An Architecture for Managing Database Evolution*

Eladio Domínguez, Jorge Lloret, and María Antonia Zapata

Dpt. de Informática e Ingeniería de Sistemas, Facultad de Ciencias
Universidad de Zaragoza. E-50009 – Zaragoza, Spain
{noesis,jlloret,mazapata}@posta.unizar.es

Abstract. This paper presents an architecture for managing database evolution when all the components of the database (conceptual schema, logical schema and extension) are available. The strategy of evolution in which our architecture is based is that of 'forward database maintenance', that is, changes are applied to the conceptual schema and propagated automatically down to the logical schema and to the extension. In order to put into practice this strategy, each component of a database is seen under this architecture as the information base of an information system. Furthermore, a translation information system is considered in order to manage the translation of conceptual elements into logical schema elements. A current Oracle implementation of this architecture is also presented.

Keywords: Information Systems, Database Evolution, Forward Database Maintenance, Meta-modelling.

1 Introduction

The requirements of a database do not remain constant during its life time and therefore the database has to evolve in order to fulfil the new requirements. In general, database evolution activities are considered of great practical importance since they normally consume a large amount of resources [10]. As a consequence, much research has been focused on analysing ways of facilitating this type of activity [1,16].

Several problems related with databases evolution have been outlined in [6]. In particular, we are interested in the forward database maintenance problem ('redesign problem' according to [16]), that is, how to reflect in the logical schema and in the extension changes that have occurred in the conceptual schema of a database. Although a lot of research papers have been written in relation with this problem (see, for example, [16] and [10]) no completely satisfactory solution has been proposed.

* This work has been partially supported by DGES, projects TIC2000-1368-C03-01 and PB-96-0098-C04-01, and by University of Zaragoza, project UZ-00-TEC-04.

A. Olivé et al. (Eds.): ER 2002 Ws, LNCS 2784, pp. 63–74, 2003.

As a contribution towards achieving a more satisfactory solution, in this paper we propose an architecture for managing database evolution within the context of forward engineering. The main difference with respect to other proposals is that we consider a translation component besides the conceptual, logical and extension components. The translation component stores information about the way in which a concrete conceptual database schema is translated into a logical schema. This component plays an important role in enabling the automatic propagation of the conceptual database schema evolution down to the logical database schema making it possible to reflect in the extension of the database the changes performed in its conceptual schema.

Another important difference with respect to other authors [16,8] is that a meta-modelling approach [3,11] has been followed for the definition of the architecture. We have chosen this approach because it allows modelling knowledge to be represented and because it has been proven that it facilitates the definition of data model translations [11]. Within the architecture, three meta-models are considered which capture, respectively, the conceptual, logical and translation modelling knowledge. At the same time, the notion of information system, such as is defined in [5], is brought into play not only at the model level (which is the way in which it is normally used) but also at the meta-model level (like in [10]).

So as to show a concrete application of our architecture, we present a current Oracle implementation, which follows the common approach [16,15] of modelling the conceptual (logical) schema of the database by using the ER (relational) model. It should be noted that we have chosen these concrete models only with the aim of illustrating our architecture. However, the architecture is of general applicability and therefore can also be applied to other approaches such as, for example, within the context of object oriented databases [1,2]. Like other authors [10], we have chosen to represent the meta–schemas by means of UML class diagrams [13] in the belief that they will be easily understood thanks to the fact that UML is an increasingly widely accepted standard modelling language.

The remainder of the paper is organised as follows. Section 2 explains our view of dealing with database evolution, presenting in Section 3 the architecture we propose. Section 4 is devoted to showing a current Oracle implementation of our architecture. In Section 5 we discuss related work and finally conclusions and future work are outlined in Section 6.

2 The Database Evolution Issue

In order to design a database, users' information requirements are represented by means of a conceptual database schema S_C (for example, an ER model or a UML diagram). This schema is translated into a logical database schema S_L (see Figure 1) which will be implemented by means of a DBMS. The database is then populated to create a consistent database state σ.

