Conceptual Schema-Centric Development: A Grand Challenge for Information Systems Research

Antoni Olivé

Universitat Politècnica de Catalunya
Dept. Llenguatges i Sistemes Informàtics
Jordi Girona 1-3, 08034 Barcelona (Catalonia)
olive@lsi.upc.edu

Abstract. The goal of automating information systems building was stated in the sixties. Forty years later it is clear that the goal has not been achieved in a satisfactory degree. One of the problems has been the lack of standards in languages and platforms. In this respect, the recent efforts on standardization provide an opportunity to revive the automation goal. This is the main purpose of this paper. We have named the goal "conceptual schema-centric development" (CSCD) in order to emphasize that the conceptual schema should be the center of the development of information systems. We show that to develop an information system it is necessary to define its conceptual schema and that, therefore, the CSCD approach does not place an extra burden on developers. In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system evolution. To achieve the CSCD goal it is necessary to solve many research problems. We identify and comment on a few problems that should be included in a research agenda for CSCD. Finally, we show that the CSCD goal can be qualified as a grand challenge for the information systems research community.

1 Introduction

The goal of automating information systems building was stated in the sixties [49]. Since then, the goal has been reformulated many times, but the essential idea has remained stable: To automatically execute the specification of an information system in its production environment.

Forty years later, it is obvious that the goal has not been achieved in a satisfactory degree. In the past, there has been a lot of research in "automatic programming systems", "automatic code generation", "integrated computer-aided software engineering tools" and the like. Currently, several projects are pursuing the same goal, such as those described in [32, 42]. The progress has been impressive, but it is a matter of fact that, in most current information systems development and maintenance projects, the design, programming and testing ac+tivities still require substantial manual effort. In the professional information systems community, two of

O. Pastor and J. Falcão e Cunha (Eds.): CAiSE 2005, LNCS 3520, pp. 1 – 15, 2005.

the currently most popular development approaches are the Unified Process [27] and the agile methods [4]. In neither of the two, automation of the system building plays a significant role.

It is then natural to pose the question: why has the goal not been achieved?. One possible explanation could be that the goal is unachievable. However, it is difficult to single out a characteristic in the goal that could make it so hard. The goal has a precise formulation. The size of the system that should be built is not larger than that of other systems our community has built. Probably, there are several successful software systems more complex than the one that should be built for automating the building of information systems.

Another possible explanation could be that the goal has not been considered worthwhile for the research or professional communities. However, this explanation is easily invalidated by observing the amount of research done or the large effort most organizations still spend in building their information systems.

The reason why the goal has not been achieved is that a number of important problems remain to be solved [43]. 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 automation goal. This is the main purpose of this paper.

We propose to call the goal "conceptual schema-centric development" (CSCD) in order to emphasize that the conceptual schema should be the center of the development of information systems. In conceptual modeling, a conceptual schema is basically the formal specification of functional requirements [29, 44]. In the next section, we briefly review the notion of conceptual schemas and analyze the nature of their relationship with information systems.

In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system evolution. More details are given in Section 3, where CSCD is also compared with other development approaches.

To achieve the CSCD goal it is necessary to solve many research problems. In Section 4 we identify and comment on a few problems that should be included in a research agenda for CSCD. Other relevant agendas have been presented in [12, 53].

The paper ends with an evaluation of the CSCD goal with respect to the grand challenges for computing research. According to Hoare, a grand challenge:

> "represents a commitment by a significant section of the research community to work together towards a common goal, agreed to be valuable and achievable by a team effort within a predicted timescale. The challenge is formulated by the researchers themselves as a focus for the research that they wish to pursue in any case." [25].

To be qualified as a grand challenge, a research goal has to meet a number of criteria. In section 5 we evaluate the CSCD goal according to these criteria.

2 Back to Basics

In this section, first we review the main functions of an information system (IS), and then we analyze the knowledge required by a particular IS to perform these functions. The analysis leads us to the definition of conceptual schemas [38].

