22

INFORME TECNICO

THE FILE DESCRIPTOR: USE OF A DESCRIPTIVE TOOL TO RETRIEVE GENERAL QUERIES TO FILES. *

ADOLFO GUZMAN ARENAS

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

APARTADO POSTAL: 14-740, MEXICO 14, D.F.
TEL. 754-02-00 EXT. 141; TELEX: 017-72-826 PPTME
AVE. INSTITUTO POLITECNICO NACIONAL 2508
ZACATENCO, 07000 MEXICO, D.F.

Artículo 57

CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL IPN
DEPARTAMENTO DE INGENIERIA ELECTRICA

No. 16 Serie: Amarilla Investigación Mayo 10, 1985

THE FILE DESCRIPTOR: USE OF A DESCRIPTIVE TOOL TO RETRIEVE GENERAL QUERIES TO FILES. *

ADOLFO GUZMAN ARENAS

ABSTRACT

The file descriptor is a data structure that describes or documents the characteristics (position, type, initial value, etc.) of the information within a file. Each file containing information possesses a unique file descriptor. The file descriptors for a collection of related files are stored together in a file named descriptors.

This paper explains the use of the file descriptor for:

a) Answering general queries to a file "Retrieve all records satisfying predicate P". This is embodied in a tool called the consultator.
b) A general tool for data input -the capturer- which allows inputting appropriate data for any file.
c) A tool for report production -the report printer- which allows design of arbitrary output reports.
d) A specification for transformations from several input files into one or more output files -the file transformer- and its corresponding tool (program to interpret such notation).

The above tools allow for input, consultation or retrieval, transformation ("computation" or "processing" of data into results) among files, and report generation, all of these without wirting programs in high level languages (Pascal, Cobol, ...). Instead, we use a simpler description of (a) the screen to be presented for inputting data; (b) the query or predicate; (c) the transformation; (d) the report.

The description used in (a) through (d) can be considered as a concise language for specifying data-manipulation (e.g., administrative or managerial) tasks. If the tools prove to be useful, one can expect considerable savings in the time and effort spent in specifying and programming such data-manipulation tasks. The system is being implemented in Pascal for an IBM-PC network.

---

* Presented at the Third International ACM Conference on Systems Documentation, SYSDOC 84, Mexico City. May 1984.

INTRODUCTION

Programming costs being on the rise, it is important to find ways to increase programming productivity. One manner is to use languages or notations more appropriate for the task at hand, with more expressive power, less prone to errors from programmers, and where "understood" or "natural" actions need not be specified or programmed. Indeed, it can be possible for the final user to use such languages, thus obviating the need for programmers.

In this paper some of these notations are discussed, along with the tools for using them. All of them employ the "file descriptor", a collection of data about a particular file.

The main application of these tools seems to be for data manipulation programs, which move data around and do little calculation.

DATA MANIPULATION TASKS

One large use of a computer is to "compute" or "calculate" i.e., to convert data into results; for instance, to find the root of an equation.

Another use is to "order" data ("ordinateur" in French, "ordenador" in Spanish), meaning to present data into more usable forms. For instance, from a file of clients, to produce a report of only those clients owing more than $20,000. A third use is for "control", i.e., deciding what to do, taking into account current input, present state, available (stored) information, etc. For instance, to decide to cut into a half

the amount of fuel supplied to a burner, given the current temperature of the furnace and the amount of material remaining to be heated. "Decision making" in another name for it. A fourth and fifth uses of a computer are possible; the three uses cited here need not be all there is. These uses often occur all in a single program, most likely mixed and intertwinned.

Along the suggestion in the Introduction, man has designed:

* Languages for computation or calculation: Fortran (Formula Translation), and most "scientific notations" of other languages: Algol, Pascal, APL.

* Languages for data manipulation: Cobol, RPG, CONVERT [1].

* Languages for control of actions: REC [2], Forth [3], AFL [4].

Most high level languages (i.e. Pascal) can do all of above tasks, but it looks fruitful to further develop languages or notations that specialize in only one of them. In this manner, additional expressive power may be gained, and simplifying assumptions can be taken, hopefully to increase programmer productivity.

In this paper we concentrate in notations that simplify the tasks for data manipulation. We now will look more closely at them.

THE FLUXOGRAM

When we look at the flow of information through a company or department, disregarding its main activities (for instance, curing people is the main activity of a hospital, not information flow), and we draw a diagram of such flow, we produce a *fluxogram* of such department ("flujograma", in Spanish). Generally, a fluxogram contains actors [who does what], actions [about the information], forms, files and reports. Most organizations already have a well-defined data flow, and almost all important information travels in written form.

