened. Finally, the specific file names are substituted for the generic ones in the query, and the query is transmitted to the remote machine.

If certain types of errors occur during the prosecution of the query, FAM will attempt to recover. FAM currently handles two types of error conditions. The first is a failure of the network connection, which is usually noticed by the TENEX operating system as a lack of interaction over the network for a given interval of time. In this case

FAM attempts to find alternative locations for the files referenced in the query, establishes links to them, and retransmits the query. The second type of error is an explicit complaint from the Datacomputer. In practice, this usually arises when FAM's model is inaccurate, and a file that was expected to be in a particular location in fact was not. In this case, FAM updates its model and attempts to recover as before.

FAM is implemented by making strong use of the features of INTERLISP that support multiple control and access environments [3] [15]. When FAM opens a connection to a particular machine, it builds a piece of pushdown stack that contains as locally bound variables the appropriate information about that connection, and whose control environment is poised to interact with the remote machine. When a request is received by FAM that involves a file on that machine, the relevant stack fragment is hooked up with the stack representing the calling sequence of FAM through IDA through INLAND. Then the stack fragment is given control, it interacts with the remote machine, and finally control and the appropriate results are returned to the calling module.

F. Directions for Further Work

As of March 1977, the LADDER system has been brought to a stage of development where it can be used with some success and enjoyment by casual users. It accepts a rather wide range of queries against a simple data base, and exhibits a degree of robustness found in few Artificial Intelligence systems. This has been achieved by making many simplifying assumptions along the way. The language component does not understand the user's queries in any fundamental sense; rather, it reflexively invokes IDA with the appropriate arguments. The data access component assumes that all queries can be answered by joining records from various files. Both systems make strong assumptions that the user knows the kinds of information that are in the data base and is asking relevant questions.

Now that an initial system has been developed and demonstrated, we can concentrate on efforts to improve its robustness, generality, and coverage of the language. As we began our efforts in language understanding in this domain almost two years ago, we were faced with a clear trade-off between building two kinds of language systems. On the one hand, systems existed that ran reliably in real time but had very meagre semantic underpinnings, whose extensibility was clearly limited, and which did not truly understand inputs to them, in the sense that they did not compose an internal representation of their meanings. On the other hand there were systems that covered the language much more thoroughly, were better grounded linguistically, and developed a representation of what the inputs meant, but that could not be made to run in real time. What was worse, there was no clear way to integrate the efforts being put into the two approaches: the underlying control structures and language defintition systems were incompatible.

After evaluating the benefits of the LIFER approach and reexamining the requirements and behavior of the more semantically based systems, we have developed a "core language system" that is capable of supporting both approaches, and of supporting systems at intermediate positions on the tradeoff between real-time performance and linguistic grounding.

The core system, which is being developed by Bill Paxton, accepts a wide range of styles of language definition, ranging from the semantically oriented syntax of the INLAND grammar to a very rich and complex amalgam of multiple knowledge sources similar to that used by the SRI speech understanding system [16]. What is most important is that the core system accepts language definitions at intermediate points within that range as well, and it should thus constitute a vehicle for bringing more linguistically and semantically oriented styles of language processing into actual use in a staged fashion. We are developing a research plan that should enable us to simultaneously explore the issues involved in true language understanding while augmenting the power, coverage, and linguistic relevance of the demonstration system.

Our plans for data access include extensions to the input language of IDA to permit quantified queries. This will enable the system to distinguish between such queries as "What is the last reported position of each sub?" and "What is the last reported position of any sub?"

We will attempt to demonstrate the generality of the IDA approach to data base access planning by interfacing it to a CODASYL-type [17] data base in addition to the relational data base currently on the Datacomputer.

In addition to these efforts, which we expect will improve our performance system, we are continuing to progress in our longer range research. An integrated language understanding and access planning system built around the representation of knowledge in semantic network form is being designed. The longer term efforts will benefit from the tool-building involved in the performance-oriented work. Development of the performance system is guided and prioritized by the results and problems encountered in our longer term research. The early successes of this program have provided an initial demonstration of the beneficial effect of simultaneously pursuing lower risk research aimed at cost-effective performance and higher risk research aimed at advancing the state of the art.