2.1 Functions of an Information System

ISs 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 performs three main functions [6, 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 some 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 some generating condition is satisfied.

2.2 Knowledge Required by an Information System

To be able to perform the above functions, an IS requires some general knowledge about its domain, and knowledge about the functions it must perform. In the following, 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, then the IS has to know the entity and relationship types to be represented, and their current population. Some ISs may require to know that population also at some or all past time points. The entity and relationship types of interest are general knowledge about the domain, while their (time-varying) population is particular knowledge.

In conceptual modeling, we call Information Base (IB) the representation of the state of the domain in the IS. A non-temporal IB represents only the current state of the domain, while a temporal one represents also the state of the domain at any past time.

The representation of the state in the IB must be consistent. This is achieved by defining a set of conditions (called integrity constraints) and requiring that the IS satisfies them at any time. Such integrity constraints are general knowledge about the domain.

The domain state is not static. Most domains change through time, and therefore their state changes too. A domain event is a state change that consists of a set of elementary changes in the population of entity or relationship types that are considered as a single change in the domain. When the state of a domain changes, the

IB must change accordingly. The IS must know the types of the possible domain events and the effect of each event instance on the IB [39]. This is also general knowledge about the domain.

If the informative function has to communicate some information or commands on request, then the IS must know the possible request types and the output it has to communicate. On the other hand, if there are generated communications then the IS must know the generating condition and the output it has to communicate when it is satisfied. Note that this knowledge is not about the domain, but about the functions required to the IS.

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 some 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 some action must be performed when a generating condition is satisfied then the IS must know this condition and the action it has to perform. Note again that this knowledge is not about the domain, but about the functions required to the IS.

2.3 Conceptual Schemas

The first conclusion of the above analysis is that in order to perform its required functions, an IS must have some general knowledge about its domain and about the functions it has to perform. In the information systems field, such knowledge is called the Conceptual Schema[1] (CS).

Every IS embodies a CS [33, 31, 48 (p.417+)]. Without a CS, an IS could not perform any useful function. Therefore, developers need to know the CS in order to develop an IS.

Unfortunately, however, the need of CSs in IS development is often overlooked or ignored. The consequences are negative, both in theory and in practice. To help remedy the situation we propose to reformulate the need of CSs as a principle, that we propose to call the *Principle of Necessity*:

> "*To develop an information system it is necessary to define its conceptual schema*".

The principle of necessity can be seen as a consequence of the *100 Percent Principle* stated in the report [26]:

> "*All relevant general static and dynamic aspects, i.e. all rules, laws, etc. of the universe of discourse should be described in the conceptual schema. The information system cannot be held responsible for not meeting those described elsewhere, including in particular those in application programs*."

[1] In the fields of Knowledge Representation and Semantic Web the name given to an (almost) equivalent concept is Ontology [38].

The same report stated also the *Conceptualization Principle*, which says that:

> "*A conceptual schema should only include conceptually relevant aspects, both static and dynamic, of the universe of discourse, thus excluding all aspects of (external or internal) data representation, physical data organization and access as well as all aspects of particular external user representation such as message formats, data structures, etc*."

The main purpose of the activity of conceptual modeling is to elicit the CS of the corresponding IS. Given that, as we have seen, any useful IS needs a CS, we easily arrive to the conclusion that conceptual modeling is an essential activity of information systems development.

The CS must always exist, but it may take several forms, both externally and internally to the IS. Externally, the CS may be mental (exists only in the developers' heads) or explicit (written in some conceptual modeling language). When it is explicit, the CS documents the common understanding that users, analysts, and designers have about the domain and the functions imposed to the IS [29]. Internally, the CS may be diffused among the code of the IS, or be an explicit executable component. This paper advocates explicit and executable CSs.

3 Conceptual Schema-Centric Development

In this Section we reformulate the vision of a conceptual schema-centric development (CSCD) of information systems. We first present the characteristics of CSCD and then we compare it with some of the current development approaches.

