PUC

Série: Monografias em Ciência da Computação
Nº 4/77
(antiga/formerly: Monographs in Computer Science and Computer Applications)

AN INCREMENTAL APPROACH TO THE CONSTRUCTION OF DATA BASE SOFTWARE

by

A.L. Furtado
C.J. Lucena

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 4/77

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

AN INCREMENTAL APPROACH TO THE CONSTRUCTION OF DATA BASE SOFTWARE*

by

A.L. Furtado

C.J. Lucena

Editor: Michael F. Challis

April, 1977

* Work partially supported by the brazilian government agency FINEP

For copies contact:

Rosane T.L. Castilho
Head, Setor de Documentação e Informação
Deptº de Informática - PUC/RJ
Rua Marquês de São Vicente, 209 - Gávea
20000 - Rio de Janeiro - RJ - BRASIL

RESUMO:

Uma metodologia de projeto de linguagens é proposta para aplicação no desenvolvimento de sistemas de programação de bancos de dados. A metodologia envolve uma abordagem em três níveis e utiliza o conceito de "clusters".

PALAVRAS CHAVE:

Bancos de dados, modelos de dados, projeto de linguagens, tipos de dados.

ABSTRACT:

A language design methodology is proposed for application in the development of data base software. The methodology involves a three-level approach and makes use of the cluster concept.

KEY WORDS:

Data bases, data models, language design, data types.

CONTENTS

1. INTRODUCTION

The need for a programming discipline, which is widely recognized today in all areas of computer applications, is very critical in the field of data bases. The complexity of the data structures involved has led to questioning if reliable data base software is at all feasible [1].

In this report some preliminary considerations are made on the general problem of elaborating data base software, with the help of a methodology developed for language design.

2. DEVELOPING DATA BASE SOFTWARE

General purpose data base software systems are widely used. They commonly adhere to some data model, particularly to the hierarchical (IMS, MRI 2000), network (DBTG, IDS, DMS) or relational (INGRES, system R) models [2].

One such system can be used for creating and manipulating a specific data base in some installation. In this process, the special characteristics of the information (say, natural resources, general accounting, military supplies, etc) and its patterns of usage (e.g. frequency of queries versus frequency of updates) must be taken into consideration. Some systems give the user some flexibility for "tuning" the system according to these special characteristics.

In some cases an installation may decide instead to use special purpose software, either purchased or developed by the installation itself.

Along the opposite direction, recent research aims at general purpose systems. One motivation is to support all data models, as recommended in [3]. Examples of such systems, which will be termed basic systems, are reported in [4, 5].

Basic systems are explicitly proposed as a low level facility on top of which other systems, committed to one or other model, will be created. We can thus imagine a construction process, whereby one would develop data base software moving

through the basic, general purpose and special purpose stages, deriving each stage from the previous one with an increasing degree of specialization.

Conceivably the amount of effort in this gradual process would be less than that which a "start from scratch" policy would require for the development of software for each specific model, or for each specific data base. A gradual approach would also seem to facilitate program verification.

Figure 1 displays the proposed construction process, which clearly proceeds in a bottom-up fashion.

