Elementary translations: the seesaws for achieving traceability between database schemata *

Eladio Domínguez[1], Jorge Lloret[1], Ángel L. Rubio[2], and María A. Zapata[1]

[1] Dpto. de Informática e Ingeniería de Sistemas.
Facultad de Ciencias. Edificio de Matemáticas.
Universidad de Zaragoza. 50009 Zaragoza. Spain.
ccia@posta.unizar.es
[2] Dpto. de Matemáticas y Computación. Edificio Vives.
Universidad de La Rioja. 26004 Logroño. Spain.
arubio@dmc.unirioja.es

Abstract. There exist several recent approaches that leverages the use of model transformations during software development. The existence of different kinds of models, at different levels of abstraction, involves the necessity of transferring knowledge from one model to another. This framework can also be applied in the context of metadata management for database evolution, in which transformations are needed both to translate schemata from one level to another and to modify existing schemata. In this paper we introduce the notions of translation rule and elementary translation which are used within a forward database maintenance strategy.

1 Introduction

Several recent research efforts focus on automating the generation and management of transformations between models or schemata representing the same system at different levels of abstraction [3, 6, 13, 16, 22]. For example, the importance of transformation management has been recently acknowledged by the UML community with the advent of the Model Driven Architecture (MDA) [16] in which transformations between platform independent models (PIM) and platform specific models (PSM) must be managed. The transformation modeling framework proposed in [3] within the 'model management' approach [2] is another example. In general, the management of model transformations is recognized as a necessary goal for achieving the ambitious goal of automating system implementation. This issue is particularly difficult when models or schemata evolve over time and consistency between them must be kept.

* This work has been partially supported by DGES, project TIC2002-01626, by Ibercaja-University of Zaragoza, project IB 2002-TEC-03, by the Government of La Rioja, project ACPI2002/06, by the Government of Aragon and by the European Social Fund

In the database field, the evolution issue is related to the existence of changes within the different phases of the life cycle, from early design stages to exploitation and maintenance. In particular, a recent paper highlighted the existence of "a lack of support (methods and tools) in the database maintenance and evolution domain" [13]. Among the several problems related to evolution activities (see [11]), one of the most important is the 'forward database maintenance problem' (or 'redesign problem', according to [20]). This problem is how to reflect in the logical and extensional schemata the changes that have occurred in the conceptual schema of a database. One way of tackling this problem is to ensure the traceability of the translation process between levels. But there is no agreement about which artifacts and mechanisms are needed for assuring traceability [19, 22].

As a contribution towards achieving a satisfactory solution to this problem, in this paper we propose a metadata approach to ensure the traceability of the translation between the conceptual and logical levels. Some ideas about our approach are presented in [7], and its relationship with a model–driven framework is in [5]. The way in which we propose to achieve traceability is making use of a specific translation component in which information related with the translation process is stored by means of elementary translations. When an evolution process is carried out in a conceptual model, it is propagated to the logical model applying a propagation algorithm which makes use of the information stored in the translation component.

The remainder of the paper is organized as follows. Section 2 is devoted to the presentation of both an outline of our proposed architecture and an example that we use throughout the paper. In Section 3 we present concepts we use for creating the component which ensures traceability, whereas in Section 4 we detail how the elementary translations of this component are used in the evolution process. The paper is completed with some related work and conclusions.

2 Preliminaries

2.1 Evolution Architecture Overview

Many current research efforts aim at contributing to the solution of the several problems related to database evolution activities [4, 13, 10, 15]. As a contribution towards achieving a satisfactory solution to one of these problems (that of 'forward database maintenance problem'), some of the authors of the present paper have presented in [7] an architecture and a prototype tool for managing database evolution whose graphical representation is shown in Figure 1. The way of working of this architecture is as follows: the conceptual component reacts to database evolution related external events and changes the conceptual database schema according to the semantics of the received event. The information of the fact that these changes have taken place is propagated through the rest of the components and they are appropriately changed in order to reflect the changes at the conceptual level.

