An Entity-Relationship Approach to Schema Evolution

C. T. Liu, S. K. Chang and Panos K. Chrysanthis*
Department of Computer Science
University of Pittsburgh, Pittsburgh, PA 15260

Abstract

A dynamically evolving information system supports changes to its database schema in order to facilitate the needs of new application programs. This paper presents an approach to schema evolution through changes to the Entity-Relationship schema of a database. This approach has the advantage of being closer to the designer's perception of data, rather than to the logical database schema which describes how data are stored in the database. The underlying database structure is re-organized, if necessary, to accommodate new data without the changes affecting existing objects. In this way and through the construction of views, modifications of existing programs are avoided while all objects in the database are accessible to all application programs, both new and old. Thus, the invariance of the programs' interface is preserved by the views.

1 Introduction

A dynamically evolving information system must support changes to its database schema in order to facilitate the needs of new application programs. At the same time, it should be able to continue supporting existing application programs providing access to objects created under the previous or new database schemas. An information system stores objects in its database for a long time and, in general, it is neither easy nor practical to modify complex application programs such as those found in these types of systems [9]. Thus, schema changes should be transparent to the application programs.

Various approaches to the problem of changing schemas and maintaining consistency between instances created under different schemas have been proposed, particularly in the context of Object-Oriented databases [10, 12, 2, 13, 4]. This paper discusses another way to support *schema evolution* based on the Entity-Relationship (ER) approach for data modeling [3] and is graphic oriented.

In this ER-based approach, changes are specified on the *ER diagram* representing the ER database schema which is closer to the designer's perception of data, rather than being directly specified on the database schema which describes how data are stored in the database. Changes to the ER diagram are subsequently mapped into changes to the underlying database structure. The underlying database structure is re-organized, if necessary, to accommodate new data without the changes affecting existing objects. That is, existing objects retain their representation whereas new data are represented in *complementary* representations and related to existing objects. Each version of an object schema is expressed as a view to the underlying database and thus, the invariance of the programs' interface is preserved. In other words, through the construction of (database) *views*, modifications of existing programs are avoided while all objects in the database are made accessible to all application programs, both new and old. In addition to data independence, views are used in maintaining object consistency.

In some respects, our approach combines aspects of the object-oriented type evolution approaches of Skarra and Zdonik [10, 12] and it can be adopted in the same context via the use of the Extended ER model [11]. However, in the rest of this paper, we introduce our approach in the context of the basic ER model and an "objectified" relational database model in which database instances are associated with a system-generated, systemwide unique object identifier (Oid).

2 Evolution through the ER Diagram

The basic ER data model supports two semantic primitives, *entities* and *relationships* between entities, in terms of which the structure of a database, i.e., the database schema, is described. An *ER diagram* is a graphical representation of a ER schema consisting of rectangular boxes representing entity types and diamond-shaped boxes connected to rectangular boxes representing relationship types (see Figure 1). An *entity type* is a set of entities having the same properties or attributes. Similarly, a set of relationships having the same attributes forms a relationship type. The connections of a relationship type with entity types indicate how the entities in the entities types are involved in the relationship.

Attributes are used to describe an entity or a relationship. Each attribute is associated with a domain that defines the possible values for the attribute. Each entity usually has an *identifying* or *key* attribute whose value is distinct from any other entity in the same entity type. The instances of a relationship type can be identified by the identifiers of the entities involved in the relationship and are called *foreign attributes* to the relationship. In Figure 1 we show the attributes of

*This work is partially supported by the National Science Foundation under grant IRI-9210588.

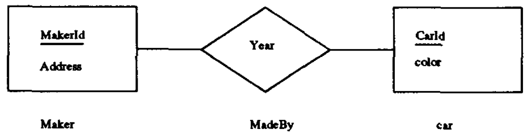

Figure 1: A partial ER diagram for modeling cars

entity or relationship types inside the *nodes* of the ER diagram, i.e., boxes and diamonds. The identifying attribute is underlined.

We will refer to an instance of an entity type or a relationship type as an *object*, and the description of an entity type or the relationship type as the *object schema*. Thus, an ER schema consists of a set of object schemas which provide the interface for application programs to create and access objects in the database.

