A Research Agenda for Conceptual Schema-Centric Development

Antoni Olivé[1], Jordi Cabot[2]

[1]Universitat Politècnica de Catalunya, Spain
[2]Universitat Oberta de Catalunya, Spain

Abstract. Conceptual schema-centric development (CSCD) is a research goal that reformulates the historical aim of automating information systems development. In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system's evolution. To achieve the CSCD goal, several research problems must be solved. In this paper we identify and comment on sixteen problems that should be included in a research agenda for CSCD.

1 Introduction

The goal of automating information systems (ISs) building was established in the 1960s [51]. Since then, the goal has been reformulated many times, but the essential idea has remained the same: to automatically execute the specification of an information system in its production environment.

Forty years later, it is clear that this goal has not been achieved to a satisfactory degree. The main reason is that a number of major problems remain to be solved [41]. Most of these problems are technical, but others are related to the lack of maturity in the information systems field, such as the lack of standards. The insufficient standardization of languages and platforms has hampered advances in the automation of systems building. Fortunately, however, the last decade has seen the emergence of new standards related to information systems development. The progress made in standardization provides an opportunity to revive the goal of automation [50].

In [37] we proposed to call the goal "conceptual schema-centric development" (CSCD) in order to emphasize that the conceptual schema should be the focus of information systems development.

To achieve the CSCD goal, numerous research problems must be solved. In this paper we propose a research agenda with sixteen main research problems that we believe it is necessary to solve in order to achieve that goal. This agenda extends, refines and updates the one proposed in [37].

The paper is organized as follows. In the next section we briefly review the role and contents of conceptual schemas. In Section 3 we characterize the CSCD goal. We then present the proposed research agenda in Section 4. Finally, in Section 5 we summarize the conclusions of this paper.

2 Conceptual Schemas

In this section, we first review the main functions of ISs and then analyze the knowledge required by a particular IS to perform these functions. Through this analysis we will be able to define and establish the role of conceptual schemas.

2.1 Functions of an Information System

Information systems can be defined from several perspectives. For the purposes of conceptual modeling, the most useful is that of the functions they perform. According to this perspective, an IS has three main functions [3, p.74]:

- *Memory*: To maintain a consistent representation of the state of a domain.
- *Informative*: To provide information about the state of a domain.
- *Active*: To perform actions that change the state of a domain.

The memory function is passive, in the sense that it does not perform actions that directly affect users or the domain, but it is required by the other functions and it constrains what these functions can perform.

In the informative function, the system communicates some information or commands to one or more actors. Such communication may be explicitly requested or implicitly generated when a given generating condition is satisfied.

With the active function, the system performs actions that change the state of the domain. Such actions may be explicitly requested or implicitly generated when a given generating condition is satisfied.

2.2 Knowledge Required by an Information System

In order to perform the above functions, an IS requires general knowledge about its domain and knowledge about the functions it must perform. In the following sections, we summarize the main pieces of knowledge required by each function.

If the memory function of an IS has to maintain a representation of the state of the domain, the IS must know the entity and relationship types to be represented and their current population. The entity and relationship types that are of interest are general knowledge about the domain, while their (time-varying) population is particular knowledge.

In conceptual modeling, an Information Base (IB) is the representation of the state of the domain in the IS. The representation of the state in the IB must be consistent. This is achieved by defining a set of conditions (called integrity constraints) that the IS is required to satisfy at any time. Such integrity constraints are general knowledge about the domain.

The domain state is not static. Most domains change over time, so their state must also change. When the state of a domain changes, the IB must change accordingly. There are several kinds of state changes. If they are caused by actions performed in the domain, they are called external domain events. If they are caused by actions performed by the IS itself, they are called generated domain events. The IS must know the types of possible domain event and the effect of each event instance on the IB. This is also general knowledge about the domain.

If the informative function has to provide information or commands on request, the IS must know the possible request types and the output it must communicate. On the other hand, if there are generated communications then the IS must know the generating condition and the output it has to return when the condition is satisfied.

In general, in order to perform the informative function, the IS needs an inference capability that allows it to infer new knowledge. The inference capability requires two main elements: derivation rules and an inference mechanism. A derivation rule is general knowledge about a domain that defines a derived entity or relationship type in terms of others. The inference mechanism uses derivation rules to infer new information.