3.1 Characteristics

The conceptual schema-centric development of an information system has three main distinguishing characteristics, that we call *Explicit, Executable* and *Evolving Schema*.

Explicit Schema. Once the functions of the IS have been determined there is an explicit, complete, correct and permanently up-to-date conceptual schema, written in a domain-independent, formal and declarative language. There is a development environment with tools that facilitate the validation, testing, reuse and management of (large) schemas.

Executable Schema. The schema is executable in the production environment. This may be achieved by an automatic and complete transformation of the CS into software components (including the database schema) written in the languages required by the production environment, or by the use of a virtual machine running on top of that environment. In either case, the CS is the only description to be defined. All the others are internal to the system, and need not be visible externally.

According to the conceptualization principle, CSs exclude all aspects related to information presentation. Therefore, the presentation layer of an IS is outside the scope of CSCD, although it may be based on the CS [16].

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

3.2 Comparison with Other Approaches

The CSCD approach may be used either as an alternative to, or in conjunction with, other development approaches. In what follows we give examples of the two cases, taking as reference some of the currently most popular development approaches.

Architecture-Centric. One of the distinguishing aspects of the Unified Process [27] is that it is architecture-centric. This means that the system's architecture is used as a primary artifact for conceptualizing, constructing, managing, and evolving the system under development. An architecture-centric approach develops the functions (use cases) and the system's architecture in parallel. The rationale is that the functions drive the architecture, but at the same time the architecture guides the functions. On the other hand, an early focus on architectural issues is necessary for an iterative and incremental development process.

In the CSCD approach, the functions of the system and the CS are determined without taking into account architectural issues. The architecture of the system is either predefined or generated automatically. In either case, it is likely that the architecture is based on one or more architectural patterns widely used in the information systems field.

Test-Driven Development. Test-Driven Development (TDD) is one of the core practices of Extreme Programming (XP), a well-known agile method. TDD is an iterative process. Each TDD cycle is composed of five steps: (1) Write a test that defines how a part of the software should behave; (2) Run the test and see that it fails; (3) Make a little change; (4) Run the test and succeed; and (5) Refactor to remove duplication or any other problems that were introduced to get to test run [5].

CSCD can be used in conjunction with TDD. In CSCD, a test is a sequence of action requests along with comparisons of actual results with expected results. Running a test is straightforward because the schema is executable. Changes and refactoring are done at the schema level [54].

Model-Driven Architecture. The OMG's Model Driven Architecture (MDA) defines an approach to system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. In the MDA there are two kinds of models: Platform Independent Models (PIM) and Platform Specific Models (PSM). The PIMs provide formal specifications of the structure and behavior of the system that abstracts away technical details. A conceptual schema is a PIM. The PSMs specify the system in terms of the implementation constructs that are available in one specific implementation technology. Both PIMs and PSMs are expressed in the UML [40, 41].

One of the key features of MDA is the notion of mapping. A mapping is a set of rules and techniques used to transform a model into another one. A PIM to PSM

mapping is used when the PIM is sufficiently refined to be projected to the execution infrastructure. The projection is based on the platform characteristics. Going from the conceptual to a domain layer in the J2EE platform is an example of PIM to PSM mapping. The goal of the MDA is to make the transformation from PIMs to PSMs as automatically as possible. This goal is expected to be achieved in many cases.

When the conceptual modeling language is the UML, the CSCD approach is MDA-compliant. However, in some respects CSCD is broader than (the current version of) MDA because it includes not only the initial development of information systems, but also their evolution.

Domain-Driven Design. The *domain-driven design* approach [15] advocates the use of the CS as the domain layer. The idea is that if the CS is executable in the platform in which the IS is implemented, then the CS can be the domain layer. The objective is that the domain layer is literally the CS, without any mapping between both. To make such a close correspondence of CS and design possible, it is essential to use languages that serves both purposes.

