PUC

PRE-EDIÇÃO

SIMPOSIO INTERNACIONAL SOBRE METODOLOGIAS PARA O PROJETO E CONSTRUÇÃO DE SISTEMAS DE SOFTWARE E HARDWARE

PATROCINADOR

CONSELHO NACIONAL DE DESENVOLVIMENTO
CIENTIFICO E TECNOLOGICO

28 DE JUNHO A 2 DE JULHO DE 1976
RIO DE JANEIRO
BRASIL

Pontificia Universidade Catolica do Rio de Janeiro
ua Marques de São Vicente, 209 — ZC-20
Rio de Janeiro — Brasil

005.106
I61p

PREPRINTS

INTERNATIONAL SYMPOSIUM ON METODOLOGIES FOR THE DESIGN AND CONSTRUCTION OF SOFTWARE AND HARDWARE SYSTEMS

SPONSOR : CONSELHO NACIONAL DE DESENVOLVIMENTO
CIENTIFICO E TECNOLOGICO

CO-SPONSORS: CANADIAN INTERNATIONAL DEVELOPMENT AGENCY
GESELLSCHAFT FUR MATHEMATIK UND DATENVERARBEITUNG
NATIONAL SCIENCE FOUNDATION
IBM DO BRASIL

ORGANIZING COMMITTEE:

PROFESSORS

C.J.P. LUCENA
W. BRAUER
N. MACHADO
A.L. FURTADO

DATES

JUNE 28 TO JULY 2, 1976

NOTE

THESE PREPRINTS HAVE BEEN EDITED. THE AUTHORS ARE KINDLY INVITED TO REQUEST ANY CORRECTIONS OR CHANGES FOR THE FINAL EDITION.

TOWARDS A DBMS BASED ON ABSTRACT DATA TYPES

Rubens N. Melo

Pontificia Universidade Catolica - Rio de Janeiro

I. INTRODUCTION

Much work has recently been devoted to the development of DMBS's (Data Base Management Systems).

Some consensus seems to have been reached on some general points. For example many researchers agree on the multilevel structure nature of DBs (Data Bases). The agreement is extented to the associated concepts of INTERNAL, CONCEPTUAL and EXTERNAL SCHEMATA [ANSI 75] . Although there is no general agreement on the number of levels, their characteristics, their interrelationships and the related terminology, we can observe that the terminology and ideas introduced in [COD 71] and improved later in ANSI 75 have been largely accepted.

The meaning of the three concepts mentioned above as proposed in ANSI 75 can be summarized as follows.

The INTERNAL SCHEMA | is the description of the physical D.B. i.e the description of how the information is really stored in the D.B. There are some proposals about the manner in which those descriptions should be made. They range from descriptions that include details about physical aspects of the D.B. [COD 71] to some high level descriptions [SAA 75] .

The <u>EXTERNAL SCHEMA</u> is the description of the portion of information stored in the DB which is of interest to a class of users.

An EXTERNAL SCHEMA is characterized by two aspects: First, it describes only the subset of information which is of interest for a particular class of application. Second, it must describe this subset in a structured way. The structure should be oriented towards this class of applications.

The CONCEPTUAL SCHEMA consists of a complete description of all the information contained in the DB. It should carry no information about the way it is really (physically) stored. The principal function of the conceptual schema is to serve as an intermediate mapping between the several EXTERNAL SCHEMATA and the INTERNAL SCHEMA.

The objects of the EXTERNAL SCHEMA are implemented in terms of the objects of the CONCEPTUAL SCHEMA which in turn are implemented in terms of the objects of the INTERNAL SCHEMA.

In a DB design, the "Enterprise Administrator" first defines the CONCEPTUAL SCHEMA taking into account the intrinsic characteristics of the information to be stored in the DB, as well as the characteristics of the applications. The "Data Base Administrator" then follows the directives of the Enterprise Administrator and uses the available technological resources to define the INTERNAL SCHEMA. He actually builds up the objects of the CONCEPTUAL SCHEMA. The "Application Administrators" then define the EXTERNAL SCHEMATA based on the objects of the CONCEPTUAL SCHEMA and finally the EXTERNAL SCHEMATA are used by the different classes of users.

