Data Model Evolution as a Basis of Business Process Management

Volker Gruhn[1], Claus Pahl[2], Monika Wever[1]

[1] LION Gesellschaft für Systementwicklung mbH, Universitätsstraße 140, D–44799 Bochum
[2] Universität Dortmund, Fachbereich Informatik, Baroper Straße 301, D–44221 Dortmund

Abstract. In this article we propose an approach to business process management which meets the demands of business process evolution. This approach allows for on–the–fly modifications of business processes. In contrast to many other approaches, we do not only concentrate on activities to be carried out in business processes, but also on the data created and manipulated by these activities. We propose to apply data model analysis and improvement strategies well–known from the information system field in the context of business process management.

Keywords
business processes, data model, data model analysis, evolution

1 Motivation

Business process (re–)engineering [10], process innovation [5] and continuous process improvement are buzzwords meaning more or less the same: to organize business processes in an efficient way. This may mean to completely redesign processes in order to achieve breakthroughs in productivity or it may mean to adapt an existing process to changing circumstances. Software processes are business processes in a software house. The question how to manage software processes has been addressed with increasing interest [17] during the last decade .

In the following we use the notion of *process* to denote general business processes as well as software processes. Process modeling, process model analysis, and the enaction of process models (i.e. to govern a real world process on the basis of an underlying model) is summed up as *process management*.

Analysis of process models turned out to be necessary in order to avoid the enaction of faulty process models. Process evolution turned out to be a central issue in order to ensure that process models can be adapted to changing circumstances. A closer look at existing process management approaches reveals that some provide analysis facilities for process models [3] while others somehow care for on–the–fly modifications of activity models [7]. Most however, do not

support the analysis of data models and evolution of data which has already been produced by processes. Thus, the process management perspective could be called *process–oriented* (data models and database schemas underlying these processes are usually considered less important).

In contrast, research in the field of information systems deals with questions of what *good* data models should look like and how evolution of database schemas can be managed [1] without considering the business processes supported on top of the data stored in the database. Thus, in modeling information systems one usually starts from a *data–driven* perspective (i.e. what is the information to be administrated, how can this data be organized efficiently). Dynamic aspects of information systems are either — if at all — expressed in the form of integrity constraints or they are hard–wired in the form of application programs.

In this article we describe how the $\mathcal{L}eu$ approach [8] has been extended to cover data model analysis, improvement, and schema evolution.

In section 2, we briefly introduce our process management approach. In section 3, we discuss an example of a $\mathcal{L}eu$ process model in detail. Section 4 deals with requirements related to data model analysis and schema evolution. In section 5, we discuss the improvement mechanisms implemented in $\mathcal{L}eu$. Section 6 illustrates database evolution in relation to the example in section 3. Finally, section 7 concludes this paper with a discussion of experiences.

2 The $\mathcal{L}eu$ approach to process management

In the $\mathcal{L}eu$ approach we consider activity models, data models and organization models as constituent parts of process models [8]. In detail this means:

Data modeling: In $\mathcal{L}eu$, data models are used to describe the structure of objects which are manipulated within a process. Data models in $\mathcal{L}eu$ are described by means of extended entity–relationship diagrams [8].

Activity modeling: In $\mathcal{L}eu$, activity models are used to define activities to be executed in a process. Moreover, activity models define the order in which activities of a process have to be carried out. Activity models in $\mathcal{L}eu$ are described by means of FUNSOFT nets [6], which are high level Petri nets (compare section 3.4).

Organization modeling: Organization models in $\mathcal{L}eu$ are used to define which organizational entities are involved in a process. Organization modeling is based on a hierarchical role concept. Roles are sets of permissions for the execution of activities. Roles can be attached to organizational entities. Organization models in $\mathcal{L}eu$ are described by means of organizational diagrams which identify organizational entities and their hierarchical relationships.

Once these aspects of processes have been modeled, it is necessary to integrate them. A data model and an activity model are integrated by associating data model entities and activity parameters in a typing relation. An activity model and an organization model are integrated by associating organizational entities to those activities for which they are responsible.