In order to support schema evolution through changes to an ER diagram, we considered the following possibilities: (1) A node and its associated connections (edges) may be added to an ER diagram corresponding to the introduction of a new object schema or a new relationship. (2) A node and its associated edges may be removed from an ER diagram corresponding to the dropping of an existing object schema or relationship. (3) Edges may be added to or removed from an ER diagram reflecting changes in the relationships among the existing entity types in the database. (4) A node may be split into several nodes and a number of nodes may be merged into a single node as a result of aggregation and specialization of entity types respectively. (5) Attributes may be added to or drop from nodes corresponding to the gain and lost of attributes because of new properties of the objects and requirements of the application programs. For example, attributes are added to or removed from nodes when edges are added or removed in an ER diagram. Finally, (6) The domain of attributes may be changed, widen or reduce over time.

A close examination of the above possibilities shows that in the last three cases, Cases 4 to 6, the changes may involve existing attributes. For example, when a node is split into two nodes, the attributes of the newly created nodes may be duplicated or derived from the attributes of the original node. In general, the dependencies between attributes are classified into four types: *shared, independent, derived* and *dependent* [4]. In the above example, a duplicated attribute is an instance of shared dependency. A dependent attribute cannot be derived from other attributes, but its value is affected by the modifications to the value of other attributes. An independent attribute can neither be derived from other attributes nor is its value affected by changes to the values of other attributes. To facilitate the specification of these attributes, and in particular, the derived and dependent ones, we extended the ER diagram (and consequently the ER model) so that one can express the *relationships between attributes* of different versions of the same object schema in the form of functions. As we will see in the next sections, these functions are used in the construction of views

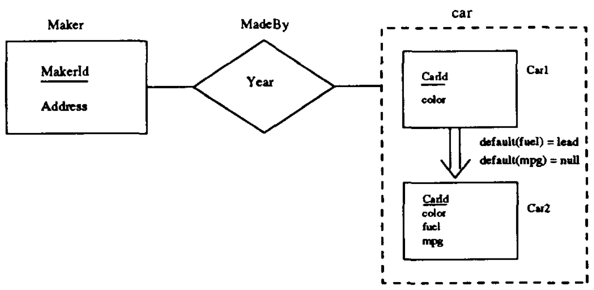

Figure 2: An Evolved ER diagram for modeling cars

as well as the database re-organization.

Furthermore, our approach does not really remove a node, an attribute or an edge from an ER diagram but treats them as defunct constructs and requires from the designer to specify, again in the form of a function, the way that the missing values of the affected attributes are generated.

Functions in an ER diagram are denoted by annotated directed double edges between nodes where the annotation captures the functions and the direction expresses the derivation relationship between nodes. Nodes representing different versions of an object schema are specified by being enclosed within a dotted box and all relationship edges being redirected to the enclosing dotted box. For instance in Figure 2 which captures the evolution of the ER diagram in Figure 1, the object schema *car* is evolved by adding the attributes *fuel* whose value can be *leaded* or *unleaded*, and *mpg* (mileage rating) whose value is an integer. The edge associated with the *default(fuel)* and *default(mpg)* functions specifies that the version *car2* is derived from the version *car1* (the initial object schema). The two functions state that the missing value of the attributes *fuel* and *mpg* in objects created under the object schema *car1* are *leaded* and *null* respectively.

3 Object Representation

3.1 The Underlying Database

The schema of the underlying database, here assumed to be relational, is derived from the ER diagram by using a mapping algorithm. The relational database is "objectified" so that it can effectively support this mapping as well as the construction and use of database views representing the different versions of object schemas. That is, we assume that each object, i.e. instance of entity or relationship type, is associated with a systemwide unique and immutable identifier (Oid) not visible to application programs.

The mapping between an ER diagram and the corresponding relational database schema is described by a *version derivation graph (VDG)*. Changes to an ER diagram are first mapped onto the VDG and then into relations in the underlying database (See Figure 3).

A version derivation graph consists of a set of nodes, N, and a set of directed edges, A. A node represents a version of an object schema and an edge represents

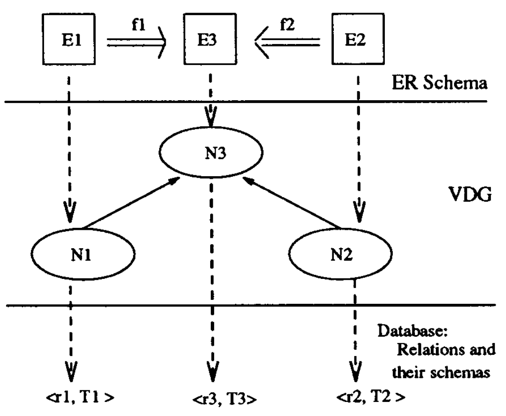

Figure 3: Three Level Schema Mapping

the derivation relationship between nodes. For example, the VDG in Figure 3 captures the derivation of a new schema represented by node N_3 from two other schemas represented by nodes N_1 and N_2.

