Zuerst ersch. in : MFDBS 91 : 3rd Symposium on Mathematical Fundamentals of Database and Knowledge Base Systems, Rostock, Germany, May 6 - 9, 1991; proceedings / B. Thalheim ... (eds.). - Berlin; Heidelberg [u.a.] : Springer, 1991. - S. 377-388. - (Lecture notes in computer science ; 495). - ISBN 3-540-54009-1

The Background of the DASDBS & COSMOS Projects[1]

H.-J. Schek, M.H. Scholl, G. Weikum

ETH Zurich, Department of Computer Science
Information Systems - Databases
CH-8092 Zurich, Switzerland
e-mail: <lastname>@inf.ethz.ch

Abstract

This survey describes the conceptual framework behind the DASDBS and COSMOS projects. COSMOS is the current research program of the database research group at ETH Zurich. These activities are a natural follow-on of DASDBS, the Darmstadt Database System project at the Technical University of Darmstadt. While most emphasis in DASDBS was on a database *kernel* system, the project at ETH focuses on the *cooperation* between a database system and its environment. The environment consists of clients asking for database service and other systems offering service to the database system. The research objective is the exploration of the architecture of a COoperative System for the Management of ObjectS (COSMOS). In short, we are on the way "from the kernel to the cosmos".

1. DASDBS — A Family of Database Systems

1.1 Overall Architecture of DASDBS

DASDBS is designed as a family of database systems as shown in Figure 1. A storage management *kernel* serves as the lowest common denominator of the requirements of the various application classes, and a family of *application-oriented frontends* provides semantically richer functions on top of the kernel.

- The *DASDBS kernel* is a storage manager that provides only basic data management functions. In particular, data independence, access optimization, and integrity checking are not included. A simple but efficient query processing strategy and storage structures for NF^2 relations are implemented in the kernel. Thus, the kernel is already a low-level platform for building high-performance applications that would otherwise bypass a full-fledged DBMS. In this sense, the kernel can also be viewed as an extended file system.
- An *Application(-oriented) Object Manager* (AOM) together with the kernel implements a full-fledged DBMS tailored to an individual application class, such as business transactions, geographical information systems, office automation, or expert system support. The services provided in an AOM include an end-user and application programmer interface, query optimization, and access path management. For business applications based on flat relations, the AOM corresponds to the RDS of System R. An application using a DASDBS system will typically interface to its AOM, not to the kernel. The data models offered by distinct AOMs may be different from each other and from the NF^2 model of the kernel.

1.2 Nested Relations as a Data Model for Storage Structures

NF^2 relations as a formal description of *storage structures* were an important concept in the DASDBS project. In this subsection we describe this role of nested relations.

1. This survey is an excerpt from [SPSW90] and [SSW90].

Konstanzer Online-Publikations-System (KOPS)
URN: http://nbn-resolving.de/urn:nbn:de:bsz:352-201360

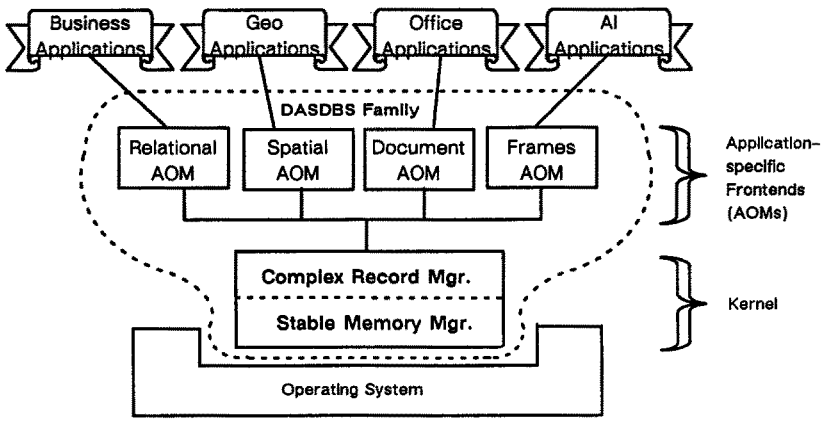

Fig.1: DASDBS Architecture

Physical Database Design and Impacts on Optimization

We were strongly convinced that a new system should not only have a more powerful data model at its interface to the applications, but also a clear and well understood interface to its storage manager. The main motivation for using NF^2 relations as the storage-level "data model" was on optimization: If both the conceptual *and* the internal model can be rigorously described in a formal framework, it is possible to describe the mapping between them algebraically. This allows us to apply and generalize techniques of algebraic query optimization towards the physical level. As indexes can also be described by NF^2 structures, we hoped to include access path selection in the framework of algebraic optimization. The NF^2 model has been extended to cope with addresses (stored object identifiers) and address materialization was included in order to support direct access to stored objects given their addresses. It was shown in [DPS86,DPSW86,SPS87] that a large variety of known storage structures for physical database design is encompassed by NF^2 data structures. Thus, using the NF^2 model at the kernel level was supposed to simplify the physical database design and the related problem of query transformation and optimization.