Within this framework, the database evolution issue can be stated in a general way rephrasing the ideas explained in [16] as follows. Due to varied reasons (changes in the real world [16], optimisation procedures for improving the performance of the system [7]...), the conceptual schema S_C is modified generating

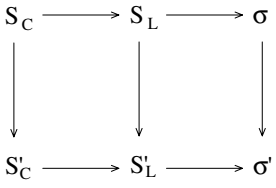

Fig. 1. Database Evolution

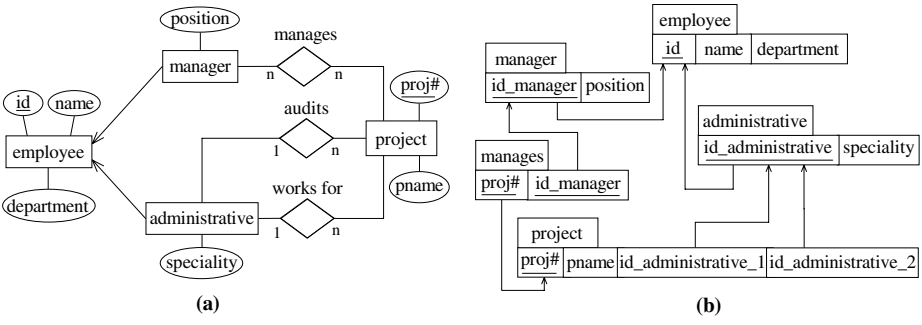

Fig. 2. Examples of Conceptual and Logical Database Schemas

a new conceptual schema S'_C. Ideally this modification at the conceptual level should be managed following the strategy of 'forward database maintenance' [6] according to which changes at the conceptual level should be automatically propagated down to the logical schema and its population. That is, the logical schema S_L has to be modified in order to generate a new logical representation S'_L and the database state σ has to be mapped into a new database state σ' consistent with S'_L.

The changes to be performed in the conceptual database schema in order to carry out the desired evolution can be expressed by means of schema transformations [7]. A schema transformation accomplishes modifications in the structure of the database and maps the population of the source schema into an allowable population of the resultant schema [2,7].

In order to illustrate an example of database evolution, we will use the schema of Figure 2(a) as the conceptual schema S_C, which has been obtained combining different examples included in [9]. In this example the E/R model has been used as the conceptual modelling technique and, as usual, entity types, relationship types and attributes are represented, respectively, by rectangles, diamonds and ovals. This schema represents a company where it is perceived that there are employees and projects. The employees can be managers (managing projects) or administrative staff (working for projects or auditing projects).

Traditionally the relational model has been used as the logical modelling technique when the E/R model is used at the conceptual level. Following this criteria, the proposed example of E/R schema has been translated into a relational model (see Figure 2(b)) using the algorithm proposed in [4]. In Figure 2(b) the foreign key constraints have been represented by means of arrows. With re-

gard to this example we only want to emphasise that the n–n relationship type manages has been translated into a relational table which is not the case with the 1–n relationship types works_for and audits.

In the course of this paper we are going to consider two examples of evolution of the conceptual schema of Figure 2(a). One is the case in which the audits relationship type, together with its instances, must be deleted. This example seems to be very simple but we will see later on that it is more complex than it appears. Moreover, it will serve to illustrate the suitability of the translation component we include in our proposed architecture. As for the other example, we are going to suppose that the attribute department of the entity type employee must be transformed into an entity type. This transformation (1) adds to the conceptual schema an entity type department described by means of two attributes (id_department and department), id_department being its primary key, (2) adds a relationship type employee has department and, (3) deletes the attribute department. With respect to the extension, this transformation maps each distinct non–null value of the attribute department in the old schema into a distinct 'department' entity in the new schema. Furthermore, the corresponding 'employee has department' relationships are added.

3 Database Evolution Architecture

The architecture we propose aims at providing a general framework which makes it feasible to manage database evolution following a forward maintenance strategy. Therefore, the architecture has to be defined in such a way that the changes performed in the conceptual database schema can be reflected in the logical schema and its extension. It is more or less obvious that some component has to allow conceptual, logical and extensional information to be stored. Let us go on to illustrate, by means of an example, the necessity of also storing knowledge with regard to the translation process from the conceptual into the logical schema.