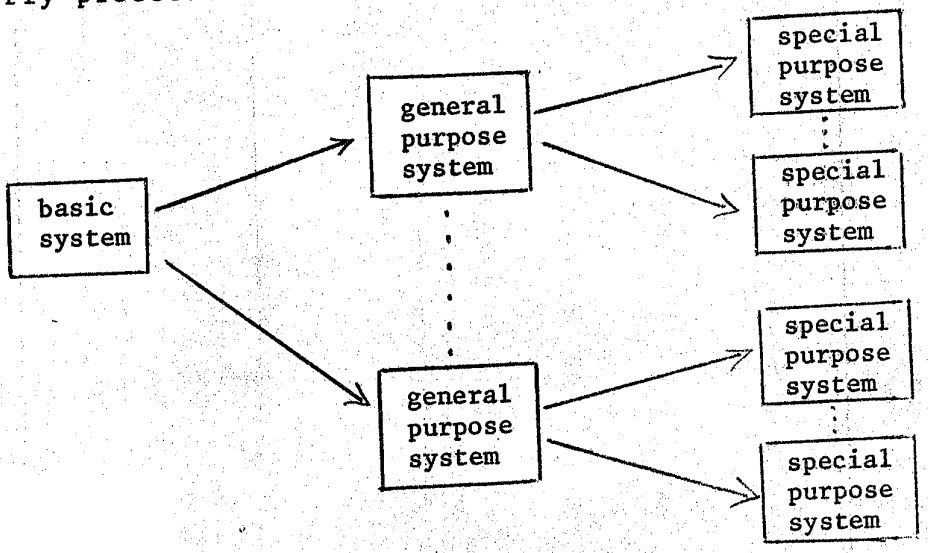

Fig. 1: Construction process

The situation shown in figure 1 becomes acceptable in practical terms if a subsequent process - an optimization process - can be performed, which may involve backtracking in a top-down fashion against the arrows in the figure.

Both construction and optimization are difficult processes to implement. In the sequel we shall investigate a methodology, developed in the area of language design, whose employ seems to be most helpful here.

3. THE THREE-LEVELS APPROACH TO DATA STRUCTURES

Several programming techniques have recently been proposed to reduce the gap between the specification of an algorithm for the solution of some problem and its coding into a machine acceptable form. Programming language constructs are also being investigated to suggest the application of those techniques. A very useful idea, which has been presented under various forms in the literature, involves the use of different linguistic levels to model the different levels of abstraction that occur in the process of program development. The concept of cluster [6] as a way of defining and representing types is a central programming language concept that helps to support this idea.

A three linguistic level approach, based on the concept of cluster was proposed in [7] to allow the construction of programs that are simultaneously easy to verify, efficient and portable.

According to this approach, the topmost linguistic level, designed to support the specification level of programming (or relational in Earley's terminology [8]), consists of a language of very high level of the CLU family. In fact, we expect to try the ideas discussed in the present work on a PL/I extension [9] that only allows the use of abstract data types and primitive (unstructured) PL/I types at the specification level. That gives a highly non procedural [10] flavor to the specification programs, since associative referencing can be simulated and highly aggregate operators defined in connection with the abstract data types can be used. A particular application program can be specified in a top-down fashion through layers of clusters that define the data abstractions appearing at the different levels of abstraction of the application program specification. At the bottommost level of the specification a set of unimplemented terminal clusters associated with the application's set of basic types will be found. We call this set of basic types and associated clusters the base application's machine.

Often, the operations of a base application's machine are well established and can be characterized via precisely stated axioms. The knowledge of the precise semantics of the base

application's machine is instrumental for the synthesis of a correct specification program. The scheme in figure 2 tries to convey the above ideas in a compact form.

