Entity-Relationship Approach to Software Engineering
C.G. Davis, S. Jajodia, P.A. Ng and R.T. Yeh (eds.)
Elsevier Science Publishers B.V. (North-Holland)

A VIEW OF DATABASE MANAGEMENT SYSTEMS AS ABSTRACT DATA TYPES

by

Paul K. Blackwell, Sushil Jajodia*, and Peter A. Ng

Department of Computer Science
University of Missouri-Columbia
Columbia, Missouri 65211
U.S.A.

In this paper, we attempt to outline a proposal of a database management system which supports an entity-relationship model, as a collection of abstract data types. Each abstract data type in this collection can be formalized using algebraic axiom specification technique of Guttag; each can then be implemented to yield a set of interactive tools which will provide a formal and stepwise process in the specification of a database system.

I. INTRODUCTION

For nearly a decade, the abstract data type (ADT) concept has been investigated as a promising tool in software engineering and programming language design. Its ability to provide precise implementation-independent specifications of complex data structure led to the suggestion that it could be used to describe a database management system (DBMS). Indeed, in recent years, several efforts have been made to use the ADT concept to characterize particular aspects of a DBMS [3,9,16]. However, we are not aware of any work that has succeeded in integrating these partial results into a complete specification for a DBMS.

We attempt to outline a proposal of a DBMS which supports the entity-relationship (E-R) model as a collection of ADT's. It is our goal to formalize each ADT in this collection using the algebraic axiom specification technique of Guttag [11]; each can then be implemented to yield a set of interactive tools which will provide a formal and stepwise process in the specification of a database system. We have made some progress toward our goal and describe it here.

We have chosen the E-R model [4] since it is considered more natural and "closer to the user's conceptual model of the data" than other data models [17]. This model is however regarded as informal [19], thus, our aim is to formalize it.

The organization of this paper is as follows. In Section II, we briefly describe the E-R model and the E-R diagram. In Sections III and IV, we discuss the similarities and differences between the ADT approach and the current approaches to database design, and how a DBMS can be viewed as a collection of ADT's. In Section V, we briefly describe the algebraic specification of the ADT Erd and some advantages of its implementation. We discuss the application of the ADT approach to verification and testing of database design in Section VI. Finally, the conclusion is given in Section VII.

*The work of S. Jajodia was supported, in part, by a University of Missouri Summer Research Fellowship.

II. THE ENTITY-RELATIONSHIP DIAGRAM

In this section, we briefly describe the E-R model and the E-R diagram. In the E-R model, information is described in terms of four primitive concepts, viz., entity, relationship, attribute and value. Entities represent things or objects in the real world, and relationships are described by means of attributes and their underlying value sets. Entities are classified into sets of different types, and the relationships among entities are classified into various relationship sets.

An E-R model can be represented by a graph, called an E-R diagram, as shown in Fig. 1. Each node in the graph either corresponds to an entity set, a relationship set, or a value set. Entity sets are represented by rectangular boxes, relationship sets by diamond-shaped boxes, and value sets by circles. Participation of an entity set in a relationship set is denoted by an edge connecting the two. An arc pointing from an entity or relationship set to the appropriate value sets constitutes an attribute.

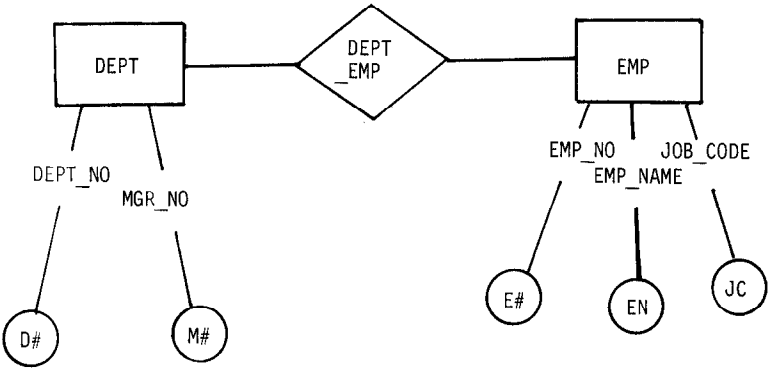


Figure 1 An E-R Diagram

The information about entities and relationships among entities at any instant can be organized in a collection of tables: entity set relations and relationship set relations. For example, a database instance for the E-R diagram in Fig. 1, is displayed in Fig. 2.

