

A Framework for Database Evolution Management

Isabelle COMYN-WATTIAU* Jacky AKOKA** Nadira LAMMARI***

* *Laboratoire CEDRIC-CNAM et ESSEC, wattiau@cnam.fr*

** *Laboratoire CEDRIC-CNAM et INT, akoka@cnam.fr*

*** *Laboratoire CEDRIC-CNAM, 292 Rue Saint Martin, F-75141 Paris Cedex 03, lammari@cnam.fr*

Abstract

This paper addresses the crucial problem of changes and evolution in the context of database systems. We argue that the importance of change depends primarily on its nature. The nature of change impacts one or several phases of the database life-cycle. We propose a typology of changes based on three dimensions, namely the nature, the significance, and the time frame. These dimensions enable us to characterize the changes and to propose a set of evolution techniques facilitating the propagation of change in the database system. A framework for database evolution is presented. It allows us to take into account in particular unanticipated evolutions. More generally, the framework provides a first step in the development of a new generation of guidance tool exploiting the potential of evolution techniques such as reverse engineering, forward engineering, schema integration, and data integration. Our aim is to provide maintenance teams with realistic experiential database evolution environments. Finally, the implications for practice and for further research are discussed.

1 Introduction

The speed of adaptation of information systems, and more specifically of database systems, is generally considered as being determinant for enterprise competitiveness. Database systems enable data changes. The latter can be anticipated and controlled by triggers in relational databases and by methods in object-oriented databases. Data structure changes are performed by Data Base Management Systems (DBMS) commands. However, all changes are not allowed. Moreover, the DBMS does not perform code changes. As a consequence, database systems are not evolving and sufficiently adaptable. Database systems are built on the constraints of the past, in order to be used in the present. They generate constraints for the future. This is the main reason for the huge budgets dedicated to database maintenance, and more generally, to software maintenance. About 50% to 70% of companies software budgets are absorbed by the maintenance and evolution [7] and 80% of companies times and efforts are devoted to maintenance [27]. We claim that these costs are highly due to the lack of methodology guiding the designers in unanticipated database evolution and maintenance management. Our objective is to propose a framework allowing the maintenance engineer to characterize the unanticipated changes and to guide him/her by providing him/her with a list of techniques to be used in order to deal with the major types of change.

A database system is defined and constrained by its users, its owners, the other sub-systems that interact with it, and finally by the infrastructure on which it operates. Therefore each modification brought by these components or actors affects its internal composition. Our objective is to understand these evolutions and to propose adequate responses to these modifications within a framework. The latter constitutes a formal basis for database systems evolution. In this paper we address the problem of database systems evolution in order to provide some kind of flexibility leading to an effective management of database evolution.

Database design is one of the numerous activities in the development of an information system within an organization [2]. Our aim is to allow the maintenance engineers to take into account changing requirements at the three design steps: requirements definition, conceptual and logical design, and physical design. We define a set of techniques that enable schema evolution, whatever the moment when the change occurs. We also differentiate between minor and major changes depending on their impacts on the information systems.

The remainder of this paper is organized as follows. In the second section we describe the usual phases of the database design process. Section 3 is devoted to the description of our framework for database evolution. Section 4 presents related works. Finally, Section 5 concludes the paper and gives some research perspectives.

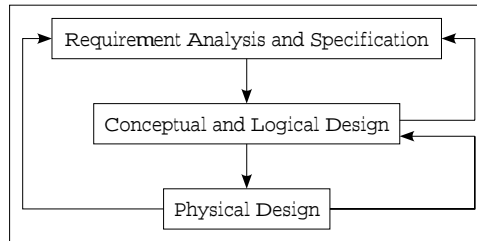


Figure 1: The database design process

2 The Database Design Process

The design of a database is a long and fastidious process. A consensus is now existing about the main three phases to be performed : requirement analysis and specification, conceptual and logical design, and physical design (Figure 1). This process is mainly iterative.