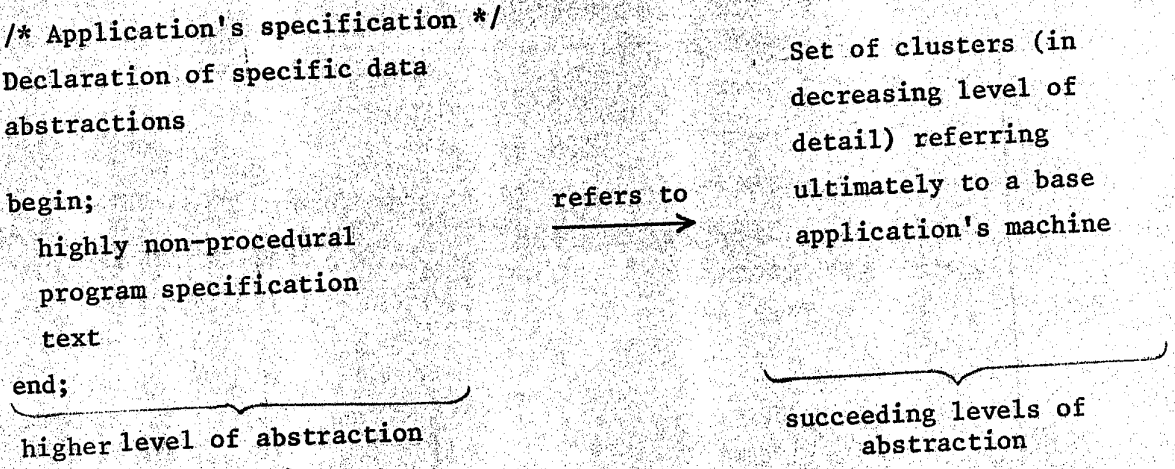

FIG. 2: Different levels of abstraction expressed in the topmost linguistic level: the specification level.

Once the program specification is completed, we are left with the implementation of the operations (clusters) of the base application's machine. We propose that this development should be carried out in yet a different linguistic level [7, 8]. At this linguistic level, called the access path level, the programmer describes the effects of the operations defined for the type through a generic representation that refers to a common base representation (defined in terms of standard operations). At this level, besides the types available for the specification level, the linguistic level supports the type rep (generic representation) and the standard PL/I array and structure (used to model the access paths).

This linguistic level is succeeded by the concrete representation level, where the language used within the clusters makes use of the "lowest level" features of the host language (e.g. pointers) to implement as efficiently as possible the common base representation.

This approach to the implementation of the base

application's machine has a number of advantages:

a. A data type is often described in terms of an access path to a representation (e.g., stacks and queues of objects). The approach suggested avoids that the same access path be described in several different ways, one for every concrete representation used, even though the "abstract" behavior of the access mechanism is the same.

b. Since the access path level clusters do not make any assumption about the concrete representation, they constitute better modules for the decomposition of larger programs; the common base language that refers to the concrete representation level serves as an adequate programming mechanism for the achievement of modular programming, as discussed in [11].

c. Abstract machine modelling is a well known technique for achieving portability [12]. The concrete level of the representation can be regarded as defining an abstract machine. Since a compiler is able to generate code for both the abstract and primitive data types used in a program (expressing it in terms of the primitive commands that handle the common base representation) a high degree of portability can be obtained.

d. By gathering run time statistics about a program and attributing cost functions to the standard operations implemented through concrete representation level clusters (in a way similar to [13, 14, 15]), it is possible to select the best concrete representation cluster for a given application. In fact, it is not difficult to envisage a system in which this selection is done automatically from a library of concrete representation clusters (as in [15]). Thus, run-time efficiency can be predicted and controlled in a machine-independent fashion.

e. The situation mentioned in a above is also reflected in proofs of correctness of the referred clusters - one has to prove the same "abstract" behavior several times, once for each concrete representation used. It is expected that the mappings used in proving the correctness of the original concept of cluster [16] will be decomposed into simpler mappings under the proposed extension. This concept was first

introduced in the language PEP [17].

In the present work we shall be dealing with the application of the three level methodology to data base systems. The particular base application's machine will be therefore a data base machine (the basic system of the previous section). The adoption of the above methodology will contribute to the systematic construction of a reliable operational implementation, and later to transform this preliminary implementation into an efficient one. Other software properties such as integrity, security, and data independence can also be enforced by means of the proposed techniques.

4. OUTLINE OF THE PROPOSED SYSTEM

4.1 A Data Base Machine

Regerdless of the adopted model we can view a data base as a collection of inter-related files. A file is an ordered or unordered set of records, which consist of a number of items of certain types (integer, real, character, Boolean, etc). Two files are inter-related by defining links between pairs of records, one from each file; links between records of the same file can also be defined (cf. [4] and the section "elements of logical structures", pages 13-15 of [18]*).