Appendix

Transcript of Sample Session

```
@ladder

Please type in your name: X. S. Data
Do you want instructions? Yes
This program has access to 14 files which comprise a facsimile of a
Navy command and control data base. The data is stored on the
Datacomputer at NELC, with backup at CCA in Cambridge, Massachusetts.
The data base includes physical characteristics and position
information for all ships, and more detailed operational information
for U.S. Navy ships. Data about embarked U.S. Navy units, convoys of
merchant ships, and ports of departure and destination are also
available.

The system will respond to the question,
DESCRIBE THE FIELDS
with a description of all 72 fields in the data base. The information
in the data base is described in detail in a Technical Note available
from NELC.

IMPORTANT NOTE: The current version of the query answering system can
only perform some simple calculations on the values in the data base.
For example, it cannot answer questions about the composition of
organizational units or find the nearest ship to a given point.

Example questions include:
HOW FAR IS THE CONSTELLATION FROM CHARLESTON?
WHERE IS THE LOS ANGELES
THE LONGEST SHIP CARRYING VANADIUM ORE
WHEN WILL THE PHILADELPHIA REACH PORT?
WHAT U S SHIPS ARE WITHIN 400 MILES OF GIBRALTAR?


1_where is the kennedy?
PARSED!
Parse time: 501 milliseconds.
 Connecting to Datacomputer at NELC
        * FAM indicates which computer is being accessed.  The next 14
        * lines are interactions between the remote datacomputer and
        * the local FAM.
>> ;0031 770217200155   IONETI: CONNECTED TO USC-ISIR1-5400010
>> ;J150 770217200159   FCRUN: V='DC-3/00.00.5' J=2 DT='THURSDAY,
**FEBRUARY 17, 1977 12:01:59-PST' S='USC-ISIR1'
>> !0041 770217200159   DNCTNX: DATACOMPUTER GOING DOWN IN 1636 MIN
**BECAUSE TENEX IS GOING DOWN AT FRI FEB 18 77 3:30:00PM-PST FOR 330
**MIN DUE TO DEBUGING SOFTWARE
>> ;J200 770217200159   RHRUN: READY FOR REQUEST
>> .I210 770217200159   LAGC: READING NEW DL BUFFER
*> SET PARAMETERS
*< X    EXIT
<< ^Z
*> SET PARAMETERS
*< V    VERBOSITY (-1 TO 4):  1
*< P    PROCEED WITH DATALANGUAGE
         [CONFIRM WITH <CR>]
        * FAM has now established the network connection.  It proceeds
        * to log in and open the appropriate file.
<< LOGIN %TOP.BLUEFILE.GUEST ;
<< OPEN %TOP.BLUEFILE.SHIP READ;
<< OPEN %TOP.BLUEFILE.SAGALOWICZ.STDPORT1 WRITE;
        * STDPORT, STDPORT1, STDPORT2, and STDPORT3 are Datacomputer
        * 'ports' which serve both to define the network connection
        * to the Datacomputer and to specify the user's (in this case
        * IDA's) view of the data. FAM is now finally ready to
        * transmit the query.
<< FOR STDPORT1 , SHIP WITH (NAM EQ 'JOHN#F.KENNEDY') BEGIN STRING1 =
<< UIC STRING2 = VCN END;
*> TOTAL BYTES TRANSFERRED: 13
<< OPEN %TOP.BLUEFILE.TRACKHIST READ;
<< OPEN %TOP.BLUEFILE.SAGALOWICZ.STDPORT2 WRITE;
<< FOR STDPORT2 , TRACKHIST WITH (UIC EQ 'N00002') AND (VCN EQ '0')
<< BEGIN STRING1 = PTP STRING2 = PTD END;
*> TOTAL BYTES TRANSFERRED: 30
Computation time for query: 9211 milliseconds.
        * This counts cpu time spent in IDA and FAM
Real time for query: 354881 milliseconds.
        * This counts clock time from when requests are sent to the
        * Datacomputer until replies are received.
(PTP '6000N03000W' PTD 7601171200)
        * The answer means that the ship was at 60 degrees north
        * latitude, 30 degrees west longitude at noon on January 17,
        * 1976.