3 Example of a process model

In this section we take a closer look at a data model and an activity model. This is illustrated by a simplified process model from the building construction administration area. The organizational structure is not discussed here since it is irrelevant for analysis, improvement and evolution of a data model.

3.1 Data modeling in $\mathcal{L}eu$

Data modeling in $\mathcal{L}eu$ is based on extended entity–relationship diagrams (ER diagrams) [8]. Figure 1 shows a simplified data model for *lease management*.

Relationships can be of the cardinalities 1:1, 1:n, or n:m. This is depicted in single–ending and multi–ending edges. Furthermore, they can be mandatory or optional. This is represented by solid and dashed lines.

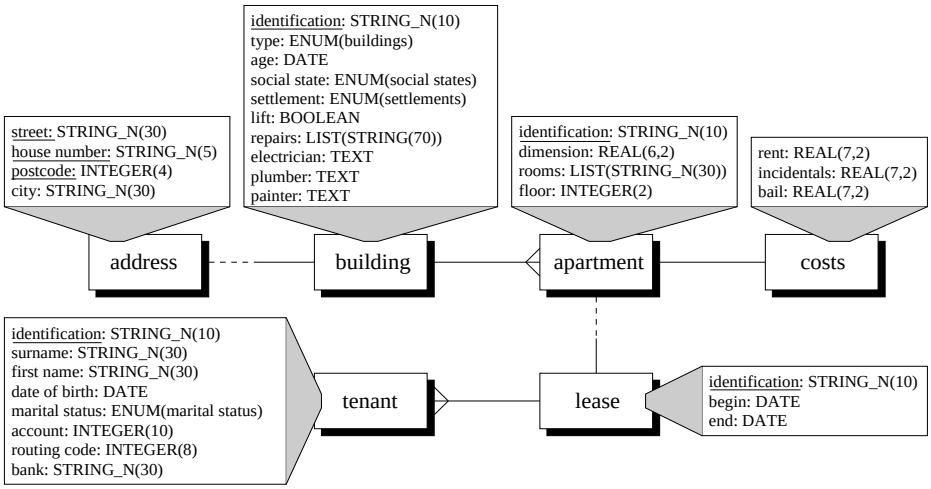

Fig. 1. Data model for lease management

Some object–oriented features have been introduced into the entity–relationship modeling as supported by $\mathcal{L}eu$. First of all, we understand entity types as object types, i.e. each entity has a unique object identity. We use the term *object* to denote an entity. Objects can be of a predefined format (Postscript, WordPerfect documents, etc.) or of a complex type. Objects of these complex types are described by means of their attributes. Attributes can be mandatory or optional. Mandatory attributes can be used to compose a user–defined primary key of an object type. In figure 1 the primary keys are underlined. Non–primitive attribute types are lists that can be composed of primitive types and enumeration types which can be defined by the user.

Another object–oriented feature of $\mathcal{L}eu$ data modeling is the encapsulation of object types by means of user–defined operations. Each object can only be manipulated by means of operations attached to the corresponding object type. Moreover, we use the concept of delegation between object types [20] in order to define relationships between closely related object types. By means of delegation it is possible to enforce the execution of certain operations to be delegated to another object. Thus, the concept of delegation is applied to the type level *and* to the object level. This is the main difference to the inheritance concept. A more detailed motivation for using the delegation concept is described in [21].

3.2 Generating a database schema

A data model in $\mathcal{L}eu$ is used to describe the structure of objects which are manipulated within processes. To store these objects, a relational database schema is generated from the data model which is accessed within running processes interacting with process participants (compare figure 2).