All these notions must be regarded in a very broad sense. For example, links are not to be confused with any structure that can implement them; for relational da ta bases a link may be achieved by actual record concatenation, or through an "indirect join" [19], etc.

Thus, the data base machine involves the data types file, record, item, and link, where the characterization of each type comprises the operations defined on the respective sets of objects. Such operations are typically additions, deletions, updates and selections.

---

* We have no counterpart to repeating groups, which have been demonstrated unessential by some DBTG analyzers.

For arriving at the data base machine one starts at the specification level. One possible axiom here, which restricts the operation of deleting a record from a file is that the record must not be referred to in any link.

At the access path level provision would be made, among others, for two-way paths between linked records. At the concrete representation level, we could decide to use pointer lists (attached to the represented records) for implementing the links.

The implemented data base machine will consist of clusters corresponding to the four (basic) data types.

4.2 The General Purpose and Special Purpose Systems

When moving to a more restricted system the three-level approach is still used to some extent. Additional clusters are created to implement data types such as, for example, CODASYL sets (at the general purpose stage).

A specification level is clearly needed for defining the additional restrictions. Thus, CODASYL sets use links, but they must be 1:n and cannot relate records of the same file.

However, there is no need to develop the two other levels- they are already given in the data base machine. This means that the operations defined with the CODASYL set cluster will invoke operations defined for the data base machine as primitives; furthermore these CODASYL set operations will contain control statements enforcing the additional restrictions (keeping the links 1:n , etc). In other words, no extra access paths or concrete representations are added.

The same goes for special purpose systems. It may happen, for example, that items employee - number and employee - salary appear (redundantly) in two different files, and we wish to impose the restriction that the same employee must have the same salary in both files (an integrity constraint); this additional restriction can be incorporated into the clusters especially coded for these files. Such clusters would invoke operations belonging to general purpose clusters (such as the CODASYL set cluster), and would employ control statements for the new restrictions.

Thus the general purpose and special purpose stages merely bring additional restrictions, expanding the specification level of the data base machine but still using its access paths and concrete representations.

A remark must be made here about security constraints. They can be partly incorporated via control statements, as with integrity constraints. One can also assign, to each individual user, rights to invoke certain operations in a cluster while preventing him from using the remaining operations in the cluster [20]; so a user can be allowed to perform selections and additions and prevented from performing deletions or updates with respect to a given file.

4.3 Optimization

Consider again the CODASYL set example. In section 4.1 we said that lists of pointers are a possible concrete representation for links, but we had the general case of n:m links in mind. For the 1:n links needed by CODASYL sets more economical representations are known (e.g. one pointer from the owner to the first member, one pointer from each member to the next with the last member closing the ring by pointing back to the owner - see figure 3 below).

Again in the files-with-redundant-information example it might be convenient to create an auxiliary path between records with the same employee-name to make it easier to check for consistency.

Finally, chained calls can become expensive; one would like to "flatten" the code of the cluster operations perhaps by in-line expansion instead of invoking lower level operations.

These are three possible ways to optimize performance and storage requirements. The first way is particularly interesting because it suggests a top down process which goes back to the data base machine to change its concrete representations (or possibly its access paths and concrete representations).

The second strategy requires the access path level and representation level to be used also when reconstructing the general or special purpose systems (with the auxiliary structures).