There are two kinds of nodes in VDG: *virtual* and *non-virtual* nodes. A non-virtual node corresponds to an object schema which is either the initial one or is derived from existing object schemas and enhanced with new attributes. A virtual node corresponds to a derived object schema from existing ones without additional attributes. Formally, if E_1, E_2 and E_3 are the sets of attributes of object schemas corresponding to nodes N_1, N_2 and N_3 of Figure 3 respectively, then N_3 is a virtual node if $E_3 \subseteq E_1 \cup E_2$.

Non-virtual nodes are associated with a relation in the underlying database. This is because objects created under a schema that maps onto a non-virtual node cannot be stored in the relations associated with the nodes from which the object schema is derived. On the other hand, the objects created from schemas mapped onto virtual nodes can be completely stored in the relations associated with the nodes from which is derived.

Let us illustrate the derivation of the relation schema T_i associated with a node N_i through an example. Assume that nodes N_1 and N_2 in Figure 3 are non-virtual nodes corresponding to object schemas in the initial ER diagram.

Since N_1 and N_2 are non-virtual nodes, they are associated with a relation in the underlying database. Being an initial object schema, E_1 maps to a relation r_1 whose schema T_1 contains all attributes of E_1 plus one extra attribute, the Oid. Similarly, E_2 maps to a relation r_2 whose schema T_2 contains all attributes of E_2 plus an Oid. The mapping algorithm used in the case of initial object schemas is similar to the algorithms proposed by [11, 8] modified to generate and include object identifiers into the relation schemas.

As mentioned above, the mapping of E_3 into a relation depends on whether E_3 maps into a virtual or non-virtual node:

```
if $E_3 \subseteq E_1 \cup E_2$
    then $N_3$ is a *virtual* node,
    else  $N_3$ is a *non-virtual* node.
```

If N_3 is a virtual node, it shares all its attributes with either N_1 or N_2. Since both N_1 and N_2 are associated with relations, N_3 does not require additional space in the database to represent the attributes of E_3 and therefore relation r_3 is not created in the database. Instead, relation r_3 (or equivalently object schema E_3) is represented as a view on relations r_1 and r_2.

If N_3 is a non-virtual node, then it has attributes that it does not share with either N_1 or N_2 and hence it does require additional space to represent them. For this reason, the database is re-organized by creating a new relation r_3 with schema T_3 to represent the additional attributes:

$$r_3 : T_3 = (E_3 - \{E_1 \cup E_2\}) \cup Oid$$

That is, T_3 consists of an Oid and the attributes belonging to the set difference between attributes of E_3 and the union of attributes of E_1 and E_2. In other words, T_3 contains all the attributes of E_3 shared with neither E_1 nor E_2 and an Oid. Thus, E_3 is represented as a view on r_1, r_2 and r_3 and in one sense, r_3 *complements* the representations of r_1 and r_2 for objects created under the object schema E_3.

To summarize, the initial ER diagram is translated into a VDG consisting of non-virtual nodes representing the initial object schemas and associated with their corresponding relations. The view of initial object schemas is the corresponding relation. When the ER schema evolves, the changes to ER schema are mapped into a set of new VDG nodes connected to existing VDG nodes according to the derivation edges in the ER diagram, the corresponding relations are then created, if necessary, and finally the views of the different versions of object schemas are reconstructed.

3.2 Object Schemas as Database Views

Each object schema, irrespective of whether it maps into a virtual or non-virtual node in the VDG graph, is expressed as a view of the relations in the database. The basic idea underlying the construction of views for each version of an object schema is the notion of a *complete object schema* S_c which is the union of all attributes of the different versions. All objects created under any version can be expanded into complete objects as if they had been created under the complete schema. In this way, the view associated with a version of an object schema E_i is defined as a selection of a subset of complete objects based on the constraints associated with the particular version followed by a projection over the attributes E_i. For example, the constraint for selecting objects under the object schema *car1* is *fuel = leaded.*

The conversion of an object from a version E_i to the complete S_c is achieved through the *Expand()* operation which generates the values of those attributes in S_c that are not attributes in E_i by using the functions specified at the time of the evolution of the ER diagram, e.g., *default(fuel)* and *default(mpg)* in Figure 2 and stored in the corresponding VDG node. Each version of an object schema is associated with an *Expand()* operation.