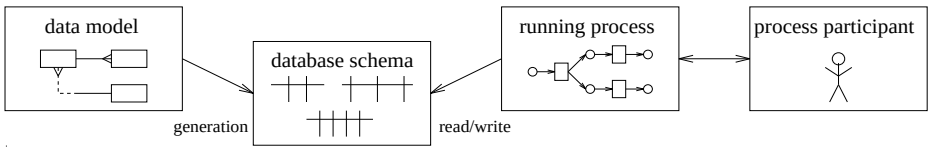

Fig. 2. Generation and use of a database schema

For each object type at least one table is generated. This table contains one column for each attribute and one additional column for a primary key called *surrogate*. A surrogate is an automatically generated identifier for an object. Several types of attributes offered by $\mathcal{L}eu$ are not supported by a relational database. For example, attributes like a list cannot be stored in one column of a flat relational table. Therefore, only the first element of a list is stored in the column of the list attribute. Additionally, a further table is generated to store the whole list. Attributes of enumeration or text types are treated similarly. For each n:m relationship one table is generated. It contains two columns for the surrogates of the two related object types. For other relationships the table of one object type contains an additional column to store the surrogates of the other object type as foreign keys. Figure 3 shows some of the tables generated for the *lease management*.

Basic operations for creation, update and deletion of objects are also generated out of the data model. Thus, the internal representation of objects in the database does not have to be known in order to store and retrieve objects. These operations (and some more sophisticated operations like *get–all–objects–of–type*) build a programming interface up to the generated database schema.

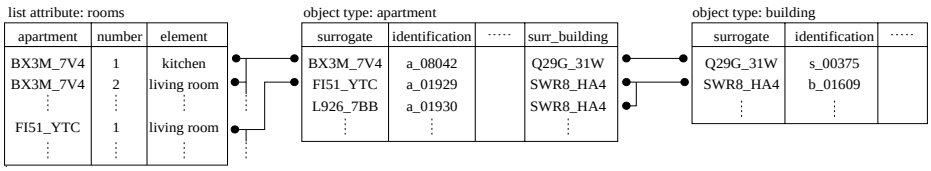

Fig. 3. Database schema for lease management

3.3 Access paths

To manipulate the objects stored in the generated database the activities carried out in the processes either use operations of the database programming interface or they use database queries. While the operations are sufficient for simple retrievals, individual queries covering objects of several types are needed for more complicated retrievals. They are formulated on the abstraction level of data models. Details of the internal database schema do not have to be known. In order to avoid potential performance problems caused by too complicated queries, the process modeler should describe object access paths. We define the access path of an object query as the set of attributes used in the query together with a quantification specifying how frequent the access path is used.

3.4 Activity modeling in $\mathcal{L}eu$

In $\mathcal{L}eu$, FUNSOFT nets are used to specify activities to be carried out and their order. FUNSOFT nets are high level Petri nets, of which semantics is defined in terms of Predicate/Transition nets [9]. In FUNSOFT nets, the T–elements (represented as rectangles) are called agencies. Agencies represent activities. S–elements (represented as circles) are called channels. They are used to store objects. FUNSOFT nets are hierarchically structured by means of T–element refinement [11]. FUNSOFT nets do not only contain a definition of activities and their parameterization, but also an order of activities. They allow to define that activities have to be carried out sequentially, concurrently or alternatively. For more details about FUNSOFT nets we refer to [6].

Figure 4 shows a FUNSOFT net which describes the activity model of agreeing on leases. In the upper branch of the net an apartment is selected. In the lower branch a potential tenant is selected. If an object of type *apartment* is available in channel *Apartment*[3] and if an object of type *tenant* is available in channel *Cleared tenant*, then agency *Write lease* can be started. The result is a lease which is then subject to the legal check.

[3] The annotation above a channel symbol displays the type of objects to be stored in this channel, the annotation below a channel symbol is the channel name.