It is worth noting the two main characteristics of this approach that make it different from other database evolution proposals. On the one hand, it includes an explicit translation component that stores information about the way in which a concrete conceptual database schema is translated into a logical schema. This component ensures the traceability of the translation process. On the other hand, a metamodeling approach [8] has been followed for the definition of the architecture. Within this architecture, three meta–models are considered which capture, respectively, the conceptual, logical and translation modeling knowledge.

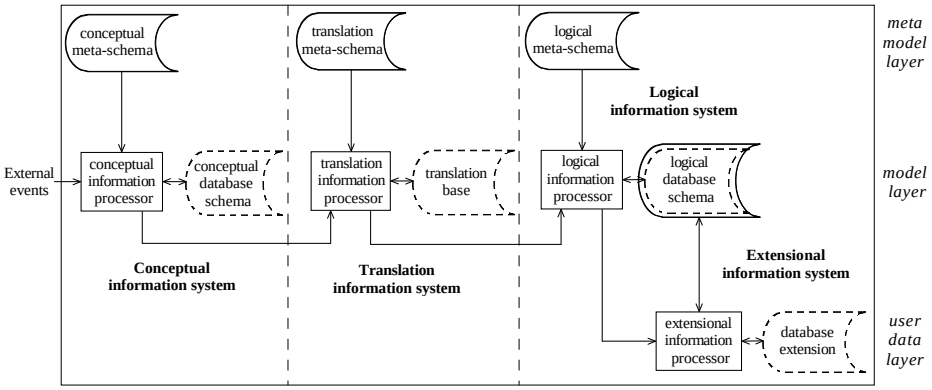

Fig. 1. Architecture for Database Evolution

2.2 Running example

The architecture described in [7] and outlined in the previous subsection has been developed with the property of being independent of any particular modeling technique, by means of the adoption of a metamodeling approach. However, in order to clarify the way the architecture works we will be considering a particular example, and for this reason some concrete techniques must be established. For this example, we have chosen the most common techniques in the context of the database field: the E/R technique as the conceptual modeling technique and the relational model as the logical modeling technique.

The conceptual evolution example we are going to use during the paper is the following. The initial conceptual schema consists of an isa hierachy `H` with supertype `E` and subtype `S`. The entity type `E` has two attributes, `e1` and `e2`, and the primary key is `e1`. We assume the existence, within our architecture, of the external event type 'change of primary key in the root of an ISA hierarchy'. This event is issued against the initial conceptual schema, resulting in a modified conceptual schema where the attribute `e2` is the new primary key of `E`. A comprehensive, detailed study about this event and its processing within the architecture is included in [6]. This archetypal example has been chosen with the aim of covering two aspects that, to our knowledge, have not been fully explored in the evolution context. These aspects (that are being incorporated into our current implementation) are the evolution of ISA relationships and the evolution of integrity constraints.

3 Creation of elementary translations

Traditional database engineering processes include, among others, the transformation of conceptual schemata into logical schemata. Usually this transformation process consists of applying a set of transformation rules to a conceptual schema in order to obtain a logical one. In the literature, there is a multitude of sources where sets of transformation rules are presented (see [9, 18]).

However, the way in which the transformation rules are described needs to be enriched when we want to use them in a forward database evolution context. In this context several problems arise. Some of these problems are technical matters (related, for instance, to the decisions that must be taken when naming conflicts occur); others involve the essence of evolution issues. As an example of this last case, let us consider a relationship type A with cardinality 1–n that has been translated using a particular rule. Suppose that as a consequence of a change in the domain, the cardinality of A must be changed to n–n. This means that now a new transformation rule must be applied. However, if there is no metadata information about the transformation rule that was originally applied to relationship type A, the new transformation can not benefit from the original one and the complete translation process must be redone from scratch.

Within our architecture, we have enriched the notion of transformation rule giving rise to a notion we call *translation rule* (see Section 3.1). These translations rules are used for building the *translation algorithm* (see Section 3.2). When this algorithm is applied to a conceptual schema, it produces not only a logical one (as in the traditional transformation process) but also a set of *elementary translations* stored in the translation base. An elementary translation is the smallest piece of information reflecting the correspondence between the conceptual elements and the logical ones.