The idea was to broaden the scope of algebraic query optimization, since now the physical database layout is taken into account in algebraic transformations. We concentrated on the case of a relational frontend supported by the kernel, that is, we are given the choice to hierarchically materialize frequent joins on the internal level. We developed a theory for algebraic query optimization, that is, a set of transformation rules. These rules enable the query optimizer to eliminate unnecessary joins [Scho86].

Functionality of the DASDBS Kernel

The concept of using a formal model as the interface of the storage subsystem also guided the choice of the operational part of the DASDBS kernel interface. The idea was to provide only basic functions that do not incur much overhead, yet are powerful enough so that semantically richer application-oriented functions could be implemented with only a few kernel calls. What we had in mind was a storage system similar to the Research Storage System (RSS) of System R. Unlike RSS, however, we wanted to support efficient access to complex objects, and we strove for more flexibility with regard to index and transaction management. The solution that we finally came up with was to implement a subset of the nested relational algebra within the kernel, and to exclude index management from the kernel.

The notion of *"single pass" queries* was introduced in [Sche85,Scho86] to encompass all expressions of the nested algebra that can be computed in a single hierarchical scan over the data (linear–time, constant–space processible). A detailed analysis of single pass queries is contained in [Scho88]. Roughly, we can use nested selections and nested projections, but not in all of their possible combinations. Set comparisons have to be excluded from selections unless one of the operands is a constant. Any select–project–join query involving only materialized joins can be mapped to *one* single pass expression [Scho86].

An efficient evaluation strategy for single pass queries is implemented in the DASDBS kernel [PSSWD87,Pau88]. The main objective here was to detect non–qualifying tuples as soon as possible. Due to the nesting of conditions deep in the hierarchy, this was not a trivial task. The formal definition of the operator semantics and their algebraic properties, such as commutativity, helped in designing the optimal evaluation strategy.

1.3 Key Concepts and Overall Design of the DASDBS Kernel

Because of the kernel's central role in the architecture of the DASDBS family, it was clear from the beginning that performance was very crucial. Our key concepts for achieving high performance have been the following:

- complex–object support,
- set–orientation,
- multi–level transaction management,
- kernel extensibility.

By *complex–object support* we mean that the kernel should have knowledge about the structural aspects of complex objects, to allow physical clustering of related objects on disk. Because structurally related objects are often accessed together, structure–driven clustering can indeed improve disk I/O efficiency significantly. Complex objects directly serve as the physical storage unit, rather than being decomposed into smaller units. In an application–oriented frontend, we can therefore achieve any desired schema–driven clustering scheme by choosing the appropriate nested relational schema as the kernel representation of complex objects [SPS87]. Such a storage scheme may, of course, be quite different from the structures seen at the interface of the frontend.

Set–orientation was conceived as a complement to the concept of complex objects. As complex objects are constituted by sets of subobjects, we anticipated a potential bottleneck in the repeated invocation of certain services while iterating over such a set. To avoid this bottleneck, most functions in the DASDBS kernel deal with sets of objects at a time. The performance advantages that we expected were optimized data transfers, especially between disk and memory [Wei89], and a minimum amount of crossing interfaces during the processing of a set of objects.

Advanced applications posed new challenges also in the area of transaction management. We wanted to exploit the semantics of applications, so as to cope well with long–lived transactions that access complex objects. Moreover, the architecture of the DASDBS family suggested that transaction management should be provided already in the kernel, but should be extensible in the application–oriented layers. By carefully studying details and tricks of conventional high–performance transaction managers, we finally came up with a general framework that is known as *multi–level transaction management* or layered transactions [Wei86,Wei87a,Wei87b,BSW88,WHBM90].