Fluxograms are appropriate chiefly for representing data manipulation activities. We want to use fluxograms for two different purposes:

a) A fluxogram is an accurate description of the transformation that the information suffers, within a department or an organization. As such, it can be used to answer inquiries about administrative procedures: "How can I obtain a driver's license?"

b) Given a fluxogram; it may be desired to automate the procedures that handle the information of a part of it., that is, given some information handling activities in an organization, we want to automate part of them. For this purpose, the fluxogram is a description of "what happens" to the information inside that part, and in some sense it may be possible to produce the desired computer programs directly (i.e., automatically; by machine, not by hand) from the fluxogram.

We address both aspects (a) and (b) in this paper, but we focus first in a simpler concept which is nevertheless necessary for both tasks: that of how to describe the way the information is stored inside a file. This is called the file descriptor for that file.

THE FILE DESCRIPTOR

We consider files where all records are of the same length, and where each one contains information organised in the same manner. This information is about a particular item, and a file is a collection of (information about) such items; thus, the files CLIENTS (Figure 1) contains 215 records, which is to say, 215 clients (a client is an item of file CLIENTS).

Each client contains a client-name (bytes 1-30), an address (bytes 31-99), etc. Also, file PROJECTS contains a collection of 68 records; each record is a project, and contains 8 fields: project number (bytes 1-5), leader of the project (bytes 6-30),..., project description (bytes 66-180).

The above descriptions are contained in the file DESCRIPTOR. Each of its records begins with the name of the file it describes, followed by some information about the whole file : size of file in records, size of a record in bytes, etc. Then, an area of field descriptors follows. Each field descriptor contains the name of the field it describes, the initial position in the item, its size in bytes, its type, and range value useful for validation. All of this should be clear from figure 1.