3.1 Translation rules

As architect designers, we specify the set of translation rules which can be applied to each kind of conceptual building block. Basically, a translation rule defines how the conceptual elements of an instance of a conceptual building block (of the conceptual schema) are translated into logical elements of instances of logical building blocks (of the logical schema).

Table 1 shows a sketch of the syntax of a translation rule. A translation rule determines the procedures that have to be applied to instances of the specified conceptual building block B and each procedure calculates the value of the different logical elements (table name, column name, column type, constraint...). In order to determine the procedure to be applied, exactly one of the conditions from 1 to m must be *true*. For instance, a particular translation rule applicable to relationship types must establish the procedure for determining the name of the table into which the relationship type is buried. The conditions serve to decide which procedure is selected according to the cardinality of the relationship.

Table 1. Sketch of a translation rule

<rule_name> (b instance of conceptual building block <B>)
  when <condition 1> then
    $a_1 \leftarrow procedure_{11}$;
    $a_2 \leftarrow procedure_{12}$;
    ...
    $a_n \leftarrow procedure_{1n}$;
  endwhen
  ⋮
  when <condition m> then
    $a_1 \leftarrow procedure_{m1}$;
    $a_2 \leftarrow procedure_{m2}$;
    ...
    $a_n \leftarrow procedure_{mn}$;
  endwhen
endrule

3.2 Translation algorithm

This algorithm takes as input conceptual building block instances of the conceptual schema and creates 1) the elementary translations that relate the conceptual elements and the logical ones and 2) the logical elements of the logical schema. For the modeling techniques we have chosen, the conceptual building blocks are entity type, relationship type and ISA hierarchy, and the only logical building block is table.

In general, for each conceptual building block there are different translation rules that can be applied. In our approach, the translation rule to be applied is chosen by the user. This process can be modeled by the following algorithm sketch:

```
For each set C of instances of building block B
 For each instance b of C
   (a) election=get_building_block_translation_rule(B)
   (b) apply_elementary_translation_creation_procedure(b, election)
   (c) apply_logical_element_creation_procedure(b, election)
```

In the first iteration of the outer loop of this sketch, `C` represents the set of entity types not involved in any ISA hierarchy. In the second iteration, `C` represents the set of ISA hierarchies and, in the third, `C` represents the set of relationship types. As a consequence, each building block instance is translated by exactly one translation rule: the entity types that belong to an ISA hierarchy are translated by the translation rule for the ISA hierarchy chosen by the DBA and the rest of building blocks instances are translated by the corresponding translation rules chosen by the DBA.

At the time of this writing, this algorithm is completely automated for the chosen metamodels and we think it is easily extensible to other metamodels like UML [17] and object–relational [21].

Let us describe the steps of the algorithm.

Step (a). The user selects the translation rule to be applied to the corresponding building block instance of the initial conceptual schema. The metadata stored in the conceptual information base of the architecture representing the initial conceptual schema example are shown in Figure 2(a). Let us suppose that the designer has chosen to apply a translation rule called *isa1_tr* to the ISA hierarchy number 7 in Figure 2(a). This rule is based on Elmasri's rule on chapter 7 of [9] and transforms each entity type of the hierarchy into a table.

Step (b). Taking into account the translation rule selected in step (a), the appropriate elementary translations which model the transformation process of each building block instance are stored in the translation base. To achieve this, each translation rule has an elementary translation creation procedure associated to it. For example, we have a translation rule for entity types (namely *entity_type1_tr* rule) and a translation rule for ISA hierarchies (namely *isa1_tr* rule) whose elementary translation creation procedures are shown below. In these procedures several auxiliary functions are used (shown in Table 2) and the created elementary translations can be of different types (some of them are shown in Table 3).