Most of the work in this field tries to find the best types for the objects at the different levels. Most of the discussions concern the conceptual schema to which several approaches have seen proposed [COD 70 , COD 71, ANSI 75, ABR 74, BFP 74, SEN 75].

However little work has been done in the specification of relations between the objects at the different levels. Furthermore the specification of the operations on these objects has been overlooked. We conjecture that this is one of the reasons why certain areas of DB technology are still underdeveloped (concurrency control, integrity control, etc).

In parallel with the development of DBMS there has been some efforts in the field of software engineering, directed to the area of (Extensible) Programming Languages. Here tools are described which permit the structuring of large programs in a modular fashion. Among these tools, the concept of Abstract Data Types seems to be one of the most promising.

An Abstract Data Type (ADT) is essentially a set of objects characterized by a common behaviour. The construction of an ADT is basically top-down where objects and operations in one level are defined in terms of the objects and operations of lower levels. In contrast to the policy adopted in the DBMS field, the behaviour of the objects of an ADT is stressed through the precise definition of the operations defined on them and the description of the objects is not emphasized.

The purpose of this paper is:

- To call attention to the similarities between the multilevel structure usually proposed for DB and the multilevel structure existing in the concept of Abstract Data Types.

- To suggest that the adoption of a model whcih gives more emphasis to the operations on the objects than to their representations may lead to better solutions of important problems in less developed areas of DBMS field.

- To show informally that the desired requirements of a DBMS can be achieved through the concept of Abstract Data Types.

- To suggest that the formalism of ADT when adapted to the problems of DBMS field may help in the formal specification of important DB problems.

- To suggest the notion of restriction of an abstract type as a useful concept for DBMS.

In section 2 we give an introduction to the concept of a database and give the principal requirements for a good DBMS according to a recent paper by Nijssen [NIJSSEN 76]. In section 3 we present some notions about Abstract Data types and in section 4 we indicate how the concept of ADT can be applied to DBMS. Finally in section 5 we present some concluding remarks.

II. DBMS REQUIREMENTS

After examining the discussions in the DBMS field we find it possible to formulate a comprehensive definition for the concept of Data Base. Based on this definition we can then deduce some desirable requirements for a DBMS.

A recent paper [NIJSSEN 76] states that a DB is "<u>a collection of data, used by a certain number of individuals, each applying his preferred mental model and language, during a certain period of time at a reasonable cost</u>". From this working definition the following requirements for a DBMS are deduced.

1. <u>Liberty of user model</u>

The DBMS must provide the user with facilities that permit him to construct his own model from a certain available basic model.

For example the DBMS should allow the users to apply to the same DB, a t least the relational, "network" and "hierarchical" models.

2. <u>User model completeness</u>

According to [NIJSSEN 76] a DBMS provides user model completeness if it provides structural completeness and constraint completeness. The structural completeness exists if the DBMS allows the descriptions of all the structures which can be mapped (forward and backward) onto a strucutre consisting of a finite number of sets (of n-tuples) and functions without resorting to dummies. The constraint completeness exists if the DBMS permits the description of any consistency and validity constraint.

3. <u>Data Independence</u>

Data independence has been recognized as a primary goal for DBs. Basically it reflects the immunity of user programs to changes in the physical aspects of the objects they are manipulating. For instance a program should not need to be modified if the representation of an object changed say, from an indexed-sequential file to a direct access file.

Taking into account that the reprogramming effort, wasted in large enterprises has been reported to occupy from 20% to 50% of the available programming staff, there is no doubt that data independence is an important fundamental goal.

4. <u>Language Independence</u>

The DBMS must facilitate the communication between the DB and the users by means of several languages. The user should be allowed to utilize his preferred language such as Cobol, Fortran and PL/I.

5. <u>Central Control over Security and Integrity</u>

The use of the DB "by a certain of individuals" which may be interested only in portions of the DB not necessarily disjoint may raise problems of security and integrity.