Let us suppose that the audits relationship type must be deleted from the conceptual schema of Figure 2(a). This modification must be automatically reflected in the logical schema of Figure 2(b) deleting some element. According to the translation algorithm applied to the conceptual schema, it is known that the audits relationship type has been translated into an attribute of the project table, this attribute being a foreign key referencing the administrative table. The problem is that this table contains two columns (id_administrative_1 and id_administrative_2) verifying these conditions. If there is no information about the specific process followed for obtaining the logical schema of Figure 2(b), it is not known which attribute should be deleted. Our proposal is to store knowledge about the translation process explicitly in a component of our architecture, in the same way that the conceptual, logical and extensional information is stored. This component will include, for example, information about the column which the relationship type audits has been translated into.

A meta-modelling approach has been followed for the definition of the components storing conceptual, translation and logical information (a modelling ap-

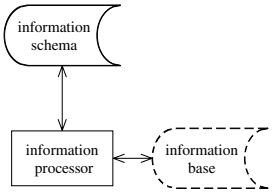

Fig. 3. Components of an Information System

proach is used with regard to the extensional information). The meta-modelling approach consists of representing modelling knowledge by means of a meta-model, where a *meta-model* is a conceptual schema of the elements constituting a data model or technique [3]. Following this approach the elements of a conceptual database schema, logical database schema or translation process are seen as instances of the corresponding meta-model. In order to capture this fact, [10] inspired us with the idea of bringing into play the notion of information system, such as is defined in [5], not only at the model level (which is the way in which it is normally used) but also at the meta-model level.

According to [5] an *information system* (see Figure 3) consists of three components: an information schema, an information base and an information processor. The information schema[1] defines all the knowledge relevant to the system, the information base describes the specific objects perceived in the Universe of Discourse, and the information processor receives messages reporting the occurrence of events in the environment. In order to respond to the events received, the information processor can send structural events towards the information base and/or towards the information schema and can generate internal events that inform other information systems of the changes performed in it.

The notion of information system is used within our architecture giving rise to four information systems which are used to store, respectively, the conceptual modelling knowledge, the translation process, the logical modelling knowledge and the extension. The corresponding components of each one of these information systems as well as the way in which they are related appear in Figure 4. The name of each one of these components has been modified in an attempt to capture the type of knowledge that they store (in any case the graphical symbol that surrounds each component stands for the type of component it represents).

It must be noted that three different abstraction levels are involved in the architecture. On the one hand, the information schemas of the three former information systems are situated at the most abstract level (meta-model layer according to [13]) and, on the other hand, the information base which stores the population of the database is situated at the least abstract level (user data layer [13]). All the other elements are situated at the model layer [13]. Let us now explain each one of the four information systems of the architecture.

[1] In fact, in [5] this component is called 'conceptual schema'. However, with the aim of avoiding misunderstandings, we have considered it inappropriate to use the term 'conceptual' since we are going to use this component not only at the conceptual level of the database but also at the logical and physical levels.

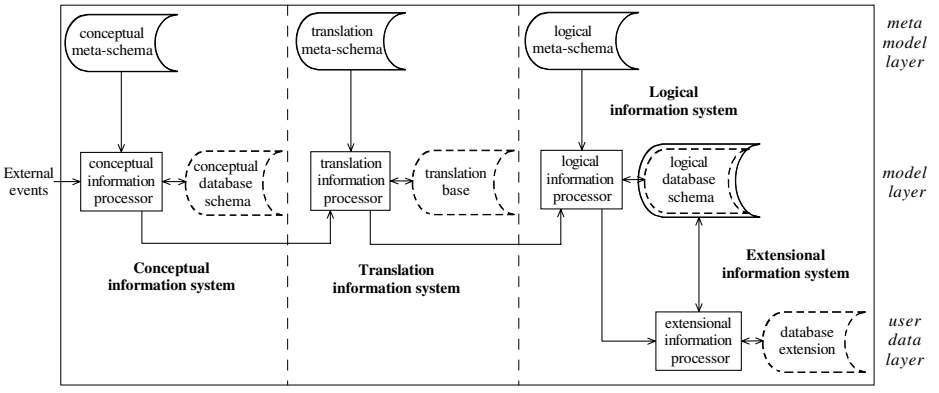

Fig. 4. Architecture for Database Evolution

3.1 Conceptual Information System

The population of this information system (that is, its information base) represents the constituent elements of a given conceptual database schema. For example, with respect to the E/R schema of Figure 2(a), facts such as that employee is an entity type and that department is an attribute have to be stored. These facts will be stored following the structure established in the conceptual meta–schema.