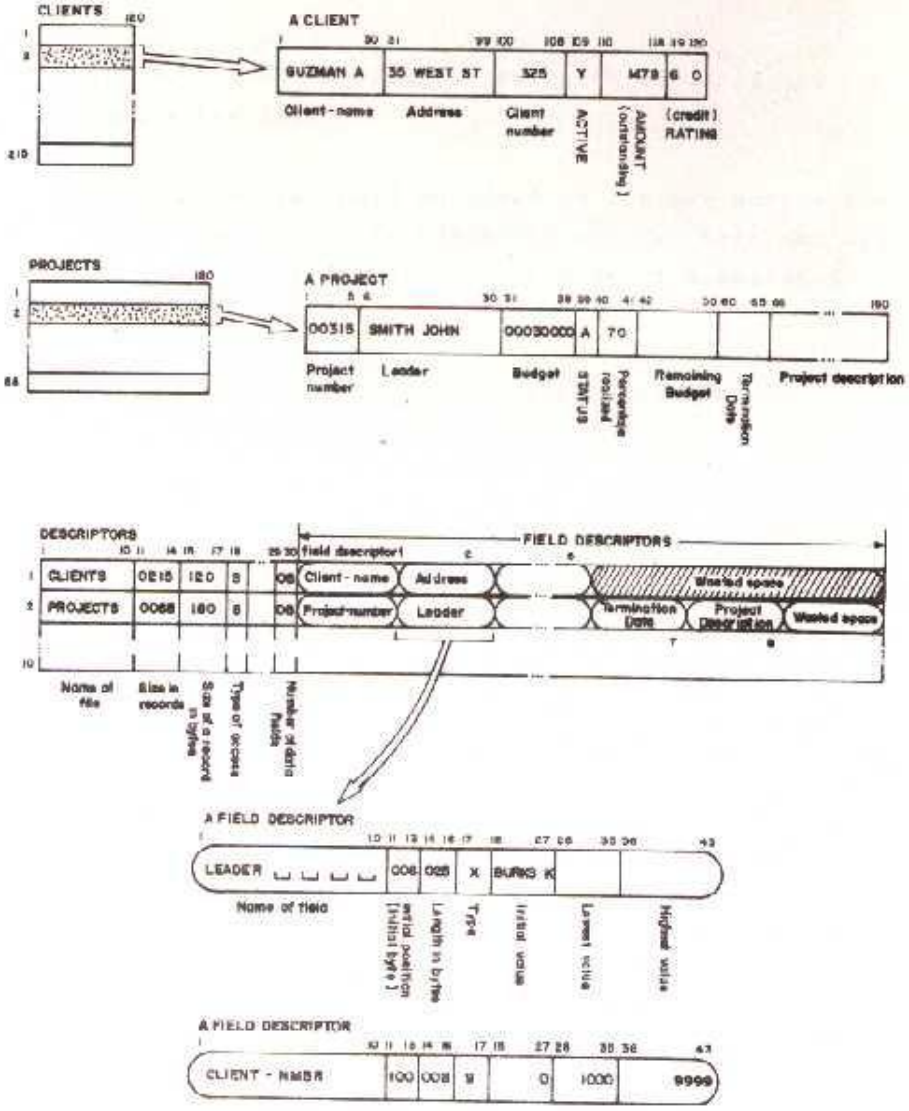

FIGURE 1
THE FILE DESCRIPTOR

There is a field descriptor for each field in the record storing the item.

Other parameters could be added to further enrich the information stored in the file descriptor. We have not done so in figure 1, to keep it simple. A more sophisticated field descriptor can contain :

* Description of data (in the item) which are not fixed in position ("explicit" properties).

* Number of occurences of each field, for multivated fields.

* Conditions upon which a particular record can be deleted. For instance, certain accounts cannot be deleted unless AMOUNT OUTSTANDING = 0.

THE FUNCTION VAL

The main use of the file descriptors is to access the information by name (using the name of the field) and not by position (byte position in the record). Thus, SMITH JOHN (see figure 1) is accessed by VAL ('LEADER', 'PROJECTS', 2), meaning "the value of the field LEADER in the second project (register # 2 of file PROJECTS).

Often, the name of the file is specified only at the beginning; thus the consultation CONSULT CLIENTS specifies the file CLIENTS for all further queries. Also it is common to access the records in a pre-determined order, thus the record number, too, its implicit. Hence, it is sufficient to say VAL ('LEADER'), VAL ('ADDRESS'), and so on. Even

when we are using several files simultaneously, VAL ('ADDRESS') is unambiguous as long as no two file descriptors have the field ADDRESS. That is, file name and record number are mostly implicit (omitted) parameters of VAL.

How VAL works. When a file (CLIENTS, let us say) is specified -- for instance, at the start of a session with the Consultator--, a function LOAD-DESCRIPTOR (CLIENTS) tries to bring the descriptor of such file into memory. Upon success, the function tries to see if the file exists, and if it can be opened. Once open, the file is available for further processing : to answer general questions from the Consultator, to receive new records from the Capturer, etc. The records of the file are accessed (read into memory or written from memory) in some specific way (sequentially; only one record, by key; start from a given key and go upwards; idem but go downwards; go from a lower to an upper limit; etc.) which needs not concern us now.

VAL has the important duty to access the information stored at particular byte positions, given the name of the field. For instance, VAL ('ACTIVE') will access byte 42, producing 'Y' for record 2. VAL ('CLIENT-NMBR') will produce 325 for same record, accessing bytes 100-108. The translation of 'CLIENT-NMBR' into "bytes 100-108" is done via the descriptor : VAL searches the "name of field" of each field descriptor of CLIENTS, until CLIENT-NMBR is found. There, the field descriptor will inform VAL that the information is found beginning in byte 100, and has a length of 9 bytes (Figure 1). It also gives "9" as the type of CLIENT-NMBR, and further says that the field has to have a value between 1000 and 9999. VAL uses the information to access bytes 100 through 108, and perhaps to perform type conversion.

VAL therefore has the ability to give the value of any field of any record for any file whose description is in DESCRIPTORS.

This useful program accesses any file, selects among its items those having a given property and prints some parts of them.

Suppose we want to know what clients are active and owe us more than $15,000, we want a list of them specifically, their name, address, and amount standing. This list should be printed alphabetically by client lastname. The inquiry is as follows :

```
CONSULT CLIENTS

* (clients) *                          Ok-file clients exists. The system
What question?                         asks for the predicate which
                                       specifies the particular conditions
                                       a client must satisfy.

ACTIVE = Y & AMOUNT    15000

What do you want to know?
CLIENT-NAME, ADDRESS, AMOUNT           print three properties of clients
                                       selected.

any ordering?                          Use value of NAME to sort the output
                                       list.
CLIENT-NAME

    NAME                ADDRESS                AMOUNT
Alvarez Juan      5 de Mayo 18. Mexico City    16520
     .                    .                      .
     .                    .                      .  this is the answer
     .                    .                      .
*(clients)*
What question?                         ready to accept new question
```

FIGURE 2.

