BRNO UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering and Computer Science
Department of Computer Science and Engineering

METHODS FOR DESIGN OF AN OBJECT ORIENTED DATABASE SYSTEM

Ing. Karel Obluk

Supervisor: Prof. Ing. Jan M. Honzík, CSc.
Opponents: Prof. Ing. Tomáš Hruška, CSc.
Prof. Ing. Jiří Šafařík, Ph.D.

TABLE OF CONTENTS

1 INTRODUCTION

During the last decade data management technology has entered a brand new world of applications like CAD/CAM, GIS, multimedia and others that pose different requirements on the database management system. Moreover, even business applications now require more sophisticated technologies, such as active rules support, distributed management or data mining. The object-oriented technology is one of the major candidates to fulfil all the requirements posed by these applications. In this work I summarize various methods used for the design of object-oriented database systems and I also propose some new methods and algorithms, mainly in the area of active rules processing and transaction models.

2 STATE OF THE ART

2.1 Object model and Persistence

2.1.1 Object model

One of the most important object model concepts is the support for multiple type objects. From the modeling perspective, this comports best with the reality and makes it possible for an object instance to play different *roles* during its lifetime. This support allows for some very useful constructs and it is also quite important for some advanced features. Closely coupled with the object model issues is the problem of object persistence. Two most noticeable features that distinguish object oriented database systems from object-oriented languages are *persistence independence* and *persistence orthogonality*.

2.1.2 Object Identity

Object identity is one of the fundamental properties of an object. It identifies each object instance, all references are expressed by use of object identity. It is used also for collections, a complex object modeling and support functionality like indexes. The operation of dereference, i.e. the transformation of object identity to the object instance (its data) itself has to be designed very carefully. Implementation of this transformation has a big influence on the performance of the whole database system and any inefficiency in object dereference can dramatically impact all other components. The design and implementation of

object identity has a lot in common with an object allocation and clustering. There are basically two possible implementations of object identity – logical and physical.

2.1.3 Logical Storage Model

An object is essentially a conceptual entity and should provide high independence of the physical data. A user of the logical object (i.e. usually a programmer of an application) should not have to take into account the physical format of the object data and the mapping necessary to store the data on the disk. This mapping problem is very well known from all database systems, no matter if they are relational or object. In object-oriented databases, however, there exist two kinds of relationships: between types and between object instances – composition. Object data can be grouped in their physical containers together to provide better access performance based on type relations or on composition relation. This grouping is usually called *object clustering*. There are basically three storage models for object clustering: the *Decomposition Model*, the *Normalized Storage Model* and the *Direct Storage Model*. Quite often, a mixture of two models is used to achieve best results.

2.1.4 Physical Storage Model

Independently of the logical storage model and implementation of the object identity, there are two basic ways of physical storage implementation. The server that transfers whole data pages to the client is called *page server*. Page server groups object data in pages and those pages are usually the basic units for caching and locking algorithms. If, on the other hand, the server provides also query optimization and an object manager, it is able to provide exactly the object instance that the client required. Those servers are usually called *object servers*. Currently, it is still not clear which strategy is better. Experience from various systems is different and whilst some authors argue that page server architecture is better, others claim that object servers provide higher throughput. There still exist too few implementations of object-oriented database systems to support either of these two opinions.

2.2 Garbage Collection

It is very well understood that any object instance has to be created first. It then exists in the database because one of the main object-oriented database

properties is the ability to support persistence. But when should such instance be deleted? There are generally two methods – *explicit object destruction* and *automatic garbage collection*. While explicit destruction is quite straightforward from the implementation point of view, the automatic garbage collection is much more complicated.

This method assumes that the object instance should exist until the last reference to it is removed. Once the last reference is broken, the object instance should be deleted automatically. This is very convenient for an application programmer and an end user as the system automatically collects the garbage, that is, removes all objects instances that are no more of any use. However, efficient implementation of automatic garbage collection is a very difficult task, mainly in database systems that support distributed processing. I describe several methods that are used most often and in more detail I discuss the method named *GC consistent cuts* that was proposed for the O_2 object-oriented database system.

2.3 Active Rules

Another important area of an object oriented database system is the support for active rules processing. Most of the current ECA system implementations consist of a separate engine built on top of an existing system. The ECA engine monitors *events* in the database and if a specified *condition* is met, the corresponding *action* takes place. The vast majority of the ECA systems concentrate on improving the condition evaluation engine and optimisation of the ECA systems performance.