2_pecos
 Trying Ellipsis: WHERE IS PECOS
Parse time: 1632 milliseconds.
<< FOR STDPORT1 , SHIP WITH (NAM EQ 'PECOS') BEGIN STRING1 = UIC
<< STRING2 = VCN END;
*> TOTAL BYTES TRANSFERRED: 17
<< FOR STDPORT2 , TRACKHIST WITH (UIC EQ '*') AND (VCN EQ '13003') BEGIN
<< STRING1 = PTP STRING2 = PTD END;
*> TOTAL BYTES TRANSFERRED: 21
Computation time for query: 2797 milliseconds.
Real time for query: 87120 milliseconds.
(PTP '2131S00234E' PTD 7601171200)
        * The connection had been established and the files opened, so
        * the entire operation proceeded more quickly.
```

```
3_the fastest russian ship within 400 miles of gibraltar?
 Trying Ellipsis: WHERE IS THE FASTEST RUSSIAN SHIP WITHIN 400 MILES OF
GIBRALTAR
Parse time: 1738 milliseconds.
<< OPEN %TOP.BLUEFILE.PORT READ;
<< FOR STDPORT1 , PORT WITH (DEP EQ 'GIBRALTAR') BEGIN STRING1 = PTP
<< END;
*> TOTAL BYTES TRANSFERRED: 18
Computation time for query: 1197 milliseconds.
Real time for query: 40142 milliseconds.
<< FOR STDPORT2 , TRACKHIST WITH((( (PTPNS EQ 'N') AND (PTPX LT
<< '04245') AND (PTPX GT '02845') AND (PTPEW EQ 'W') AND (PTPY LT
<< '01230') AND (PTPY GT '00000')) OR( (PTPNS EQ 'N') AND (PTPX LT
<< '04245') AND (PTPX GT '02845') AND (PTPEW EQ 'E') AND (PTPY LT
<< '00170') AND (PTPY GT '00000')))) BEGIN STRING1 = PTP STRING2 = PTD
<< STRING3 = UIC STRING4 = VCN END;
*> TOTAL BYTES TRANSFERRED: 41
<< OPEN %TOP.BLUEFILE.SAGALOWICZ.STDPORT WRITE;
<< BEGIN DECLARE X STRING (,100) ,D=' ' DECLARE Y STRING (,100) ,D=' '
<< DECLARE Y1 STRING (,100) ,D=' ' DECLARE Y2 STRING (,100) ,D=' ' Y =
<< '*' Y1 = '*' Y2 = '*' X = '00.0' FOR SHIP WITH (NAT EQ 'UR') AND
<< (UIC EQ '*') AND (VCN EQ '99005') IF MCSF LT '99.9' AND X LT MCSF
<< THEN BEGIN Y = NAM X = MCSF END STDPORT.STRING1 = Y STDPORT.STRING2
<< = X END;
*> TOTAL BYTES TRANSFERRED: 45
Computation time for query: 5805 milliseconds.
Real time for query: 143126 milliseconds.
(NAM 'AMPERMETR' MCSF '15.0' PTP '3600N01130W' PTD 7601171200)

4_who commands the biddle
PARSED!
Parse time: 711 milliseconds.
          * FAM keeps track of the number of open files and ports.  Since
          * there is a limit to the number of these that the Datacomputer
          * can support, FAM maintains a working set of open files and
          * ports.  The least recently used is the one to be closed.
<< CLOSE PORT ;
<< OPEN %TOP.BLUEFILE.UNIT READ;
<< FOR STDPORT1 , UNIT WITH (ANAME EQ 'BIDDLE') BEGIN STRING1 = RANK
<< STRING2 = CONAM END;
*> TOTAL BYTES TRANSFERRED: 32
Computation time for query: 1754 milliseconds.
Real time for query: 36638 milliseconds.
(RANK 'CAPT' CONAM 'J.TOWNES')
```

```
5_what is his lineal number?
               HIS => ((NAM EQ 'BIDDLE') (? RANK) (? CONAM))
        * INLAND's interpretation of 'his' is the call to IDA for 'Who
        * commands the Biddle?'
PARSED!
Parse time: 902 milliseconds.
<< FOR STDPORT1 , UNIT WITH (ANAME EQ 'BIDDLE') BEGIN STRING1 = LINEAL
<< STRING2 = RANK STRING3 = CONAM END;
*> TOTAL BYTES TRANSFERRED: 36
Computation time for query: 1218 milliseconds.
Real time for query: 32573 milliseconds.
(LINEAL 4850 RANK 'CAPT' CONAM 'J.TOWNES')

6_what ships have destination luanda
PARSED!
Parse time: 1075 milliseconds.
<< CLOSE TRACKHIST ;
<< OPEN %TOP.BLUEFILE.MOVES READ;
<< FOR STDPORT1 , MOVES WITH (DST EQ 'LUANDA') BEGIN STRING1 = UIC
<< STRING2 = VCN END;
*> TOTAL BYTES TRANSFERRED: 34
<< FOR STDPORT1 , SHIP WITH (UIC EQ '*') AND
<< (VCN EQ '22014' OR VCN EQ '22012') BEGIN STRING1 = NAM STRING3 =
