# MANAGING NETWORK ACCESS TO A DISTRIBUTED DATABASE

Technical Note 144

June 1977

By: Daniel Sagalowicz
P. Morris

Artificial Intelligence Center

## ABSTRACT

This paper describes a program, FAM--for File Access Manager--, used to access to data distributed over a computer network. FAM is part of a system which allows a casual user to express queries in a restricted subset of English, about a database of fourteen files stored redundantly on several Datacomputers. FAM is responsible for the data distribution aspects of the whole system: it establishes the network connections, decides which files in redundant groups to use, opens and closes them as needed. This paper presents some of the techniques which are used and discusses the limitations of this particular approach.

ACKNOWLEDGMENTS

The work described in this paper benefited from discussions with various members of the Artificial Intelligence Center at SRI International. Special mention should be given to Koichi Furukawa (now at ETL), Earl Sacerdoti, Jonathan Slocum, amd Michael Wilber. The research reported herein 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.

# Managing Network Access to a Distributed Database.

P. Morris* and D. Sagalowicz
Artificial Intelligence Center
SRI International
Menlo Park, California

## A. INTRODUCTION

This paper discusses one of the components of LADDER (Language Access to Distributed Data with Error Recovery) [1], a database access system being developed at SRI. The goal of this system is to provide to casual users easy access to information stored on multiple computers, under various database management systems. (The need for such a system has already been amply covered in the pertinent literature.) A simple overview of LADDER (see Figure 1) is briefly explained below.

The first component INLAND (Interactive Natural Language Access to Navy Data), provides the user with the ability to ask questions about Navy information contained in databases similar to those used by the Navy. Although INLAND does comprehend Navy terminology and semantics, it is not aware of the specific structure of this information. In particular, it does not know how the database is subdivided

---

* P. Morris is currently associated with the Computer Science Department, University of California, Irvine, California.

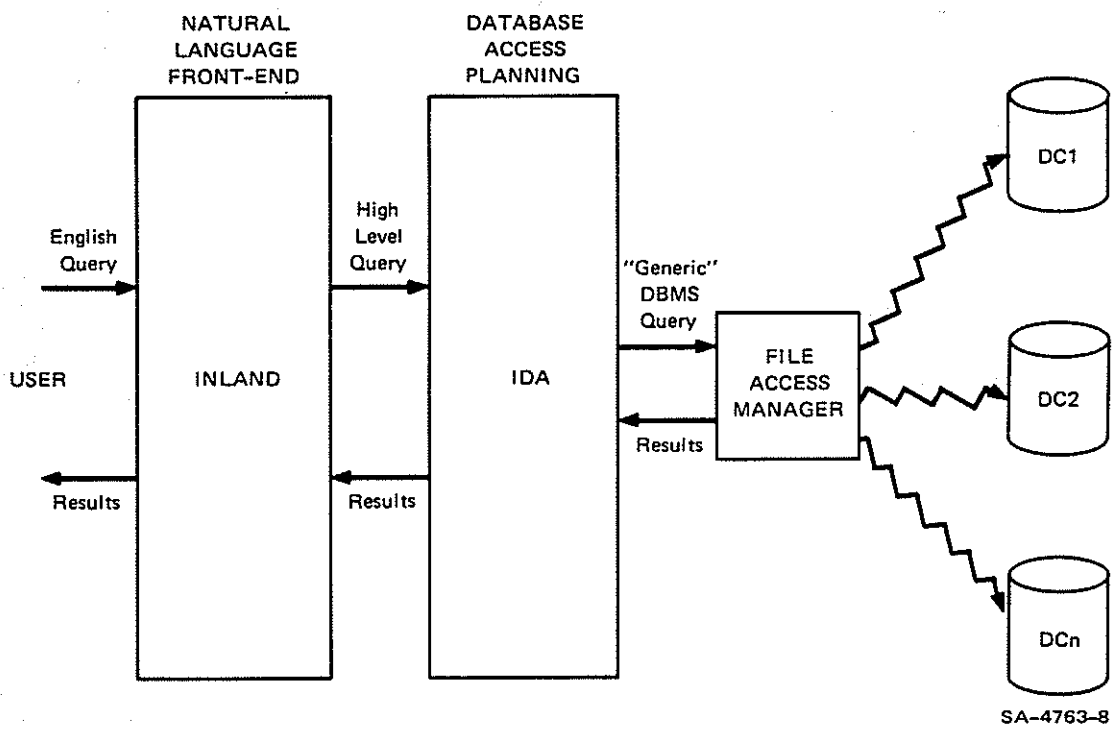
NATURAL
LANGUAGE
FRONT-END
DATABASE
ACCESS
PLANNING
English
Query
USER
INLAND
Results
High
Level
Query
Results
IDA
"Generic"
DBMS
Query
Results
FILE
ACCESS
MANAGER
DC1
DC2
DCn
SA-4763-8

