Armazenamento de Informação - SBD
Banco; Dados
Modelo; Entidade; Relacionamento
Projeto; Banco; Dados
CNPq 1.03.03.00-6

A State-Space Approach for Database Redesign

José Mauro Volkmer de Castilho*

1 INTRODUCTION

The E-R approach [Chen76] has nearly become a standard for the specification of database's conceptual data requirements in database systems design, due to its simplicity, easy understanding, and wide use within the software development community. Most of the information systems in use today, specially those that have been properly designed, have E-R diagrams documenting their conceptual schemes, describing the structure and interdependencies of corresponding conceptual data.

It seems then adequate that such diagrams ought to be used when one has to study the redesign of available databases, adjusting them to new requirements defined by the user. Modifications, new data structures, and enhancement of existing data structures result in the specification of new E-R diagrams, which normally show strong similarity with the original diagrams.

Data conversion from an old version to a new version of a database is a critical task, whose results may have heavy impact on the existing set of application programs that manipulate the database. The ideal situation is one in which only exactly the programs that access data involved in the modification of data requirements, and the corresponding stored data structures, should be adapted, recreated or recompiled, and in which all still meaningful data is moved to the new stored data structures, with small computational effort (meaning few time and few additional storage space).

Chen has briefly studied diagrams transformation in an old paper [3], presenting a set of *operations* on E-R diagrams which correspond to diagram modifications performed more frequently during the enhancement of a software system's data description. Batini et al., in [1], present a more complete study on E-R diagram transformations, classifying them as transformations that preserve information contents of the corresponding data descriptions (seeking minimality or redundancy avoidance, diagram normalization, better expressivity and legibility), and as transformations that increase information contents of the corresponding schemes.

The above mentioned studies worried basically with the increase of quality of the schemes themselves, a task to be performed by the system's designer during the conceptual data design phase. Chen's study considers the availability of an already existing database, but Batini's study seems to be applicable only during the conceptual data design phase.

Instituto de Informática-UFRGS, Caixa Postal 15064- CEP91501-970 Porto Alegre-RS

The redesign of a database, adapting it to new data requirements, may make use of diagram modifying operations mentioned in both studies, for better results:

(1) the operations in [3], and those in [1] that increase scheme's information contents, may be used to modify the diagrams, adjusting them directly to satisfy the new requirements;

(2) the operations in [1] that preserve information contents of the old schema, may be used to enhance the quality of the new schema.

A simple way to map old->new diagram modifications to corresponding old->new modifications on stored data structures could be: associate to each diagram transforming operation, from [3] and [1], a programming language procedure that should create the contents of new stored data structures, or adapt the contents of old ones, to adjust them to the new conceptual diagram. Studies in this direction were done by [Gazola92] and [2], considering data stored as database relations. The general expression for data conversion could be specified as:

reference to a new stored data structure <- (programming language procedure whose execution retrieves and uses a set of data values extracted from old stored data structures, to create the contents of the new stored data structure)

The study and definition of such programming language procedures for database conversion are outside the scope of this work.

Considering the available sets of diagram transformation operations, there are usually many alternate ways of using them to construct a new diagram from an old one. The set of operations actually used by the systems designer is only one of them, maybe the one that seemed more "natural" to him for this task. Alternate diagram transformation sets of operations may be grouped in *classes*, which have in common the particular set of operations used, together with their *opposite* operations (well-defined sets of operations have *positive*, or concept including, operations, and *negative*, or concept excluding, operations. This does not imply that any well-defined set of operations should have an even cardinality, because two or more operations may have as *opposite* a single operation. This will not be explored here any further, but the reader may examine the sets of operations proposed by Chen and Batini for a hint on this statement).

In any class there is at least one *minimal* transformation set, which is, simply, the smallest set of elements in the class. Any transformation set which is not minimal in the class, must exhibit some sort of *transformation cycle*, involving operations and their opposites (transformations done by some operations are undone by others, and vice-versa).

It is important to observe that the set of diagram transformation operations that the designer has used may not be one of the best, that is, one that corresponds to a sequence of programming language procedures whose execution produces a new database in less execution time and using less auxiliary storage space during database conversion.