In the requirement analysis and specification phase, business domain experts select strategically relevant operational database attributes and specify the purpose to use them. During this phase, the users express their view of the universe of discourse in order to produce specifications of the application requirements which will be obtained through the future database. More precisely, two types of requirements are specified : data requirements and operational requirements. Data requirements impact the content of the database, and operational requirements are concerned with the utilization of the database by users and programs.

The second phase called conceptual and logical design or data modeling phase involves formulating the data objects resulting from the previous stage in terms of data modeling formalisms. The resulting logical schema is built on the data and the operational requirements. It actually contains all the information needed to meet the operational requirements. Moreover, depending on the model, it represents some business rules related to these data. If these constraints cannot be modeled by the schema, they will be embedded in programs. In relational databases this phase is split into two steps. The first step leads to a conceptual schema using Entity-Relationship (ER), Extended Entity-Relationship (EER) or Unified Modeling Language (UML) formalisms. The second step maps the conceptual schema to a relational schema.

The last phase deals with implementation issues, considering object structures, and inter-object links. The logical schema is transformed into a physical one by taking into account storage considerations and performance aspects imposed by the DBMS under consideration.

Looking back at the first two stages, one of the most important tasks is to uniquely identify the data semantics of the target applications and describe them using a data formalism. However the target applications are not fixed and the requirements are not always clear enough. Some divergence of the data semantics due to different viewpoints of the business policies may occur. Besides, it remains difficult to know if a logical schema meets completely business requirements. As a consequence, numerous unanticipated changes occur during the design process or even later when the system already exists.

By separating the process into these three phases, the designer is faced successively with different kinds of problems. The requirement analysis and specification phase aims at defining and delimiting the scope of the information system. The conceptual and logical phase is only concerned with the precise representation of the data structures and links. Finally, only the physical phase deals with performance constraints. This splitting mechanism has been widely experimented and was "proved" to be an efficient way to design database applications from scratch. We claim that this separation process can be applied to change and evolution management in database systems. As a consequence, the design process has to be inserted in an iterative cycle.

3 Unanticipated Change: A Characterization and Management Framework

As for any component of an information system, a database can be subject to unanticipated changes that can occur either during its design or during its exploitation. More precisely the evolution management must take into account the different dimensions of a change. We consider three main dimensions allowing us to characterize this change in order to take it into account by using appropriate evolution techniques :

- the nature of change,
- the change time frame,

- the significance of change.

Depending on these three dimensions, a set of techniques is recommended to handle unanticipated changes. In this section, we first describe the different characterizations of changes. Then we briefly define the different techniques used to take into account these changes. Finally we match the different techniques with the different types of changes.

3.1 The change dimensions

The *nature of change* defines the level concerned by the change. An evolution can deal with requirement specifications, or with conceptual and logical, or physical aspects. For example, adding a new concept is a specification change in the sense that it is due to new requirements. Of course it will have some consequences on the conceptual schema, and therefore on the logical and physical schemas. The data structures may be affected and performance considerations may have to be updated. However, we have to deal with this change at the specification level and to derive the different impacts on subsequent levels. As a second example, let us consider a business rule modification that may lead the designer to define the multiplicity a relationship should have. As a consequence, we can witness changes in the logical schema and even in the code. Either this change is considered as a conceptual change and translated into logical concepts, or it is expressed in terms of logical concepts and our framework will suggest to reverse engineer this change at a conceptual level. According to the nature of change, the evolution process has to deal with reverse and/or forward engineering techniques in order to keep equivalent descriptions of the system at the different abstraction levels. More generally, the forward engineering approach will anticipate such changes by using a trace mechanism.

The *change time frame* is another dimension used to characterize possible database evolutions. We distinguish between three main periods: the requirement analysis and specification period, the design and development period, and the maintenance period. Along this time frame, changes become more and more difficult to take into account due to running constraints. During the first two periods, the changes must be integrated into the different representations of the database, forward and backward. If the change occurs during the maintenance phase, it must be propagated through the system involving data structures and/or programs. However, it must also be integrated in the system documentation at the different abstraction levels. Let's consider two different examples:

- (a) a new domain is inserted into the system,
- (b) the original disk has to be removed and a new disk is installed.