The E-R model has been shown to be an acceptable model for defining the logical view of a database [5,6]. This model, in some sense, generalizes the three major data models that have been used in database systems--hierarchical, network, and relational. The E-R diagram can be translated in a straightforward manner into a hierarchical, network, or relational database scheme. Also, it can be implemented directly [17, 18]. The main advantage of the direct implementation is that the user can formulate queries directly from the diagram, thus permitting the user to have a basic understanding of the underlying logical organization.

III. DATABASE MANAGEMENT SYSTEMS AND ABSTRACT DATA TYPES

The ADT's have been shown to play a significant role in the development of software that is reliable, efficient, and flexible. Since many complex systems can be viewed as ADT's, it was suggested that ADT's be used to characterize a DBMS [1, 8].

We can define an ADT as a mathematical model together with various operations

DEPT_NO	MGR_NO
11	112233
12	223344
13	334455

EMP_NO	EMP_NAME	JOB_CODE
112233	JANE KNIGHT	10
223344	JIM SAND	5
334455	DON SMITH	12
445566	JON ZIMMER	9

DEPT_NO	EMP_NO
11	112233
12	223344
13	334455
11	445566

Figure 2 An Instance of the E-R Diagram

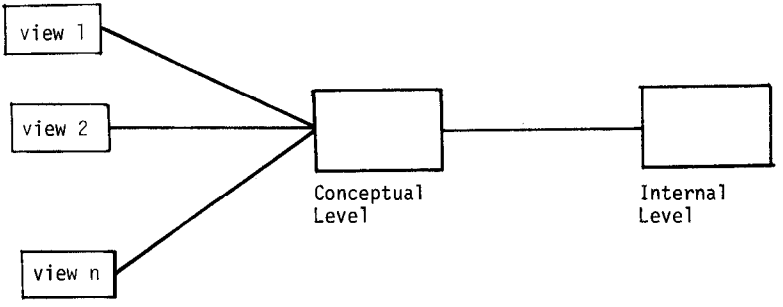
defined on the model [1, 11]. For example, we can view a stack as a sequence of zero or more elements such that all insertions and deletions take place at one end, called the top. The list of operations on a stack might include operations such as PUSH, POP, TOP, ISNEW, REPLACE, etc. Unfortunately, we cannot view a DBMS as an ADT in such a simple and straightforward way. There are several levels of abstraction which exist in the specification of a database structure, and each level can be viewed as an ADT. For example, the ANSI/SPARC study group proposed a three scheme mode to support a DMBS: a set of external schemas to meet the needs of the application programs, a single conceptual schema to define the information needs of the organization, and a single internal schema to describe the physical structure of the database. Each of these schemas can be described in terms of ADT's.

There are many similarities between the ADT approach and the current approaches to database design. An ADT permits formal specification of a data type without any reference to an implementation. This is very close to the data independence objective in a database, of providing a sharp and clear separation between the physical and logical aspects of a database. The user should be able to interact with the database at a convenient and abstract level without any knowledge of the physical data structures, storage organizations, and access methods used to store the data on a storage device. The ADT approach uses the "top-down" design methodology which imposes a hierarchical structure on the program development; each level of the structure represents abstractions which suppresses all irrelevant detail while clearly exposing the relevant concepts and structures. This approach is consistent with the usual database design methodology which calls for dividing the database design into two steps: logical design and physical design; each step is further subdivided into smaller steps of manageable proportions. A major advantage of the ADT approach is that it provides a formal basis for proving the correctness of the implementation of a data type. This can be very useful in the

design of a database which usually is an extremely complex task. Because of these similarities and obvious benefits, it is natural to investigate whether the abstract data type approach can be used in the database context.