In fact, what one has in hand here is a rather complex optimization problem, whose ultimate goal is the systematic construction of a complete program that should create a new

database from the old one, in less time and using less auxiliary storage space. This paper proposes an approach to solve this problem in two steps:

(1) to identify all sets of diagram transformation operations that may transform an old diagram in a new one, using available operations from a given *operations' universe* (i.e., the sets of operations presented in [3] and [1]);

(2) for each minimal set found, identify what order should be defined on the operations of the set, so that the optimization goals may be achieved. This order will determine the order of execution of the corresponding set of database conversion procedures.

Any one of the two problem steps may be expressed as a state-space search problem [7] or plan-formation problem [6]. For both steps, the state space is the set of all correct E-R diagrams (those that do not have entity boxes directly connected to other entity boxes, or relationship boxes connected directly to other relationship boxes). In the first step, the state-space search should look for sets of diagram transformation operations that generate the new diagram from the old one. Each operation in the set should take the diagram produced by another operation in the set and make its resulting diagram "closer" to the new diagram, in the sense that the corresponding operation introduces a diagram part that the new diagram has and the old one has not, or takes out a part that the old diagram has and the new one has not.

In the second step, the state-space search looks for sequences of diagram transformation operations, each sequence taken from one of the sets identified in the first step, with the initial state represented by the *old* diagram and the final state represented by the *new* diagram.

The first step has not yet been examined in depth. It should group in sets all possible and *productive* combinations of instances of diagram transformation operations (operations with instantiated arguments) taken from a given set of *operation schemes* (parameterized operations, defined like the ones proposed by [3] or [1]). *Productive* here means that the corresponding combination of instantiated operations should exhibit the "closeness" property outlined above.

Any set of (instantiated) operations identified in this first step should also have a sort of *sequencing* property: it should be possible to establish at least one sequence of operations in the set, where the first one uses as initial state the old diagram, the second uses the diagram produced by the first operation as its initial state, and so on until the new diagram is produced by the last operation in the sequence. Operation sets without the sequencing property should be discarded. In general, the identification of productive combinations of operations, and the verification of the sequencing property of a set of operations seems to be a hard task.

To simplify the presentation of the whole idea, it was decided here to make this first step trivial, using a small set of *atomic* operations schemes, instead of using the more *natural* sets proposed by [3] or [1]. This set has only operations for the creation and exclusion of entity and relationship elements from diagrams. Just by inspecting the

differences between old and new diagrams, one can determine a set of instantiated operations which has the sequencing property described above.

This work then concentrates on the outlining of a solution for the second step, through the presentation of a small example. To make the presentation still more simple, it was used, instead of the full E-R model, a shortened version of it, having only the Entity and Relationship modeling concepts, with no attributes or relationship cardinalities, and no extensions to deal with data abstractions.

It may be shown that such simplifications do not invalidate the generality of the approach taken. For instance, semantic extensions, like specialization/generalization or aggregation, may be modeled as special relationships. Attributes may be modeled by defining value domains as entities and attributes themselves as binary relationships between normal entities and domain entities. Cardinalities are more difficult to deal with. Their treatment may need the addition of some sort of *cardinality constraint* defining mechanism to our simple E-R model. This will be left for later studies on the subject.

It is supposed, in some explanations made during the example presentation, that data are stored as relations, following the relational model of data [4].

The paper is organized in four sections, including this introduction. Section 2 gives a general idea of the proposed approach for the second step, and section 3 details the formalisation of the problem as a state-space search problem, with the use of a small example. Section 4 presents conclusions and further research directions.

2. A QUICK OUTLINE OF THE STATE-SPACE APPROACH TO DATABASE REDESIGN.

This work's proposal is based on a simple basic idea: having at hand the E-R diagram that describes the current contents of the software system's database, the data designer defines how it should be modified to satisfy new data requirements from the user. The modifications introduced should preserve the properties of legibility, expressiveness and minimality in the resulting diagram [1].