The conceptual information system has to react to external events received from the environment. The type of events in which we are interested are those related with database evolution. Each event issued from the environment is handled by the information processor which checks its validity according to the restrictions imposed by the conceptual meta–schema. If it is valid, the information processor induces a collection of structural events necessary to change the conceptual database schema (that is, the information base) according to the semantics of the received event. The conceptual information processor also generates a collection of internal events that inform the translation information system of the changes performed in the conceptual database schema.

3.2 Translation Information System

The goal of this information system is to store all the information necessary to enable any change performed in the conceptual database schema to be automatically reflected in the logical schema. This goal is achieved by storing the way in which conceptual schema elements are translated into logical schema elements. In order to do this, the translation accomplished by the chosen translation algorithm is specified as a set of elementary translations each of which represents the translation of only one conceptual element into a logical element. For example, the translation of the E/R schema of Figure 2(a) following the algorithm proposed in [4] performs, among others, the elementary translations of transforming the entity type employee into a table with the name employee, and transforming the attribute department into a column with the same name. The elementary

translations are stored in the translation base (that is, the information base) specifying the type of translation, the element of the conceptual schema it is applied to and the element of the logical schema it gives place to.

The translation information system has to react to the internal events generated by the conceptual processor which inform it about the changes performed in the conceptual database schema. In accordance with these events, the information processor determines the elementary translations that must be added, deleted or updated from the translation base. After these changes, the translation base contains the set of elementary translations that translates the new conceptual schema (resulting from the evolution process) into a logical one. The information processor also generates a collection of internal events that inform the logical information system of the changes performed in the translation base.

It must be noted that the new set of elementary translations is determined without it being necessary to apply once again the translation algorithm from scratch. The knowledge stored in the translation information system avoids having to recalculate the logical elements that result from the conceptual elements that have not been modified. The idea of using an information system to store the elements related with the translation process and the way in which this knowledge is used during the database evolution process are, from our point of view, a significant contribution of our work.

3.3 Logical and Extensional Information Systems

The logical information base stores the elements of the logical schema obtained as a translation of a given conceptual database schema. For example, with respect to the relational schema of Figure 2(b), facts such as that employee is a table and that department is a column of it have to be stored. These facts will be stored following the structure established in the logical meta–schema.

The logical information processor receives a collection of internal events from the translation information processor, according to which it generates the structural events necessary to change the logical database schema in order to reflect the evolution performed at the conceptual level.

In Figure 4, the information base of the logical information system is surrounded with both the information base and information schema symbols. This is because the logical database schema can be seen as the information base of the logical information system (as we have explained) or as the information schema of the extensional information system. For this reason two different components of our architecture store the same information. This fact obliges us to define some rules, called *correspondence rules* (in the same sense as in [10]). These rules govern the correspondence between the elements of each one of the two components. In order to hold these rules, the logical information processor sends the internal events reporting the changes made in the logical database schema, and the extensional information processor, according to the received events, changes the database extension and the database schema (that is, the information schema). This is the only case in which the information processor changes the information schema of its information system.