Creation procedure for the translation rule entity_type1_tr

1. For the entity type E add to the translation base $new\_elementary\_translation(\text{`}ETT01\text{'}, get\_id(E), new\_id(E), get\_id(E))$
2. For each attribute $a_i, i = 1..n$, add to the translation base $new\_elementary\_translation(\text{`}ETT20\text{'}, get\_id(a_i), new\_id(a_i), get\_id(E))$
3. For the primary key pkE of E add to the translation base $new\_elementary\_translation(\text{`}ETT60\text{'}, get\_id(pkE), new\_id(pkE), get\_id(E))$

Creation procedure for the translation rule isa1_tr

1. For the supertype E of the ISA hierarchy H, apply the creation procedure for the translation rule *entity_type1_tr*, but *get_id(H)* is used as the final parameter instead of *get_id(E)*
2. For each subtype $S_i, i = 1..m$ of the ISA hierarchy H, add to the translation base $new\_elementary\_translation(\text{`}ETT01\text{'}, get\_id(S_i), new\_id(S_i), get\_id(H))$
3. For each attribute b_{ki} of $S_i, k = 1..r_i, i = 1..m$, add to the translation base $new\_elementary\_translation(\text{`}ETT20\text{'}, get\_id(b_{ki}), new\_id(b_{ki}), get\_id(H))$
4. For each attribute $a_i, i = 1..j$, of the primary key of E and each subtype $S_i, i = 1..m$, add to the translation base $new\_elementary\_translation(\text{`}ETT22\text{'}, get\_id(a_i), new\_id(a_i), get\_id(H))$
5. For the primary key of the entity type E and for each subtype $S_i, i = 1..m$, add to the translation base $new\_elementary\_translation(\text{`}ETT62\text{'}, get\_id(pkE), new\_id(pkE), get\_id(H))$
6. For each isa constraint with source S_i and target $E, i = 1..m$, add to the translation base $new\_elementary\_translation(\text{`}ETT65\text{'}, get\_id(isa_i), new\_id(isa_i), get\_id(H))$

(a) Conceptual database schema

entity_type

entity_type_id	name
1	E
2	S

attribute

attribute_id	name	datatype	entity_type_id
3	e1	integer	1
4	e2	integer	1

isa_hierarchy

isa_hier_id	supertype	subtypes
7	1	2

conceptual_constraint

conc_constr_id	type	name	source	target
5	pk	pk1	1	3
6	isa	isa1	2	1

(b) Translation base

elementary_translation

elem_transl_id	type	conceptual_element	logical_element	belongs_to
16	ETT20	E.e1	E.e1	7
17	ETT20	E.e2	E.e2	7
18	ETT01	E	E	7
19	ETT60	pk1	pk1	7
20	ETT22	E.e1	S.e1	7
21	ETT62	pk1	pk2	7
22	ETT01	S	S	7
23	ETT65	isa1	fk1	7

(c) Logical database schema

table

table_id	name
8	E
9	S

column

column_id	name	datatype	table_id
10	e1	integer	8
11	e2	integer	8
12	e1	integer	9

logical_constraint

logic_constr_id	type	name	source	target
13	pk	pk1	8	10
14	pk	pk2	9	12
15	fk	fk1	12	10

Fig. 2. Information bases

Table 2. Functions used in the translation algorithm

Function	Meaning
new_elementary_translation	creates a new elementary translation in the translation base
new_id	provides a unique identifier for a table, column or logical constraint
get_id	queries the conceptual base in order to find the identifier of an attribute, entity type, isa hierarchy or conceptual constraint

When we apply step (b) to the ISA hierarchy numbered 7 in Figure 2(a), the elementary translation creation procedure generates the elementary translations of Figure 2(b).

Step (c). The appropriate logical elements are stored in the logical database schema. We omit for space reasons details about this step. The metadata representing the logical elements that are obtained in step (c) for our example are shown in Figure 2(c).

4 The elementary translations act as seesaws

In this section we explain how the traceability achieved by means of the translation component is used to propagate the changes between the conceptual and the logical levels. The changes in the conceptual database schema are specified by means of external events. We show the *propagation subalgorithm for the translation information system* and how the changes made in the conceptual database schema are used by this subalgorithm in order to change the translation base appropriately. Then we show how the changes in the translation base are used

Table 3. Types of elementary translations

Type	Meaning
ETT01	translation of an entity type into a table
ETT20	translation of an entity type attribute into a column of an entity type table
ETT22	translation of an entity type primary key attribute into a column of a subtype table
ETT60	translation of a conceptual primary key into a logical primary key
ETT62	translation of a conceptual primary key into a primary key of a subtype table
ETT65	translation of an isa constraint into a foreign key constraint

by the *propagation subalgorithm for the logical information system*, which makes the appropriate changes to the logical database schema. Some functions used in these subalgorithms are shown in Table 4.

