PUC

Série: Monografias em Ciência da Computação

Nº 5/77

(antiga/formerly: Monographs in Computer Science and Computer Applications)

TOWARDS THE DESIGN OF DATA BASE INTERFACES FOR NON-PROGRAMMERS

by

A. L. Furtado

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rua Marquês de São Vicente 225 — ZC 19

Rio de Janeiro — Brasil

Série: Monografias em Ciência da Computação

Nº 05/77

(Antiga/Formerly Monographs in Computer Science and Computer Applications)

TOWARDS THE DESIGN OF DATA BASE INTERFACES FOR NON-PROGRAMMERS*

by

A. L. Furtado

Series Editor: Michael F. Challis

April, 1977

*This research is part of the HYADES project, and is partially sponsored by the braziliam government agency FINEP.

For copies contact:

Rosane T. L. Castilho
Head, Setor de Documentação e Informação
Depto. de Informática - PUC/RJ
Rua Marques de São Vicente, 209 - Gávea
20.000 - Rio de Janeiro - RJ - BRASIL

RESUMO:

É apresentada uma proposta para o atendimento das necessidades de não programadores, na área de sistemas de gerência de bancos de dados. Tenta-se conciliar a liberdade da linguagem natural com as limitações impostas por considerações de eficiência, dado o presente estado da arte.

PALAVRAS CHAVE :

Bancos de dados, não-programadores, compiladores, modelo relacional, algebras relacionais.

ABSTRACT:

A proposal for meeting the requirements of non-programmers, in the area of data base management systems, is presented. It attempts a compromise between the freedom of natural language and the limitations imposed by efficiency considerations, given the present state of the art.

KEY WORDS:

Data bases, non-programmers, compilers, relational model , relational algebras.

CONTENTS

1. INTRODUCTION

Attempts are being currently made to facilitate the access of non-programmers to data base management systems (DBMS).

The motivation behind such efforts is to avoid (or to minimize) the need to train such users. In order to access information in a data base a user must normally be aware of some artificial language, and of the logical organization of part of the data base. For slightly more complex queries he must also possess a measure of programming skill.

In the ideal situation non-programmers would use natural language for interrogating the DBMS. To make this possible, the data base would store information about its own organization, and would be capable of logical inference [1,2].

It is unfortunate that such systems are still unavailable for practical usage. Given the present state of the art, they cannot be efficient enough and their added size is considerable. On top of that, the inherent ambiguity of natural languages can defeat even the largest system.

Clarification dialogues have been devised for reducing the necessity for the machine to be "intelligent" and to unravel ambiguities [3]. In such an environment, from an imprecise request made by the user, the machine and the user working together proceed to a formulation which is both correct and phrased in the official terminology of the specific data base. However, this dialogue may be time-consuming and there is no guarantee that the (untrained) user will recognize as correct some machine-produced reformulation of his request.

We are much in favor of the natural language efforts, possibly enhanced by clarification dialogues. Our point is that, while research is in progress, some less ambitious solution should be devised.

As in other man-machine environments, the question is to determine how much "intelligence" should be expected from each component. Since neither the machine nor the non-programming user can cope entirely with the problem, an additional human component is suggested - the Non-programmer Interfaces Administrator (NIA)*, whose job is to generate, through a software tool called the Interface Generator (IG), one or more Specific Interfaces(SI) for each user or class of users.

Figure 1 depicts the four strategies indicated here, showing where (between man and machine) befalls the main burden.

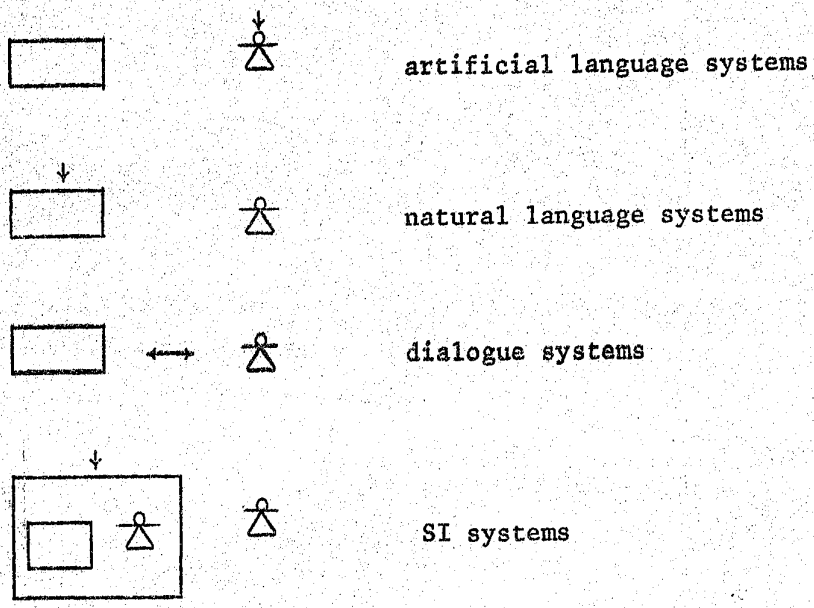

Fig.1: the DBMS machine and the user

* A variety of the Application Administrator proposed in [4].