2.4 Transactions

One of the most important areas of a database system is the transaction processing support. In contrast to relational database systems, object oriented databases have different needs and requirements. This crucial topic is covered in a great detail.

A transaction is an execution of a set of programs that access shared *recoverable resources* (e.g., data items). The concurrency transparency and failure atomicity of transactions are due to four properties – *Atomicity*, *Consistency*, *Isolation* and *Durability*. Together, these are commonly referred to as the *ACID test* for transactions.

2.4.1 Correctness Criteria

Correctness criterion indicates the notion of correctness that is employed to achieve a certain degree of *concurrency transparency* in the system. Full concurrency transparency means that each user transaction could be repeated with the same results. This could be achieved only if two concurrent transactions do not interfere and the implication is that the concurrent execution does not compromise database consistency. There are basically two correctness criteria: *serializability-based* and *nonserializable*.

Generally, there are two different ways of achieving serializability, based on the definition of "equivalence" between two histories. The two types of history equivalence that have been proposed are *view equivalence* and *conflict equivalence*. Since determining if two histories are view equivalent has been shown to be NP-complete problem, for practical reasons we are only concerned with conflict equivalence.

There are a number of serializability-based correctness criteria that differ in how they define a *conflict*. We concentrate on three criteria discussed in the literature: *commutativity*, *invalidation* and *recoverability.* Commutativity states that two operations conflict if the results of different serial executions of these operations are not equivalent. Invalidation defines a conflict between two operations not on the basis whether they commute or not, but according to whether or not the execution of one invalidates the other. The conflict relation established on the basis of recoverability seems to be identical to that established by invalidation, however there is no formal proof of this equivalence.

Serializability requires that the execution of each transaction must appear to every other transaction as a single atomic step. This requirement may be unnecessarily strong for many applications. The semantic information, in this case, is the *application semantics* rather than the *data semantics* and it could be used to weaken serializability and achieve a higher level of concurrency.

2.4.2 Selected Transaction Models

Transaction models, in one sense, specify the user interface to the transaction management system (TMS). The user is required to write the transactions according to the model restrictions. In another sense, the transaction model determines the capabilities of the TMS. Classification of the

transaction models could be based on the transaction structure or on the object structure. In this paper, we will use the transaction structure classification although there exist many extensions to each model and some of them simply do not fit strictly in any category.

According to this classification, we distinguish four broad categories in increasing complexity: *flat transactions*, *closed nested transactions*, *open nested transactions* such as sagas, and *workflow models* which, in some cases, are combinations of various nested forms.

2.4.3 Workflow Models

The most general transaction model is usually referred to as a *workflow model*, sometimes called also an *ATM model – advanced transaction model of activities*. The traditional ACID test properties are not sufficient or they are rather too restrictive for modern OODBMS systems. Although they are quite sufficient for modeling relatively simple and short activities, they are much less appropriate for modeling longer and more elaborate activities. It has been argued that even the nested transaction models are not sufficiently powerful to model current business activities. Therefore, new models, which are combinations of open and close transactions, have been proposed. The name *workflow model* is an answer to arguments that these cannot be properly called transactions since they break the ACID properties which are assumed to be the basic rule of all transactions.

A workflow model is modeled as an *activity*, which has open nesting semantics in that it allows other concurrent activities to see its results. The components of an activity can be either other activities or closed nested transactions. The components of a closed nested transaction can be only other closed nested transactions. The structure of an activity can be represented as a directed graph, although some models restrict it to be a hierarchy.

Since activities have the open nested semantics, it is necessary to define *compensating transactions* that semantically undo the effects of an aborted activity. However, it is not always possible to define a compensating transaction for an activity, mainly in the case of activities that interact with the real world (bank transfers, mail etc.).

2.5 Interoperability

Interoperability is becoming a very important issue nowadays when almost every computer is connected to the Internet or at least to a local area network (LAN). On the one hand, the object-oriented database systems with their encapsulation and abstraction capabilities may vastly assist in providing interoperability. On the other hand, they add to the complexity being yet another form of data repository. There is a number of ways that object-orientation can play a role in assisting interoperability. A common approach is to define a common object model that can be used to represent various objects from other repositories. A popular approach is to define an object model that is powerful enough to model others. Another alternative is to use composite objects that integrate data from various data sources.