Extensibility considerations became important in the DASDBS project when we were confronted with real-life spatial data from geosciences [DSW90]. These applications use a variety of different kinds of lines, regions, or surfaces, and, orthogonal to it, many different discrete representations of these continuous geometric objects exist. Supporting only basic primitives such as points, point sets, or polygon lines was not considered to be a satisfactory solution, for we wanted to avoid that every user of our system had to convert his favorite data structure and his geometric representation into ours in order to get database service for his application. Rather the system should take any geometric representation and store it into the database without (much) conversion and interpretation. Vice versa, upon retrieval from the database, data should be represented in the application format. We called this requirement the support for *externally defined types* (EDTs) [WSSH88], to emphasize the combination of the type systems of the application's implementation language and the database kernel. For the DBMS, an EDT is regarded as an abstract data type, only known by its functions. It is implemented outside the DBMS with the type system of the programming language used in the application layer. Thus, the DBMS has just the necessary knowledge, and the application retains its favorite data structure.

2. COSMOS – A Cooperative DBMS Framework

Much research in databases in the past years was devoted to areas like logic and databases, complex objects and the object-oriented approach, and their support by new system architectures. The common goal has been to increase the usability of database systems for applications beyond "classical" business applications. While we will see much influence of this research in future database systems, we observe that the cooperation between a database system and the "rest of the world" is often cumbersome and has mostly been disregarded. More research is needed to make database systems more cooperative for users (application programmers and data base administrators), for other system services (operating system, file services, programming languages, application tools, other database systems), and for the database system itself in that it takes better advantage of foreign services.

A database system today requires that all data are converted and loaded into database data structures. No coexistence with other data repositories is supported, and no interoperability with other services is provided. Research in extensible databases is an important area but only a first step towards the adaptation of a database system to an application environment.

The principles of teamwork, of cooperation by delegation, and of autonomy and responsibility have been proven advantageous in human enterprises. It is a question to what extent these principles are a reasonable basis for the cooperation of computer subsystems as well [ACM1,Sch90]. In our project we want to find out how much autonomy can be left to a subsystem, depending on the service that this subsystem offers. We want to see how we can gradually integrate heterogeneous subsystems and how we can gradually turn a federated cooperation into a tight one. Within this overall framework, the goal of making database systems more cooperative includes a variety of requirements:

- *Coexistence of Multiple Data Representations:*
  Cooperation between the database system and other data handlers such as CIM tools requires that any kind of application object (a vehicle, a turbo engine, a document, a folder, a map, a bank account, ...) should be supported in any representation a user wants. Multiple representations, views, and exchange formats must be provided for different applications, programming languages, and heterogeneous system environments.
- *Transaction Model:*
  Cooperation between multiple subsystems requires a cooperative transaction model. Isolation,

atomicity, and durability should be properties that can be relaxed for each individual transaction rather than being hard-wired in the transaction paradigm.

- *Schema Evolution:*
  Long-term cooperation between the database system and the application developer requires support for dynamic schema evolution. At any point of time, the schema of the database must be changeable or refineable, i.e., it must be possible to add new applications and new data or new representations. Cooperation between applications needing the old schema and those working with the refined one is necessary. The database system is responsible for providing the appropriate views and/or keeping several representations consistent.

- *Adaptive Performance Tuning:*
  Performance tuning is an important issue in all database systems, but even more crucial in an environment where applications with multiple data representations and multiple access demands are to be supported. Better cooperation between the database system and the DBA is crucially needed, ultimately aiming at the automation of performance-tuning decisions. For example, storage structures should be automatically reorganized, and self-adaptation of the physical schema is a further feature of the envisioned cooperative system.

- *Cooperation with the Operating System:*
  Cooperation is also needed between the database system and the operating system. For example, intra-transaction parallelism should be automatically exploited, taking advantage of multiprocessor systems and operating-system features.

The COSMOS Subprojects

In the sequel, we give short descriptions of the four subprojects that are already established within COSMOS. Figure 2 gives the global picture of cooperating services together with a rough approximation of what ends of the COSMOS the subprojects belong to. In COCOON, the main efforts are devoted to the development of a core object model and the related optimization issues; COMFORT is oriented towards automating performance tuning and better cooperation between the database system and the operating system; in the COAX subproject, we analyze cooperative and application-specific extensible databases with geographical and CIM applications in mind; finally, HYDRA tries to evaluate and generalize the extensibility concepts developed in the geo area in the context of hypermedia documents in a creative office environment.