The first change can easily be propagated as long as the information system is not running, even if it impacts heavily the different design levels. If the system is already under operation, a migration process must be conducted. The maintenance team must ensure that both the system is correct and its documentation is up-to-date. Change (b) should not impact neither the documentation nor the structure of the information system.

The *significance of change* is the third relevant dimension. Depending on the impact of the change on the information system, we define it as being minor or major. A major change will generate an important workload both on the documentation and on the database application. A minor change has a limited impact. It can be taken into account without defining a fastidious process. In this case, a trace mechanism is useful in order to define precisely the impact sphere of the change at each abstraction level. For instance, let's consider the following three types of change:

- (a) A new concept is added in the conceptual schema.
- (b) A new domain is inserted into the system.
- (c) The system must migrate from a DBMS platform to another one.

Even if it impacts all the abstraction levels, change (a) is a minor one. Change (b) is major. First the new domain must be conceptually represented. The second step consists in integrating this conceptual representation with the existing conceptual schema. Finally, change (c) leads to an important migration process. However, it has theoretically no impact on the conceptual level.

3.2 The techniques needed to manage evolutions

These sets of techniques are used either to ensure the correct documentation of the system or to propagate the changes in the software.

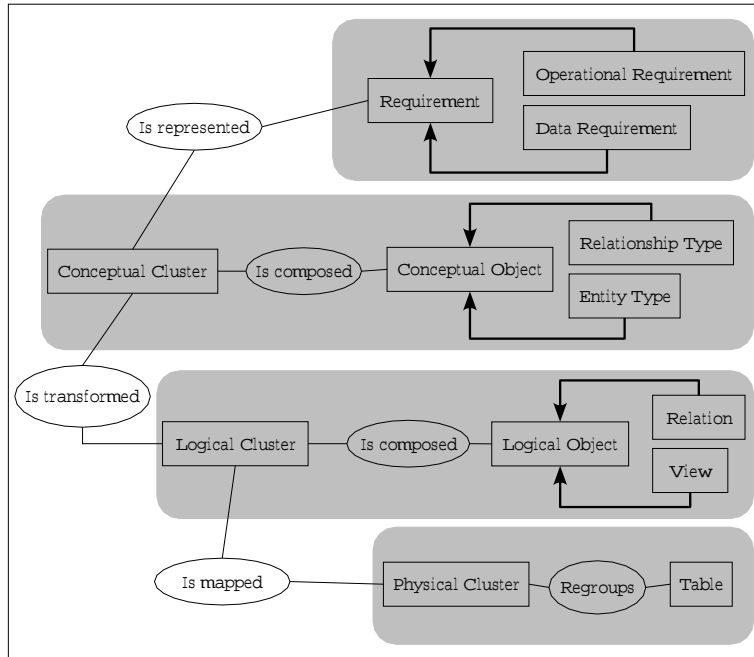


Figure 2: The chain between abstraction level components

Reverse engineering. It consists in providing the system with a high level description by reversing the normal design process. For instance, starting from a physical description of data, a reverse engineering process leads to a conceptual schema of these data [1]. In particular, reverse engineering encompasses program comprehension, structure comprehension, and meta-modeling techniques.

Forward engineering. It refers to the design process starting from a high level description of data and enabling the elicitation of an efficient physical organization of data. In the context of Unanticipated Software Evolution (USE), it contains a trace mechanism allowing the minor change propagation forward and backward. Moreover, the forward engineering mechanism must be based on an iterative cycle.

Schema integration. This technique consists in defining a global schema integrating several initial partially overlapping schemas. It involves the resolution of numerous problems, such as terminology conflicts or different user perceptions [6]. It can be used to confront the conceptual representation of an existing system with the abstraction of a new one (the one modeling the evolution) to be merged in this system.

Data integration. When a database is obtained from the merging of several existing systems, a data integration process is required in order to deal with data format conflicts, heterogeneity of representations, etc.