3 OBJECTIVES

Main objective of this work was to summarize existing methods for design of an object oriented database systems and propose some new methods and extensions to the existing ones. These new approaches should improve the overall quality and usability of an object oriented database system with main focus on general information systems as opposed to specialised information systems (i.e. GIS, CAD, CAE and similar). The new features should allow both for better application development environment and for greater user friendliness.

Therefore, three important areas of an object oriented database system were targeted:

1. **Event system** – improve its modelling capabilities and reduce maintenance obstacles, mainly in distributed environment
2. **Data persistence** – improve user friendliness, allow for some common tasks that users often require
3. **Transaction support** – extend the workflow model so that it is more suitable for general IS applications and try to reduce its drawbacks as much as possible

Another significant topic is the three-tier architecture support. Quite many systems overlook the security issues tied to the data transmitions in a heterogenous network. Moreover, extensibility very often does not mean that

really any part of the system could be extended and support for new technologies could not be added to existing systems.

The author gained practical experience in design of both a relational and an object oriented database system. Some of the methods and approaches described in this work were proposed during the development of an OODBMS G2; extensions to the transaction system have their origin even back in the relational system G1.

The proposals in the area of data persistence utilize the input from users of the database systems VEMA and practical experience with various other database systems.

4 METHODS

There was no one universal method that could be used to study and evaluate the various approaches in all areas of this work. For many parts of this work it was necessary to collect user input and utilize experience achieved during development and usage of other systems. However, as most of current information systems are built using relational technology, it was necessary to combine knowledge of object-oriented approaches to software development together with the latest advances in database technology. For some areas it was possible to make use of theories and methods that are equally applicable to relational and object-oriented approaches.

For description of transactional properties and the workflow model I used the ACTA framework. Chrysanthis and Ramamritham originally proposed it for description of the *Saga model* properties. However, it is generally accepted as a very good formal apparatus for description of transaction systems in general.

Other areas of the work, like the event model, database areas and usage of object deltas, stem mainly from the user input together with facts obtained from various open sources. In this context, there is one fact that is really worth noting. Most books and papers describe what the designers and programmers think about what the users need and want and their assumptions and conclusions in this area are very often incorrect. On the other hand, the same books and papers describe very well what really was implemented or proposed and as such could be assumed as reliable sources. Quite the opposite, the users frequently want something completely different from what they are talking about. Moreover, it is interesting that very often this is not because of insufficient skills of the end user but rather because the user is “lying”. The

user is simply feeling shame that she wants something unusual, old-fashioned or contrariwise that she does not want something modern that all others want or already have. This symptom has to be seriously taken into account when collecting and evaluating the user input. It is necessary not only to simply ask the user but also to check all the answers several times and make sure that the conclusions are correct.

5 MAIN RESULTS

5.1 Events

5.1.1 Events: Integral Part of the Object Definition

I describe the role of active rules in a database system and I propose a new concept of active rules definition and processing system. In the proposed model, the object definition contains also definition of all possible events; corresponding actions are part of the object implementation. The system contains automatic support for some typical events like handling the *inverse* or *is-part-of* relation; other events can be added freely. The most important thing is that the model captures all the object behavior including that in irregular, or unusual conditions; and the object implementation contains all code that the object requires. This is very important in a distributed environment where the object "takes with itself" all the code it needs to behave properly and keep its state consistent. Moreover, if the definition or implementation of an object changes, all events and corresponding actions can be changed as well and at the same time.

5.1.2 Flexibility and Extensibility

One common argument against this model is its potential lack of flexibility when it comes to definition of business rules. However, I show that this need not be true. Using the concept of inheritance, the proposed model achieves both high flexibility and extensibility. Moreover, rules can easily be defined and implemented in the same development environment as the whole application; therefore, it is not necessary to create any specialized tools. The only problem is with the introduction of new rules in an existing database where already exist instances of objects that should be subclassed. This is easily solved in a system that supports multiple type objects, where the new

type can be added to existing object instances. In addition, the existing rules can easily be discarded by a simple removal of the type information from existing object instances. Systems that do not support multiple type objects have to cope with this problem by changing the type of all existing object instances, which can be quite time-consuming operation. This could be considered as the only serious disadvantage in systems that do not support multiple type objects and where the rules change quite frequently.