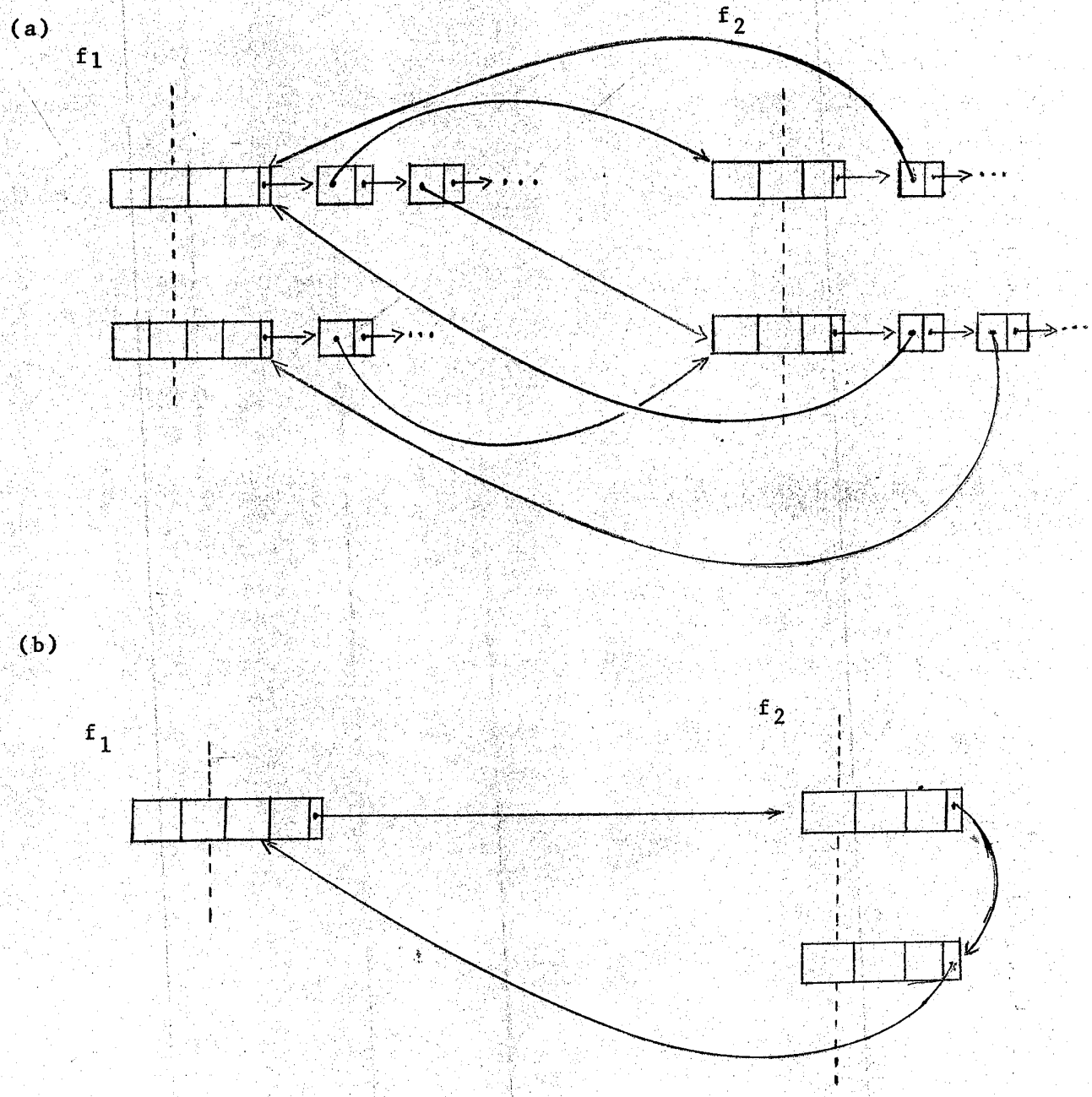

FIG. 3: Possible link representations for
(a) n:m links
(b) 1:n links

4.4 Other Considerations

Changing access paths and concrete representations can become a faster and safer task if libraries of alternative clusters are kept (cf. section 3, item d).

Assume now that one general special purpose system S which does not comprise the entire work in an installation, would benefit from a change in the data base machine. Assume further that the information involved in S is often handled together with other information whose manipulation is done through the unmodified (or differently modified) data base machine. In this case translation operations must be provided.