Change implementation. By change implementation, we designate the transformations of the system needed to take into account a minor modification of it. It is based on a mechanism allowing us to evaluate the impact sphere of this change on the objects, and to chain this sphere between abstraction levels. Figure 2 illustrates partially the model underlying this mechanism in the case of relational systems.

Migration process. Such a process encompasses the set of tasks enabling the transformation of a system when a major change is needed. In such a case, the impact sphere is so important that the whole system has to be reconsidered.

3.3 The database evolution framework

In our opinion, a typology of all the different unanticipated changes that can occur in an information system must be established and linked to possible impacts in the database representing this information system. This kind of guidance can help the maintainer of the database to trigger off an adaptive or perfective maintenance process.

The table below (Figure 3) proposes the appropriate techniques to be used in the case of a requirement, a conceptual and logical, or a physical change. The utilization of these techniques depends also on the change time frame, such as changes occurring at one of the design stages of the database or changes occurring at the maintenance phase of the database. For example, if we are faced with a major conceptual change, when a relational system is under design, a schema integration process may be conducted in order to obtain a global conceptual representation encompassing this change. Then, a

		NATURE OF CHANGE			
		CHANGE TIME FRAME	<i>Requirement Analysis and Design Change</i>	<i>Conceptual and Logical Design Change</i>	<i>Physical Design Change</i>
SIGNIFICANCE OF CHANGE	Minor Change	<i>Requirement Analysis and Specification Phase</i>	Regular Iteration	Regular Iteration	Not relevant
		<i>Design and Development Phase</i>	Forward Engineering	Forward Engineering Reverse Engineering	Change Implementation
		<i>Maintenance and Evolution Phase</i>	Forward Engineering Change Implementation	Forward Engineering Reverse Engineering Change Implementation	Reverse Engineering Change Implementation
	Major Change	<i>Requirement Analysis and Specification Phase</i>	Regular Iteration	Schema Integration	Not relevant
		<i>Design and Development Phase</i>	Schema Integration Forward Engineering	Schema Integration Forward Engineering	Reverse Engineering
		<i>Maintenance and Evolution Phase</i>	Schema Integration Data Integration Forward Engineering Migration Process	Schema Integration Data Integration Forward Engineering Migration Process Reverse Engineering	Reverse Engineering Migration Process

Figure 3: Techniques to be used for each change type

forward engineering technique transforms this conceptual schema into a logical schema of a database and finally into a physical description. If the system is under maintenance, a migration process must follow.

As illustrated at Figure 4, our framework aims at guiding the designer and/or the maintenance engineer to face different types of change. Depending on its nature, the change impacts on one or several phases of the database life-cycle. Different evolution techniques have to be used depending on the three dimensions of this change. These techniques will ensure the correct propagation of the change on the whole database life-cycle. The database evolution framework is sketched in Figure 4. This framework encompasses all the dimensions of changes. There are five major components in the framework. Since this framework represents the way database changes can be taken into account, it is important to highlight the nature of changes. Clearly, the nature of changes influences the choice of the adequate database evolution techniques and impacts the database life-cycle. The second component is the change time frame which directly influences the choice of the evolution technique. The third component is the significance of the change. How the framework leads to an evolution technique depends on the scope of the change, either major or minor. In this framework, the fourth component, the database life-cycle is the focal point. Understanding database changes can be a very complex task due to the number of variables present in the database environment. By considering each phase of the database life-cycle, the framework allows us to tackle this complexity. The last component refers to evolution techniques. Using the mappings provided by the framework, the maintenance team can manage database changes issues. For each combination of the three dimensions characterizing database changes, the framework suggests the right evolution techniques for each phase of the database life-cycle. Figure 4 exhibits only the macro mappings between the components.