The approach works well when it is acceptable to express the CS in an implementation language, such as Java. In general, this is not the case. CSs need to represent knowledge declaratively, while it must be represented procedurally in the domain layer. For example, declarative constraints, dynamic and multiple classification, operation specifications by pre/postconditions or state transition diagrams do not have a literal representation in object-oriented programming languages. Different language constructs and different languages are needed in the CS and in the domain layer.

When the conceptual modeling language used is declarative and executable in the production environment, the CSCD and the domain-driven design approaches coincide. However, in some respects CSCD is broader than domain-driven design because it includes not only the initial development of information systems, but also their evolution.

4 Towards a Research Agenda for CSCD

CSCD is still a research goal. There are many research problems that must be solved before CSCD is a widely used approach in the development of industrial information systems. In this Section, we identify some of the research problems in each of the three characteristics of CSCD. We emphasize here some problems related to CSCD; see [12, 53] for other relevant research agendas in conceptual modeling.

4.1 Explicit Schemas

Very Large Conceptual Schemas. The CS of a large organization may contain thousands of entity types, relationship types, constraints, etc. The development and management of (very) large CSs pose specific problems not found in small CSs. 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 [13]. We need methods, techniques and tools to support designers and users in the development, reuse, evolution and understanding of large schemas.

So far, work on this topic has focused mainly on CSs for databases [1, 10, 47]. In CSCD we have to deal with ISs and we have to 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 some aspect of a business. From the information system perspective, business rules are elementary pieces of knowledge that define or constrain the contents and the change of the IB. 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 [7, 45]. Given that business rules are part of CSs, we can say that, in what respects to business rules, that community already follows the CSCD approach.

It is necessary and convenient to integrate the business rules and the CSCD approaches. It should be possible to extract the rules embedded in a schema, and to present them to users and designers in a variety of ways and languages, including the natural language. Automated support for this extraction and presentation is necessary. On the other hand, it should be easy to capture a particular rule and to integrate it into the schema. Automated support for this integration is desirable.

Complete and Correct Conceptual Schemas. Completeness and correctness are two of the quality factors of CSs. A complete CS includes all knowledge relevant to the IS. A correct CS contains only correct and relevant knowledge. Consistency is subsumed by validity and completeness [28]. In CSCD, completeness and correctness are the prime 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 CSs at least to the same extent that has been achieved in software.

There has been some work on testing CSs [23, 32, 17, 54]. There are automatic procedures for the verification of some properties of CSs in Description Logics [9]. Model checking is being explored as an alternative verification technique [14]. In all of these topics, a lot of work remains to be done [34].

4.2 Executable Schemas

Materialization of Derived Types. In general, CSs contain many derived entity and relationship types, with their corresponding derivation rules [36]. For efficiency reasons, some of these types must be materialized. The determination of the derived types to materialize should be as automatic as possible. On the other hand, changes in the population of base types may imply 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 to materialize in data warehouses [21] is highly relevant to the determination of the derived types to materialize in ISs. Similarly, the large body of work done in the incremental

maintenance of materialized database views [20] is highly relevant to the more general problem of change propagation in ISs.

Enforcement of Integrity Constraints. Most CSs contain a large number of integrity constraints [37]. The IS must enforce these constraints in an efficient way. This can be achieved in several ways [50]. 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 no constraint will be violated.

In CSCD, the analysis (whichever approach is taken) should be fully automatic and able to deal with any kind of constraint. A general method for this analysis does not exist yet. However, there has been a lot of research and development work in the enforcement of constraints in the database field, for relational, deductive and object-oriented databases [11, 51, 30]. The general method is likely to be an extension of this work.

4.3 Evolving Schemas

Concepts evolution. The most fundamental changes to a CS are adding or dropping concepts (entity, relationship or event types or states in state machines) and adding or dropping edges in the concept generalization hierarchy. These changes must be propagated to the logical schema(s) of the database(s) and to its (their) instances. The changes may affect also the existing derivation rules, constraints, event effects, etc. and, thus, the program code that implements them. Change propagation should be as automatic as possible. On the other hand, changes to the generalization hierarchy may imply a change (increase or decrease) in the population of some concepts such that some integrity constraints are violated. The IS should (efficiently) detect these violations and produce an appropriate response.