The predicate is formed by primitive questions joined by + (or) and & (and).

We do not use parenthesis, instead, + has more precedence over &. We do not use "not"; instead, we reverse the relation in the primitive question. Thus, instead of not RATING>60 we write RATING>=60.

Each primitive question is a binary relation between two fields or constants : RATING>60;
6000<AMOUNT; CLIENT-NMBR>RATING.

For fields holding text, we use @ (contains), [(begins)], (ends with) and = (is equal) : ADDRESS @ MICH "address contains MICH" (Michigan).

The list of properties to be printed is a collection of field names, separated by commas.

If an order is specified, records satisfying the predicate are sent, in the order they appear in the file, to the printer. If an order is specified, let us say RATING, AMOUNT then the output is sent to a temporary disk file, to be sorted and printed. They will appear on paper first by ascending credit rating, and, for the same rating, by ascending amount outstanding. Subtotals will appear at appropriate levels, properly indented.

HOW THE CONSULTATOR WORKS

The query is analyzed by a small syntactic analyzer, and a table is formed in memory, which contains the same information as the query, but in condensed form. Also the list of properties to be printed is converted into other table.

The file to be consulted is opened, and its records are brought sequentially to memory.

As each record is present, the query or predicate is applied to it; if true, the list of properties of that record is printed. If false, nothing is printed. In any event, the next record is brought into memory, and the process iterates until the end of file is met.

IMPROVEMENTS TO THE CONSULTATOR

There are several useful improvements:

*Produce more beautiful reports, using the report writer for this.

*Produce large wall-paper reports.

*Send output to disk, for further processing or for obtaining several copies of a report.

*Catalog some popular reports; i.e., ask the consultator to obtain its input from a (catalogued) file, not from the console: in this manner periodical reports can be obtained by a command such as :

RUN REPORT 14

instead of typing lots of input to the Consultator each time the report has to be produced.

THE CAPTURER

This program fills any record of any file, obtaining

the raw data from the keyboard, and prompting the user the appropriate names of the field descriptors. It is used to enter new data into a file, to modify existing records, or to display them. It also validates the input.

Simple capture. A simple way to capture and validate data is to say for instance CAPTURE CLIENTS, after which the different fields are displayed for input, one below the other:

```
CLIENT NAME:
    ADDRESS:
CLIENT-NMBR:
      (etc)
```

This input is validated on-line, checked against the limits specified for each field in the corresponding field descriptor. Needless to say, the name of each field, its length, etc, are obtained from the file descriptor of CLIENTS.

Capture with screen. A more elegant way to capture data is to produce a screen which resembles the document containing the data. For this, a file containing the screen format is needed. This file contains "commands" to the screen, specifying what gets displayed where. Each command is of the form mm, nn $TEXT$ or mm, nn VARIABLE which mean "display TEXT in line mm, column nn" or "capture the variable which will be input at line mm, column nn". The capturer checks to see whether the screen format exists, defaulting to "simple capture" if it does not.

Variations for the Capturer. Some useful variations to the capturer are : CAPTURE CLIENTS
Append data after the end of the file.
MODIFY STUDENTS 300

Modify our update some records of file STUDENTS, starting at record 300. "Old" data is displayed and open for modification (by rewriting upon it); new data is validated.

DISPLAY INVOICES 223-250

Display (no changes allowed) invoices 223 through 250.

Improvements. Sometimes we do not access records (CLIENTS, say) by record number, but with a key (the client-name, for instance). This data is part of the input data. To handle this case, the file descriptor has to contain information about what fields are keys; the capturer has to know, too, whether the presence of a record with the same key is forbidden (for instance, in entering new clients), required (as in updating a client) or allowed.

How the Capturer works. Each command of the screen format is essentially outputted as it is, in the specified line and column. If a command contains a variable, the variable is read, validated and stored in a buffer. When all commands are processed, the buffer is written onto the file, and the screen format is interpreted again, in order to capture information for the next record.

All displaying takes place first, then all variables are captured.

THE REPORT WRITER

The output of the consultator, already in a given (sort) order, can be rendered more readable with the use of the Report writer, which can establish headings, variables which go into columns, subtotals, totals ("breaks"), etc. The report writer can also be used by itself, in this case the

complete file is swept.

To specify a report, a file containing the report format is needed. Each "command" in the report format can be thought of as a description of a particular line of the final output. Some commands are:

PA: nlin, ncol
Set the page size
AR: file 1, file 2, ...