5. SUMMARY

The present paper describes a software development approach oriented towards the implementation of data base systems. The proposed approach needs to be further formalized and its validity has to be tested experimentally. The ideas contained in this work constitute the framework of a project aimed at providing a software environment for the development of custommized data base systems. The system to be designed will be centered on a PL/I extension [9] which supports the three linguistic level model described in section 3.

APPENDIX

Some possible operations of a data base machine

- Creation of structures

create data file
create link file
create inversion file

- Destruction of structures

drop data file
drop link file
drop inversion file

- Record selection

select data record (given its identification-result: the record)
select link record (given the identification of one record-result: identification of a record linked to the given one).
select inversion record (given a value sequence-result: identification of a record having the given values)

note: a select can be re-applied for obtaining another record, if any, meeting the given requirements.

- Record insertion

insert data record
insert link record
note: insertion of inversion records is caused by insertion of data records

- Record removal

remove data record
remove link record
note: removal of inversion records is caused by removal of data records; link records are also removed when data records are.

- Record update

  update values in data record

  note: removal and insertion of inversion records may follow the update of values in data records.

REFERENCES

[1] HOARE, C.A.R. - "Data Reliability" - Proc. of International Conference on Reliable Software" - Los Angeles (1975) 528-533.

[2] SIBLEY, E.H. (ed.) - "Special issue: Data base management systems" - Comput. Surveys, Mar. (1976).

[3] ANSI/X3/SPARC Study Group on Data Base Management Systems - Interim Report - FDT (1975).

[4] TSICHRITZIS, D. - "A network framework for relation implementation" - in "Data Base Description" - Douqué and Nijssen (eds.) - North-Holland (1975).

[5] CHALLIS, M. - "The Jackdaw database package" - University of Cambridge - Tech. Report 1 (1974).

[6] LISKOV, B. and ZILLES, S. - "Programming with abstract data types" - Proc. of SIGPLAN Symposium on Very High Level Languages (1974).

[7] LUCENA, C.J., SCHWABE, D. and BERRY, D. - "Method and facilities for data definition" - to appear (1976).

[8] EARLEY, J. - "Relational level data structures for programming" - Acta Informatica 2 (1973).

[9] SCHWABE, D. and LUCENA, C.J. - "Design and Implementation of a data abstraction definition facility - PUC - Tech. Report 14/76 (1976).

[10] LEAVENWORTH, B.M. and SAMMET,J.-"Overview of nonprocedural languages" - Proc. of the SIGPLAN Symposium on Very High Level Languages (1974).

[11] DENNIS, J.B. - "Modularity" in "Software Engineering, An Advanced Course" - Springer Verlag (1975).

[12] POOLE, P.C. and WAITE, W.M. - "Portability and adaptability" - in "Software Engineering, An Advanced Course" - Springer Verlag (1975).

[13] GOTLIEB, C.C. and TOMPA, F.W. - "Choosing a storage schema" - Acta Informatica 3 (1974).

[14] TOMPA, F.W. - "Evaluating the efficiency of storage structures"- University of Waterloo - Tech. Report CS-75-16 (1975).

[15] LOW, J.R. - "Automatic coding: choice of data structures" - Stanford University - Tech. Report CS-74-452 (1974).

[16] BERRY, D.M., ERLICH, Z., LUCENA, C.J. - "Structured data representations - proposed modifications to the concept of cluster" - Proc. of the SIGPLAN Conference on Data Abstractions, Definition and Structure (1976).

[17] LUCENA, C.J., SCHWABE, D. and BERRY, D. - "Issues in data type construction facilities" - PUC - Tech. Report 4/75 (1975).

[18] FRY, J.P. and SIBLEY, E.H. - "Evolution of Data-Base Management Systems" - Comput. Surveys, Mar. (1976).

[19] PALERMO, F. - "A data base search problem" - in "Information Systems" - Tou (ed.) - Plenum (1972).

[20] JONES, A.K. and LISKOV, B.H. - "A language extension for controlling access to shared data" - IEEE Trans. on Software Engineering 4, December (1976).