As shown, the user of an SI sees a customized "virtual" machine, pre-constructed for him by the NIA.

Taking the several classes of end users in [4], one would say that our proposal is especially applicable to the inquiry specifier. If the NIA's help is easily secured, end users of unstructured processes can also use it. Finally, an IG-generated interface can be an inexpensive first iteration for meeting the requirements of a parametric user, on an experimental basis; after testing possibly several SIs, a final version might be approved which would then be rewritten by an application programmer for maximum efficiency.

2. DESIGN GOALS

In order to accomodate different kinds (and even different personalities) of users, an SI will allow ample freedom of language, with the few restrictions to be introduced shortly.

An SI consists of a set of query types $q_1, q_2, \ldots, q_m$. Each q_i has a fundamental formulation, which is simply a natural language sentence meaningful to the user; the paraphrases of q_i are all other natural language formulations that will be accepted by the system as equivalent to (i.e. expressing the same meaning as) q_i*. Clearly the system should also be able to distinguish the fundamental formulation and all paraphrases of q_i from those of another query type q_j.

A query type is to be understood as a generic pattern. An actual query submitted at a given moment to the system by the user is an instance of some type q_i. Perhaps the main difference between the aspect of a query type and its instances is the possible occurrence of constant places in the query type, to be replaced by constants in the instances.

We are now in a position to enumerate the restrictions embodied in the SI system:

* The possibility that two query types be distinct to the system and yet express in fact the same meaning is not excluded.

- an instance must be terminated by some punctuation symbol;

- the fundamental formulation and all paraphrases of a query type q_i must be characterized by certain words or phrases, denominated key terms; the set of key terms of q_i and those of any other q_j cannot be identical;

- constants must appear between quotes;

- if more than one constant appears in a sentence, each appearance must be distinguished by some preceding or succeeding term, denominated constant definers.

The first restriction is a rather common one. The three other restrictions have to do with the efficiency of the process of recognizing query instances.

In fact, freedom of language and efficiency are two of the design goals adopted here, and the trade-off attainable in view of the above restrictions is the object of the discussion that follows. The third goal is extensibility of the set of query types, which is provided by the TG translator and which is the topic of the next section.

The recognition process is efficient because the detection of key terms, quotes, and constant definers requires only a very rudimentary pattern-matching capability (essentially what is provided by the PL/I INDEX).

A considerable degree of freedom is left. Apart from the key terms and quoted constants (possibly coupled with constant definers), which can appear in any order, all the remaining text is ignored by the system. The possibility of using paraphrases arises from the irrelevance to the system of such portion of the text: a paraphrase may involve a different choice of words, a different syntax, and even misspellings, syntactical and other kinds of errors in that portion. The user may choose to be concise or verbose on different occasions.

Computations that are commonly associated with queries can also be indicated, such as average, maximum, minimum, total, count, etc. In keeping with the free-language goal their appearance need not resemble a procedure call.

A few examples of query types that a user might require are given below, key terms and constant definers being underlined:

find the teachers who make more than ☐ .
give the average salary of teachers in department ☐ who lecture in the ☐ course.
retrieve all legal course-number/teacher-name pairs.

The third query (given in [5]) is a good example of the vagueness involved in the query - formulation process. The user would presumably explain to the NIA that 'a pair is "legal" if a teacher with the indicated name teaches at least one offering of the indicated course'.

3. THE GENERATION AND UTILIZATION OF THE INTERFACES

Phase a. The NIA finds from the user what types of queries the latter considers useful for his work. As phrased by the user, these constitute the fundamental formulations.

Presumably the user already knows what information is kept in the data base, and in some cases he may even have participated in the design of the information system leading to the creation of the data base.

In any case, the NIA will give all necessary explanations and will assist the user in defining the types of queries, with special care in the identification of key terms and the use of constants.

The NIA will not try to "teach" the user what is the correct terminology for the particular data base. He will simply ask the user to explain what he means by each type of query, to be phrased in the user's

own style; however, he will mention to the user the need to include the convened key terms, etc. in his instance queries.

Phase b. The NIA (alone) constructs, for each type of query of the given user a corresponding precise expression in some very high lan - guage notation. For relational data bases we are strongly biased in favor of some relational algebra (particularly our algebra of quotient relations [6]), but other precise formalisms are also eligible. The chosen formalism must be extended to allow the invocation of computational procedures (such as average, etc.).*

Thus this phase results in a set of transformation rules of the form:

$q_1 \rightarrow a_1$
$q_2 \rightarrow a_2$
.....
$q_m \rightarrow a_m$

where the q_i are the original query types and the a_i are the corresponding algebraic expressions.