Declare file 1 as the main file, and file 2, ... as secondary files, to be used for producing the report.

Example : AR: CITIES. Process the file CITIES.
EN : col, "TEXT"; col2, "TEXT2"; col3, VARIABLE;...

Heading (to be repeated at each page). Print texts and variable names at specified columns.

CT: level1, VARIABLE1; level2, VARIABLE2;...

Establish breaks (to print subtotals) at level level1 when variable1 changes value; at level level2 when variable2 changes, etc. The higher the level number, the innermost the break. Example: CT: 1, COUNTRY; 2, STATE

Break printing subtotals for each country, and within a country, for each state. What to print at level i is specified by the command Ti.

Ti: col, variable, col, variable; col, "TEXT". Print variables and texts (at specified columns) when a break at level i occurs (as defined by a previous CT command).

"Variable" is a name for totalizers at different levels; the appropriate totalizer is chosen according to i in Ti.

Suppose we say :

```
T1: 10, INHABITANTS
T2: 10, INHABITANTS
```

Then T1 says that the number of inhabitants at level 1 (which was established above at the country level) will be printed when a break at level 1 (ie., a change in country) occurs, and T2 says that the number of inhabitants at level 2 (state level) will be printed each time a break of level 2 (change in state) occurs. Reset to zero takes place after printing.

The interpreter accumulates the INHABITANTS (presumably a field of the file) into the internal variables 1INHABITANTS, 2INHABITANTS, 3INHABITANTS, etc. and prints the variable iINHABITANT when a break at level i takes place. In this case, as each city is processed, accumulation takes place in all iINHABITANTS accumulators. When a change of state (level 2) occurs, 2INHABITANTS (accumulation of the inhabitants of all cities belonging to this state) gets printed and reset.

A redundant way to say the same is :

```
T1: 10, 1INHABITANTS
T2: 10, 2INHABITANTS
```

A clearer way to specify the printing of subtotals is :

```
T1: 2, "TOTAL OF PEOPLE IN THIS COUNTRY:";
    40, INHABITANTS

T2: 2, "TOTAL OF PEOPLE IN THIS STATE:";
    40, INHABITANTS

DE: col, "TEXT"; col, VARIABLE; col, VARIABLE,
```

Detail line. Prints texts and values of variables at specified columns, each time a record is read from the primary file.

It is also possible to print variables which are internal to the report writer (i.e., they are not fields of the file). These begin with #, for instance, #TOTAL. How to compute these variables is specified with the help of the commands "cn" (compute expression, after a break of level n) and "IF".

Reports can be wider than the width of the printer line; in this case, the reports are printed piecewise (to be glued later by hand).

THE FILE TRANSFORMER

We have covered so far tools to simplify input (the capturer), output (the report writer) and inquiries (the consultator) to an arbitrary collection of files. It remains to take care of processes that convert or transform one (or more) input files into one (or more) output files.

We will not go into the detail of how the transformation is carried on; the general idea is that, in the notation of the *File Transformer*, when a field with the same name occurs in an output file and in an input file, it gets copied

from input to output file. Otherwise, we have to specify what fields get their value from what other fields; the notation of the File Transformer also has CN (computations) and IF (conditionals), as well as useful file handling "primitives", such as append, replace, sort, join or merge, delete or destroy, etc.

The File Transformer uses several input files, and several output files.

OTHER TOOLS

The tools so far described are by no means all we need to simplify processing of administrative data; we can add:

* A virtual file handler, that joins lengthwise two records belonging to two different files; for instance, suppose we have in one file information about an employee, its age, salary, etc., and in other file, information about its past jobs; using a primary key, we can think that we have a single long file with no redundant information (this is the same as the *join* function of data base systems).

* A virtual record handler, that handles unexistant records. For instance, suppose we define a virtual record which is the sum (accumulation) of another twenty records (say, a Department divided into twenty offices); then every change to data in the subordinate records is "automatically" reflected in the virtual record. What really happens is that the virtual record does not get updated each time a change occurs in one of its twenty sons; the changes get

postponed until the virtual record gets consulted (read). This being the case, the system "quickly" recomputes the values of the virtual record, and produces them to the function requesting the data, just as if they were read from disk. A virtual record has zero updating time, but a long read time. Virtual records cannot be written.

* A virtual property handler. Some properties can be declared "virtual", i.e., they can be computed from "real" or virtual properties. Suppose we have as properties of an employee its salary and the number of children; from this, we can compute the "virtual" property "salary per capita", using a formula; in this case, we do not need to store this data into the record of the employee. Virtual properties have instant update time, and they do not occupy space in disk. Nevertheless, they consume computer time (the time to evaluate the formula) when they are "read" from the record.