IV. VARIOUS LEVELS OF ABSTRACTION IN A DATABASE MANAGEMENT SYSTEM

It is generally agreed that there are at least three levels of abstraction which exist in the specification of a DBMS: EXTERNAL LEVEL (individual user views), CONCEPTUAL LEVEL (community user view), and INTERNAL LEVEL (storage view) [7,19].



The internal level is the one which is closest to the physical storage, i.e., the one concerned with how the data is stored permanently on secondary storage devices. The internal schema is specified by means of a data storage description language. The conceptual level is the abstraction of the database in its entirety. A DBMS provides users with a data definition language (DDL) to describe the conceptual database in terms of a data model such as network, relational, or E-R model. The external level is the one closest to the users and represents an abstraction of that portion of the conceptual database which is of interest to them. The user has at his disposal a subschema data definition language for declaring views.

Now each level of abstraction can be viewed as a distinct abstract data type. For example, we can think of the conceptual model and the operations defined on it (DDL) as analogous to the stack concept and its corresponding operations (push, pop, etc.). This is the justification for the next section where we formalize an E-R diagram, which is the abstract level of an E-R model, as an abstract data type Erd.

In addition to these three levels, there is yet another dimension to our perception of the database: instance of the database which is concerned with the actual data present in a database. The data manipulation language (DML) is the interface employed by users to access or modify the contents of a database. The DML is designed so that it allows the manipulation of data structures of the sort supported by the underlying data model. The DML allows users to add new records to the database, and to look up, modify, insert or delete existing records. Again, we can treat a database as an ADT by viewing files (or relations) in the database as a mathematical model and the DML as a collection of operations on the objects of the model. It might appear that databases are different from data types such

as arrays or stacks in that data in a database change frequently and operations on the objects (i.e., queries) can be quite different. A moment of reflection shows however that contents of a database changes frequently, the conceptual schema does not and basic operations on the database remain the same. For example, any query in a language based on the relational algebra is equivalent to an expression involving the basic relational operators such as select, project, join, etc.

V. THE ALGEBRAIC SPECIFICATION OF ABSTRACT DATA TYPE ERD

In this section, we briefly describe the syntactic specification of the abstract data type Erd. We urge the interested reader to consult [2] for details which are omitted here. By definition, an algebraic axiom specification of a data type T consists of a syntactic and semantic specification [11]. The syntactic specification defines the names, domains, and ranges of the data type T. The semantic specification contains a set of axioms in the form of equations which relate the operations of T to each other.

The algebraic specifications of Erd is based on the following assumptions:

- In an E-R diagram, all entity set and relationship set names are distinct.
- Also all pairs of attribute and value set names are distinct.

The operations on the data type Erd belong to one of these four categories:

1. CONSTRUCTOR SET OPERATIONS

   The abstract data type Erd contains these constructor set operations, i.e., operations which satisfy the property that all instances of the data type Erd can be represented using only constructor set operations.

   a. NEWERD - Create and initialize a new E-R diagram.
   b. ADDESET - Add an entity set to the E-R diagram provided it has not been added to the E-R diagram previously.
   c. ADDRSET - Add a relationship set to the E-R diagram provided all entity sets involved in the relationship exist in the diagram and it has not been created in the E-R diagram previously.
   d. ADDAVSET - Add a (attribute, value set) pair to a particular entity or relationship set provided the latter exists but the pair does not in the E-R diagram.

2. HIDDEN (INTERNAL) OPERATIONS

   The abstract data type Erd has these hidden operations:

   a. INSERTESET - Add a particular entity set to the E-R diagram.
   b. INSERTRSET - Add a particular relationship set to the E-R diagram.
   c. INSERTAVSET - Add a particular (attribute, value set) pair to an entity or relationship set in the E-R diagram.