FIGURE 1

into files and records, or where those databases are currently located. It therefore translates the query into a formal, high-level query to the next component of the system, IDA, without any mention of the database structure.

The next component, IDA (Intelligent Data Access), [2], is given a description of the database structure in a structural schema, which is used to translate the query into a query program to be issued to database management systems (DBMSs). One query to IDA may typically generate several queries to the DBMSs which will then access several files. Moreover, in our case we assume that these files may be distributed on a computer network, such as the ARPANET.

In our implementation, information about the precise location of each file is not given to IDA. Instead, because IDA is given the illusion that all files belong to one computer, its resulting query program does not include the actual name of any file on any computer. Rather, it uses generic file names, and the next component issues each query to the appropriate computer, replacing the generic file names by the actual file names used on those particular computers.

The final component is FAM (File Access Manager), is the central topic of this paper. FAM is in charge of connecting with various computers, finding the most recent versions of pertinent files, issuing the actual DBMSs queries, and recovering from errors.

We now use only one actual DBMS, the Datacomputer developed by CCA [3], but hope to extend the system soon so as to be able to access several DBMSs.

Because FAM has been developed to be used independently of the other components of the system, we will assume in this paper that a user may interface to FAM either directly or via a different front end than that provided in LADDER.

## B. THE FILE ACCESS MANAGER

### 1. Overview

FAM's purpose is to provide reliable access to a distributed database while insulating the user or calling program from inessential details concerning that database. FAM allows the user to specify files by means of generic names: each generic file name corresponds to a primary file and perhaps also to secondary, backup copies of it. A local model of actual files and locations corresponding to these references is maintained. Given a request, FAM decides how best to meet it in terms of choosing actual files and a single location in which to assemble them. It then proceeds to make the necessary connections, logging into remote locations, opening and closing files, and moving data as required. In the event of certain classes of mishaps FAM will take appropriate recovery action. Finally, FAM passes on the query, with such simple translations as are necessary, to the database and returns the answer to the caller.

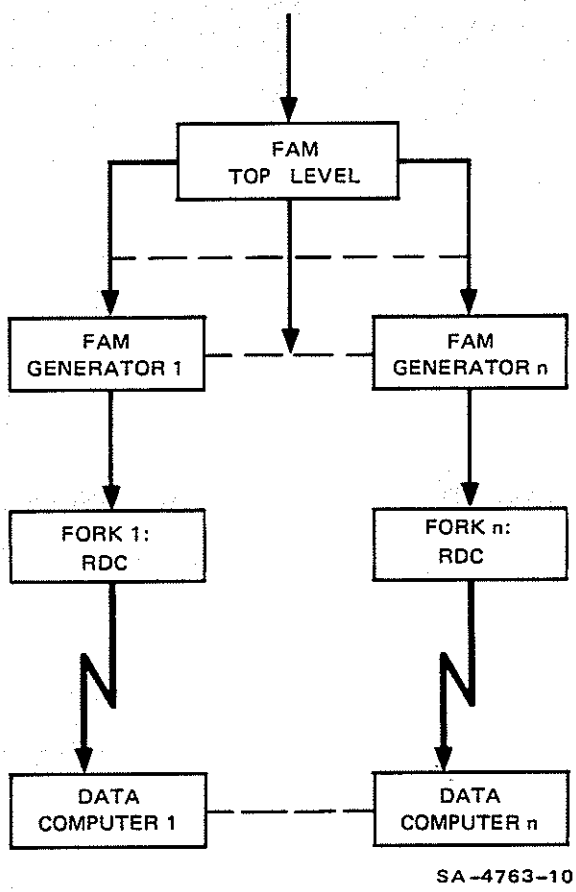
FAM
TOP LEVEL
FAM
GENERATOR 1
FAM
GENERATOR n
FORK 1:
RDC
FORK n:
RDC
N
N
DATA
COMPUTER 1
DATA
COMPUTER n
SA-4763-10