2.1 COCOON – A Core Object-Oriented DBMS

COCOON is a recursive acronym standing for COcoon ... Complex-Object-Orientation based on Nested relations. The overall objective of COCOON is to investigate the principal foundations and architecture of a cooperative object-oriented database system. Within COCOON, our particular focus is on:

- *Data Model Evolution:*
  Our approach towards an OODB model follows an evolutionary path from relations via nested relations (or complex objects) to an object model that is perceived as a synthesis of concepts from complex object models, object-orientation, nested query languages, and recursion. COCOON is considered a *core* object model: we include only a limited number of concepts, aiming at an extensible model that facilitates inclusion of new (application-specific) concepts, such as new base types (geometry, text, graphics, ...) or type constructors (array, list, ...). "Evolutionary"

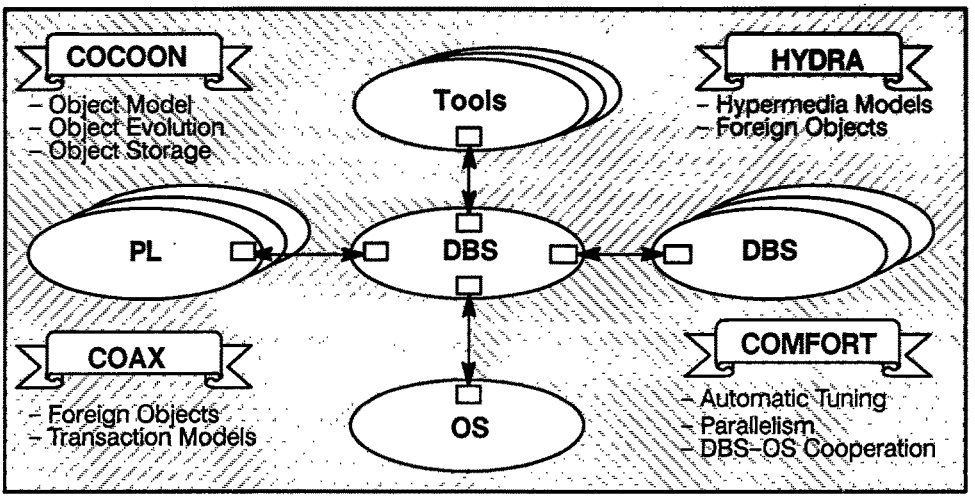

Fig. 2: The COSMOS Project and its Subprojects

means that the *concepts* and some of the *formal foundations* of the object model have been carried over from the predecessor models; that is, rather than re-inventing known concepts, we try to integrate and extend them, if necessary. Thus, cooperation between COCOON and relational DBMSs, for instance, is facilitated by "talking the same language". Not only do the data models evolve, so do the applications. Support for dynamic schema modifications is another cooperative aspect of COCOON.

- *Optimization:*
  Identifying good storage structures for a given object schema is the "data-definition-time" part of the overall optimization problem. We exploit nested relations as the internal storage structures to implement composite objects. This provides a variety of schema-driven clustering options at the object level. (In addition, the COMFORT subproject investigates clustering and declustering at the page and file level.) The "query-time" part of the optimization problem consists of mapping the object-oriented queries to efficient query execution plans that take into account storage clusters, object indices, replicated data, and streamed, buffered, or parallel low-level query operators. Here again we investigate the use of existing technology for the description of access plans in terms of nested relational algebra operators.

Foundations of the COCOON Object Model

As the COCOON model is an evolution from nested relations [SS90a,SS90b,SS90c], the principles of an object-oriented query algebra could be drawn from the nested relational algebra. However, there are subtle differences: For example, a distinction between object-preserving and object-generating semantics of query operators is necessary [SS90c,HS91]; query results need to be classified in the object lattice; the update language has to integrate generic and type-specific operations; updatability of views has proven much easier than in relational databases [SLT90]. Statically type-checking the query language while providing enough flexibility, such as meta-level querying or even mixed data- and meta-level operations, are another concern.

Monitoring Database Integrity during Updates

In this subproject, we investigate efficient update processing. We focus on the updatability of views, updates in the presence of multiple representations of objects, and integrity monitoring, particularly the need for dynamic re-classification of modified objects: Rather simple update operations may cause quite a large number of objects to be re-classified in the class lattice, based upon class predicates. This generally undecidable problem should be alleviated by accounting for the semantics of update operations.

Support for Dynamic Schema Evolution

A basic precondition for providing schema modifications is that the meta-schema of the objectbase may be represented in the same way as the other objects. As a consequence, we can apply the COCOON operations to the meta-level objects too. Schema changes are then just updates on the meta level. To provide *efficient* support for evolving schemas, however, means to avoid having to transform the objects affected by the schema change (propagation to the instance level). We are developing a framework for schema modifications including views (screening), lazy (deferred) propagation, and eager (immediate) propagation [Tre91]. Particularly interesting in the context of COSMOS are the possibilities and limitations of gradually changing from flat relational to full-fledged object schemas. The latter aspect reflects the intent to support a smooth migration path, one of the ways in which COCOON is cooperative.