   Like the constructor set operations, it is possible to define all instances of the data type using only hidden operations. The crucial difference is that the execution of these operations does not require a check as to whether they are valid operations, so a user is not allowed to use these operations.

3. DELETE UPDATE OPERATIONS

These operations allow the user to modify the existing E-R diagram. The operation names are DELETEESET, DELETERSET, DELETEAVSET, REPLACEESET, REPLACERSET, and REPLACEAVSET. Their functions are self-explanatory so their description is omitted.

4. SUPPLEMENTARY OPERATIONS

These are supporting operations which allow the user to check whether, for example, a particular entity set has already been added to the E-R diagram or simply to list all the entity sets, relationship sets and their attributes and value sets. The operation names are LISTESET, LISTRSET, NEIGHBORHOOD, HASESET, HASRSET, AND HASAVSET.

The algebraic specification of the abstract data type Erd has been implemented at the University of Missouri - Columbia, using the PL/1 language. It can be used as an interactive tool which can play an important role in the E-R approach to database design. Some of its advantages are given below:

a) It takes the responsibility of linking together facts (entity sets, relationship sets, attribute and value sets), thus leaving some of the tasks involving the completeness and consistency of the collected requirements to the computer.

b) It checks the formal correctness of the E-R diagram which is being built and detects inconsistencies and redundancies.

c) It produces, at the end of the whole process, an E-R diagram which can be used to design subsequent phases.

d) With graphic capability, an E-R diagram can be generated or modified interactively with a graphic display terminal. This provides a more natural graphical form for the requirement engineer to work with, rather than using a one-dimensional textual language. This approach employs a graphical notation for better communication without any reduction of emphasis on rigorous formalism.

We have also given the algebraic specification of operations required for information retrieval queries. It allows users to insert, delete, or update tuples from entity or relationship set relations. We omit their description.

VI. APPLICATION OF THE ABSTRACT DATA TYPE APPROACH TO VERIFICATION AND TESTING OF DATABASE DESIGN

Database design is a long and tedious process, and one of the biggest problems is that of getting from the informal requirements of a required system to a detailed database design in such a way that the consistency, viz., whether the E-R diagram corresponds to the given description of the enterprise, completeness, viz., whether both the description of an enterprise and the E-R diagram are well defined, and the correctness, viz., whether the implementation of an application is correct.

At present, in the E-R approach to logical database design, information of interest about the real world is organized by some ad hoc method into an enterprise schema which is expressed in terms of an E-R diagram which can then be transformed into an appropriate database schema, e.g., E-R schema, relational schema etc. Because the translation from the requirements to the E-R diagram is not a well-defined process, it is based on the intuitive knowledge of the meaning of the description. Thus, it is difficult to verify, for example, whether the requirements are consistent with the resulting E-R diagram and the subsequent implementation; however, formalization of the system requirements in early stages of the software development process can lead to specifications which can be checked for completeness, consistency, and correctness. We describe this process next.

Given a description of the requirements (in a certain form) of a database system, we can view it as an abstract data type T. Thus, T can be thought of as a collection of sets, e.g., set EMP consisting of employees, set DEPT consisting of various departments etc., and a collection of allowable operations permitted on these sets. At this point, only the different sets and how the various operations act on these sets need to be identified, not how they will be implemented. In this way, T can be expressed formally using algebraic axioms.

The next step in the design should be to check that the specification is consistent with the original requirements and sufficiently complete. Although this cannot be performed rigorously, even an informal check will tend to expose shortcomings or inconsistencies between the formal and informal specifications.

The final set in the design is to provide implementation of the abstract data type T. If the DML has been formalized as an abstract data type, it can be viewed as a type concept which is available to the user for defining the application data type T, and we can write down an implementation of T in terms of the data type. The advantage to this approach is that now the correctness of the implementation can be verified formally [10, 11, 12, 13].