If, in the active function, the IS has to perform a certain action on request, then the IS must know the possible request types and the action it

has to perform in each case. On the other hand, if a certain action must be performed when a generating condition is satisfied, the IS must know this condition and the action it has to perform.

2.3 Conceptual Schemas

The first conclusion from the above analysis is that in order to perform its required functions, an IS must have general knowledge about its domain and about the functions it has to perform. In the field of information systems, such knowledge is referred to as the Conceptual Schema (CS).

Every IS embodies a CS [29, 34, 48, p.417+]. Without a CS, an IS could not perform any useful functions. Therefore, developers need to know the CS in order to develop an IS.

The main purpose of conceptual modeling is to elicit the CS of the corresponding IS. As we have seen, given that all useful ISs need a CS, we can easily reach the conclusion that conceptual modeling is an essential activity in information systems development.

3 Conceptual Schema-Centric Development

In this section we reformulate the vision of the conceptual schema-centric development (CSCD) of information systems. To achieve this vision, we must be able to specify the initial conceptual schema, to execute it in the production environment and to evolve it in order to support the new functions of the IS. We call these three main distinguishing characteristics *explicit, executable* and *evolving schema*.

Explicit schema. Once the functions of the IS have been determined, there must be an explicit, complete, correct and permanently up-to-date conceptual schema written in a formal language. We need a development environment with tools that facilitate the validation, testing, reuse and management of (potentially large) schemas.

Executable schema. The schema is executable in the production environment. This can be achieved by the automatic transformation of the conceptual schema into software components (including the database schema) written in the languages required by the production environment, or by the use of a virtual machine that runs over this environment. In either case, the conceptual schema is the only description that needs to be defined. All the others are internal to the system and need not be externally visible.

According to the conceptualization principle [27], conceptual schemas exclude all aspects related to information presentation. Therefore, the software responsible for handling user interactions (the presentation layer) is outside the scope of CSCD.

Evolving schema. Changes to the functions of the IS require only the manual change of its conceptual schema. The changes to this schema are automatically propagated to all system components (including the database schema and data) if needed.

4 Towards a Research Agenda for CSCD

CSCD is still an open research goal. There are many research problems that must be solved before CSCD can become a widely used approach in the development of industrial information systems. In this section, we identify some of the research problems found related to the three CSCD features presented above. Our starting point is the agenda presented in [37], which we extend, refine and update here. We highlight the problems related to CSCD; see [9, 14, 55] for other relevant research agendas in conceptual modeling.

4.1 Explicit Schemas

Very large conceptual schemas. The conceptual schema of a large organization may contain thousands of entity types, relationship types, constraints, and so on. The development and management of (very) large conceptual schemas poses specific problems that are not encountered when dealing with small conceptual schemas. Conceptual modeling in the large is not the same as conceptual modeling in the small. The differences are similar to those observed between programming in the large and programming in the small [16]. We need methods, techniques and tools to support conceptual modellers and users in the development, reuse, evolution and understanding of large schemas.

So far, work on this topic has focused mainly on conceptual schemas for databases [1, 11, 46]. In CSCD we have to deal with ISs and take into account both the structural (including constraints and derivation rules) and behavioral schemas.

Business rules integration. A business rule is a statement that defines or constrains certain aspects of a business. From the information systems per-

spective, business rules are elementary pieces of knowledge that define or constrain the contents of and the changes to the information base. Business rules are the main focus of a community that advocates a development approach in which the rules are explicitly defined, directly executed (for example in a rules engine) and managed [8, 42]. Given that business rules are part of conceptual schemas, we can state that the community already follows the CSCD approach as far business rules are concerned.

It is both useful and necessary to integrate the business rules and CSCD approaches. It should be possible to extract the rules embedded in a schema and to present them to users and conceptual modellers in a variety of ways and languages, including natural language. Automated support for this extraction and presentation is necessary. It should also be easy to pick up on a particular rule and to integrate it into the schema. Automated support for this integration is desirable.

Similarly, the workflow community fosters the use of workflow specifications as the primary artefact in the software development process. Workflow specifications define a set of activity ordering rules that control the workflow execution. These rules are usually executed and managed with the help of dedicated workflow management systems. Workflow specifications should be also integrated with the CSCD approach.