In order to formally present the approach taken in this work, a sort of *E-R diagram transformation algebra* is defined. Its domain is the set of all possible E-R diagrams, and its operations set comprehends operations of creation and exclusion of entity and relationship concepts on diagrams. Each transformation operation of this algebra has at least one input argument denoting an E-R diagram, and produces as resulting value a correspondingly transformed E-R diagram. The operations set should exhibit the property that, given any pair of correctly expressed E-R diagrams, one called *old* and the other *new*, there should exist at least one sequence of transformation operations, where each operation uses the resulting diagram produced by the previous operation in the sequence, which could create the *new* diagram from the *old* one. Normally many sequences satisfying this condition may exist for the same set of operations.

As already said, the conversion of the old database version to the new one is done with the use of a set of predefined programming language procedures, arranged in a sequence that follows the sequence of corresponding diagram transformation operations.

The redesign problem then reduces itself to: given an old diagram, that describes the contents of an (old) relational database, look for the *best* sequence of diagram transformation operations. Such sequences are intuitively characterized as those that correspond to a sequence of data conversion procedures, which produces a new version of the stored database, with the minimum of computational effort (access time, auxiliary storage space), and using the maximum of information contained in the old database.

The approach taken in this work is also based on the premise that *old* and *new* versions of a database cannot exist at the same time: the new version should be obtained by "on the spot" modifications performed on the old version. This premise must in fact be true when the database is stored in relatively small secondary memory, as happens quite often in software systems implemented on personal microcomputers.

Several criteria may be used to characterize what is the best sequence of diagram transformation operations: number of operations in the sequence; size of auxiliary storage space, necessary during data transport between old and new versions of the database; size of the total amount of bytes transported between versions (remember that both occupy more or less the same areas on secondary storage!); amount of secondary memory physical accesses, necessary to seek and get/write data. To simplify the discussion, only the first two are used here: the *best* transformation sequence is one that is short enough, and uses minimum amount of secondary storage space (none, if possible).

3 FORMALISATION OF THE DIAGRAM TRANSFORMATION PROBLEM AS A PROBLEM OF STATE-SPACE SEARCH, WITH AN EXAMPLE

To study the problem of E-R diagram transformation and databases conversion as a state-space search problem [7], the syntax of a convenient First Order Language [5] is defined, to specify the algebra of diagram transformations. The language is many sorted, and the main sort is *E-R diagram*. To this sort corresponds the domain of "correct E-R diagrams", including the empty, or initial, diagram. Other sorts are: *entity* (meaning: *entity name), relationship* (meaning: *relationship name), entity set* (meaning: *set of entity names).* To each one of those sorts correspond, respectively: the domain of entity names; the domain of relationship names; the domain of sets of entity names. The syntax of the language's functional symbols (which represent diagram transformation operations) is defined below:

create-e (entity, diagram)

create-r (relationship, entity set, diagram)

exclude-e (entity, diagram)

exclude-r (relationship, entity set, diagram)

Considering exactly these operations, there exists always only one class of sets of transformation operations for any pair *old--new of* E-R diagrams. Hence, the first step of the redesign problem is easy to perform, since any instantiated operation includes or excludes a single concept from the diagram.

So, just by inspecting the diagrams, one may identify the elements of the class: are all those sets of instantiated operations which contain operations that perform the inclusion of concepts existing in the new diagram and not existing in the old one, and which contain operations that perform the exclusion of concepts not existing in the new diagram, but existing in the old, plus any quantity of operations forming pairs of *opposite* operations. The minimal element of the class is also easily identifiable: it is the smallest set in the class, and is unique.

Figures 1 and 2 show a pair of *old-new* diagrams to illustrate the approach. Both diagrams represent a database for an Academic Management application. The old diagram (figure 1) has three entities (**S**, for *Students*; **D**, for *Disciplines*; **T**, for *Teachers*), and two relationships (**m**, for *inscribed-in*; **t**, for *teach*).