There are five steps in constructing views representing versions of object schemas involving the traversal of the VDG which stores all the necessary information associated with each version. Let us illustrate these five steps by constructing the views for E_1, E_2 and E_3 in the above example (Figure 3). We assume that E_3 maps into a non-virtual node and hence it is associated with a relation r_3.

Step 1: The complete schema is constructed by building the union of attributes of E_1, E_2 and E_3: $S_c = E_1 \cup E_2 \cup E_3$.

Step 2: For each version E_i, we identify the relations involved in storing objects created under E_i. For version E_1 the relation is r_1. For E_2 is r_2. For E_3 all three relations r_1, r_2 and r_3.

Step 3: We derive the selection condition for the set of objects O_i^* created under each version. Objects created under version E_3 O_3^* are selected by joining r_1, r_2 and r_3 on Oid. Since r_2 is used to store the objects created under E_2 and part of objects created under E_3, objects created under version E_2 O_2^* are selected by discarding the objects created under version E_3 from relation r_2. Similarly, objects created under E_1 are selected by removing the objects created under version E_3 from relation r_1.

Step 4: Once we have identified the objects created under a version, we can expand each object to a complete object. Here, f_1 is used to expand objects in O_1^* and f_2 to expand objects in O_2^*. Objects in O_3^* are complete objects and do not require any conversion.

Step 5: Let r_c be the entire set of complete objects generated in step 4. Each view $View_i$ is defined uniformly as follow: $View_i = \Pi_{E_i}(\sigma_{C_i}(r_c))$,
where Π stands for projection, σ stands for selection and C_i specifies the constraints on E_i.

Each view is stored in the corresponding VDG node and it may need to be reconstructed after each database re-organization.

Because of the uniqueness of the Oids, there is always a one-to-one correspondence of views to the relations in the underlying database. Thus, there is no problem of mapping updates on a view to updates on relations in the database involving a join. However, there is a problem with updates in the case of derived attributes in which there is no inverse function [1, 5, 6]. In this case, the designer is expected to specify the manner in which updates on derived attributes should be performed in the form of functions.

4 Conclusion

In this paper, we presented an approach to schema evolution through changes to the ER diagram representing the schema of a database. In order to facilitate changes to the ER schema we enhanced the basic constructs of ER diagrams with constructs that specify versions of entity and relationship types, and relationships between attributes in different versions. We described a method for mapping changes to ER diagram into relations in the underlying database and for constructing database views expressing the various schema versions. Through views schema changes are made transparent to the application programs while all objects in the database are accessible to all programs.

This ER approach to schema evolution has the advantages of being closer to the designer's perception of data as well as graphic oriented and hence easier to use.

References

[1] F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. *ACM Trans. on Database Systems*, 6(4), 1981.

[2] J. Banerjee, W. Kim, H. Kim, and H.F. Korth. Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In *Proc. of ACM SIGMOD*, 1987.

[3] P. Chen. The Entity Relationship Model - Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1), March 1976.

[4] S. M. Clamen. Type Evolution and Instance Adaptation. Technical report, School of Computer Science, C.M.U., June 1992.

[5] S. S Cosmadakis and C. H. Papadimitriou. Updates of Relational Views. *J. ACM*, 31(4), Oct. 1984.

[6] A. M. Keller. *Updating Relational Databases Through Views*. PhD thesis, Department of Computer Science, Stanford University, 1985.

[7] W. Kim, E. Bertino, and J. F. Garza. Composite Objects Revisted. In *Proc. of ACM SIGMOD*, 1989.

[8] V. M. Markowitz and A. Shoshani. On the Correctness of Representing Extended Entity-Relationship Structures in the Relational Model. In *Proceedings of ACM SIGMOD*, 1989.

[9] M. E. Segal and O. Frieder. On-the-Fly Program Modification: Systems for Dynamic Updating. *IEEE Software*, March 1993.

[10] H. A. Skarra and S. B. Zdonik. Type Evolution in an Object-Oriented Database. In *Research in Object-Oriented Databases*. Addison-Wesley, 1987.

[11] T.J. Teorey, D. Yang, and J.P Fry. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Survey*, 18(2), June. 1986.

[12] S. B. Zdonik. Object-Oriented Type Evolution. In *Advances in Database Programming Languages*. Addison-Wesley, 1990.

[13] R. Zicari. A Framework for Schema Updates In an Onject-Oriented Database System. In *Proc. of Conference on Data Engineering*, 1991.