5.1.3 Gains

The most important advantage of the proposed event system is the improvement of modeling capabilities. At the same time, installation and maintenance requirements are reduced, mainly in distributed systems and heterogeneous environments. This notion of events that are part of the object model is one of the original contributions of this work; its core part was presented at the MOSIS ’99 conference.

5.2 Database Areas and Object Deltas

5.2.1 Database Areas

The idea of database areas is relatively simple and is very similar to a widely used concept of *namespaces* in programming languages. The motivation is purely practical and its main purpose is to help to an end user and to improve the user-friendliness of the whole database system. Database areas have no strict boundaries, they are not restricted by storage allocated to them or any other placement specification. They could be viewed upon as a kind of collection that is maintained automatically by the system. Each object instance is marked as a member of a certain database area. The database area is then recursively defined as a collection of all object instances that are marked as members of that area. The definition of database areas employs multiple inheritance, so that a new area can be defined as an ancestor of one or more existing areas. An end user can define a new area any time they need and select the active area.

5.2.2 Object Deltas

In a multiversion database, each object state is stored as a new value. All states of one object instance are linked together and the system can return to

older values. This offers an end user the possibility of accessing older versions directly, enabling object history inspection, and comfortable undo operations. It also provides very good support for transaction processing. The new state also does not have to be stored directly but it can rather be stored as a difference from previous value. Another area where object deltas can be used is the ECA rules processing in active databases, where the object modifier can have access to the previous object state. This idea was proposed independently for the G2 database system. The last but not least interesting feature of object deltas is the possibility to offer various views of the database to the end user, based on the time scale. For this purpose, we propose a concept of *object generations*. Object generation is defined by a time span in which the particular value is valid. Each object can have one or more generations and it is possible to access either the last one or any of the older ones. However, there are still some unsolved issues with object generations that have yet to be solved. The main problem is that of inter-generation object relations.

5.2.3 Gains

The concept of database areas together with object generations represents a quite novel approach and together with the usage of object deltas can improve the overall functionality, efficiency and user friendliness of the whole database system. This concept was proposed during the development of OODBMS G2 and I took part in its design and implementation. The usage of object deltas was designed independently of other similar solutions; the idea of database areas and object generations is completely new.

5.3 The SCAT Transaction Model

5.3.1 Disadvantages of the Workflow Model

The workflow model (ATM) has many properties that are required for object-oriented database systems and modern applications; however, the problem of compensating transactions cannot be overlooked. Sometimes, it is very difficult to specify a compensating transaction for an open activity. If compensating transactions are parts of the object model, as proposed in this work, some problems may be solved. But still relaxing the axiom of transaction isolation is a big issue. Therefore, it is necessary to provide some level of isolation for certain activities.

5.3.2 The SCAT Transaction Model

The proposed model tries to cope with this issue. Its main idea is to modify the workflow model slightly by allowing the activity itself to control if it is affected by other activities. In the proposed *Self-Controlling Activities Transaction Model* we define two kinds of activities. *Isolated Activity* is an activity that allows other activities to see its results but it does not use partial results of other activities. Isolated activities are always independent on all other activities. *Open Activity* is the "traditional" activity of the workflow model. It allows other activities to see its results and it uses results of other (committed) activities.

Other then this, the basic workflow model is unmodified. That means that activities may contain other activities or closed nested transactions. Components of a closed nested transaction may be only other closed nested transactions. The structure of an activity can be represented as a directed graph, although it seems to be quite sufficient to restrict it to be a hierarchy.

5.3.3 Gains

The SCAT method is one of the main contributions of this work. On practical examples I demonstrate its advantages and the benefits for an object-oriented database system and information systems built on top of it. The original idea of this concept was presented at the conference *Objekty '96*, its core part was also presented at the international conference *MOSIS '98*.

5.4 Extensibility and Scalability

Most current applications built upon relational or object-oriented database system use the client – server architecture. However, this model has several drawbacks. In the first place, the client – server model does not scale well. Secondly, the server is a critical spot of the whole system. Object orientation offers much better apparatus to overcome all these shortcomings. Because of location transparency, the objects can easily be distributed to different processors and systems. Therefore, it is possible to distribute the application processing to another system or even multiple systems. This is usually referred as a *three-tier architecture*, meaning client – application – server layers. Sometimes, it is also called *multi-tier architecture* to point out that the application layer does not have to be concentrated on one processor or computer system only.