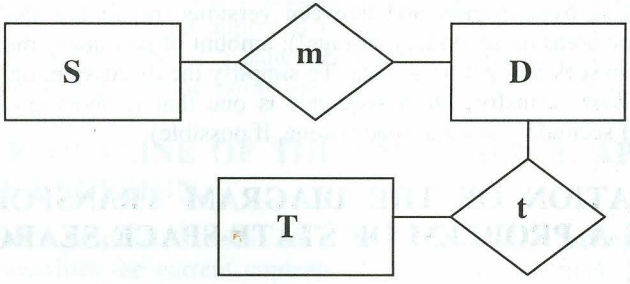

Figure 1: Old diagram

The new diagram (figure 2) is similar to the old one, but has one entity more (**P**, for *Presentations* of disciplines), and three new relationships (**ad**, for *advise*; **p**, for *is-presented-at*; **a**, for *attend*). One relationship has disappeared (**m**), because, according with information given by the user, it became redundant with the information contained in the path **a-P-p**. Also, the meaning of some elements of the old diagram has changed slightly, but no name was changed. For instance, the entity *Disciplines* was representing in the old diagram also the notion of *Presentation*. There are no redundancies, since the set of students a teacher advises may be different from the set of students he or she teaches.

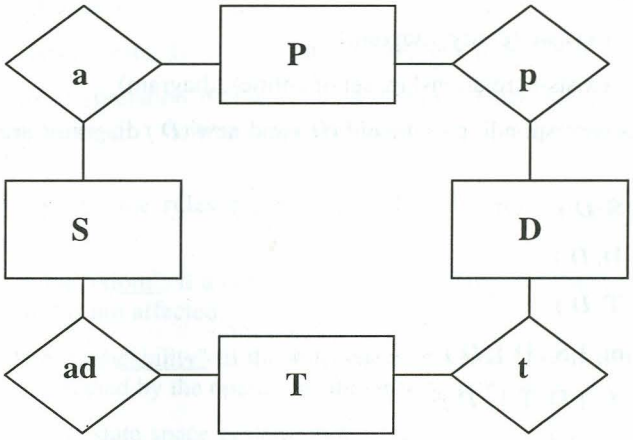

Figure 2: New diagram

The minimal alternative set of diagram transformation operations for this example is composed by the following operations:

(a) create-e (**P**, d)

(b) create-r (**a** , { **S** , **P** } , d)

(c) create-r (**p** , { **P** , **D** } , d)

(d) create-r (**ad** , { **S** , **T** } , d)

(e) exclude-r (**m** , { **S** , **D** } , d)

where "d" above corresponds to the E-R diagram received as argument by the operation.

If the set of transformation operations was another (like, for example, the set proposed by Chen in [3]), then the identification of such a minimal set would have been much more difficult. To complicate things, there would exist several alternative classes of transformation operations, each one of them with more than one minimal set.

A *state* is an E-R diagram instance. Each state may be described by a set of atomic, variable-free, formulas, written in the logic language, that are true in it. The set of atomic formulas should show all the state's relevant information, in such a way that two states are the same if and only if they are described by exactly the same set of atomic formulas. In

this work's example, the alphabet of the logic language should contain the following set of predicate symbols:

(e) exists-e (entity, diagram)

(f) exists-r (relationship, set of entities, diagram)

The states corresponding to the old (***D***) and new (***D***') diagrams are described below. For ***D*** :

exists-e (**S**, ***D***)

exists-e (**D**, ***D***)

exists-e (**T**, ***D***)

exists-r (**m**, { **S** , **D** }, ***D***)

exists-r (**t** , { **D**, **T** } , ***D***)

For ***D*** ' :

exists-e (**S**, ***D***')

exists-e (**D**, ***D***')

exists-e (**T**, ***D***')

exists-e (**P**, ***D***')

exists-r (**a**, { **S** , **P** } , ***D***')

exists-r (**p**, { **P** , **D** } , ***D***')

exists-r (**t**, { **D** , **T** } , ***D***')

exists-r (**ad**, { **T** , **S** } , ***D***')

The rules governing the construction of search paths in the state space are described in the following, and the logic language needs one more auxiliary predicative symbol, *belongs*, which tests pertinence of an element to a set:

(g) belongs (entity, set of entities)

In the sentences below, variables *e*, *r*, *d*, and *eset* correspond, respectively, to the sorts *entity*, *relationship*, *E-R diagram*, and *set of entities*. To simplify the expression of the sentences, universal quantifiers are not explicitly represented. Variables that seem to be free are actually universally quantified.