External events. When the DBA wants to change the database, (s)he issues an external event. An external event is composed of a set of conceptual modification primitives. For example, according to our running example, we suppose that the DBA wants to make the attribute `E.e2` the new primary key of entity type `E`. Then, (s)he issues the external event change_of_primary_key_in_ISA('E','e2').

We, as architect designers, have designed the external event type change_of_ primary_key_in_ISA so that it performs only one conceptual modification for evolution that in our running example is:

$$change\_conc\_constr\_target(\text{'}pk\text{'},\text{'}E\text{'},\text{'}E.e1\text{'},\text{'}E.e2\text{'}) \tag{1}$$

where the target of the primary key of the entity type `E` is modified so that the attribute `E.e2` is the new target.

For this external event type, only one conceptual modification is performed, but, in general, an external event type can perform more than one modification. For example, we, as architect designers, could have designed the external event type change_of_ primary_key_in_ISA so that it also retains an uniqueness constraint for the old attributes of the primary key. In this case, another conceptual modification for evolution, apart from (1), must be performed. In our particular example this modification is new_conceptual_constraint('uniq','E','E.e1'), which declares a new unique constraint on `E.e1`.

Propagation subalgorithm for the translation information system. This subalgorithm takes as input the conceptual changes produced by an external event and updates the translation base so as to reflect these changes. Its input is the set of changes (specified as a set of conceptual modifications) which has been made in the conceptual schema. The output is the added, deleted and affected elementary translations. An *affected elementary translation* is an elementary translation that must be modified or an elementary translation that is not modified but that involves a logical element which will have to undergo some change. Here we sketch this subalgorithm:

```
For each change in the conceptual database schema
  (a) Detect new elementary translations to be added to the
      translation base and add them.
```

Table 4. Functions used in the propagation algorithm

Function	Meaning
change_conc_constr_target	changes the target of a conceptual constraint
change_logic_constr_source_and_target	changes the source and the target of a logical constraint
change_logic_constr_target	changes the target of a logical constraint

(b) Detect those elementary translations that are affected by the conceptual change and modify them when needed.
(c) Detect elementary translations to be deleted from the translation base and delete them.

Let us show how this subalgorithm works for our external event example. The subalgorithm is applied to the conceptual modification (1) mentioned above. In step (a) for (1), since the new primary key attribute of the entity type `E` must be translated into a column of the primary key of the subtype table `S`, the following elementary translation is added to the translation base:

$$new\_elementary\_translation(\text{`}ETT22\text{'}, get\_id(\text{`}E.e2\text{'}), new\_id(\text{`}S.e2\text{'}), 7) \quad (2)$$

In step (b) the elementary translations involving the primary key of `E` and the ISA between `S` and `E` (translations numbers 19, 21 and 23 in Figure 2(b)) are identified as affected. In our example, the elementary translations 19 and 21 are affected because their logical element (the primary key constraint of E) will change their target. The elementary translation 23 is affected because its logical element(the foreign key `fk1`) will change its source and its target.

In step (c), the elementary translation that informs the attribute `e1` has been added to the table of `S` (translation 20) is deleted because the conceptual element `E.e1` is no longer an attribute of the primary key of the entity type `E`.

Here we explain the seesaw metaphor. A seesaw is an artifact by means of which the actions performed on the left side (for example, putting a weight on it) provoke a reaction on the right side (the right side goes up). In our framework, each elementary translation acts like a seesaw. The changes made in the conceptual part (the left side of the seesaw) provoke modifications in the logical part (the right side). This reaction is captured by the propagation subalgorithm for the logical information system, explained in next subsection, which updates the logical database schema.