Schema integration. A conceptual schema is very rarely developed by a single conceptual modeller [47]. Instead, several sub-schemas are (separately) developed by different modellers, each of whom addresses a specific part of the IS. To apply the CSCD approach, these sub-schemas must subsequently be integrated in a single schema that represents the overall view of the IS.

A first step in integrating the schemas is to identify and characterize the relationships between the different sub-schemas (schema matching [40]). Once these have been identified, matching elements can be linked in a coherent schema (schema merge [39]).

Previous research on this topic focuses on the integration of database schemas [6, 38]. More recently, the problem has been studied at a more abstract level (for example, [4] presents general operators for model matching and merging). Nevertheless, much work remains to be done on schema integration in the presence of general integrity constraints and derived elements. Moreover, research on the integration of behavioural schemas is still in a preliminary stage [49].

Complete and correct conceptual schemas. Several factors affect the quality of a conceptual schema, as stated in the framework presented in the seminal paper [28] and validated in [32, 33]. Completeness and correctness

are two of the quality factors of conceptual schemas. A complete conceptual schema includes all knowledge relevant to the IS. A correct conceptual schema contains only correct and relevant knowledge. Correctness is also referred to as validity. Consistency is subsumed by validity and completeness. In CSCD, completeness and correctness are the principal quality factors. They can be achieved by using a very broad spectrum of approaches, including testing and verification. It should be possible to test and verify conceptual schemas to at least the same degree that has been achieved with software.

Several studies have focused on testing conceptual schemas [25, 30, 20, 57]. There are automatic procedures for the verification of some properties of conceptual schemas in description logics [10]. Model checking is being explored as an alternative verification technique [18]. Nevertheless, in all these topics, a lot of work remains to be done [35].

Refactoring of conceptual schemas. In general, several complete and correct conceptual schemas may exist for the same IS. However, some are better than others in terms of quality. Therefore, in some cases an initial conceptual schema may be improved if it is first transformed into a better (semantically-equivalent) alternative schema.

For this purpose, the application of refactorings at the model level has been proposed. Refactoring was initially proposed at the code level [19] as a disciplined technique for improving the structure of existing code (using simple transformations) without changing the external observable behaviour. More recently, work has been done to apply this technique to design models instead of to the source code [31]. In CSCD, we need specific refactoring operations that take into account all the components in a conceptual schema. General guidelines have not yet been developed to determine *when* and *where* to apply refactorings in order to improve the quality of the conceptual schema.

Reverse engineering. Most legacy applications do not have an explicit conceptual schema. To benefit from the CSCD approach, we must elicit the explicit conceptual schema from the internal schema embodied in the software components that form the legacy application. This process is known as reverse engineering.

Reverse engineering applied to entity and relationship types and to the taxonomies of conceptual schemas has been extensively studied for relational databases [15] and object-oriented languages [53]. However, much work remains to be done regarding the reverse engineering of general integrity constraints and derived elements of schemas. Moreover, a complete understanding of the application code in order to elicit the behavioural part

of the schema is also needed. Ideally, the interactions between the application's various software components should also be considered during the reverse engineering process.

4.2 Executable Schemas

Materialization of derived types. In general, conceptual schemas contain many derived entity and relationship types, with their corresponding derivation rules [36]. For reasons of efficiency, some of these types must be materialized. The process to determine the derived types that need to be materialized should be as automatic as possible. Moreover, changes in the population of base types may require changes in that of one or more materialized types. The propagation of these changes should be completely automatic.

The work done on the selection of database views that need to be materialized in data warehouses [23] is highly relevant to the determination of the derived types to materialize in ISs. Similarly, the large body of work on the incremental maintenance of materialized database views [22] is highly relevant to the more general problem of change propagation in ISs.

Enforcement of integrity constraints. Most conceptual schemas contain a large number of integrity constraints. The IS must enforce these constraints efficiently. This can be achieved in several ways [52]. The main approaches are integrity checking, maintenance and enforcement. In integrity checking and maintenance, each constraint is analyzed in order to (1) determine which changes to the IB may violate the constraint; (2) generate a simplified form of the constraint, to be checked when a particular change occurs; and, (3) (in maintenance) generate a repair action. In integrity enforcement, each event (transaction) is analyzed in order to (1) determine which constraints could be violated by the effect of the event; and, (2) generate a new version of the event effect that ensures that none of the constraints will be violated.