<< VCN END;
*> TOTAL BYTES TRANSFERRED: 74
Computation time for query: 3431 milliseconds.
Real time for query: 78071 milliseconds.
(NAM 'TARANTED')
(NAM 'TARU')

7_what ships faster than the kennedy are within 500 miles of naples?
PARSED!
Parse time: 1232 milliseconds.
        * One question from the user's viewpoint can involve many
        * data base queries.  First, LADDER asks, 'Where is Naples?'
<< CLOSE STDPORT2 ;
<< OPEN %TOP.BLUEFILE.PORT READ;
<< FOR STDPORT1 , PORT WITH (DEP EQ 'NAPLES') BEGIN STRING1 = PTP END;
*> TOTAL BYTES TRANSFERRED: 18
Computation time for query: 2301 milliseconds.
Real time for query: 91551 milliseconds.
        * 'What is the maximum cruising speed of the Kennedy?'
<< FOR STDPORT1 , SHIP WITH (NAM EQ 'JOHN#F.KENNEDY') BEGIN STRING1 =
<< MCSF END;
*> TOTAL BYTES TRANSFERRED: 10
Computation time for query: 1371 milliseconds.
Real time for query: 29867 milliseconds.
        * 'What are the data base keys of the ships within 500 miles
        * of Naples?'
<< CLOSE STDPORT ;
<< OPEN %TOP.BLUEFILE.TRACKHIST READ;
<< FOR STDPORT1 , TRACKHIST WITH(( (PTPNS EQ 'N') AND (PTPX LT
<< '05345') AND (PTPX GT '03545') AND (PTPEW EQ 'E') AND (PTPY LT
<< '02330') AND (PTPY GT '00530'))) BEGIN STRING1 = UIC STRING2 = VCN
<< END;
*> TOTAL BYTES TRANSFERRED: 60
        * 'Return the name of any of the four ships within 500 miles of
        * Naples whose maximum cruising speed exceeds 35 knots.'
<< FOR STDPORT1 , SHIP WITH (MCSF GT '35.0') AND
<< (UIC EQ 'N00003' OR UIC EQ 'N00001' OR UIC EQ '*') AND
<< (VCN EQ '0' OR VCN EQ '99025' OR VCN EQ '99024') BEGIN STRING1 = NAM
<< STRING2 = UIC STRING3 = VCN END;
*> TOTAL BYTES TRANSFERRED: 0
Computation time for query: 4392 milliseconds.
Real time for query: 149401 milliseconds.
NONE

8_how far is the kitty hwk from gibraltar
            spelling-> HAWK
        * Spelling correction is performed using the INTERLISP spelling
        * corrector with a list of candidates composed of valid words
        * that could have led to parses of the sentence if they had
        * replaced the misspelled word.
PARSED!
Parse time: 2077 milliseconds.
<< FOR STDPORT1 , PORT WITH (DEP EQ 'GIBRALTAR') BEGIN STRING1 = PTP
<< END;
*> TOTAL BYTES TRANSFERRED: 18
Computation time for query: 1577 milliseconds.
Real time for query: 39213 milliseconds.
<< FOR STDPORT1 , SHIP WITH (NAM EQ 'KITTYHAWK') BEGIN STRING1 = UIC
<< STRING2 = VCN END;
*> TOTAL BYTES TRANSFERRED: 13
<< CLOSE UNIT ;
<< OPEN %TOP.BLUEFILE.SAGALOWICZ.STDPORT2 WRITE;
<< FOR STDPORT2 , TRACKHIST WITH (UIC EQ 'N00003') AND (VCN EQ '0')
<< BEGIN STRING1 = PTP STRING2 = PTD STRING3 =
<< (GCDIST (3545 , 'N' , 530 , 'W' , PTPX , PTPNS , PTPY , PTPEW)) END;
*> TOTAL BYTES TRANSFERRED: 34
Computation time for query: 3129 milliseconds.
Real time for query: 55606 milliseconds.
(((PPTP '3545N00530W' PTP '3700N01700E' PTD 7601171200 GCDIST 1087)))
        * The distance was 1087 nautical miles.
```

```
9_is there a doctor embarked in the jfk
        * 'jfk' is not in the lexicon (yet).
THE PARSER DOES NOT EXPECT THE WORD "JFK" TO FOLLOW
"IS THERE A DOCTOR EMBARKED IN THE"
OPTIONS FOR NEXT WORD OR META-SYMBOL ARE:
end-of-list

10_define jfk to be like kennedy
        * The lexicon is augmented by the user, in natural language.
        * 'jfk' will henceforth be accepted by INLAND, and will be
        * interpreted in the same way that 'kennedy' is.
FINISHED

11_redo 9
        * The INTERLISP redo feature is used to reinvoke question 9.
PARSED!