FIGURE 2

The basic structure of FAM is as shown in Figure 2. It is a set of LISP functions that communicates with multiple datacomputers by using RDC, a low-level program developed by CCA that provides the user with the basic functions necessary to access the datacomputer over the ARPANET. Several TENEX forks, one per datacomputer, are set up by FAM, running RDC: each such RDC receives its commands from FAM, located on an upper fork, via a LISP-RDC interface and transmits them to the datacomputer to which it is connected.

FAM maintains a local model of the files it can access in a table stored in a disk file. This table contains information necessary for logging on and accessing files, as well as the locations corresponding to generic files. In addition, some files are identified as temporary, i.e. subject to later deletion. In addition to this table and the direct inputs, FAM's action is controlled by certain global parameters, such as verbosity, which can be set by the user or calling program.

An important aspect of FAM is its ability to recover from certain types of errors. In a system involving intermachine communication in a network, certain kinds of errors inevitably occur as a result of violent termination by one or more participants. FAM deals with two basic error types. A type 1 error involves the datacomputer crashing during a communication episode, or being down when the episode begins. The crash must be detectable at the

interface. FAM responds by finding a new, hopefully more placid, location for the file in question. Access to the distributed database system is thus protected from the vagaries of operating computer systems by redundant storing of files on different machines.

Type 2 errors are more subtle. They occur when the datacomputer is unable to interpret FAM's command. Generally this means that the local model retained by FAM has become inaccurate. For example, a temporary file created by FAM in a previous interaction may have been deleted by another user, without a corresponding updating of FAM's local model. Depending on the circumstances, FAM will take action to continue the interaction and restore the model. There is thus a tendency for inaccuracies to disappear from the model over time.

## 2. Capabilities

This section presents a user's view of FAM's operations*. A typical user will employ the commands FAMINIT, FAMSEND, and FAMEND (a second level of commands is described in the user's guide; generally they allow more detailed control over FAM).

---

* Unless otherwise noted, statements concerning files and generic files also hold for ports and generic ports. A port is an input/output path, as well as a logical view of a file, on the datacomputer; many datacomputer commands do not distinguish between files and ports.

FAMINIT is executed once to initialize the system. It may be given optional arguments specifying the model to be used (if these are not given, a default is assumed) and generic files to be immediately located and opened. FAMSEND may then be used repeatedly to query the datacomputers and return responses. Two arguments are passed to FAMSEND: a list of generic names (of files and ports) and an expression in generic datalanguage representing the query. Generic datalanguage is identical to ordinary datalanguage, except that references to files are replaced by references to generic files. The list of generic names includes all those occurring in the generic datalanguage expression.

FAM will now search the table for the primary locations of the generic names mentioned (a location is a pair consisting of a machine and a specific file/port available on that machine).

First FAM will choose a machine in which to assemble the referenced files and ports. This is necessary because the datalanguage expression must ultimately be handed to one machine for action. Currently, this decision is made on the basis of the least number of files and ports to be copied. Aware of copies in temporary files that it has made on previous occasions, FAM will first try to open these. In general, FAM tries to avoid doing unnecessary work. If this fails (type 2 error), FAM will recopy the file. For each location needed, FAM will attempt the following:

1. If the machine is not already connected FAM will make the network connection and log in under a suitable account.
2. If the file or port is not open, FAM will open it (files are opened for reading, ports for writing).
3. If steps 1 and 2 are successful FAM is said to have accessed the location. For any generic name, if accessing of the primary location fails, the secondary locations are attempted in turn, with FAM doing the bookkeeping necessary to keep track of connected machines and opened files. Datacomputers allow a maximum number (six, currently) of opened files and ports; FAM will close least-recently used ones as necessary to open new ones.