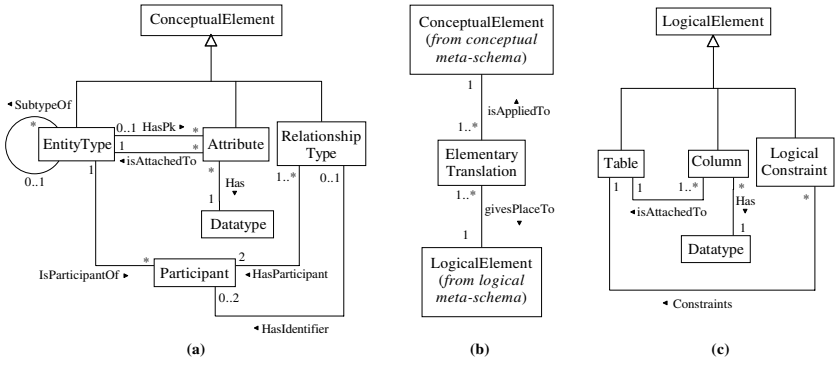

Fig. 5. Meta-models of the Information Systems

4 An Implementation of Our Architecture

In this section, we describe a concrete implementation of our architecture. This implementation is based on the RDBMS Oracle 8i Release 8.1.7 and the programming language PL/SQL Release 8.1.7 [17]. Within this implementation, the E/R technique has been chosen as the conceptual modelling technique and the relational as the logical modelling technique. We will use UML meta–schemas in order to conceptually describe the structure of the three information schemas that belong to the meta-model level. The graphical representation of these meta–schemas appears in Figure 5 in which only the name is included for each class (its attribute compartment does not appear).

The meta–schema of the **conceptual information system** (see Figure 5(a)) conceptualises the different modelling elements of the E/R model (the version we use is based on the E/R model proposed in [9]). For example, it is represented that a relationship type is related with exactly two participants and that an entity type is related with the attributes that conform to its primary key (PK). This meta–schema also has several meta–constraints associated to it. For example, two constraints (expressed by means of OCL [13]) with context EntityType are the following:

1. If an entity type is not a subtype of any entity type, then it must have one primary key.
   `SubtypeOf → size = 0` **implies** `HasPK → size ≥ 1`
2. An entity type cannot be a subtype of itself.
   `SubtypeOf → forAll( e | e <> self )`

In order to carry out database evolution tasks the external event types that we have implemented appear in Table 1 (their arguments have been omitted). Six of these external events are basic operations of addition or deletion of modelling elements (entity type, attribute or relationship type), two of them allow the modification of a primary key, one adds an entity type as a specialisation of another entity type and, finally, one transforms an attribute into an entity type.

Table 1. External Event Types

For entity types	For attributes	For relationship types	For primary keys
NewEntityType	NewAttribute	NewRelType	AddAttrToPk
DropEntityType	DropAttribute	DropRelType	DropAttrFromPk
NewEntitySubtype	AttributeToEntityType		

We are aware that this is a relatively small number of external events, and that more are needed for facilitating the database evolution.

As an illustration of database evolution, the example of schema evolution described at the end of Section 2, according to which the attribute department is transformed into an entity type, will be performed using the external event:

AttributeToEntityType('employee.department')

This event has only one parameter which expresses the attribute that has to be transformed into an entity type. As a consequence of this event the following tasks have to be performed: (1) create a new entity type called department with two attributes (id_department and department) and primary key id_department, (2) create a new relationship type, called has, between employee and department, and (3) delete the attribute department of the entity type employee.

The information processor of the conceptual information system has been implemented as a set of PL/SQL procedures, one for each of the established external event types. For example, there exists a procedure that is executed when the event AttributeToEntityType occurs and this procedure accomplishes the tasks associated with the event.

The meta–schema of Figure 5(b) represents that the **translation information system** stores the elementary translations that have to be applied to the given conceptual database schema in order to be translated into a logical schema (the elementary translations are determined following the translation algorithm proposed in [4]). Each translation is related with the conceptual element to which it is applied and with the logical element to which it gives rise.

The information processor of this information system is implemented as a collection of PL/SQL triggers which are fired by the insert, delete or update operations performed in the conceptual database. For example, the addition of the entity type department fires a trigger which adds to the translation base an elementary translation that translates the entity type department into a table. The deletion of the attribute department fires a trigger which deletes the elementary translation that translates the attribute department into a column.