There has been an impressive amount of work on database schema evolution, focusing mainly on concepts evolution [3]. In CSCD the evolution starts at the conceptual level and it must be automatically propagated to the logical level [24]. Furthermore, more work is needed on the impact of changes to the generalization hierarchy on general integrity constraints.

Constraints evolution. Adding a constraint may make the IB inconsistent. Changing a constraint can be seen as a removal (which cannot lead to any inconsistency) and 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 may need to be efficient. If one or more fragments of the IB violate the constraint, the IS has to produce some response (reject the constraint, ignore the inconsistency, repair the fragment or handle the fragment as an exception).

In the database field, the problem of adding constraints has been studied for some particular constraints and database models [52]. 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, involving both base and/or derived types.

Derivability evolution. The derivability of entity and relationship types may change. A base type may change to derived, or the other way round. Besides, a derivation rule may change. Changing the derivability of a type may imply a change in its population, and indirectly in that of other types. If the change affects a materialized type, then it is necessary to recompute it. For large IBs the recomputation may need to be efficient. On the other hand, changing the population of a type may induce the violation of some integrity constraints. The IS should (efficiently) detect these violations and produce an appropriate response.

There has been some work on this topic [18], but much more needs to be done. A partially similar problem in the database field is that of "view adaptation" after view redefinition [22].

5 Is This a Grand Challenge?

CSCD is a research goal. Even if aspects of the CSCD approach may be found in current information systems development projects, there is a long way to go until its full potential may be realized in industrial projects.

In this section we assess the CSCD as a research goal. To this end, we use two sets of criteria:

1) The key properties of a good long-range research goal, proposed by Jim Gray [19] .
2) The desirable properties of a research goal to qualify as a grand challenge, proposed by Tony Hoare [25].

5.1 A Good Long-Range Research Goal?

We evaluate first the CSCD research goal with respect to the five key properties of a good long-range research goal described in [19].

Understandable. The CSCD goal is very simple to state to people working in the information systems field. The goal may also be understood by the general public. The distinction between specification and implementation is sufficiently familiar, and this facilitates the understanding of the goal of easing the development and the evolution of information systems by focusing only on the specification, and leaving the implementation details to "the machine".

Challenging. It is not obvious how to achieve the goal. The ideas of automatic programming, code generation, model compilers, etc. have been around for a long time, but the results obtained so far are not sufficient for the needs of most

information systems. Progress made in the automatic evolution of information systems has been even less satisfactory.

Useful. If the goal is achieved, the results will be useful to most organizations.

Testable. The goal will be achieved when we have a system that (1) facilitates the development of complete, possibly very large, declarative, tested and verified conceptual schemas; (2) automatically generates efficient information systems from these schemas, in the chosen platforms; and (3) evolves (also automatically) existing information systems from the changes to their schemas.

Incremental. The goal can be decomposed into intermediate milestones, in several ways. There are several ambition levels in (1) each aspect of the support to the development of CSs: completeness, coping with largeness, declarativity, testing and verification; (2) automatic generation: extent covered, platforms supported, degree of automation and level of performance; and (3) automatic evolution: types of changes supported and degree of automation.

5.2 A Grand Challenge?

Tony Hoare [25] suggested a set of seventeen criteria that must be satisfied by a research goal to qualify as a grand challenge. These criteria include the five key properties indicated above. The additional criteria relate to the maturity of the discipline and the feasibility of the project. In what follows we evaluate the CSCD goal with respect to the additional criteria. The order of the criteria is not significant.

Fundamental. Design, implementation and evolution of ISs is a fundamental concern in the information systems engineering field.

Astonishing. Most of the general public, and even many information systems professionals, are unaware of the possibility that computers might generate their own ISs and, above all, that changes to them can be done automatically.

Revolutionary. If the goal is achieved, it will lead to a radical change in the way ISs are developed and maintained in most organizations.