As an example, let us consider a library management system. At the end of the design phase, a validation process conducted with the users pointed out the lack of differentiation between scientific editor and publisher. This change can be defined as a conceptual minor change occurring at the design phase. Applying our framework leads us to recommend:

- a reverse engineering technique to propagate this unanticipated change in the specifications and
- a forward engineering technique

Let us consider a second example consisting of adding a book order process into the system. It is a major unanticipated change occurring on the maintenance phase. Our framework suggests to perform :

- a schema integration between the previous conceptual schema and the conceptual representation of the new process,

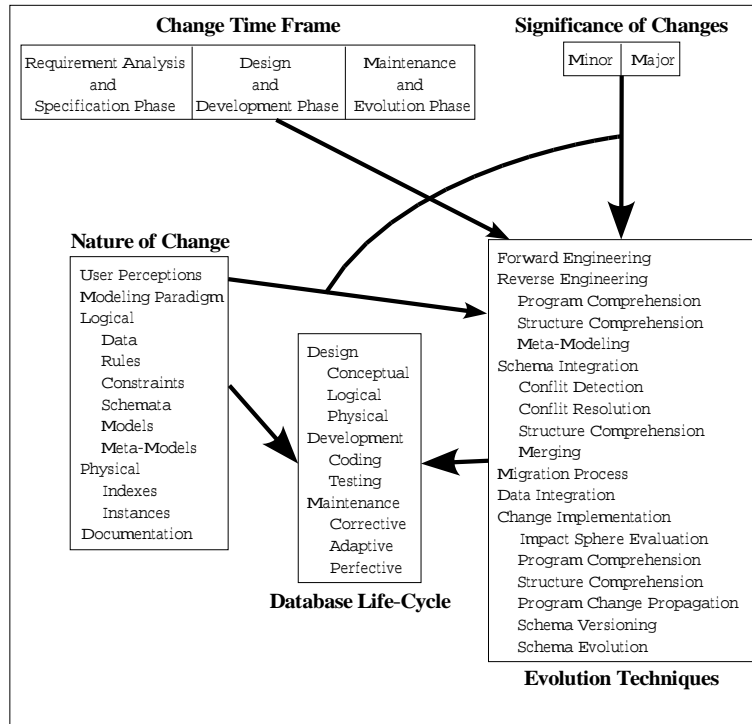


Figure 4: Techniques to be used for each change type

- a forward engineering of the new data model and,
- a migration data process from the current system to the new one.

These examples illustrate the usefulness of our framework both for designers and maintainers of evolving information systems.

4 Related Works

Past work on change evolution has focussed mainly on the late phases of software life-cycle [16]. Our approach encompasses the whole life-cycle and especially the early phases (requirements specification and design).

The problem of database evolution has been widely addressed during the last twenty years. The literature supplies a set of approaches and techniques for schema evolution. Approaches for database transformations have been proposed [8]. Among these approaches we can mention those for ER schema transformation proposed by Batini et al. [4], Kim et al. [15], and Kashyap et al. [14]. Some aspects of these approaches have been formalized only for transformation and are independent of database content [20, 21]. Other approaches considered only certain types of transformations and assume that specific types of dependency constraints are employed [5, 13, 17, 29]. In contrast to these approaches, McBrien et al. [18, 19] have proposed a formal framework for ER transformation that allows arbitrary constraints on instances to be specified on part of the transformation rules. They also propose a prototype ER schema transformation tool that automatically translates queries, updates and data from the source schema into the transformed schema.

In the area of object-oriented databases, Banergee et al. [3] introduce a taxonomy of schema modification operations. Rules preserving invariant of the schema are proposed to put constraints on schema evolution. Zicari [32] proposes a set of primitives to manage schema evolution in the O2 database systems. Randal et al. [24] address the semantics of change by defining an axiomatic model enabling a formal specification of dynamic schema evolution in Object-Oriented systems. A change impact analysis approach is proposed in [9, 10]. This approach uses a multi-graph that describes the links between the software components and a knowledge-base system to represent the knowledge of the evolution experts. Yougopuspito et al. [31] describe a transformation process from a relational database to an object relational database.

Some works go beyond the definition of taxonomy of changes [28]. Roddick et al. [26] describe the changes through different characteristics like the effect or the cause of changes. Neumann et al. deal with software changes in order to analyze the change impact [22]. In the same objective, Weideneijer et al. [30] propose a set of semantics patterns changes

that can occur in a conceptual schema. The use of these patterns should contribute in the determination of the best way to change the conceptual schema. Rashid suggests an aspect-oriented approach to deal with unanticipated changes [25].