Propagation subalgorithm for the logical information system. The information about the added, deleted and/or affected elementary translations is sent to the logical information system, which applies the propagation subalgorithm to the logical information system. We do not include this subalgorithm here and only show how it works in our example. Thus, for the new elementary translation added in (2), the subalgorithm adds to the logical information schema the following logical element:

$$new\_column(\text{`}e2\text{'}, \text{`}integer\text{'}, \text{`}S\text{'}) \quad (3)$$

For the affected elementary translations 19 and 21, their logical elements must change their target. For this reason, the subalgorithm performs the following changes in the logical database schema:

$$change\_logic\_constr\_target(\text{`}pk\text{'}, \text{`}E\text{'}, \text{`}E.e1\text{'}, \text{`}E.e2\text{'}) \tag{4}$$

$$change\_logic\_constr\_target(\text{`}pk\text{'}, \text{`}S\text{'}, \text{`}S.e1\text{'}, \text{`}S.e2\text{'}) \tag{5}$$

where the parameters are the type of constraint, the table on which the constraint is specified and the old and new targets.

For the affected elementary translation 23, its logical element must change its source and its target. For this reason, the subalgorithm performs the following change in the logical database schema:

$$change\_logic\_constr\_source\_and\_target(\text{`}fk\text{'}, \text{`}S\text{'}, \text{`}S.e1\text{'}, \text{`}S.e2\text{'}, \text{`}E.e1\text{'}, \text{`}E.e2\text{'}) \tag{6}$$

where the parameters are the type of constraint, the table on which the constraint is specified, the old and new sources and the old and new targets.

Finally, the deletion of the elementary translation 20 produces the following deletion in the logical information system:

$$delete\_column(\text{`}S.e1\text{'}) \tag{7}$$

By means of the two algorithms explained above, the translation base now reflects the correspondence between the new conceptual and logical schemas.

5 Related work

The necessity of capturing information about translations performed between models has been recognized in the literature and very varied approaches have been proposed. In the MDA approach [16], mapping techniques between PIM's or between PSM's are considered as association classes so that a mapping will be represented as an object (relating models) and not only as a link. With respect to PIM to PSM mappings, the MDA approach proposes the 'incremental consistency' as a desirable feature [14]. Within our architecture this property is achieved by means of the translation component.

The consideration of our architecture has lead us to propose in [5] several modifications in the MDA architecture. For example, it seems surprising to us that in the original MDA Metamodel Description there are explicit association classes to represent 'PSM Mapping Techniques' (from PSM to PSM) and 'PIM Mapping Techniques' (from PIM to PIM), but there are no analogous association classes to represent 'PIM to PSM Mapping Techniques' or 'PSM to PIM Refactoring Techniques'. According to the proposal we present in this paper, the class 'PIM to PSM Mapping Techniques' would also be included in the MDA architecture. In our opinion, the addition of this association class is suitable for ensuring the traceability needed for propagating the changes in a PIM model to its corresponding PSM model.

The innovative trend of model management [2, 3] also advocates the representation of mappings by means of objects. It is argued that this reification is often needed for satisfactory expressiveness [2]. This proposal aims to be generic in the sense that it can be applied to different kinds of models while our work is specific to database evolution issues.

Within the specific database evolution field, several papers deal with the evolution of object-oriented databases and relational databases [1] but, in general, they lack the consideration of a conceptual level which allows the designer to work at a higher level of abstraction [15]. A proposal that also considers the conceptual level is presented in [13]. In this case, the traceability is achieved storing all the sequence of operations (called history) performed during the translation of the conceptual schema into a logical one. In order to use this history for propagating the conceptual schema modifications, a process of history cleaning (eliminating redundant operations) must be performed [12]. The main difference between this approach and ours is the type of information stored for assuring traceability. Whereas in [13] the idea is to store the history of the process performed (probably with redundancies), in our case the goal of the elementary translations is to reflect the correspondence between the conceptual elements and the logical ones (in this case, there is no room for redundancies). Another difference is that we follow a meta–modelling approach. An in depth explanation of the use of meta–models within our proposed architecture appears in [7].

6 Conclusions and future work

The main contribution of this work is the presentation of a method for ensuring traceability in the context of database evolution. We explain how to use this traceability for propagating changes between database design levels. More specifically, we have explained how to create a component that reflects the correspondence between conceptual and logical database schemas. For this purpose, we have used elementary translations that reflect the relations between the conceptual elements and the logical ones and that facilitate evolution tasks.