FAM now transfers such files/ports as it needs, moving them first to the local computer--the one on which FAM resides--and then on to the new machine. As of this writing, the datacomputer does not allow the transfer of files directly from one datacomputer to another. FAM gives the copies names that reflect their origins and notes them in the model. If all goes well, the references to generic names in the generic datalanguage are now replaced by specific names in the selected machine, and the revised datalanguage is sent to that machine for action. The reply is read and passed back to the calling program.

When the user wants to end the session, FAMEND is called, causing the deletion of all temporary files (i.e., all files copied from one machine to another as described above), including those created during previous sessions that have not yet been deleted--for example, in case of a computer crash in the middle of a session. One available feature allows copies to be created as permanent files which are not deleted at FAMEND. After the deletions all the connections to datacomputers are cleanly closed. FAM cleans up the local model and then terminates.

To summarize, FAM gives a user the semblance of dealing with a single reliable datacomputer that is connected and whose files and ports are always open.

## 3. Implementation Information

The file access manager communicates with datacomputers via set of INTERLISP functions that communicates with a slightly modified version of the RDC package [4]. For each datacomputer accessed, the subsystem RDC is started up on a separate fork, which is then synchronised with a pseudoteletype (PTY). A PTY is a local buffer that is treated by the fork as though it were a terminal. That is, commands placed in the buffer are executed by the fork, and output from the fork that would normally go to the terminal is placed in the buffer allowing FAM to operate several RDC forks as though it were a human user with several terminals (and jobs).

To assist in the bookkeeping associated with each fork, FAM maintains separate control and data environments for each datacomputer, implemented by the Spaghetti-Stack capability of INTERLISP [5]. A new generator is initialized whenever a datacomputer is accessed for the first time. The generator handle is stored as the value of an atom naming the datacomputer. Much of the information pertaining to the datacomputer is held in the form of variable bindings within the generator. The control state of the generator is also utilized to represent certain conditions. For example, because of restrictions on the number of PTYs available, it is sometimes necessary to desynchronize a fork so that its PTY can be used elsewhere. When this happens the generator is suspended in a state such that when it is revived it will acquire a PTY from the available pool.

In order to economize on stack space, when FAMINIT is executed a suitable control environment is created and a stack pointer set to it. This is intended to be close to the top level of INTERLISP, i.e. the stack is shallow at this moment. When the generators for each datacomputer are subsequently created they are "hung" from this point on the stack, i.e., the expressions creating them are evaluated in the control and access environment saved by the stack pointer. Individual generators can then communicate with each other through common variable bindings. The top level of FAM is able to communicate by evaluating SETQs in these access environments, as well as by passing arguments directly.

Each generator keeps a local model of the datacomputer situation: it has a list of the open files and ports, the directories to which they belong, the directory to which the user is logged in on the particular datacomputer, and so on.

Backtracking, needed to deal with such errors as datacomputer crashes, is accomplished via the Spaghetti Stack: a stack pointer is set to the appropriate position for return in the event of failure. Note that during FAMEND all outstanding stack pointers created by FAM are restored and the stack is returned to its preFAM state.

## 4. FAM Control Structure

The top level function of this structure is FAMSEND, which is passed a list of files and ports to be used, and a generic datalanguage query. The locations of each file and each port are then determined from the model. Subsequently, a central machine for assembling files and ports is selected. Then, each location is handled in turn; subsequent evaluation takes place in the access environment of each machine as set up by FAMINIT. For each location FAM decides whether it needs to do any copying and sends appropriate commands to the datacomputers. Files are opened as needed. Finally, FAM substitutes a specific name for the corresponding generic name in the datalanguage. All commands to each datacomputer pass through a generator. The first time each datacomputer is referenced, FAM establishes a copy of the generator with appropriate arguments and hangs

it from the correct position on the stack. Thereafter, FAM merely revives the generator and passes on commands. Each generator evokes the functions to set up the forks, control the datacomputer, react to errors, and perform local bookkeeping.