Research-directed. Significant parts of the goal can be achieved only by the methods of research. Among them there are improving conceptual modeling languages, devising methods for dealing with largeness, verification of CSs, generating efficient code from highly-declarative knowledge specification and the propagation of CS changes down to implementation.

Inspiring. The goal has the support from (almost) the entire information systems research community.

International. The goal has an international scope. Researchers from all around the world can participate in it.

Historical. The goal of automating information systems building was already formulated in the late sixties [35, 49] and since then it has been reformulated many times [8, 23, 42, 19]. The goal of evolving information systems from their specifications was already stated in the late seventies [46, 2] and, as before, it has been reformulated many times.

Feasible. The goal is now more feasible than ever. The progress recently made in standardization of languages and platforms has boosted industry interest in the goal. The basic constructs of conceptual modeling languages are already well known. It is feasible to improve current development environments. There is a lot of experience in code generation. Data persistence techniques and patterns are well known and successfully used in databases and in many data management layers. Many of the techniques of constraint enforcement, view definition and materialization, and schema evolution developed in the database field could be adapted to the broader context of ISs.

Cooperative. The work can be parceled out to teams working independently on (among others) conceptual modeling languages, testing and verification tools, persistence management, constraint enforcement and derived types.

Competitive. CSCD encourages and benefits from competition among teams. Most of the problems in the research agenda admit several solutions, with different range of application and/or efficiency.

Effective. The promulgation of the CSCD challenge is intended to cause a shift in the attitudes and activities of the relevant academic and professional communities. Development teams could consider the explicit definition of CSs a necessary step that needs to be done at professional quality level, using the right development environment. IS students could learn that CSs are closer to "working software" than to just "documentation". Researchers in conceptual modeling could focus on the theoretical issues to be solved in order to build CS development environments for professional use. Researchers of constraint enforcement, materialized views and evolution in databases could accept the challenge of extending current solutions to the broader context of ISs.

Risk-Managed. The main critical risks to the project arise from difficulties in achieving automatic systems evolution, in particular when evolution affects existing instances. Given that ISs need to evolve very often, the CSCD goal would fail (in theory and in practice) if it were not possible to define most of the changes only at the conceptual level. An early and constant focus on the evolution issues is essential to the success of CSCD.

6 Conclusions

The main purpose of this paper has been to revive the goal of automating information systems building. We have named the goal "conceptual schema-centric development" (CSCD) in order to emphasize that the conceptual schema should be the center of the development of information systems.

We have shown that to develop an information system it is necessary to define its conceptual schema. In CSCD, conceptual schemas would be explicit, executable in the production environment and the basis for the system evolution. To achieve the CSCD goal it is necessary to solve many research problems. We have identified and made some comments on a few problems that should be included in a research agenda for CSCD.

Finally, we have shown that the CSCD goal can be qualified as a grand challenge for the information systems research community.

Acknowledgements

I wish to thank the GMC group (Jordi Cabot, Jordi Conesa, Dolors Costal, Xavier de Palol, Cristina Gómez, Anna Queralt, Maria-Ribera Sancho, Ruth Raventós and Ernest Teniente) for many useful comments to previous drafts of this paper. This work has been partially supported by the Ministerio de Ciencia y Tecnologia and FEDER under project TIC2002-00744.

References

[1] Akoka, J.; Comyn-Wattiau, I. "Entity-relationship and object-oriented model automatic clustering". Data & Knowledge Engineering, 20 (1996), pp. 87-117.

[2] Balzer, R.; Cheatham, T.E.; Green, C. "Software Technology in the 1990's: Using a New Paradigm", IEEE Computer, 1983, pp. 16-22.

[3] Banerjee, J.; Kim, W.; Kim, H-J.; Korth, H.F. "Semantics and Implementation of Schema Evolution in Object-Oriented Databases". Proc. ACM SIGMOD 1987, pp. 311-322.

[4] Beck, K. Extreme Programming Explained. Embrace Change. Addison-Wesley, 2000, 190 p.