There are several possible directions for future work. One is how to apply our ideas to other metadata management problems, in particular to the problems of schema integration or round–trip engineering [2]. Another direction is how to apply the architecture when more modeling constructs are defined in the conceptual and logical meta–schemata. For example, we could consider models with richer constructs, such as UML [17] and object-relational models [21].

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. P. A. Bernstein, Applying Model Management to Classical Meta Data Problems, *First Biennial Conference on Innovative Data Systems Research- CIDR 2003*, Online Proceedings, 2003.

3. K. T. Claypool, E. A. Rundensteiner, Gangam: A Transformation Modeling Framework. *International Conference on Database Systems for Advanced Applications-DASFAA 2003*, IEEE Computer Society, 2003, 47–54.
4. K. T. Claypool, E. A. Rundensteiner, G. T. Heineman, ROVER: flexible yet consistent evolution of relationships, *Data Knowl. Eng.* 39(1), 2001, 27–50.
5. E. Domínguez, J. Lloret, A. L. Rubio, M. A. Zapata, An MDA–Based Approach to Managing Database Evolution (position paper), in A. Rensink, (Editor), *Proceedings of MDAFA 2003. Model–Driven Architecture: Foundations and Aplications*, CTIT Technical Report Series, No. 03-27, 2003, 97–102.
6. E. Domínguez, J. Lloret, A. L. Rubio, M. A. Zapata, Evolving the implementation of ISA Relationships, submitted for publication.
7. E. Domínguez, J. Lloret, M. A. Zapata, An architecture for Managing Database Evolution, in A. Olivé et al. (eds) *Advanced conceptual modeling techniques- ER 2002 Workshops*, LNCS 2784 , 2002, 63–74.
8. E. Domínguez, M. A. Zapata, J. J. Rubio, A Conceptual Approach to Meta–Modelling, in A. Olivé, J. A. Pastor (Eds.), *Advanced Information Systems Eng.-CAiSE'97*, LNCS 1250, 1997, 319–332.
9. R. A. Elmasri, S. B. Navathe, *Fundamentals of Database Systems (4th ed.)*, Addison-Wesley, 2003.
10. F. Ferrandina, T. Meyer, R. Zicari, G. Ferran, J. Madec, Schema and Database Evolution in the O2 Object Database System, *Very Large Data Bases- VLDB'95*, 1995, 170–181.
11. 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*, LNCS 881, 1994, 112–131.
12. J.M. Hick, *Evolution of relational database applications: Methods and tools*, PhD Thesis, University of Namur, 2001 [in French].
13. J.M. Hick, J.L. Hainaut, Strategy for Database Application Evolution: The DB-MAIN Approach, in I.-Y. Song et al. (eds.) *Conceptual Modeling- ER 2003*, LNCS 2813, 2003, 291–306.
14. Anneke Kleppe, Jos Warmer, Wim Bast, *MDA explained. The Model Driven Architecture: Practice and Promise*, Addison–Wesley, 2003.
15. 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*, LNCS 1789, 2000, 369–386.
16. J. Miller, J. Mukerji (eds.), MDA Guide Version 1.0, Object Management Group, Document number omg/2003-05-01, May 1, 2003.
17. OMG, *UML Specification version 1.5 formal/2003-03-01*, available at http://www.omg.org, March, 2003.
18. H. A. Proper, Data Schema Design as a Schema Evolution Process. *Data Knowl. Eng* 22(2), 1997, 159-189
19. B. Ramesh, Factors influencing requirements traceability practice, *Communications of the ACM*, 41 (12), December 1998, 37-44.
20. 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*, LNCS 1157, 1996, 292–308.
21. M. Stonebraker, D. Moore, P. Brown, *Object Relational DBMSs: Tracking the next great wave (2nd ed.)*, Morgan Kaufmann Publishers, 1999.
22. W. M. N. Wan-Kadir, P. Loucopoulos, Relating evolving business rules to software design, *Journal of Systems Architecture*, article in press, 2003.