In CSCD, the analysis—regardless of the approach taken—should be fully automatic and able to deal with any kind of constraint. A general method for this analysis does not yet exist. However, a great deal of research and development work has been carried out on the enforcement of constraints in the database field for relational, deductive and object-oriented databases [12, 44]. The general method is likely to be an extension of this work. A recent step in this direction (for the integrity checking strategy) is [13].

From declarative to imperative behaviour specifications. There are two different approaches for specifying the effect of the domain events of an IS: the *imperative* and the *declarative* approaches [56]. In an imperative specification, the conceptual modeller explicitly defines the set of changes (insertions of entities and relationships, updates of attribute values, etc.) to be applied over the IB. In a declarative specification, a contract for each domain event must be provided. The contract consists of a set of pre and postconditions. A precondition defines a set of conditions on the event input and the IB that must hold when the domain event is issued, while postconditions state the set of conditions that must be satisfied by the IB at the end of the domain event.

In conceptual modeling, the declarative approach is preferable since it allows a more abstract and concise definition of the event effect and conceals all implementation issues [56]. Nevertheless, in order to execute the conceptual schema, these declarative specifications must be automatically transformed into their equivalent imperative specifications. The main problem of declarative specifications is that they may be non-deterministic, i.e. there may be several possible states of the IB that verify the postcondition of a contract. This implies that a declarative specification may have several equivalent imperative versions, which hampers the transformation process.

Up to know, there is no general method that automatically provides this translation. Current solutions are mainly limited to deal with the *frame problem* [7], which discusses the possible IB states for types that are not referred to in the event contract. The automatic transformation for types that do appear in the contract needs further investigation.

Reusability. The possibility of reusing previously developed software pieces in the implementation of a new IS is one of the long-standing goals in the software community. In CSCD, reusability could help to reduce the effort required to transform the conceptual schema into an appropriate set of software components by means of studying the commonalities between the schema and a given set of existing software elements [2].

The main obstacle to a broader adoption of the reusability goal is the problem of selecting the right software component/s to reuse. Currently, the selection process is not completely automatic and requires a formal definition of the software components and a semantic comparison between the components and the conceptual schema [43]. This kind of analysis is still under development, particularly for two specific types of software components: commercial-off-the-shelf (COTS) components and web services. Ideally, the selection process should also consider possible non-functional requirements of the IS and the cost of integrating the selected component into the rest of the system.

4.3 Evolving Schemas

Concept evolution. The most fundamental changes to a conceptual schema are the addition or removal of concepts (entity, relationship or event types or states in state machines) and the addition or removal of edges in the concept generalization hierarchy. In CSCD, evolution must be automatically propagated to the logical level [26]. Therefore, these changes must be propagated to the logical schema(s) of the database(s) (and/or to other software components generated for the execution of the CS) and to its (their) instances. Changes to the generalization hierarchy may induce a change (increase or decrease) in the population of some concepts such that certain integrity constraints are violated. The IS should (efficiently) detect these violations and produce an appropriate response. Further work on these topics must take into account the considerable amount of existing work on database schema evolution, which focuses mainly on concept evolution [5].

Furthermore, concept evolution may also affect other elements in the conceptual schema. For instance, changes to a generalization hierarchy may affect the general integrity constraints defined in the schema (some constraints may become unnecessary while others may now be required). Additionally, the formal definition of constraints, derivation rules and domain events may need to be adjusted after a concept evolution since they may refer to elements that no longer exist in the schema or whose specification (cardinality, data type, changeability, etc.) has been changed during the evolution process.

Constraints evolution. Adding a constraint may turn the IB inconsistent. Changing a constraint may be considered as a removal (which cannot lead to any inconsistencies) plus an addition. When a constraint is added, the IS has to check whether or not the current IB satisfies it. For very large IBs, the checking procedure may need to be efficient. If one or more fragments of the IB violate the constraint, the IS has to produce a response (to reject the constraint, to ignore the inconsistency, to repair the fragment or to handle the fragment as an exception).

In the database field, the problem of adding constraints has been studied for particular constraints and database models [54]. In CSCD, we need to be able to deal with particular constraints (like cardinalities) but also with general constraints expressed in a conceptual modeling language, including base and/or derived types.