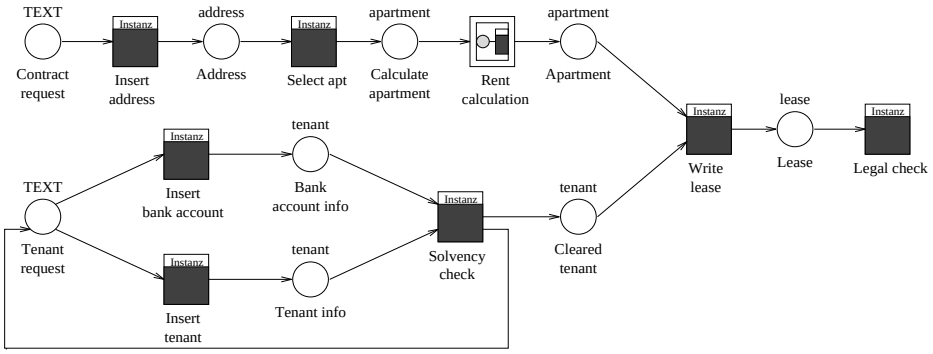

Fig. 4. Activity model for agreeing on leases

4 Related work

Process models used for governing real world processes are crucial for the efficiency of any business operation. Thus, process model analysis is necessary in order to avoid the enaction of inefficient and/or inconsistent process models. The analysis of activity models and organization models has been discussed intensively in the process management literature [13, 16]. In the following we will discuss some approaches which focus on data model analysis.

In principle, we can distinguish between data model analysis that is independent from an individual customer and customer–specific data model analysis. While the first kind of analysis can be realized without considering any details of customer circumstances, the latter requires detailed knowledge of the customer situation.

In [14] efficiency of data models is identified as one of the key success factors of any information system. Abstract properties of data models such as completeness, comprehensibility, simplicity, flexibility and others are considered as substantial properties of data models. The necessity to measure these properties on the basis of metrics and to develop improvement strategies is illustrated.

Unfortunately, properties like efficiency and completeness can only be checked with respect to individual customer situations. A data model consisting of a set of object types may, for example, be an efficient basis for processes being carried out at one company, while it can be awfully inefficient as a basis of the same processes carried out at another company. This may be due to differences in execution frequencies of processes or it may be due to different arrival frequencies of objects to be processed. Thus, information about the customer specific situation is needed in order to detect such data model shortcomings.

Once process models have been customized, the customer situation does not remain completely stable. Internal organization as well as legal changes and modified business targets demand process model flexibility. This flexibility concerns activity models, organization models, and data models as well as their integration. We realize an additional problem regarding the evolution of data models:

once a database has been generated out of a data model and once data is stored in this database, the evolution of a data model requires the transformation of the data according to the data model evolution.

Process conditions tend to change during the process' lifespan. Reasons for changes are, for example, availability of personnel, organizational changes, concentration on certain customers and markets. In order to cope with such highly dynamic circumstances, it is necessary that process models can be modified whenever required. General tendencies like globalization of markets and empowerment of people reinforce the demand for flexible processes. Flexible processes allow to react to market opportunities very quickly and thus obtain competitive advantages.

While the evolution of activity models is intensively discussed in the software process literature, the question how data models evolve is hardly considered in this context. In contrast, the question of schema evolution is discussed with respect to information system design and maintenance [12, 1]. Some approaches relevant to the work proposed in this article are discussed in the following.

In [15] different types of database schema evolutions for object–oriented database schemas are distinguished. Evolution of class definitions, evolution of relationships between class definitions and evolution of the set of class definition (add a class, delete a class, change name of a class) are identified as typical kinds of schema evolution. In [15] it is pointed out that most object–oriented database systems do not support the complete propagation of schema modifications to the object level. Mechanisms to manage this propagation are

- the use of an explicit *convert* operator which can be used to convert objects concerned when modifying a data model [19],
- an automatic start of propagation after each data model modification [2], or
- a propagation to an object as soon as this object is accessed [4].

The advantage of the manually started propagation is its efficiency because only objects actually needed in the future are converted. Its disadvantage is the danger of inconsistencies because old and not converted objects may exist at any time. The advantage of automatic propagation to all objects is its consistency, its disadvantage is its bad performance. In the $\mathcal{L}eu$ approach some object–oriented features like delegation between classes and objects (compare [20]) and association of object types and operations have been introduced into an entity–relationship oriented modeling. Thus, the types of modifications are obviously the same as discussed in [15]. Our conversion strategy is to convert all objects interactively in cooperation with the data modeler to ensure that information loss and inconsistencies can be minimized (compare section 6).