## 5. Special Problems

Many difficulties stemmed from the desirability of maintaining correctness in the local model. A single disk file was used to store the model, with open access to all users. In one instance however, simultaneous asynchronous use allowed one FAM use to overwrite the alterations another had just made. To avoid this, when one user of FAM wishes to alter the table it (1) opens the file for both reading and writing in a mode that temporarily locks out other users, (2) reads the file, (3) changes the core image, (4) rewrites the file, and finally (5) closes the file so it becomes available to other FAM users. If another user tries to access the file while it is locked, this query will hang until the table becomes available.

Deleting file copies after a datacomputer had crashed also created a difficulty. It was decided to note these not-yet-made deletions in the table and take care of them in subsequent sessions. Note that it is impossible to ensure absolute truth in the model. To see this, one need only consider the possibility of a crash of the local machine at the critical moment between making a change in the

datacomputer and updating the model. We attempted as far as possible to allow for this and to automatically make later corrections. The problems looms even larger when one contends with reconciling multiple models at different sites using the FAM system.

A minor irritation arose in connection with error messages from the datacomputer. Although the datacomputer supplied sufficient information in English for the user to distinguish among kinds of errors, the error code available to the program was uniform for a wide range of errors. Not wishing to parse the English messages, we were fortunately able to guess the nature of many of the errors by their contexts.

Aspects of the Spaghetti Stack forced certain inelegancies on the FAM system. Communication is awkward between the generators and FAM's top level. We suggest that generators should be able to freely access variables bound at the top level.

Another problem with generators is their loss when a control-D is executed within the generator, i.e., when the user wishes to force INTERLISP to go back to its top level. This was prevented by redefining the Spaghetti GENERATE function.

## 6. Suggested Improvements

One simple improvement would allow the system to choose to operate in fast or reliable modes. Currently, with each new query the system first tries to open primary locations even if secondary ones are already connected, because the information in the primary locations is presumed to be better. Since attempting access to another datacomputer can be slow, some users might prefer to have connected secondary locations always used for certain files. A related question involves distinguishing between rapidly changing and static files, and using this information to determine the acceptability of secondary locations.

Considerable room for improvement exists in selecting the best machine for assembly of files and ports residing on multiple machines. One scheme involves modeling the "weight" of each file, i.e., the approximate time of transfer, and using it to compute a minimum time assembly.

Temporary files and ports could also be dated when they are created or reused and then protected against deletion by other users until a certain period has elapsed. This would prevent the current possibility of a user's temporary files being deleted by another evocation of FAM during a job. This is actually very unlikely: the files would have to be closed at the time. In any case, FAM would recreate them as needed.

We indicated earlier that certain datacomputer failures cannot be detected at the FAM-RDC interface. These generally involve failure of the particular job but not the overall system. When RDC times out, it is not possible to distinguish these conditions from those of a slow, heavily loaded datacomputer. When sufficient facilities for checking on the datacomputer, are provided FAM can be extended to recover in these situations.

The greatest deficiency in FAM is of course that it is a file ACCESS manager: it does not provide for the creation or update of the database. It should also be understood that we have addressed here only a small portion of the problems arising with a true distributed database facility.

## C. CONCLUSION

We have described a file access manager that gives its users the capability of accessing files distributed over a computer network. Using a local model, FAM is able to decide which computers to use to execute the user query. It also recovers in many cases of error that may occur during the response to a query. Although FAM falls far short of the goals of a real distributed database management system, it has been an extremely useful and attractive part of the LADDER system.

## REFERENCES


1. E. Sacerdoti, "Language Access to Distributed Data with Error Recovery," Fifth International Joint Conference on Artificial Intelligence, Cambridge, Mass., Aug. 1977.

2. D. Sagalowicz, "IDA: an Intelligent Data Access Program," (proposed for presentation at the Third International Conference on Very Large Data Bases, Tokyo, Japan, Oct. 1977.

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

4. J. Farrell, "RDC: a Program to Run the Datacomputer," Technical Memo, Computer Corporation of America, Cambridge, Mass. (June 1974).

5. D. G. Bobrow and B. Wegbreit, "A Model and Stack implementation of Multiple Environments," pp.591-603, Comm. of the ACM, Vol. 16, No. 10 (Oct. 1973).