Derivability evolution. The derivability of entity and relationship types may change. A base type may become a derived type or vice versa. Fur-

thermore, a derivation rule may also change. Changing the derivability of a type may produce a change in its population and, indirectly, in that of other types. If the change affects a materialized type it must be recomputed. For large IBs, recomputation may need to be efficient. Changing the population of a type may also induce the violation of certain integrity constraints. The IS should (efficiently) detect these violations and produce an appropriate response.

Some work has been carried out on this topic [21], but much more needs to be done. A partially similar problem in the database field is that of "view adaptation" after view redefinition [24].

Completeness and correctness of the evolved schema. After evolving the CS, it is necessary to check that the conceptual schema is still complete and correct. This verification should be done efficiently. In particular, it is only necessary to consider the evolved subset of the schema (together with other schema elements that may have been affected by the evolution-induced effects).

Approaches for the efficient verification of evolved schemas focus on the detection of *consistency* (see, for example, [17]). These approaches state that a conceptual schema is consistent if it satisfies a set of integrity constraints (usually referred to as well-formedness rules) predefined by the conceptual modeling language used in the specification of the schema. These constraints restrict the possible structure of schemas defined with that modeling language. Efficiency is achieved by checking the relevant constraints on the evolved part of the schema. A constraint is relevant to an evolved schema if changes in the schema could induce a violation of this constraint.

Much work is required to efficiently verify other quality factors of the evolved schema.

4.4 Other Research Problems

Benchmarks for CSCD. In most areas of computer science (databases, computer architectures, programming and so on), an extensive set of benchmarks have been developed to test the performance (or the covering or any other property) of a method that addresses a research goal in that area. Benchmarks are also useful for comparing different proposals tackling the same goal.

In CSCD, benchmarks could help to measure the progress of the community regarding the different research goals presented in this study. Additionally, conceptual modellers could benefit from benchmarks when select-

ing a tool to specify the conceptual schemas. Several benchmarks are needed, depending on the research goal concerned.

Education for CSCD. When the conceptual schema is placed at the centre of the development process, the focus of software engineering education needs to be shifted from code-centric to model-centric. As expressed in [45], each role in the software development process requires appropriate education. In CSCD, the main role is that of the conceptual modeller. Therefore, we must develop appropriate teaching/learning techniques to leverage the modeling abilities of software engineering students and practitioners. This is a critical factor in the success of the CSCD approach.

We are currently witnessing an increase in the number of available modeling courses (both virtual and traditional face-to-face courses), particularly for the popular Unified Modeling Language. However, most of these courses focus on the notational aspects of modeling languages. Instead, in CSCD education, we must concentrate on clearly explaining the semantics of the different modeling constructs and how they can be combined to construct complete and correct conceptual schemas. A body of examples of "good" and "bad" schemas for well-known domains would therefore be very useful.

5 Conclusions

Conceptual schema-centric development (CSCD) is a reformulation of the goal of automating information systems building that highlights the central role of conceptual schemas in the automatic development of information systems. In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system evolution.

To achieve the CSCD goal, numerous research problems must be solved. The main purpose of this paper was to identify and comment on a list of sixteen open problems that should be included in a research agenda for CSCD.

We believe that this research agenda must be carried out before the CSCD approach can become widely used in practice.

Acknowledgements

We wish to thank the GMC group (Jordi Conesa, Dolors Costal, Cristina Gómez, Enric Mayol, Joan Antoni Pastor, Anna Queralt, Maria-Ribera

Sancho, Ruth Raventós and Ernest Teniente) for many useful comments on previous drafts of this paper. This work was partially supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIN2005-06053.