[5] Beck, K. Test-Driven Development By Example. Addison-Wesley, 2003, 220 p.

[6] Boman, M.; Bubenko, J.A. jr.; Johannesson, P.; Wangler, B. Conceptual Modelling. Prentice Hall, 1997, 269 p.

[7] BRCommunity.com (Eds.) "A Brief History of the Business Rule Approach", Business Rules Journal, 6(1), January 2005.

[8] Bubenko, J.A. jr. "Information Systems Meyhodology – a Research View". In Olle, T.W.; Sol, H.G.; Verrijn-Stuart,A.A. (Eds.) Information Systems Design Methodologies: Improving the Practice. North-Holland, 1986, pp. 289-318.

[9] 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.

[10] Castano, S.; de Antonellis, V.; Fugini, M.G.; Pernici, B. "Conceptual Schema Analysis: Techniques and Applications". ACM TODS, 23(3), 1998, pp. 286-333.

[11] Ceri, S.; Fraternalli, P.; Paraboschi, S.; Tanca, L. ”Automatic Generation of Production Rules for Integrity Maintenance”, ACM TODS, 19(3), September 1994, pp. 367-422.
[12] Chen, P.; Thalheim, B.; Wong, L.Y. “Future Directions of Conceptual Modeling”. Proc. ER 1997, LNCS 1565, pp. 287-301.
[13] DeRemer, F.; Kron, H. “Programming-in-the-Large Versus Programming-in-the-Small”. IEEE Trans. Software Eng. 2(2) 1976, pp. 80-86.
[14] 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.
[15] Evans, E. Domain-Driven Design. Tackling Complexity in the Heart of Business Software. Addison-Wesley, 2003.
[16] Fons, J.; Pelechano, V.; Albert, M.; Pastor, O. “Development of Web Applications from Web Enhanced Conceptual Schemas”. Proc. ER 2003, LNCS 2813, pp. 232-245.
[17] Gogolla, M.; Bohling, J.; Richters, M. "Validation of UML and OCL Models by Automatic Snapshot Generation". Proc UML 2003, LNCS 2863, pp.265-279.
[18] Gómez, C.; Olivé, A. “Evolving Derived Entity Types in Conceptual Schemas in the UML”. Proc. OOIS 2003, LNCS 2817, pp. 33-45.
[19] Gray, J. “What Next?. A Dozen Information-Technology Research Goals”. Journal of the ACM, 50(1), 2003, pp. 41-57.
[20] Gupta, A.; Mumick, I.S. Materialized Views. Techniques, Implementations and Applications. The MIT Press, 1999.
[21] Gupta, H.; Mumick, I.S. “Selection of Views to Materialize in a Data Warehouse”. IEEE Trans on Knowledge and data engineering, 17(1), January 2005, pp. 24-43.
[22] Gupta, A.; Mumick, I.S.; Rao, J.; Ross, K.A. “Adapting Materialized Views after Redefinitions: Techniques and a Performance Study”. Information Systems 26 (2001), pp. 323-362.
[23] Harel, D. “Biting the Silver Bullet. Toward a Brighter Future for System Development”. Computer, January 1992, pp. 8-20.
[24] Hick, J-M.; Hainaut, J-L. “Strategy for Database Application Evolution: The DB-MAIN Approach”. Proc. ER 2003, LNCS 2813, pp. 291-306.
[25] Hoare, T. “The Verifying Compiler: A Grand Challenge for Computing Research”. Journal of the ACM, 50(1), 2003, pp. 63-69.
[26] ISO/TC97/SC5/WG3 Concepts and Terminology for the Conceptual Schema and the Information Base, J.J. Van Griethuysen (ed.), March 1982.
[27] Jacobson, I.; Booch, G.; Rumbaugh, J. The Unified Software Development Process. Addison-Wesley, 1999, 463 p.
[28] Lindland, O.I.; Sindre, G.; Solvberg, A. “Understanding Quality in Conceptual Modeling”. IEEE Software, March 1994, pp. 42-49.
[29] Loucopoulos, P. “Conceptual Modeling”. In Loucopoulos, P.; Zicari, R. (Eds). Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development. Wiley, 1992, pp. 1-26.
[30] Mayol, E.; Teniente, E. "Consistency preserving updates in deductive databases", Data & Knowledge Eng., 47(1), 2003, pp. 61-103.
[31] Mays, R.G. “Forging a silver bullet from the essence of software”. IBM Systems Journal, 33(1), 1994, pp. 20-45.
[32] Mellor, S.J.; Balcer, M.J. Executable UML. A Foundation for Model-Driven Architecture. Addison-Wesley, 2002, 368 p.
[33] Mylopoulos, J. “The Role of Knowledge Representation in the Development of Specifications”. Proc IFIP-86, North-Holland, 1986, pp. 317-319.