Phase c. The transformation rules are processed by the IG. The input notation should single out the compulsory parts of the text.

From the set of transformation rules the IG produces a program - an SI - logically divided into two modules:

- the recognition module (REC) - this module is obtained by the IG from the left hand side of the rules; in pseudo-language notation its structure could be described as follows:

---

* As we shall see, the mention of such procedures is ignored in the recognition of query instances, but provision is made for them when generating the execution module of the SI.

```
do for each query instance x;
   α:do for i=1 to m;
      if key_terms_i are in x
      then do;
           extract constants;
           call EXEC_i (constants);
           exit α;
           end;
   end α;
end;
```

- the execution module (EXEC) - this module is obtained by the IG from the right hand side of the rules; it consists of m procedures, whose parameters correspond to the extracted constants, and which contain the appropriate declarative statements input/output statements and calls to procedures implementing each algebra operation, and to the required computational procedures;in particular,the declarative statements must provide space for temporary relations which will store the results of the algebra operations [6].

Phase d. The user activates his SI to process his (instance) queries. At this point the SI is an object code program (compiled by the IG), available only to the respective user (or group of users) by checking his (their) job code and/or some password. The instance query may follow either the respective fundamental formulation or some paraphrase thereof.*

Phase e. Whenever the need for an unanticipated query type arises the user asks again for the assistance of the NIA. Storing also the transformation rules makes it easier to add new query types and recompile the SI (reiteration of phase c), making sure that the new query type distinguishes itself from the other ones by appropriate key terms. To this extent,

---

* Recall that a paraphrase is any formulation containing the same key terms, required constants, and constant definers as the fundamental one.

an SI is extensible, as postulated before.

Deleting certain rules is also conceivable, in case for example, the user is no longer authorized to access certain kinds of information (notice that restricting these users to their SIs helps enforcing security constraints).

Ideally, this IG/SIs capability should be run in a multi-terminal environment, involving several users and one (or more) NIA(s). Thus, unanticipated situations would be routed to the NIA and he would supply new transformation rules at once; one would feel however that such prompt assistance is not feasible in general, due to the difficulty in, firstly, understanding the user's intention, and then finding its correct translation into an algebra expression.

4. CONCLUSIONS AND FUTURE WORK

Certain additions to the basic design may be helpful, although only very simple ones would be tolerated, given the basic goals introduced before. In the present design the result of one query cannot be directly used in subsequent ones; another version could allow further manipulation of the result of the immediately preceding query. For example, if a resulting (temporary) relation has employees and salaries the user might then ask for the count of employees, the total salary, etc.

A debugging feature for the NIA would be useful. After formulating and compiling a set of transformation rules he could call for a sample of the data involved, try some query instances on this reduced space, and perhaps discuss the output with the user.

Since the user may be employing paraphrases of his original query type he may, by error and unknowingly, obtain unexpected results. Note that words other than key terms, constants and constant definers are regarded as noise by the REC module of an SI; thus the user may have tried to formulate an entirely different query! To alleviate this problem it is expedient

to display the original query type formulation together with the result , as for example in:

the teachers who make more than 10K are:
.....
.....

so that the user will know what he is really obtaining.

Experimenting with several kinds and styles of user requirements will also provide an opportunity for investigating the human factors involved in the use of data bases by non-programmers [7,8].

REFERENCES

1. Thompson, F. and Bozena, H. - "Practical natural language processing: the REL prototype" - in "Advances in Computers", 13 (1975) 109-168.

2. Roussopoulos, N. and Mylopoulos, J. - "Using semantic networks for data base management" - Proc. Very Large Data Base Conference, Framingham (1975) 144-172.

3. Codd, E.F. - "Seven steps to rendezvous with the casual user" - Proc. IFIP, TC-2 Working Conference on Data Base Management Systems (1974).

4. ANSI/X3/SPARC/Study Group-Data Base Systems, "Interim Report", ACM/SIGMOD - Newsletter: fdt, 7,2 (1975).

5. Date, C.J. - "An architecture for high-level language database extensions" - Proc. ACM/SIGMOD International Conference on the Management of Data (1976).

6. Furtado, A.L. and Kerschberg, L. - "An algebra of quotient relations" - T.R. 10/76 - PUC/RJ (1976).

7. Schneiderman, B. - "Experimental testing in programming languages, stylistic considerations and design techniques" - Proc. National Computer Conference (1975) 653-656.

8. Gould, J.D. and Asher, R.N. - "Querying by non-programmers" - Proc. American Psychological Association Convention, New Orleans (1974).