Although the main focus of a database system is in the database engine itself, front-end considerations should not be overlooked as they can influence not only the overall notion of the system but also its reliability, efficiency and extensibility. In the work, I describe an approach that was used in the OODBMS G2. This system introduces the concept of *presentation environment* and *Abstract Dialogue Elements*, that enables a high degree of flexibility and extensibility. Both new presentation environments and abstract dialogue elements could easily be created and added to an existing system. This allows for a great extensibility and possibility to support new technologies that were not known in the time when the original system was created. The concept of independent presentation definitions also guarantees, that the user receives only the data they are permitted, thus maintaining a high degree of security. This is quite often an overlooked aspect in object oriented database systems, where usually whole object instances are sent to the client, even though only certain properties of an object are necessary.

6 CONCLUSION

In this work, I tried to summarize main methods and trends in the object-oriented database development. Having backed up with experience from development of the object-oriented database system G2, I proposed some novel methods and features, which, as I believe, can greatly enhance properties of an object-oriented database system. I assume the main contribution of this work to be the concept of event processing and the SCAT transaction model. In addition, I took part in design of some principles that were designed for the OODBMS G2 and that were described in this work, too. The most important are the ideas of object deltas, database areas and object generations. To my best knowledge, there exists no object-oriented database system exploiting these features. In addition, the idea of a multiversion database in the context of the transaction system support is to my best belief novel and has not been used yet in any commercial OODBMS implementation.

However, there are still many topics that deserve further studies and implementation considerations. In future, I would like to concentrate on further work on the proposed event system. There still exist some unsolved issues, mainly with respect to implementation. I would also like to work more on the concept of object generations that has still some important issues unsolved.

To summarize, I believe this work to be a contribution to the wide topic of the object-oriented database systems implementation. Not only does it summarize some of the most important and known design methods from all areas of database design, but it also offers some new directions and ways that could improve the quality and usefulness of the object-oriented database systems.

SHRNUTÍ

Nové aplikace z oblasti CAD/CAM, grafických informačních systémů nebo multimediálních aplikací, ale dnes již i klasické obchodní aplikace kladou na databázové systémy nároky, které se s pomocí tradičního relačního modelu vyplňují stále obtížněji. V poslední době je odbornou veřejností více akceptován fakt, že technologie objektově orientovaných databází by mohla být správnou volbou pro následující roky. Předkládaná práce popisuje metody používané v současné době při návrhu OO databázových systémů. Kromě shrnutí a zhodnocení používaných postupů obsahuje také některé nové metody a algoritmy zejména z oblasti aktivních pravidel a transakčního zpracování.

První část je věnovaná objektovému modelu a problémům dědičnosti. Rozsáhlejší diskuse je soustředěna zejména na podporu vícetypovosti, která přináší některé velmi užitečné vlastnosti a jejíž podpora je důležitá i pro některé metody a postupy navrhované v dalším textu. Kromě nesporných kladů rozebírám i některé problematické stránky vícetypovosti a možné varianty řešení. S objektovým modelem souvisí také otázky persistence, které jsou rozebrány zejména v kapitole 4. Vedle základního přehledu o možných metodách implementace objektové identity a modelu uložení dat se věnuji důkladněji také automatickému rušení objektů. V souvislosti s možnými přístupy k uložení dat diskutuji také dosud ne zcela vyjasněnou otázku volby mezi stránkovým a objektovým serverem, na kterou existují různé názory podpořené zcela rozdílnými praktickými zkušenostmi.

Další rozsáhlejší téma jsou databázové oblasti, rozdílové objekty (*object deltas*) a generace objektů. Koncepce databázových oblastí a generací objektů představuje nový přístup ve správě objektů a společně s metodou ukládání diferencí objektů znamená, dle mého názoru, významný přínos v práci s daty a v uživatelském komfortu. Na návrhu a implementaci těchto metod jsem se spolupodílel v rámci řešení objektově orientovaného databázového systému G2. Metoda ukládání diferencí objektů byla navržena a rozpracována nezávisle na jiných podobných řešeních; koncepce databázových oblastí a generací objektů je zcela původní.