This approach has essentially been carried out in [9] for a restricted version of the E-R model. The differences are that in [9], the procedural specifications [8] instead of algebraic axioms are used to specify the data types and that it is recommended that implementation rather than specification be tested against the original requirements. We recommend that the specification be checked since otherwise much effort may be wasted in implementing specifications which are inconsistent or not sufficiently complete.

VII. CONCLUSION

In this paper, we have attempted to outline a proposal of a DBMS which supports an E-R model, as a collection of abstract data types. We have addressed the architectural issues that arise in terms of different levels of abstraction; but we have not discussed facilities of database management systems which help maintain the logical and physical integrity of data. These topics are currently being studied.

REFERENCES:

1. Aho, A. V., Hopcroft, J. E. and Ullman, J. D., Data Structures and Algorithms (Addison-Wesley, Reading, Massachusetts, 1983).

2. Blackwell, P. K., Jajodia, S., and Ng, P. A., The Algebraic Specification of Entity-Relationship Diagrams, Technical Report, University of Missouri-Columbia, (1982).

3. Brodie, M. L., and Schmidt, J., What is the Use of Abstract Data Types in Data Bases?, Proc. 4th Conference on Very Large Data Bases, (1978) pp. 140-141.

4. Chen, P. P., The Entity-Relationship Model: Towards a Unified View of Data, ACM Trans. on Database Systems, (1)1(1976) pp. 9-36.

5. Chen, P. P., (ed.), Entity-Relationship Approach to Systems Analysis and Design, (North Holland, Amsterdam, The Netherlands, 1980).

6. Chen, P. P., (ed.), Entity-Relationship Approach to Information Modeling and and Analysis, ER Institute, Saugus, California (1981).

7. Date, C. J., An Introduction to Database Systems, 3rd Edition, (Addison-Wesley, Reading, Massachusetts, 1981).

8. Furtado, A. L., and Veloso, P. A., Procedural Specification and Implementations for Abstract Data Types, ACM SIGPLAN Notices, (16)3(1981) pp. 53-62.

9. Furtado, A. L., Veloso, P. A. S., and de Castiho, J. M. V., Verification and Testing of S-ER Representation, in Chen, P. (ed.), Entity-Relationship Approach to Information Modeling and Analysis, ER Institue, Saugus, California (1981), pp. 125-150.

10. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., Initial Algebra Semantics and Continuous Algebra, Jrnl. of ACM, (24)1(1977) pp. 68-95.

11. Guttag, J. V., Abstract Data Types and the Development of Data Structures, Comm. of ACM (20)6(1977) pp. 396-404.

12. Guttag, J. V., Horowitz, E., and Musser, D. R., Abstract Data Types and Software Validation, Comm. of ACM, (21)12(1978) pp. 1048-1064.

13. Guttag, J. V., Horowitz, E., and Musser, D. R., The Design of Datas Type Specifications, in Yeh, R. T. (ed.), Current Trends in Programming Methodology, Vol. IV, (Prentice-Hall, Englewood Cliffs, New Jersey, 1978) pp. 60-79.

14. Hubbard, G. U., Computer-Assisted Data Base Design, (Van Nostrand, New York, New York, 1981).

15. Liskov, B. and Zilles, S., Specification Techniques for Data Abstractions, IEEE Trans. on Software Engineering, (SE-1)1(1975) pp. 7-19.

16. Lockemann, P. C., Mayr, H. C., Weil, W. H., and Wohlleber, W. H., Data Abstractions for Database Systems, ACM Trans. on Database Systems, (4)1(1979) pp. 60-75.

17. Poonen, G., CLEAR a Conceptual Language for Entities and Relationships, Proc. International Conf. on the Management of Data (1978) pp. 194-215.

18. Shoshani, A., CABLE: A Language Based on the Entity-Relationship Model, Lawrence Berkeley Lab., Berkeley, California (1978).

19. Ullman, J. D., Principles of Database Systems, 2nd Edition, (Computer Science Press, Rockville, Maryland, 1982).