5 Analysis and improvement of data models

In this section we describe what kinds of data model analysis facilities are available in the *Leu* approach and which improvements of data models turned out to be typical in order to remove certain data model weaknesses. The work presented in this section is described in more detail in [18][4].

The data model improvement consists of two steps. First, the syntactical completeness of the data model is analyzed. It is examined whether all object types have attributes, relationships and primary keys and whether they are used by at least one access path. If all object types satisfy these demands, the data model is syntactically complete and its analysis will be continued. Then, four kinds of improvement are applied:

Removal of certain types of attributes: The representation of attributes of list, text and enumeration types in the database demands the generation of additional tables, as mentioned in section 3.2. Queries refering to attributes of such types have to access more than one table. The more tables a query accesses, the longer its execution will take. Therefore, the user should consider whether some lists, texts, or enumerations can be replaced by other types.

New relationships to avoid long access paths: The more object types a query uses, the more time takes its execution, because at least one additional table has to be accessed for each further object type. Hence, an access path should be as short as possible. In order to cut a long access path short, new relationships have to be added to the data model.

Merge of object types: Object types with a 1:1/mandatory:mandatory relationship are similar to one object type. Just looking at the syntax, they could be combined. However, this may not be appropriate looking at the object types' semantics. Hence, the mergence of object types can be proposed but the modeler must decide whether it is reasonable.

Split of object types: There are two reasons to split an object type. First, the attributes of the object type may be divided into two groups. Each of these groups of attributes is used by a group of access paths. There is no access path using attributes of both groups. In this case the unused attributes only enlarge the object type and slow down the execution time. Consequently, the object type should be split into two disjoined object types with one group of attributes belonging to each of them. Secondly, the attributes of the object type may also be divided into two groups but in a different way. One of these groups consists of very frequently used attributes. The attributes of the second group are used very rarely. In this case, the rarely used attributes enlarge the object type for access paths using only the frequently used attributes. Therefore, the object type should be split, too.

The user will be informed of all defects found by the analysis. The improvement only recommends certain data model changes to the modeler. He has to decide whether the proposed changes should be applied.

[4] Schneider Monika Wever's maiden name.

In the data model for *lease management* (figure 1) all four types of improvement can be applied. First, the building has one list attribute, three enumeration attributes and three text attributes. According to the first type of data model improvement (removal of certain types of attributes), some of these attribute types should be changed. Assume the type *STRING_N(70)* is long enough to store the names of the craftsmen. Then the text attributes can be changed into strings. Secondly, the access path leading from *tenant* to *address* across *lease*, *apartment* and *building* is too long. Therefore a new n:1/mandatory:optional relationship between *tenant* and *address* will be added. Thirdly, *apartment* and *costs* will be merged by adding the attributes of *costs* to *apartment*. This way the 1:1/mandatory:mandatory relationship between them is dropped. Finally, the *tenant* will be split into *tenant* and *bank account*. This is based on an analysis of access paths implemented by the agencies of the activity model for *agreeing on leases* (figure 4).

All changes discussed above are only performed on the data model in the first instance. They also have to be propagated to the database schema. This means the generation has to be started again to change the tables of the schema according to the changes of the model. It would also be possible to delete the whole database schema and generate it again. But in most cases only a part of the data model is changed so that it takes less time to change the existing database schema.

6 Evolution

Business process models are subject to modifications because of changing circumstances. Sometimes, only activity models have to be changed. In other situations data models are concerned. Since modifications of the first kind have been addressed in the process literature, we will restrict to modifications that cause changes of data models.

Potential changes of a data model are: adding new object types or relationships, deleting object types or relationships, adding new attributes to object types, removing attributes and changing the types of attributes.

As an example four customer–specific changes are performed on the data model for *lease management*. A new text attribute called *form* is added to the object type *lease*. The attribute *lift* is removed from the object type *building*. The *postcode* is extended from a four–digit to a five–digit format. And finally, the type of *first name* is changed from string into a list of strings.

Further changes of data models can be due to additional or modified process models. If, for example, a new process has to be established, requirements for efficient access to certain data may change substantially.