References

[1] Akoka, J., Comyn-Wattiau, I.: Entity-relationship and object-oriented model automatic clustering. Data & Knowledge Engineering, 20, 1996, pp. 87-117
[2] Basili, V., Briand, L.C., Melo, W.: How reuse influences productivity in object-oriented systems. Communications of the ACM 39 (10), 1996, pp. 104-116
[3] Boman, M., Bubenko, J.A. jr., Johannesson, P., Wangler, B.: Conceptual Modelling. Prentice Hall, 1997, p. 269
[4] Bernstein, P.A.: Applying Model Management to Classical Meta Data Problems. In Proc. CIDR 2003, pp. 209-220
[5] Banerjee, J., Kim, W., Kim, H-J., Korth, H.F.: Semantics and Implementation of Schema Evolution in Object-Oriented Databases. In Proc. ACM SIGMOD 1987, pp. 311-322
[6] Batini, C., Lenzerini, M., Navathe S.B.: A Comparative Analysis of Methodologies for Database Schema Integration. ACM Comput. Surv. 18 (4), 1986, pp. 323-364
[7] Borgida, A., Mylopoulos, J., Reiter, R.: On the frame problem in procedure specifications. IEEE Transactions on Software Engineering 21, 1995, pp. 785-798
[8] BRCommunity.com (Eds.): A Brief History of the Business Rule Approach. Business Rules Journal, 6 (1), January 2005
[9] Brinkkemper, S., Lindencrona, E., Sølvberg, A. (Eds.): Information Systems Engineering. State of the Art and Research Themes, Springer, 2000
[10] Calvanese, D., Lenzerini, M., Nardi, D.: Description Logics for Conceptual Data Modeling. In Chomicki, J., Saake, G. (Eds.): Logics for Databases and Information Systems. Kluwer, 1998, pp. 229-263
[11] Castano, S., de Antonellis, V., Fugini, M.G., Pernici, B.: Conceptual Schema Analysis: Techniques and Applications. ACM TODS, 23 (3), 1998, pp. 286-333
[12] Ceri, S.; Fraternalli, P.; Paraboschi, S.; Tanca, L. "Automatic Generation of Production Rules for Integrity Maintenance". ACM TODS, 19 (3), 1994, pp. 367-422
[13] Cabot, J., Teniente, E.: Incremental Evaluation of OCL Constraints. In Proc. CAiSE 2006, LNCS 4001, pp. 81-95
[14] Chen, P., Thalheim, B., Wong, L.Y.: Future Directions of Conceptual Modeling. In Proc. ER 1997, LNCS 1565, pp. 287-301
[15] Davis, K.H., Aiken, P.H.: Data Reverse Engineering: A Historical Survey. In Proc. Working Conference on Reverse Engineering, 2000, pp. 70-78

[16] DeRemer, F., Kron, H.: Programming-in-the-Large Versus Programming-in-the-Small. IEEE Trans. Software Eng. 2 (2), 1976, pp. 80-86
[17] Egyed, A. Instant consistency checking for the UML. In Proc. ICSE 2006, pp. 381-390
[18] Eshuis, R., Jansen, D.N., Wieringa, R.: Requirements-Level Semantics and Model Checking of Object-Oriented Statecharts. Requirements Engineering 7 (4), 2002, pp. 243-263
[19] Fowler, M.: Refactoring: Improving the design of existing code. Addison-Wesley, 1998, p. 464
[20] Gogolla, M., Bohling, J., Richters, M.: Validation of UML and OCL Models by Automatic Snapshot Generation. In Proc. UML 2003, LNCS 2863, pp. 265-279
[21] Gómez, C., Olivé, A.: Evolving Derived Entity Types in Conceptual Schemas in the UML. In Proc. OOIS 2003, LNCS 2817, pp. 33-45
[22] Gupta, A., Mumick, I. S.: Materialized Views. Techniques, Implementations and Applications. The MIT Press, 1999
[23] Gupta, H., Mumick, I.S.: Selection of Views to Materialize in a Data Warehouse. IEEE Trans on Knowledge and data engineering, 17 (1), 2005, pp. 24-43
[24] Gupta, A., Mumick, I.S., Ross, K.A.: Adapting Materialized Views after Redefinitions. In Proc. ACM SIGMOD 1995, pp. 211-222
[25] Harel, D.: Biting the Silver Bullet. Toward a Brighter Future for System Development. Computer, January 1992, pp. 8-20
[26] Hick, J-M., Hainaut, J-L.: Strategy for Database Application Evolution: The DB-MAIN Approach. In Proc. ER 2003, LNCS 2813, pp. 291-306
[27] ISO/TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and the Information Base, J.J. Van Griethuysen (Ed.), March 1982
[28] Lindland, O.I., Sindre, G., Sølvberg, A.: Understanding Quality in Conceptual Modeling. IEEE Software, March 1994, pp. 42-49
[29] Mays, R.G. “Forging a silver bullet from the essence of software”. IBM Systems Journal, 33 (1), 1994, pp. 20-45
[30] Mellor, S.J., Balcer, M.J.: Executable UML. A Foundation for Model-Driven Architecture. Addison-Wesley, 2002, p. 368
[31] Mens, T., Tourwé, T.: A Survey of Software Refactoring. IEEE Trans. Software Eng. 30 (2), 2004, pp. 126-139
[32] Moody, D.L., Sindre, G., Brasethvik, T., Sølvberg, A.: Evaluating the Quality of Process Models: Empirical Testing of a Quality Framework. In Proc. ER 2002, LNCS 2503, pp. 214-231
[33] Moody, D.L., Sindre, G., Brasethvik, T., Sølvberg, A.: Evaluating the quality of information models: empirical testing of a conceptual model quality framework. In Proc. ICSE 2003, pp. 295-307
[34] Mylopoulos, J.: The Role of Knowledge Representation in the Development of Specifications. In Proc IFIP 1986, pp. 317-319
[35] Mylopoulos, J.: Information Modeling in the Time of the Revolution. Information Systems 23(3/4), 1998, pp. 127-155