The meta–schema of the **logical information system** (see Figure 5(c)) conceptualises the different elements that conform to a relational model. The information processor of this information system is also implemented as a collection of PL/SQL triggers which are fired by the insert, delete or update operations performed in the translation base. For example, the addition of the elementary translation that translates the entity type department into a table fires a trigger which adds the table department. In the same way, the deletion of the elementary

Table 2. SQL sentences automatically generated and executed

```
1  CREATE TABLE department(department varchar2(30), id_department
   integer);
2  INSERT INTO department (department) SELECT DISTINCT department
   FROM employee WHERE department IS NOT NULL;
3  execute giveidvalues('department','id_department','department');
4  ALTER TABLE department ADD (PRIMARY KEY (id_department));
5  ALTER TABLE employee ADD id_department integer;
6  ALTER TABLE employee ADD (CONSTRAINT restr22 FOREIGN KEY
   (id_department) REFERENCES department(id_department));
7  execute matchvalues('employee','id_department','department',
   'department','id_department','department');
8  ALTER TABLE employee DROP COLUMN department;
```

translation that translates the attribute department into a column fires a trigger which deletes the column department.

The **extensional information system** stores the Oracle 8i database schema and its data. The information processor of this information system is also implemented as a collection of PL/SQL triggers which are fired by the insert, delete or update operations performed in the logical database. These triggers automatically generate and execute the SQL sentences that perform the changes that have to be made in the Oracle 8i database in order to accomplish the correspondence rules and to reach a consistent database state. For example, the SQL sentences generated in order to transform the attribute department into an entity type appear in Table 2. These sentences perform the following tasks:

1. Create the new table department with the values of attribute id_department created by means of the procedure giveidvalues (lines 1–4).
2. Create the relational structures corresponding to the new relationship type between entity types employee and department (lines 5–6).
3. Assign values to attribute employee.id_department using the procedure matchvalues (line 7).
4. Drop attribute employee.department (line 8).

5 Related Work

Database evolution has been widely discussed in the literature and therefore very varied approaches have been proposed. The evolution of object-oriented databases and relational databases, including the propagation of changes automatically down to the extension of the database, has received great attention and the research results have been included in prototypes or in commercial DBMS (see, for example [1]). However they lack the consideration of a conceptual level which allows the designer to work at a higher level of abstraction [10].

In [6] an abstract framework which takes into account both conceptual and logical levels is presented and the necessity of automatically propagating down

(forward strategy) the changes performed at the conceptual level is stated. The different papers dealing with forward engineering mainly differ in the way they address the propagation of the conceptual changes down to the logical schema and to the extension. For example, a taxonomical approach is followed in [15], which proposes a taxonomy of changes for ER structures and the impact of these changes on relational schemes is analysed. However this paper does not study how to reflect the schema evolution in the extension of the database.

Other approaches, more similar to ours, propose various ways to capture knowledge about the mappings performed to obtain the logical schema of a conceptual schema. This information is used subsequently in order to obtain the new logical schema associated to the changed conceptual schema. In [8], for example, the sequence (called history) of mappings performed in order to obtain the logical schema is stored. In this way the mappings affected by the changes can be detected and modified, whereas the rest can be reexecuted without any modification. Our approach has the same aim as this one but differs in that we follow a meta-modelling approach.

A meta-modelling approach is also proposed in [10], [14] and [12]. In the case of [10] only a conceptual meta-model is considered whereas we also make use of a logical and a translation meta-model. With respect to [14], the authors make use of a meta-modelling approach with a different goal since the paper deals with the definition of a query language for evolving information systems. In [12] a generalisation of the traditional information system notion similar to ours has been proposed. However, some differences with respect to our proposal are worth noting. Firstly, in [12] not only data modelling is taken into account (as we do) but also process and behaviour specification. Secondly, in [12] only the conceptual level is under consideration so that the proposed architecture includes only one information system. Finally, the information processor of an information system is concerned with the modification of the structure and also of the population, instead of using different information processors for each one of these processes as we propose.

6 Conclusions

In this paper we have presented an architecture for managing database evolution with a forward strategy. The architecture consists of four information systems whose information schema capture the relevant modelling elements. As the main contribution, a translation information system is considered, which reflects the translations performed between the conceptual and logical schemas of the database. Evolution changes performed in the conceptual database schema are reflected in the logical schema and the extension of the database making use of structural and internal events. An implementation of our architecture using Oracle has also been presented.

As a direction of future work, the problems related with the evolution of integrity constraints have to be analysed. Furthermore, a comprehensive support within our architecture for relationship evolution, following the ideas of [2], is a goal for further development.