In contrast with our work, little work has been done in addressing the problem of overall database evolution. The works of Dominguez et al. [11], and Gustavsson et al. [12] consider only the definition of an architecture allowing the management of database evolution. These architectures register schemas of different design levels and the transformation rules used for the mapping between elements of two adjacent design levels. However, these architectures only take into account the conceptual and the logical design levels. Finally, Noppen et al. propose a model to anticipate future requirements [23].

5 Conclusion

This paper describes a typology of unanticipated changes allowing the designer to evaluate their impacts on a database system. The typology is based on three dimensions :

- the nature of change : it can be concerned with requirement specifications, business rules, physical storage, etc. These types of changes are very different in terms of their consequences on the database system.
- the change time frame : the earlier a change occurs in the database life-cycle, the easier it can be integrated and propagated.
- the significance of change : minor changes affect a very small part of the system. They have to be taken into account differently compared to major changes whose impact sphere is an important part of the system.

For each type of change, we define one or several techniques to be used in order to ensure that the evolution is propagated both in the system and in its documentation. We propose a framework matching three dimensions of changes, and database life-cycle phases, to evolution techniques such as forward engineering, reverse engineering, migration process, etc.

Further work will consist in defining a guidance tool in order to help database designers and maintenance teams in managing database unanticipated evolution. This guidance tool will be based on our change typology and on our framework. Prototyping this tool will allow us to validate this typology and/or enrich it.

References

- [1] Akoka J, Comyn-Wattiau I, MeRCI: An Expert System for Software Reverse Engineering, Fourth World Congress on Expert Systems, Mexico, 1998
- [2] Atzeni P., Ceri S., Paraboschi S. and Torlone R., Database Systems: Concepts, Languages and Architectures. The McGraw-Hill Companies editor, 1999.
- [3] Banergee J., Kim W., Kim H. and Korth H., Semantics and Implementation in Object-oriented Databases. Proceedings of ACM SIGMOD International Conference on Management of Data, San Francisco, 1987.
- [4] Batini C., Lenzerini M. and Navathe S., A Comparative Analysis of methodologies for Database Schema Integration. ACM Computing Surveys Journal 18 (4), 1986.
- [5] Biskup J. and Convent B., A Formal View Integration Method. Proceedings of ACM SIGMOD International Conference on Management of Data, Washington, 1986.
- [6] Comyn-Wattiau I., Bouzeghoub M., Constraint Confrontation: An Important Step in View Integration. Proceedings of the 4th International Conference on Advanced Information Systems Engineering (CAISE'92), Manchester, May 1992, LNCS 593, P. Loucopoulos (Ed.), Springer Verlag.
- [7] Cremer K., Marburger A. and Westfechtel B., Graph-Based Tools for Re-engineering. Journal of Software Maintenance and Evolution: Research and Practice, Vol 14, 2002, pp 257-292.
- [8] Davidson S., Buneman P. and Kosky A., Semantics of Database Transformations. LNCS 1358, Edited by L. Libkin and B. Thalheim, 1998.
- [9] Deruelle L., Goncalves G. and Nicolas J. C., Local and Federated Database Schema Evolution - An Impact Propagation Model. DEXA'99, Florence, Italy, August 30 - September 4, 1999.