These problems (accuracy of DB, concurrent update, protected retrieval and hardware and DBMS failure) constitute areas currently less developed in the DB technology, since there is neither real experience nor a consistent theory. These areas are also not completely covered by the present DB language proposals.

6. <u>Simplicity</u>

The system should appear simple to the users in its diferent levels. The simplicity is influenced by the conceptual model adopted and by the syntax of the communication languages. The facility to permit the users to use their own abstractions is also an important factor for the simplicity of the system.

7. <u>Orthogonality</u>

A DBMS is orthogonal if it is easy to obtain subsets of it. This is an important requirement because it conveys the idea that subsets of DBMS may be constructed first.

8. <u>Cost and Performance Optimization</u>

The DBMS should offer several tuning facilities to modify the cost and performance associated with all resources.

A clearer formulation of the DB problem and the identification of the requirements of a DBMS provide a good basis to evaluate a proposal for a DBMS. In section 4 we indicate how the above mentioned qualities of a DBMS can be achieved by the use of abstract data types. In the following section a brief introduction to the concepts of ADT is given.

III. NOTIONS OF ABSTRACT DATA TYPES

An ABSTRACT DATA TYPE (ADT) comprises a class of objects with a common bechaviour. More precisely an ADT is usually defined by

1 - A set of objects

2 - A set of operators which

- are appliable to object of the ADT

  or

- yield objects of the ADT

3 - A set of properties (axioms) which these operators must satisfy.

As an example, the type INTEGER which is usually encountered in standard programming language is a particular case of an ADT. The set of objects coincides with the set of integers, the set of operations are the usual operations on the integers and the set of properties is the usual set of oxioms for the integers with some restrictions. A type such as INTEGER is said to be PRIMITIVE because it is automatically supported by most languages.

The type ARRAY which is also commonly supported by the standard languages is also a particular case of an ADT. There exists a set of operations defined on ARRAYS and these operations satisfy certain properties. In this case an object of the abstract type ARRAY is itself a set of elements of another type, for instance, INTEGER. When an object of the type ARRAY is declared, the type of the elements is also specified ("as a parameter").

The type ARRAY is an instance of a "TYPE GENERATOR" and in this case it is a "PRIMITIVE GENERATOR" because it is supported automatically by the system. These PRIMITIVE GENERATORs may be more general (as in ALGOL 68 and PASCAL) and are sometimes called MODES.

It is in the recent proposals and implementations of the EXTENSIBLE LANGUAGES (as CLU, ALPHARD, ABSURD.) [LIZ 75, WULF 74, SCHW 76] that the concept of ADT is more clearly developed.

In these languages the user is allowed to specify ADTs which are most convenient for the solution of a specific problem.

For example a user may decide to solve his problem with the help of an abstract type: Stack of Real and declare and use objects of type Stack of Real in the program.

The mechanism for the definition and implementation of new types vary in the defferent languages. As an illustration, the specification of an ADT in CLU [LIZ 75] is done by means of a module called CLUSTER that has the following appearance:

```
STACK OF REAL CLUSTER IS  POP, PUSH, EMPTY, TOP;

REP IS ARRAY [REAL]  (here the representation of the object is
                      specified)
----
CREATE: OP( ) RETURNS CVT  (This operation is supposed to be
                           invoked during the declaration of an
                           object)

----
END CREATE
PUSH : OP(S:CVT, X:REAL) RETURNS CVT      (Specification of the
TOP: OP (S: CVT) RETURNS INTEGER           Operations)
----
END TOP
----
END STACK OF REAL
```

In his program the user may then declare objects ("variables") of the type STACK OF REAL and manipulate them through the operations of PUSH, POP, etc.

It is important to separate two aspects of ADT: the type definition and the type implementation. Most languages support the type definition only through their implementation. Clusters are implementation of types in CLU. Type definition is thus expressed computationally.