(r1) not exists-e (e, d) -> exists-e (e, create-e (e, d))

(r2) (not (exists-r (r, eset, d)) and (belongs (e, eset) -> exists-e (e, d))) ->
exists-r (d, eset, create-r (r, eset, d))

(r3) (exists-e (e, d) and (exists-r (r, eset, d) -> not (belongs (e, eset))) ->
not exists-e (e, exclude-e (e, d))

(r4) exists (r, eset, d) -> not exists (r, eset, exclude-r (r, eset, d))

The intended interpretation of the rules is complemented by a set of meta-rules, presented below (following a similar approach taken in the formalisation of logical specifications in [8]):

(m1) "only-if": the rules express the only conditions in which an existence assertion is valid;

(m2) "frame axiom": if a condition is not explicitly indicated as being affected by an operation, then it is not affected;

(m3) "non-applicability": if the antecedent of a rule fails, then the condition on the consequent is not affected by the operation (the operation has no effect).

The search on the state space is done along the paths, or sequences of instantiated transformation operations, that satisfy the rules and meta-rules above. One of those sequences is described by the following expression:

create-r(**p**,{**P**,**D**},create-r(**a**,{**S**,**P**}, create-e(**P**, exclude-r(**m**,{**S**,**D**}, create-r(**ad**,{**S**,**T**},***D***))))

It is possible to demonstrate that the set of instantiated diagram transformation operations has the property of, given any pair of non-equal correct diagrams, allow the construction of at least one transformations' sequence that generates one from the other. This is a very important property, because it guarantees that a data designer may propose a new diagram, no matter what is the old diagram, and it will always be possible to transform the old in the new, using operations instantiated from the given set.

A sequence of transformations is *valid* when all intermediate diagrams generated during the execution of the sequence are structurally correct. For instance, the creation of a relationship between entities that do not exist in the old diagram may not precede the creation of the entities that will be related, and the exclusion of an entity may not precede the exclusion of any relationship in which it plays a part.

A valid sequence of transformations is *good* when it satisfies certain given *precedence rules* between operations. Such precedence rules are defined considering, for instance, conditions of minimal use of auxiliary memory for the conversion of the old database in the new database.

As a consequence, there should exist a precedence order, on the sequence of operations, between operations that create a new entity or relationship which corresponds to old stored data structures from where data will be extracted in the new stored data structures.

Such *data conversion precedence relation* should be specified previously, from information given by the system's user. To characterize which are the good sequences, it is

not necessary to know in advance how the database conversion will be done, but only what operation should come before or after another in the sequence.

An *optimal* sequence is a good sequence that satisfies other database conversion performance criteria, like: minimal quantity of physical accesses to secondary memory, smallest amount of bytes moved from one data structure to another, and so on.

The sequence

create-r(**p**,{**P**,**D**},create-r(**a**,{**S**,**P**}, create-e(**P**, exclude-r(**m**,{**S**,**D**}, create-r(**ad**,{**S**,**T**},***D***))))

is a valid sequence, but is not a good one, because the exclusion of the relationship **m** before the inclusion of the relationships **a** and **p**, and of the entity **P**, may cause the temporary storage of the contents of the data structure corresponding to **m** for the subsequent creation of the data structures corresponding to **a**, **p** and **P**.

To characterize good sequences of transformation operations for the current set of operations it is necessary to consider additional *precedence rules*. The only precedence rule for the example being presented appears below.