[36] Olivé, A.: Derivation Rules in Object-Oriented Conceptual Modeling Languages. In Proc. CAiSE 2003, LNCS 2681, pp. 404-420
[37] Olivé, A.: Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research. In Proc. CAiSE 2005. LNCS 3520, pp. 1-15
[38] Parent, C., Spaccapietra, S.: Issues and approaches of database integration. Communications of the ACM 41 (5), 1998, pp. 166-178
[39] Pottinger, R., Bernstein, P.A.: Merging Models Based on Given Correspondences. In Proc VLDB 2003, pp. 826-873
[40] Rahm, E., Bernstein P.A.: A survey of approaches to automatic schema matching. VLDB Journal 10 (4), 2001, pp. 334-350
[41] Rich, C., Waters, R.C.: Automatic Programming: Myths and Prospects. Computer, August 1988, pp. 40-51
[42] Ross, R.G. (Ed.): The Business Rules Manifesto. Business Rules Group. Version 2.0, November 2003
[43] Schumann, J. M.: Automated Theorem Proving in Software Engineering, Springer, 2001, p. 228
[44] Schewe, K-D., Thalheim, B.: Towards a theory of consistency enforcement. Acta Informática 36, 1999, pp. 97-141
[45] Shaw, M.: Software Engineering Education: A Roadmap. In Future of Software Engineering, Proc ICSE 2000, pp. 371-380
[46] Shoval, P., Danoch, R., Balabam, M.: Hierarchical entity-relationship diagrams: the model, method of creation and experimental evaluation. Requirements Eng., 2004, 9, pp. 217-228
[47] Sølvberg, A.: Co-operative Concept Modeling. In [9], pp. 305-326
[48] Sowa, J.F.: Knowledge Representation. Logical, Philosophical and Computational Foundations. Brooks/Cole, 2000, p. 594
[49] Stumptner, M., Schrefl, M., Grossmann, G.: On the road to behavior-based integration. In Proc. APCCM 2004, pp. 15-22
[50] Steimann, F., Kühne, T.: Coding for the Code. ACM Queue, 3 (10), 2006, pp. 45-51
[51] Teichroew, D., Sayani, H.: Automation of System Building. Datamation, 17 (16), 1971, pp. 25-30
[52] Teniente, E., Urpí, T.: On the abductive or deductive nature of database schema validation and update processing problems. Theory and Practice of Logic Programming 3 (3), 2003, pp. 287-327
[53] Tonella, P., Potrich, A.: Reverse Engineering of Object Oriented Code. Springer, 2005, p. 210
[54] Türker, C., Gertz, M.: Semantic integrity support in SQL: 1999 and commercial (object-)relational database management systems. VLDB Journal, 10, 2001, pp. 241-269
[55] Wand, Y., Weber, R.: Research Commentary: Information Systems and Conceptual Modeling – A Research Agenda. Information Systems Research, 13 (4), 2002, pp. 363-376
[56] Wieringa, R.: A survey of structured and object-oriented software specification methods and techniques. ACM Computing Surveys 30, 1998, pp. 459-527

[57] Zhang, Y.: Test-Driven Modeling for Model-Driven Development. IEEE Software, September/October 2004, pp. 80-86