In order to apply the concept of ADT in the DBMS field we feel that, mechanisms are necessary which separately support an implementation and a definitional (axiomatic) specification of types. In an environment where the representation of the objects can be shared by several programs with different views, it is essential to allow the axioms (definitions) to be further constrained by each user.

IV. USING ABSTRACT DATA TYPES IN DBMS

We have seen that in both the proposed structure of DB and the structure of abstract types, the basic idea is to construct user defined (High level) resources based on the system basic (lower level) resources. In the case of a DBMS more emphasis is given to the construction of objects in terms of lower level objects whereas in the case of a ADT the idea is to construct types in terms of lower level types.

The adaptation of the experience already obtained with the abstract type approach to the DBMS field presents immediately the advantage of using the same mechanism for the mapping between the different levels of a DB. Furthermore it is a clear and structured way of constructing objects in terms of other basic ones. The mechanism is reasonably formal and can be a good basis for a more precise formulation of important problems in the DBMS field.

In the sequel we give some indications for the use of abstract data type in DBMS and how this contributes to the achievement of the desirable requirements of a good DBMS.

The multilevel structure will be maintained and the objects and their types will be defined in different levels.

First Level

This level will correspond to the INTERNAL SCHEMA level.

The types utilized in this level will be:

- The primitive types already supported by the basic languages of the system.

  Ex. INTEGER, REAL, etc
- The primitive generators of types also already supposted by the basic languages of the system.

  Ex. RECORDS, ARRAYS, etc
- The Basic types derived from those mentioned above. These typesshould be an extension of the set of already supported types in order to obtain a suitable set of basic representation for the objects of the conceptual level and also of a suitable set of standard operations on these representations.

  Ex. Multilist files, inverted files, etc

Ideally, the set of basic representations and the associated set of standard operations should be kept to a minimum. This level is sometimes referred as a "basic virtual machine" [TOM 74] The DBMS must be able to maintain a library of basic types correspon- ding to the basic machine. In fact new basic representations can be added provided that the uniformity of reference [GESCHKE 75] of the operations is preserved. The uniformity of reference of a basic operation implies that it can be applied uniformly to any of the basic representation. For example if "Insert a new element" is a basic operation it can be applied equally to different representations of the basic set.

Second level

This level will correspond to the conceptual schema level In this level the DBMS must provide tools for the modelling completeness requirement. According to [NIJSSEN 76] the DBMS should have structuring facilities so that the structures preferred by the users "may be expressed in terms of sets (of n-tuples) and functions...". Moreover it must be able to describe any validity and consistency constraints.

Surprisingly this sounds like the definition of abstract data type where a structured class of objects is defined by means of sets, operators (functions) and axioms (constraints) to be satisfied. This leads us to conclude that the concept of abstract data type is a good candidate for a conceptual model of DB. As mentioned before it is important, in this level, to have mechanisms for supporting not only the implementation but also the definitions of types (where the integrity constraints should be described).

The types defined in this level must be general enough to capture the intrinsic structure of the information that the DB will contain as well as the user needs required in applications of the data. The conceptual schema should not be the addition of the different users views but their integration. At this level the objects will be constructed from basic representations chosen from those of the basic machine, and the operations on these objects will

The user never sees the physical representation that his program is manipulating nor needs to know how the operations he uses are in fact implemented (access strategy is also hidden !). This characteristic gives the system a high degree of data independence.

Third Level

This level will correspond to the EXTERNAL SCHEMA level. The existence of multiple users, each one using part of the information contained in the DB as if it were structured the way he prefers to see it, is part of the definition of DB. The abstractions used by a class of users must be able to be constructed from the abstract types of the conceptual schema. The types in this third level will be defined therefore in terms of the types or restrictions of the types of the conceptual schema.

The concept of restriction of an abstract type is here only vaguely defined. A precise definition is still missing. Nevertheless with the concept of restriction of type some important requirements of DBMS such as access control, privacy control can be achieved. The restriction must also preserve the integrity constraint defined at the conceptual level.