References

1. L. Al-Jadir, M. Léonard, Multiobjects to Ease Schema Evolution in an OODBMS, in T. W. Ling, S. Ram, M. L. Lee (eds.), *Conceptual modeling, ER-98*, LNCS 1507, Springer, 1998, 316–333.
2. K. T. Claypool, E. A. Rundensteiner, and G. T. Heineman, ROVER: A Framework for the Evolution of Relationships, in A. H. F. Laender, S. W. Liddle, V. C. Storey (eds), *Conceptual modeling, ER-2000*, LNCS 1920, Springer, 2000, 409–422.
3. E. Domínguez, M. A. Zapata, J. J. Rubio, A Conceptual Approach to Metamodelling, in A. Olivé, J. A. Pastor (Eds.), *Advanced Information Systems Engineering, CAISE'97*, LNCS 1250, Springer, 1997, 319–332.
4. R.A. Elmasri, S.B. Navathe, *Fundamentals of Database Systems (3rd ed.)*, Addison-Wesley, 2000.
5. J.J. van Griethuysen (ed.), *Concepts and Terminology for the Conceptual Schema and the Information Base*, Publ. ISO/TC97/SV5-N695, Mars 1982.
6. J.L. Hainaut, V. Englebert, J. Henrard, J.M. Hick, D. Roland, Database Evolution: The DB-MAIN approach, in P. Loucopoulos (ed.), *Entity-Relationship approach-ER'94*, Springer Verlag, LNCS 881, 1994, 112–131.
7. T.A. Halpin, H.A. Proper, Database Schema Transformation and Optimization, in M. P. Papazoglou (ed.), *Object–Oriented Entity-Relationship Modelling Conference- ER'95*, Springer Verlag, LNCS 1021, 1995, 191–203.
8. J.M. Hick , J.L. Hainaut, V. Englebert, D. Roland et al., Strategies pour l'evolution des applications de bases de donnes relationelles: L'approche DB-MAIN, *Proceedings XVIII Congres Inforsid*, La Garde, France, 1999.
9. A.H.F. Laender, M.A. Casanova, A.P. de Carvalho, L. F.G.G.M. Ridolfi, An Analysis of SQL Integrity Constraints from an Entity-Relationship Perspective, *Information Systems*, 10, 4, 1994, 331–358.
10. J.R. López, A. Olivé, A Framework for the Evolution of Temporal Conceptual Schemas of Information Systems, in B. Wangler, L. Bergman (eds.), *Advanced Information Systems Eng., CAiSE 2000*, Springer, LNCS 1789, 2000, 369–386.
11. C. Nicolle, D. Benslimane, K. Yetongnon, Multi–Data Models Translations in Interoperable Information Systems, in J. Mylopoulos, Y. Vassiliou (Eds.), *Advanced Information Systems Eng., CAISE'96*, LNCS 1080, Springer, 1996, 1–21.
12. J.L.H. Oei, H.A. Proper, E.D. Falkenberg, Evolving Information Systems: Meeting the ever–changing environment, *Information Systems Journal*, 4, 3, 1994, 213–233.
13. OMG, *UML specification version 1.4*, formal/01–09–67, 2001, `http://www.omg.org`
14. H.A. Proper, Th. P. van der Weide, Information Disclosure in Evolving Information Systems: Taking a Shot at a Moving Target, *Data & Knowledge Engineering*, 15, 1995, 135–168.
15. J.F. Roddick, N.G. Craske, T.J. Richards, A Taxonomy for Schema Versioning Based on the Relational and Entity Relationship Models, in R. A. Elmasri, V. Kouramajian, B. Thalheim (eds.), *Proc. of the 12th Int. Conf. on the Entity-Relationship Approach*, Elsevier, LNCS 823, 1994, 137–148.
16. A.S. da Silva, A.H.F. Laender, M.A. Casanova, An Approach to Maintaining Optimized Relational Representations of Entity-Relationship Schemas, in B. Thalheim (ed.), *Conceptual Modeling- ER'96*, Springer Verlag, LNCS 1157, 1996, 292–308.
17. S. Urman, *Oracle 9i PL/SQL Programming*, Osborne, 2002.