Jedním z prvků, které by neměly v moderním databázovém systému chybět, je podpora tvorby aktivních pravidel. Objektové databáze pochopitelně nejsou výjimkou. Na základě praktických zkušeností s vývojem relačních i objektových databázových systémů předkládám v kapitole 3 netradiční pojetí aktivní databáze, které zahrnuje aktivní pravidla přímo do objektového modelu. To umožňuje jednak výrazné zvýšení kvality datového modelu a současně dramaticky snižuje nároky na instalaci a údržbu, a to zejména v distribuovaných systémech. Rozsáhlá diskuse je věnována implementačním problémům souvisejícím s navrhovaným modelem aktivních pravidel, otázkám pružnosti a rozšiřitelnosti a v neposlední řadě i jejich správě. Navrhovaný systém je zcela původní, jeho koncepce byla prezentována na konferenci MOSIS ’99.

Další velmi důležitou oblastí databázového systému je podpora transakčního zpracování. Oproti relačním databázím, objektově orientované databáze kladou na transakční systém jiné požadavky. Tomuto rozsáhlému tématu se věnuji v kapitole 5. Kromě výchozího popisu transakcí a jejich stručné kategorizace využívám aparát rámce *ACTA* pro popis některých podstatných vlastností transakcí a jejich vazeb. Na praktických příkladech pak demonstruji problémy hledání kompenzačních transakcí. Objektový model sice nabízí řešení některých problémů, se kterými se potýkají relační databáze, ani on však není schopen pokrýt všechny situace. Proto navrhuji modifikaci transakčního modelu aktivit – metodu SCAT (*Self Controlling Activities Transaction Model*). Tato modifikace umožňuje aktivitám řídit stupeň jejich izolace. Tak je možné řešit jednak problémy spojené s nemožností nalezení kompenzačních transakcí pro určité aktivity, jednak obecné problémy související s požadavkem izolace aktivit a zaručení opakovatelného čtení. Tato metoda je zcela původní a považuji ji za jeden z hlavních přínosů mé práce. Základní teze této metody byly publikovány na konferenci Objekty ’96, na mezinárodním fóru byla prezentována na konferenci MOSIS ’98.

Závěrečná část práce shrnuje některé zajímavé metody z oblasti distribuovaného zpracování, replikace a provozu v heterogenních sítích.

REFERENCES

1. Özsu, M. T., Dayal, U., Valduriez P. (ed.). *Distributed Object Management*. San Francisco: Morgan Kaufmann, 1994.
2. Carey, M. J., DeWitt, D. J. Of Objects and Databases: A Decade of Turmoil. In Vijayaraman, T. M., Buchmann, A. P., Mohan, C., Sarda, N. L. *VLDB'96*. Bombay, India: Morgan Kaufmann, 1996, p. 3-14.
3. Cattell, R. G. G., Barry, D. *The Object Database Standard: ODMG-2.0*. San Francisco: Morgan Kaufmann, 1997.
4. Alagić, S. The ODMG Object Model: Does it Make Sense? *OOPSLA '97*. Atlanta, Georgia, 1997. *SIGPLAN Notices 1997*, vol. 32, no. 10, p. 253-270
5. Beneš, M. Multiple Inheritance. In Hruška, T. *MOSIS'98*. Ostrava: 1998, MARQ., p. 9-14.
6. Hruška, T.: Multiple Class Objects. In Hruška, T. *MOSIS'98*. Ostrava: 1998, MARQ., p. 15-22.
7. Kappel, G., Retschitzegger, W. The TriGS Active Object-Oriented Database System – An Overview. *SIGMOD Record*. 1998, vol. 27, no. 3, p. 36-41.
8. Meo, R., Psaila, G., Ceri, S.: Composite Events in Chimera. In Apers, P. M. G., Bouzeghoub, M., Gardarin, G. *EDBT'96*, Avignon, France, 1996. *Lecture Notes in Computer Science*. 1996, vol. 1057, Springer, p. 56-76
9. Hanson, E. N., Khosla, S. An Introduction to the TriggerMan Asynchronous Trigger Processor. In Geppert, A., Berndtsson, M. *RIDS'97*, Skövde, Sweden, 1997. *Lecture Notes in Computer Science*. 1997 vol. 1312, Springer, p. 51-66
10. Benzaken, V., Delobel, C., Harrus, G.: Clustering Strategies in O2: An Overview. In Bancilhon, F., Delobel, C., Kanellakis, P. C. *Building an Object-Oriented Database System, The Story of O2*. Morgan Kaufamann, 1992
11. Day, M. Object Groups May Be Better Than Pages. *Workshop on Workstation Operating Systems*. Napa, California: IEEE Computer Society Press, 1993, p. 119-122