The mapping of different EXTERNAL SCHEMATA onto the same CONCEPTUAL SCHEMA implies the necessity for mechanisms which permit definitions of "external types" the representation of which, are associated to the same object in the conceptual level. Some work in this direction has been done by [BP 75, PAO 76] where a "binding operator" is introduced. However more work has still to be done on this problem.

V. CONCLUSIONS

In this paper we suggest that the adoption of the concept of abstract data type for the development of DBMS is a fruitful area of research. We believe that the incorporation of a model (and its associated formalism) which emphasizes the operations on the objects will lead us to a better formulation of important problems in the less developed areas of the DBMS field.

Starting from the definition of DB we presented some of the principal requirements for a DBMS. Then we indicated how the concept of abstract data types (developed in the field of software engineering) could be used in a DBMS with these requirements.

The multilevel structure of DB is maintained and the same mechanism is used for the mapping between all levels.

The modelling completeness is achieved because the structuring facility provided by the abstract type is general and the constraint completeness could be supported in the type definition.

The data independence requirement is achieved by construction.

With regard to optimization of cost the diversity of options provided by the system both in the creation of new abstractions at the user level and in the choice of basic representations at the basic machine level offers several chances for tuning. In the future we can even think of an automatic process of selection of basic representations which optimizes a certain objective function. Some work in this direction has already been reported [TOM 74]

Finally the notion of restriction of types was introduced as an useful concept in the construction of user views and access control.

REFERENCES

ANSI 75 - Interim Report ANSI/X3/SPARC Study Group on Data Base Management Systems
ANSI, February 1975

COD 71 - CODASYL DATA BASE TASK GROUP (1971)
Report, April 1971

SAA 73 - Senko M. E., Altman E. B., Astrahan M. M., Feheder P. L.
"Data Structures and Accessing in Data Base Systems".
IBM System Journal, 12, 1, 1973

COD 70 - Codd E. F.
"A Relational Model of Data for Large Shared Data Banks"
CACM 13, 6, JUNE 1970

ABR 74 - Abrial J. R.
"Data Semantics"
in DATA BASE MANAGEMENT SYSTEMS (editors)
Koffeman and Klimbic, North Holland, 1974

BEP 74 - Bracchi G., Fedeli A., Paolini P.
"A Multilevel Relational Model for Data Base Management System".
in Data Base Management Systems
(editors) Koffeman and Klimbri, North Holland, 1974

SEN 75 - Senko M. E.
"Data description language in the concept of a multilevel structured description: DIAM II with FORAL".
in Data Base Description

NIJ 76 - NIJSSEN G. M.
"A Gross Architecture for the Next Generation DBMS".
PREPRINTS / IFIP TC - 2 working Conference on Modelling
in Data Base Management Systems - January 5. - 9. 1976
Freudenstadt (Black Forest)
Fed. Rep. of Germany.

WULF 74 - WULF, W. A.
"Alphard: Toward a language to support structured Programs"
Dept of Computer Science.
Internal Report. Carnegia-Mellon Univ. Pittsburgh
April 1974

LIZ 75 - Liskov B. H., Zilles S.
"Programming with abstract data types"
Proc. of ACM-SIGPLAN Symposium on Very High Level
Languages, in SIGPLAN Notices, 9, 4, (April 1974)

GES 75 - GESCHKE, C. M. and Mitchell, J. G.,
"On the Problem of Uniform References to Data Structures"
International Conference on Reliable Software, April 1975

SCHW 76 - SCHWABE, D.
"Aspectos de Engenharia de Software"
Projeto de Linguagem de Programaçao", PUC-RJ, Dept de
Informatica, 1976

PAO 76 - PAOLINI, P.
"Programming with abstract objects"
Internal Memorandum # 146, UCLA, Nov 1975

BP 75 - Berry, D. M., and PAOLINI, P.
"Binding in Abstract Data Types: Data Bases and JCL"
Internal Memorandum # 144, UCLA, Jan 1976

TOM 74 - TOMPA,
"Choosing a Data Storage Schema"
Phd Thesis, Univ. of Waterloo, 1974