Parse time: 1327 milliseconds.
<< CLOSE MOVES ;
<< OPEN %TOP.BLUEFILE.UNIT READ;
<< CLOSE PORT ;
<< OPEN %TOP.BLUEFILE.SAGALOWICZ.STDPORT WRITE;
<< BEGIN DECLARE X INTEGER X = 0 FOR UNIT WITH (DOCTR EQ 'D') AND
<< (ANAME EQ 'JOHN#F.KENNEDY') X=X+1 STDPORT.STRING1=X END;
*> TOTAL BYTES TRANSFERRED: 16
Computation time for query: 3572 milliseconds.
YES

12_define ($ length jfk )
    ...to be like (what is the length of the jfk)
    ...
        * Here we define a new grammatical construction by use of the
        * LIFER paraphrase feature.  Since the system's only
        * understanding of 'what is the length of the jfk' is the call
        * on the data base, the question is answered as a side-effect
        * of defining the paraphrase.
PARSED!
Parse time: 596 milliseconds.
<< FOR STDPORT1 , SHIP WITH (NAM EQ 'JOHN#F.KENNEDY') BEGIN STRING1 =
<< LGHN END;
*> TOTAL BYTES TRANSFERRED: 10
Computation time for query: 1514 milliseconds.
Real time for query: 46331 milliseconds.
(LGHN 1072)
        * The question was answered; the kennedy is 1072 feet long.
        * LIFER now prints out the new production rule and associated
        * response expression that embody the generalization of the
        * paraphrase given by the user.
LIFER.TOP.GRAMMAR  =>  $ <RELN> <ENTITY>
F0282
($ <RELN> <ENTITY>)
```

```
13_$ current position and heading all los angeles class submarines
        * The new pattern can immediately be used as part of the
        * grammar.
PARSED!
Parse time: 1508 milliseconds.
<< CLOSE TRACKHIST ;
<< OPEN %TOP.BLUEFILE.SHIPCLASCHAR READ;
<< FOR STDPORT1 , SHIPCLASCHAR WITH (SHIPCLAS EQ 'LOS#ANGELES') AND
<< ((TYPE1 EQ 'S') AND (TYPE2 EQ 'S')) BEGIN STRING1 = SHIPCLAS END;
*> TOTAL BYTES TRANSFERRED: 30
<< CLOSE STDPORT2 ;
<< OPEN %TOP.BLUEFILE.SHIPCLASDIR READ;
<< FOR STDPORT1 , SHIPCLASDIR WITH (SHIPCLAS EQ 'LOS#ANGELES') BEGIN
<< STRING1 = UIC STRING2 = VCN END;
*> TOTAL BYTES TRANSFERRED: 39
<< CLOSE UNIT ;
<< OPEN %TOP.BLUEFILE.TRACKHIST READ;
<< CLOSE STDPORT ;
<< OPEN %TOP.BLUEFILE.SAGALOWICZ.STDPORT2 WRITE;
<< FOR STDPORT2 , TRACKHIST WITH
<< (UIC EQ 'N00009' OR UIC EQ 'N00008' OR UIC EQ 'N00007') AND (VCN EQ
<< '0') BEGIN STRING1 = PTP STRING2 = PTD STRING3 = PTC STRING4 = UIC
<< END;
*> TOTAL BYTES TRANSFERRED: 114
Computation time for query: 7220 milliseconds.
Real time for query: 226777 milliseconds.
(PTP '0000N04500E' PTD 7601171200 PTC NAVAIL)
(PTP '1500S01300E' PTD 7601171200 PTC NAVAIL)
(PTP '3700S02000E' PTD 7601171200 PTC NAVAIL)

        * In addition to answering questions about the data base, LADDER
        * can answer questions about its own language definition.
14_how is <entity> used
PARSED!
"<ENTITY> " may be any sequence of words following one of the patterns:
<ENTITY> =>  <BASIC.ENTITY>
<ENTITY> =>  <BASIC.ENTITY> <AGENT>
<ENTITY> =>  <3RD.PERSON.SINGULAR.PRONOUN>
<ENTITY> =>  <3RD.PERSON.PLURAL.PRONOUN>
<ENTITY> =>  <EMBARKED.UNIT.SPEC>
<ENTITY> =>  <COMMANDER.SPEC>
<ENTITY> =>  <1ST.PERSON.PRONOUN>
<ENTITY> =>  <AGENT> <REF> <ENTITY>
Finished
```

```
15_<agent>
Trying Ellipsis: HOW IS <AGENT>USED
"<AGENT> " may be any member of the set {CAPTAIN COMMANDER CONAM
COUNTRIES COUNTRY OWN OWNER SKIPPER}
"<AGENT> " may be any sequence of words following one of the patterns:
<AGENT> =>  COMMANDING OFFICER
<AGENT> =>  <DET> <AGENT>
<AGENT> =>  <3RD.PERSON.SINGULAR.MASCULINE.PRONOUN>
Finished

16_done
File closed 17-FEB-77 15:44:03
^Z
*> SET PARAMETERS
*< Q    QUIT
          [CONFIRM WITH <CR>]
Thank you
@
```