When changes are performed on the data model, the generation procedure has to be started again to propagate them to the object level. This implements an immediate coercion strategy.

Whenever an attribute type is changed, the generation tries to minimize the loss of information. Loss of information can be avoided in the following cases:

- the length of an attribute type is extended
- an attribute type is changed from boolean, integer, real, time, or date to string,
- an attribute type is changed from single–valued to multi–valued, or
- an attribute type is changed from multi–valued to single–valued (only the first element of the list is kept).

In these cases the generation converts the values of the changed attributes automatically according to the new type. In all other cases an automatic conversion is impossible and the values are lost.

Besides the loss of information, the evolution of a data model causes other problems. Some of them are solved during the generation:

- Objects of optional attributes may contain null–values. If such an attribute is changed to mandatory, the generation will replace the null–values with dummy values.
- If a new attribute is added to an object type, all existing objects will have null–values in the column of this attribute. If the new attribute is mandatory, the generation will also replace the null–values with dummy values.

Other changes of data models are not automatically propagated to the object level in order to avoid inconsistencies in the database. Examples of such changes are given in the following. They all focus on the primary key of an object type:

- A new attribute is added to an object type without primary key. This new attribute must not be a primary key itself because its values are either null–values or dummy values and neither of them is unique.
- An existing attribute of an object type without primary key must not become a primary key because its values may not be unique.
- The number of attributes building the primary key of an object type must not become smaller because the values of the remaining primary key attributes may not be unique.
- The attribute type of a primary key must not be changed with loss of values because after the generation the lost values are replaced with dummy values that are not unique.

It is impossible to perform these changes on the database during the generation because reasonable unique values can not be created automatically. When some of these changes are necessary, the data modeler has to perform them “by hand” directly on the database.

To sum up, the generation of database schemas out of modified data models tries to minimize the loss of information. Whenever it cannot be avoided, the data modeler has to give his explicit agreement in order to avoid unexpected loss of information.

7 Conclusions

In our approach we deal with processes based on information systems. One of our main goals is a high degree of performance enacting these processes on information systems. Efficient storage and retrieval of data used by processes is crucial for achieving this goal. Therefore we presented methods for analysis and improvement of data models.

In this article we have motivated why evolution of data models is a prerequisite for flexible processes. Coping with evolution means being concerned with the problems of data integrity and loss of data. We have analysed possible changes in activity and data models and their effects on data. Our experience is that the benefits of graphical modeling of processes and the entailed easy modification of processes can only be fully exploited when data building the basis of processes can evolve according to changing circumstances and modified business goals.

Acknowledgements: We would like to thank all members of the *Leu* team for their cooperation in implementing these strategies. Moreover, we would like to thank Ernst–Erich Doberkat for fruitful discussions about focus and contents of the diploma thesis which contributed to this article.

References

1. J. Andany, M. Leonard, and C. Palissier. *Management of Schema Evolution in Databases.* In *Proceedings of the* 17^{th} *Conference on Very Large Databases*, pages 161–170. Morgan-Kaufmann, 1991.
2. F. Bancilhon. *Object-oriented Database Systems.* In *Proceedings of the 7th ACM Symposium on Principles of Database Systems*, Austin, Texas, US, March 1987.
3. S. Bandinelli, A. Fugetta, and S. Grigolli. *Process Modelling In-the-Large with SLANG.* In *Proceedings of the* 2^{nd} *International Conference on the Software Process - Continuous Software Process Improvement*, pages 75–83, Berlin, Germany, February 1993.
4. J. Banerjee, W. Kim, H.J. Kim, and H.F. Korth. *Semantics and Implementation of Schema Evolution in Object-Oriented Databases.* In *Proceedings of the Conference on Management of Data 1987*, 1987.
5. T. Davenport. *Process Innovation - Reengineering Work through Information Technology.* Harvard Business School Press, Boston, US, 1993.
6. W. Deiters and V. Gruhn. *Managing Software Processes in MELMAC.* In *Proceedings of the Fourth ACM SIGSOFT Symposium on Software Development Environments*, pages 193–205, Irvine, California, USA, December 1990.
7. W. Deiters, V. Gruhn, and H. Weber. *Software Process Evolution in MELMAC.* In Daniel E. Cooke, editor, *The Impact of CASE on the Software Development Life Cycle.* World Scientific, Series on Software Engineering and Knowledge Engineering, 1994.
8. G. Dinkhoff, V. Gruhn, A. Saalmann, and M. Zielonka. *Business Process Modeling in the Workflow Management Environment LEU.* In P. Loucopoulos, editor, *Proceedings of the* 13^{th} *International Conference on the Entity-Relationship Approach*, pages 46–63, Manchester, UK, December 1994. Springer. Appeared as Lecture Notes in Computer Science no. 881.