[34] Mylopoulos, J. “Information Modeling in the Time of the Revolution”. Information Systems 23(3/4), 1998, pp. 127-155.
[35] Nunamaker, J. F. “A methodology for the design and optimization of information processing systems”. Proc. Spring Joint Computer Conference, 1971, pp. 283-294.
[36] Olivé, A. “Derivation Rules in Object-Oriented Conceptual Modeling Languages”. Proc. CAiSE 2003, LNCS 2681, pp. 404-420.
[37] Olivé, A. “Integrity Constraints Definition in Object-Oriented Conceptual Modeling Languages”. Proc. ER 2003, LNCS 2813, pp. 349-362.
[38] Olivé, A. “On the Role of Conceptual Schemas in Information Systems Development”. Proc. Ada Europe 2004, LNCS 3063, pp. 16-34.
[39] Olivé, A. “Definition of Events and Their Effects in Object-Oriented Conceptual Modeling languages”. Proc. ER 2004, LNCS 3288, pp. 136-149.
[40] OMG. “Model Driven Architecture (MDA)”. Document number ormsc/2001-07-01. 2001.
[41] OMG. “MDA Guide Version 1.0.1”. OMG Document omg/2003-06-01. 2003.
[42] Pastor, O.; Gómez, J.; Insfrán, E.; Pelechano, V. "The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming", Information Systems, 26(7), 2001, pp. 507-534.
[43] Rich, C.; Waters, R.C. “Automatic Programming: Myths and Prospects”. Computer, August 1988, pp. 40-51.
[44] Rolland, C.; Prakash, N. “From conceptual modeling to requirements engineering”. Annals of Software Engineering, 10(2000), pp. 151-176.
[45] Ross, R.G. (Ed.) “The Business Rules Manifesto”. Business Rules Group. Version 2.0, November 2003.
[46] Ruth, G. R. "Automatic programming: Automating the software system development process", Proceedings of the 1977 annual conference, pp: 174 – 180.
[47] Shoval, P.; Danoch, R.; Balabam, M. “Hierarchical entity-relationship diagrams: the model, method of creation and experimental evaluation”, Requirements Eng. (2004) 9: 217-228.
[48] Sowa, J.F. Knowledge Representation. Logical, Philosophical and Computational Foundations. Brooks/Cole, 2000, 594p.
[49] Teichroew, D.; Sayani, H. “Automation of System Building”, Datamation, 17(16), August 1971, pp. 25-30.
[50] 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), pp. 287-327 (2003).
[51] Thalheim, B. Entity-Relationship Modeling. Foundations of Database Technology. Springer, 2000, 627 p.
[52] Türker, C.; Gertz, M. “Semantic integrity support in SQL:1999 and commercial (object-) relational database management systems”. The VLDB Journal, 10, 2001, pp. 241-269.
[53] Wand, Y.; Weber, R. “Research Commentary: Information Systems and Conceptual Modeling – A Research Agenda”. Information Systems Research, 13(4), December 2002, pp. 363-376.
[54] Zhang, Y. “Test-Driven Modeling for Model-Driven Development”, IEEE Software, September/October 2004, pp. 80-86.