12. Liskov, B., Adya, A., Castro, M., Day, M., Chemawat, S., Gruber, R., Maheshwari, U., Myers, A. C., Shrira, L. Safe and Efficient Sharing of Persistent Objects in Thor. *SIGMOD '96*. Montreal, Canada, 1996. *SIGMON Record 1996*, vol. 25, no. 2, ACM Press, 1996, p. 318-329.
13. Dijkstra E. W., Lamport L., Martin A. J., Scholten C. S., Steffens E. F. M. On-the-fly garbage collection: an exercise in cooperation. *Communications of the ACM*, 1978, vol. 21, no. 11, p. 966-975
14. Maheshwari, U., Liskov, B. H. Fault-Tolerant Distributed Garbage Collection in a Client-Server Object-Oriented Database. In *PDIS '94*. Austin, Texas, 1994: IEEE-CS Press, 1994. p. 239-248
15. Maheshwari, U., Liskov, B. Collecting Distributed Garbage Cycles by Back Tracing. *PODC '97*. Santa Barbara, California, 1997: ACM Press, p. 239-248
16. Skubiszewski, M., Valduriez, P. Concurrent Garbage Collection in O_2. In Jarke, M., Carey, M. J., Dittrich, K. R., Lochovsky, F. H., Loucopoulos, P., Jeusfeld, M. A. *VLDB'97*. Athens, Greece, 1997: Morgan Kaufmann, p. 356-365
17. Hulse, D., Dearle, A. A Log-Structured Persistent Store. *ACSC '96*. Melbourne, Australia, 1996. Australian Computer Science Communications.
18. Sundermaier, A., Abdellatif, T. B., Dietrich, S., Urban, S. D. Object Deltas in an Active Database Development Environment. In Bry, F., Ramakrishnan, R., Ramamonaharao, K. *DOOD'97*. Montreaux, Switzerland, 1997. *Lecture Notes in Computer Science*. 1997 vol. 1341, Springer, p. 211-228
19. Vingralek, R., Ye, H., Breitbart, Y., Schek, H. Unified Transaction Model for Semantically Rich Operations. In Gottlob, G., Vardi, M. Y. *ICDT'95*. Prague, 1995. *Lecture Notes in Computer Science*. 1995 vol. 893, Springer, p. 148-161
20. Elmagarmid, A. K. *Database Transaction Models For Advanced Applications*. San Francisco, 1992, Morgan Kaufmann
21. Schwarz, K., Türker, C. *Investigating Advanced Transaction Models for Federated Database Systems*. Otto-von-Guericke-Universität Magdeburg, Magdeburg, 1997. 58 p.
22. Tesch, T., Wäsch, J. Transaction Support for Cooperative Hypermedia Document Authoring. *ERCIM '95*. Trondheim, Norway, 1995