9. H.J. Genrich. *Predicate/Transition Nets.* In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties*, pages 208–247, Berlin, FRG, 1987. Springer. Appeared in Lecture Notes on Computer Science 254.
10. M. Hammer and J. Champy. *Reengineering the Corporation.* Harper Business, New York, US, 1993.
11. P. Huber, K. Jensen, and R.M. Shapiro. *Hierarchies in Coloured Petri Nets.* In *Proc. of the* 10^{th} *Int. Conf. on Application and Theory of Petri Nets*, pages 192–209, Bonn, FRG, 1989.
12. M. Jarke, J. Mylopoulus, J.W. Schmidt, and Y. Vassiliou. *DAIDA - An Environment for Evolving Information Systems. ACM Transactions on Information Systems*, 10(1):1–50, January 1992.
13. M.I. Kellner and G.A. Hansen. *Software Engineering Processes: Models and Analysis.* In B.D. Shriver, editor, *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences, Vol. II*, 1989.
14. D.L. Moody and G.G. Shanks. *What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models.* In P. Loucopoulos, editor, *Proceedings of the* 13^{th} *International Conference on the Entity-Relationship Approach*, pages 94–111, Manchester, UK, December 1994. Springer. Appeared as Lecture Notes in Computer Science no. 881.
15. G. Nguyen and D. Rieu. *Schema Change Propagation in Object-Oriented Databases.* In G.X. Ritter, editor, *Information Processing 89*, pages 815–820. Elsevier, 1989.
16. A. Oberweis, P. Sander, and W. Stucky. *Petri net based modelling of procedures in complex object database applications.* In D. Cooke, editor, *Proceedings of the COMPSAC 1993*, Phoenix, Arizona, US, 1993.
17. W. Schäfer, editor. *Software Process Technology - Proceedings of the* 4^{th} *European Workshop on Software Process Modelling*, Noordwijkerhout, The Netherlands, April 1995. Springer. Appeared as Lecture Notes in Computer Science 913.
18. M. Schneider. *Werkzeuge zur Optimierung erweiterter Entity–Relationship Modelle und deren Abbildung in ein relationales Datenbankschema (in German).* August 1994. Diplomarbeit, University of Dortmund.
19. A. Skarra and S. Zdonik. *The Management of Changing Types in an Object-Oriented Database.* In N. Meyrowitz, editor, *Object-Oriented Programming Systems, Languages and Applications (OOPSLA) 1986 - Conference Proceedings*, pages 483–495, Portland, Oregon, USA, October 1986. ACM Press.
20. R. Wieringa, W. de Jonge, and P. Spruit. *Roles and dynamic subclasses: a modal logic approach.* In M. Tokoro and R. Pareschi, editors, *Proceedings of the European Conference on Object-Oriented Programming, Bologna*, pages 32–59, Berlin, 1994. Springer. Appeared as Lecture Notes in Computer Science no. 821.
21. W. Wilkes. *Instance Inheritance Mechanisms for Object Oriented Databases.* In K. Dittrich, editor, *Advances in Object-Oriented Database Systems*, pages 274–279, Berlin, 1988. Springer. Appeared as Lecture Notes in Computer Science no. 334.

This article was processed using the LaTeX macro package with LLNCS style