(p1) exists-r(**m**,{**S**,**D**}, d) -> (exists-r(**a**,{**S**,**P**}, d) and exists-r(**p**,{**P**,**D**},d) and exists-e(**P**,d))

This rule reduces to only ten the quantity of good transformation sequences for the example. They are all those valid sequences where the exclusion of **m** happens after the creation of **a**, **p**, and **P**. One of them is shown below.

exclude-r(**m**,{**S**,**D**}, create-r(**p**,{**P**,**D**},create-r(**a**,{**S**,**P**}, create-e(**P**, create-r(**ad**,{**S**,**T**},***D***))))

Any good sequence may be used to indicate the ordering which the database conversion procedures, corresponding to each transformation operation in the sequence, will be executed. Some additional optimization criteria, like the ones mentioned above, may be applied now. The study of those criteria is outside the scope of this work.

4 CONCLUSIONS, FUTURE WORK.

This work has presented an approach for the systematic, and maybe automatic, treatment for the problem of database redesign. The approach is based on the use of E-R diagrams for the conceptual representation of data structures, and on the existence of a set of diagram transformation operations, like the ones proposed by [3] and [1]. The set should have the property that, given any pair of correct E-R diagrams, it should be possible to establish at least one sequence of instances of the diagram transformation operations that could transform the first diagram into the second one. To each transformation operation there should also correspond a convenient database conversion procedure, whose execution

could create, from the previous database contents a new database, conforming to the new diagram.

The approach uses state-space search methods, and proposes a two step procedure for the identification of sequences of diagram (and database) transformations that satisfy database conversion performance with minimal computational effort and maximal use of available stored data.

The first step looks for sets of instantiated diagram transformation operations that may create a new diagram from an old one. The second step uses state-space search procedures to identify, for all sets found in the first step, what are the *good* diagram transformation operation sequences that satisfy conditions of minimal computational effort and maximal old stored data utilization for the database conversion task.

Some topics mentioned in this work need more study, before one can really think in turning the process automatic. The first one is the choice of the set of more adequate diagram transformation operations. The operations should be *natural* (from the point of view of the user), and the set should be complete (it should be possible to build at least one *good* sequence of diagram transformations that change the old diagram into the new one, the old and new diagrams being any pair of correct diagrams). The operation sets proposed by [3] and by [1] seem to be good candidates to such a set of operations.

A second topic is the definition of the database conversion procedures corresponding to each instance of the diagram transformation operations. The definition of the several data conversion modules, and their integration in a single database conversion program, is not an easy task, considering the experiments already made. But the prognostics for the construction of a usable database redesign tool based on the ideas presented in this work seem to be good.

Solutions for plan formation problems rely on theorem proving techniques, that are normally computationally complex. The addition of semantic features, cardinalities, attributes, and integrity constraints definitions to our simple E-R model, and the treatment of big, realistic, diagrams, will require powerful optimization enhancements to our approach. A lot of research work still has to be done to achieve practical results.

REFERENCES

[1] Batini C, Ceri S, Navathe SB. Conceptual Database Design- an Entity-relationship Approach, Benjamin Cummings, Redwood City, CA, 1992.

[2] Casanova MA, Tucherman L, Laender AHF. On the design and maintenance of optimized relational representations of entity-relationship schemas. In: Data & Knowledge Engineering 11 (1993).

[3] Chen PPP. The entity-relationship model - a basis for the enterprise view of data. In: National Computer Conference, 1977, pp 77-84.

[4] Date CJ An Introduction to Database Systems, Addison Wesley, 1976.

[5] Enderton HB A Mathematical Introduction to Logic, Academic Press, New [Gazola92] Gazola LG A study and implementation of database-restructuring procedures, UFRGS, Bacharelado em Ciência da Computação, Trabalho de Conclusão, Porto Alegre, 1992 (in Portuguese).

[6] Kowalski R Logic for Problem Solving, North Holland, New York, 1979.

[7] Nilsson NJ Problem-solving Methods in Artificial Intelligence, McGraw-Hill, 1971.

[8] Veloso PAS, Castilho JMV de, Furtado AL. Systematic Derivation of Complementary Specifications. In: Proceedings of the 7th International Conference on Very Large Databases, Cannes, France, 81.

Artigo originalmente publicado em: 12th International Conference on Entity-Relationship Approach, 15-17 dezembro 93, Dallas, Texas, Proceedings, 1993.