23. Muth, P., Veijalainen, J., Neuhold, E. J. Extending Multi-Level Transactions for Heterogeneous and Autonomous Database Systems. *GMD Technical Report*, No. 739, Sankt Augustin, 1993
24. Rusinkiewicz, M., Klas, W., Tesch, T., Wäsch, J., Muth, P. Towards a Cooperative Transaction Model – The Cooperative Activity Model. In Dayal, U., Gray, P. M. D., Nishio, S. *VLDB'95*. Zurich, Switzerland: Morgan Kaufmann, 1995, p. 194-205
25. Peters, R. J., Lipka, A., Özsu, M. T., Szafron, D., Muńoz, A. TIGUKAT: A Uniform Behavioral Objectbase Management System. *The VLDB Journal*. 1995, vol. 4, no. 3, p. 445-492
26. Özsu, M. T., Valduriez, P. Distributed Database Systems: Where Are We Now? *IEEE Computer*. 1991, vol. 24, no. 8, p. 68-78
27. Soisalon-Soininen, E., Ylönen, T. Partial Strictness in Two-Phase Locking. In Gottlob, G., Vardi, M. Y. *ICDT'95*. Prague, 1995. *Lecture Notes in Computer Science*. 1995 vol. 893, Springer, p. 139-147
28. Vestenický, V. Replikace dat v distribuovaných systémech. In Merunka, V., Sklenář, V. *Objekty'98*. Praha: 1998, ČZU, p. 67-72
29. Lampa, P. CORBA. In Merunka, V., Sklenář, V. *Objekty'98*. Praha: 1998, ČZU, p. 105-126
30. Snášel, V., Sklenář, V. DCOM. In Merunka, V., Sklenář, V. *Objekty'98*. Praha: 1998, ČZU, p. 105-126
31. Obluk, K. Active Rules in Object-Oriented Database Systems. In Hruška, T. *MOSIS'99*. Ostrava: 1999, MARQ., p. 141-146.
32. Obluk, K. Metody návrhu databázového stroje pro objektově orientovaný systém správy dat. In Merunka, V. *Objekty '96*. Praha: 1996, ČZU, p. 49-59
33. Obluk, K. Transaction Support for Object-Oriented Database Systems. In Hruška, T. *MOSIS'98*. Ostrava: 1998, MARQ., p. 29-36.

PUBLICATIONS

1. Obluk, K. Active Rules in Object-Oriented Database Systems. In Hruška, T. *MOSIS'99*. Ostrava: 1999, MARQ., p. 141-146.
2. Obluk, K. Transaction Support for Object-Oriented Database Systems. In Hruška, T. *MOSIS'98*. Ostrava: 1998, MARQ., p. 29-36.
3. Obluk, K. Metody návrhu databázového stroje pro objektově orientovaný systém správy dat. In Merunka, V. *Objekty '96*. Praha: 1996, ČZU, p. 49-59
4. Obluk, K.: Metody návrhu a implementace databázového stroje OO databáze. *Tvorba Software'95*, Ostrava, 1995
5. Obluk, K.: Databázový stroj v objektově orientovaných databázových systémech. *Některé nové přístupy při tvorbě informačních systémů*, doprovodná konference mezinárodního veletrhu INVEX'95, Brno
6. Obluk, K.: OS/2 Warp - recenze operačního systému *Computer*. 1995, no. 2, Computer Press, Brno, 1995
7. Obluk, K.: CA vidí objektově - Recenze OO vývojového prostředí CA-VO. *Computer*. 1995, no. 8, Computer Press, Brno, 1995
8. Obluk, K., Honzíková, N., Honzík, J.M.: Visual Model of Controlled Respiration, *BioSignal '94*. Brno: 1994. FEECS Division, Technical University Brno, p. 136-138

   Obluk, K.: Genetické algoritmy. *Softwarové noviny*. 1993, Vol. 10, No. IV.
9. Havlíček, P., Krampol, R., Obluk, K.: Graphical Environment For Simulation Models. *MOSIS '92*. Olomouc, 1992
10. Obluk, K.: Knihovna pro paralelní zpracování více procesů a Modul interpretu. In Havlíček, P., Krampol, R., Obluk, K.: *Systém pro tvorbu výukových programů*. SVOČ, KIVT FE Brno, 1992
11. Obluk, K.: Grafika na počítačích PC. *XX. konference uživatelů malé výpočetní techniky*, Dům Techniky ČSVTS, Ostrava, 1990

12. Obluk, K.: Frequency Fall Detection Using the Single-Chip Microcontroller M68HC11. *Research Report of the IAESTE Project.* Department of Electrical and Electronic Engineering, The Queen's University of Belfast, Belfast, 1990
13. Obluk, K., Odehnal, P.: OKENA - soubor knihoven pro tvorbu uživatelského rozhraní programů v jazyce C. SVOČ, KIVT FE Brno, 1989
14. Obluk, K., Odehnal, P.: Grafika na počítačích PC. Smluvně zajištěno vydání - ZENITCENTRUM Beroun, 1988
15. Obluk, K.: *Překladač NEVADA FORTRAN - Příručka programátora*. JZD AK Slušovice, divize kybernetiky, Slušovice, 1986