USE OF FLUXOGRAMS FOR AUTOMATIC PROGRAMMING.

If we see carefully the above tools, we can say that, through a written notation, simpler than Pascal, it is possible to convey the appropriate functions of inputting, transforming, consulting, and outputting information from a set of files; operations all of them germane to "administrative data processing".

But we can go beyond this. We can draw in the screen of a terminal the transformations, the data acquisition documents, etc. These drawings should be the easiest of them to use, easier, say, than having to learn the intermediate

notations we have just finished describing. And then, a giant (or not so giant) compiler could transform the drawings into the intermediate notations. With the final result that the final (application) programs could be working, if we just supply a drawing of what the system does, or how does it do it. Sergio Chapa, one of our Ph. D. students, is beginning to explore this path to "automatic programming" of administrative processes, given a sketch of what the system does, what are the documents handled, etc.

Given a fluxogram, we need not automate "all of it". Rather more usually, only some parts are suitable (or worthy) of automation. Thus, given a fluxogram, the user "draws a dotted line" around a subpart of it, and only the boxes inside it get compiled. If the dotted line cuts along some line into the subpart, then a printed form has to be captured (by the capturer); if the dotted line cuts some line coming out from the compiled subpart, then a printed report has to be generated (by the report writer); these are the interfaces between the written world (of typed documents) and the machine-readable world.

USE OF FLUXOGRAMS TO ANSWER INQUIRIES ABOUT ADMINISTRATIVE PROCEDURES.

The fluxograms, if well used, describe what information handling goes into an office or organization; therefore, the information needed to answer inquiries about such handling is there. Queries of the type "What can I do to obtain...?" can be answered by "digesting" or inspecting the fluxogram, and finding one (or more than one) trajectories from the initial state (where the inquirer is) to the final state (where the inquirer wants to go). "How can I obtain a driver's license, given that I do not know how to drive?"

"How can I become a Pope, if I am already a Catholic Priest?" or "How can I obtain a building permit?". In the answers, one can make use of "subroutines" which supposedly the inquirer knows (or he can further ask). For instance, one answer can begin by saying:

1. Become a Catholic Christian.
2. Become a Catholic Priest.
3. . . . .

Each of these lines "invokes" a complex subroutine, which the inquirer can expand by asking further explanation.

CONCLUSIONS AND RECOMMENDATIONS FOR FURTHER WORK.

Conclusions. The fluxograms seem to be a useful notation for describing certain kinds of data processing that mainly involve transfer and reshuffling of information. Together with the fluxograms, this paper describes four tools useful for this kind of "administrative processing". These tools all are based in the file descriptor, a piece of information that describes how the information within a file is stored, and what does it mean.

With the help of the tools, we can write application programs in an intermediate notation (that the tools interpret), instead of using Pascal or other high level general purpose programming language. As they are, then, these tools already simplify the programming of this kind of application systems.

The fluxograms can be more fully used by (a) automatic compiling of the administrative programs, starting from the fluxograms -- at present, an intermediate notation has to be manually prepared --; (b) use of fluxograms to answer inquiries about administrative procedures.

Recommendations for further work. (a) Finish the tools and expand the notation that drives them; (b) produce additional tools, as outlined in Section "Other Tools"; (c) design a notation to describe fluxograms, that is amenable to compiling; (d) design the compiler for such notation; (e) write an interpreter for analysis of a fluxogram, in such a way that it can answer inquiries about administrative procedures.

REFERENCES

1. Guzmán, A., and McIntosh, H. CONVERT. Comm. ACM, Ang. 66.

2. McIntosh, H.V., A. Convert Compiler of RBC for the PDP-8 Acta Mexicana de Ciencia y Tecnología Vol. II, No. 1, Enero-Abril, 1968, pp. 33-43.

3. James, J. S., What is Forth? A tutorial introduction Byte Vol.5, No. 8 (August 1980)

4. The AFL Programming Manual.

BIBLIOGRAPHY

5. Javier, Margarito. Systems to make general queries to several files with arbitrary formats. B. Sc. Thesis, ESIME-IPN (Electronics and Communications Engineer), México City 1984. (In Spanish).

6. Chapa, Sergio. A schema for queries and data input based on the file descriptor. Technical Report AM 16, Electrical Engineering Dept., CINVESTAV-IPN. Mexico City. 1985 (In Spanish).