REFERENCES

1. J. Farrell, "The Datacomputer - A Network Data Utility," Proc. Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, Ca., pp. 352-364 (May 1976).

2. Computer Corporation of America, "Datacomputer Version 1 User Manual," CCA, Cambridge, Mass. (August 1975).

3. W. Teitelman, "INTERLISP Reference Manual," Xerox PARC, Palo Alto, Ca. (December 1975).

4. G. G. Hendrix, "Human Engineering for Applied Natural Language Processing," submitted to 5th IJCAI, Cambridge, Mass. (August 1977).

5. W. A. Woods, "Transition Network Grammars for Natural Language Analysis," CACM, Vol. 13, No. 10, pp. 591-606 (October 1970).

6. J. S. Brown and R. R. Burton, "Multiple Representations of Knowledge for Tutorial Reasoning," pp. 311-349, D. G. Bobrow and A. Collins, eds., Representation and Understanding (Academic Press, New York, 1975).

7. R. R. Burton, "Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems," BBN Report No. 3453, Boston, Mass. (December 1976).

8. D. Waltz, "Natural Language Access to a Large Data Base: an Engineering Approach," Proc. 4th IJCAI, Tbilisi, USSR, pp. 868-872 (September 1975).

9. E.F. Codd, "A Relational Model of Data for Large Shared Data Banks," CACM, Vol. 13 No. 6, pp. 377-397 (June 1970).

10. D. Sagalowicz, "IDA: An Intelligent Data Access Program," submitted to SIGMOD Conference, Toronto, Canada (August 1977).

11. M. Minsky, "A Framework for Representing Knowledge," Artificial Intelligence Memo No. 306, MIT, Cambridge, Mass. (June 1974).

12. K. Furukawa, "A Deductive Question Answering System on Relational Data Bases," submitted to 5th IJCAI, Cambridge, Mass. (August 1977).

13. B. M. Wilber, "A QLISP Reference Manual," Artificial Intelligence Center Technical Note No. 118, Stanford Research Institute, Menlo Park, Ca. (March 1976).

14. P. Morris and D. Sagalowicz, "Managing Network Access to a Distributed Data Base," forthcoming in _Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks_, Berkeley, Ca. (May 1977).

15. D. G. Bobrow and B. Wegbreit, "A Model for Control Structures for Artificial Intelligence Programming Languages," _Proc. 3rd IJCAI_, Stanford, Ca., pp. 246-253 (August 1973).

16. D. Walker et al., "An Overview of Speech Understanding Research at SRI," submitted to 5th IJCAI, Cambridge, Mass. (August 1977).

17. CODASYL Data Base Task Group, _April 1971 Report_ (ACM, New York, 1971).