- [10] Deruelle L., Goncalves G. and Nicolas J. C., A Change Propagation Model and a Platform for Multi-Database Applications. Proceedings of IEEE International Conference on Software Maintenance, Florence Italy, November 6 - 10, 2001.
- [11] Dominguez E., Lloret J. and Zapata A., An Architecture for Managing Database Evolution. Workshop on Evolution and Change in Data Management of ER'2002, Tampere, Finland, October 7-11, 2002.
- [12] Gustavsson H., Lings B. and Lundell B., An Active Approach to Model Management for Evolving Information Systems. Workshop on Evolution and Change in Data Management of ER'2002, Tampere, Finland, October 7-11, 2002.
- [13] Johannesson P., Schema Integration, Schema Translation and Interoperability in Federated Information Systems, PH. D. Thesis, DSU, Stockholm University, 1993.
- [14] Kashyap V. and Sheth A., Semantic and Schematic Similarities Between DataBase Objects: A Context-Based Approach. VLDB journal, 5 (4), 1996.
- [15] Kim W., Choi I. Gala S. and Scheeval M., On Resolving Schematic Heterogeneity in Multidatabase Systems. Modern Database Systems, ACM Press, 1995.
- [16] Kniesel G., Noppen J., Mens T. and Buckley J., WS9. The first International Workshop on Unanticipated Software Evolution. Online proceedings of USE2002 - First International Workshop on Unanticipated Software Evolution, June 2002. <http://joint.org/use/2002/ecoopWsReportUSE2002.pdf>.
- [17] Larson J. H., Navathe S. B. and Elmasri R., A Theory of Attribute Equivalence in Database with Application to Schema Integration, IEEE Transaction on Software Engineering, 15 (4), 1989.
- [18] Mc Brien P. and Poulouvassilis A., A Formal Framework For ER Schema Transformation, Conceptual Modeling. ER'97, Los Angeles, 1997.
- [19] Mc Brien P. and Poulouvassilis A., Automatic Migration and Wrapping of Database Applications - A Schema Transformation Approach. ER'99, LNCS 1728, pp 96-113.
- [20] Miller R. J., Ioannidis Y. E. and Ramakrishan R., The Use of Information Capacity in Schema Integration and Translation. Proceedings of 19th International Conference on Very Large Data Bases, Ireland, 1993.
- [21] Miller R. J., Ioannidis Y. E. and Ramakrishan R., Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice. Information System Journal, 19 (1), 1994.
- [22] Neumann G., Strembeck M. and Zdun U., Using Runtime Introspectible Metadata to Integrate Requirement Traces and Design Traces in Software Components. Online proceedings of USE2002 - First International Workshop on Unanticipated Software Evolution, June 2002. At <http://joint.org/use/2002/sub/>.
- [23] Noppen J., Tekinerdogan B., Aksit M., Glandrup M. and Nicola V., Optimizing Software Development Policies for Evolutionary System Requirements. Online proceedings of USE2002 - First International Workshop on Unanticipated Software Evolution, June 2002. At <http://joint.org/use/2002/sub/>.
- [24] Randal J. P. and Tamer Ozsu, An Axiomatic Model of Dynamic Schema Evolution in Object-Base Systems. ACM Transactions on Database Systems, 22 (1), March 1997, pp 75-114.
- [25] Rashid A., Aspect-Oriented Schema Evolution in Object Databases: A Comparative Case Study. Online proceedings of USE2002 - First International Workshop on Unanticipated Software Evolution, June 2002. At <http://joint.org/use/2002/sub/>.
- [26] Roddick F. J. et al, Evolution and Changes in Data Management - Issues and Directions. SIGMOD Record journal, 29 (1), March, 2000.
- [27] Schach S. R. and Tomer A., A Maintenance-Oriented Approach to Software Construction. Journal of Software Maintenance and Evolution: Research and Practice, Vol 12, 2000, pp 25-45.
- [28] Schaarschmidt R. and Lufter J., Schema Versioning for Archives in Database Systems, Workshop on Evolution and Change in Data Management of ER'99, LNCS 1727, Paris, France, 1999.
- [29] Spaccapietra S., Parent C. and Dupont Y., Model Independent Assertions for Integration of Heterogeneous Schemas, VLDB Journal, 1 (1), 1992.

- [30] Wedemeijer L., Semantic Change Patterns in the Conceptual Schema. Workshop on Evolution and Change in Data Management of ER'99, LNCS 1727, Paris, France, 1999.
- [31] Yugospito P. and Araki K., Evolution of Relational Database to Object-Relational Database in Abstract Level. Proceedings of the International Workshop on Principles of Software Evolution, July 1999, pp 103-107.
- [32] Zicari R., A Framework of Schema Updates. Proceedings of the 7th International Conference on Data Engineering, Japan, 1991.