Query Optimization and Execution Strategies

In this subproject, we investigate the query processing issues, that is, developing reasonable execution strategies for complex queries, such as joins between hierarchies [RRS90], depending on the nesting levels of join attributes or cost estimation for various techniques of using a nested relational storage manager. Depending on the choice of physical storage structures, the optimizer to be developed will try to identify the optimal execution strategy, including alternatives such as set-oriented vs. streamed vs. parallel mode of combining operators. Ultimately, we try to evaluate quantitatively the possible performance gains from using a nested relational storage server as opposed to a flat relational one to implement an OODBMS.

Supporting Recursive Queries

We can define derived functions on objects in the COCOON query language, COOL. These functions may be defined recursively. Additionally, COOL will be extended by some built-in constructs for expressing recursion. Our previous work in this area focused on efficient algorithms for special classes of recursive problems [Jia90a, Jia90b,Jia91]. For example, efficient graph algorithms for different types of transitive closure queries have been investigated. Now we have to integrate them into the system, and particularly into the optimizer.

2.2 COMFORT – Compile-time Performance Tuning

COMFORT stands for Compile-time Performance Tuning. The overall objective of COMFORT is to automate tuning decisions for transaction processing in database systems. Our basic approach towards this objective is to determine most tuning parameters and run-time strategies on the basis of a compile-time analysis of the transaction programs and statistical information about the data and the workload. Compared to currently available database systems, COMFORT is supposed to

- extend the range of tuning opportunities, e.g., by introducing transaction-type-specific system parameters,

- eliminate, to a large extent, the need for a human tuning expert like a DBA,
- provide guidelines for building a database system that adapts itself to the workload without incurring much overhead, and,
- ultimately, remove (the need for) manual tuning tricks from (source-level) transaction programs and make application development easier.

Database system tuning is usually considered hard. COMFORT is supposed to develop heuristic guidelines and rules of thumb for "intelligent" tuning decisions, and to automate the application of such rules in a system environment. This goal entails both classical DBMS performance issues and a better cooperation between the DBMS and the operating system, so as to take more advantage of advanced hardware technology and OS capabilities. Particular focus will be on issues that are expected to become performance-limiting factors with a wider use of advanced technologies. Currently, we concentrate on the following issues (see [WHMZ90] for details):

OPERA: The Case for Open Nested Transactions

Based on the framework of multi-level transaction management, we are investigating workload-customized concurrency control and recovery protocols that exploit semantics yet do not incur much overhead. In addition, we are investigating how one can benefit from multi-level transactions in a federated DBMS environment with both global and local transactions [SWS91,WS91] (see also the COEXIST project below).

RAPIDS: Resource and Processor Management with Intelligent DBMS Scheduling

We want to develop an approach toward "intelligent" resource management, which coordinates the policies for load control, CPU scheduling, and memory management with the transaction manager [WS91]. In particular, we have been investigating automatic load control that prevents a DBMS from thrashing caused by excessive lock conflicts or excessive buffer steals.

FIVE: Intelligent Data Allocation, Migration, and Reorganization

To optimize data access in a storage hierarchy consisting of large memory, disk arrays, and optical disks, we are studying "intelligent" strategies for data allocation and reorganization. In particular, we have developed heuristic algorithms for dynamic file creations and file expansions in disk arrays [WZS90], aiming at I/O parallelism to minimize the reponse time of file accesses and at balancing the overall I/O load across all disks.

PLENTY: Parallelism in DBMS

We are investigating how parallel computers can be exploited in order to reduce transaction response time, using multi-level transactions with parallel subtransactions as the execution model. In particular, we are investigating the performance impact of parallel algorithms for query- and update-processing in a multi-user environment [HW91].

2.3 COAX — Cooperation between Applications and Extensible DBMSs

COAX stands for COoperation between Applications and eXtensible database systems. Research on the spectrum from loose to tight cooperation with different programming languages and with different application tools is the main objective of this subproject. Geographical information systems (GIS) and databases for computer integrated manufactoring (CIM) are used as example application areas. The extensibility concepts studied in DASDBS are generalized in the following aspects:

- general structure mapping and conversion code generation,
- access methods for foreign objects, and
- foreign–object subsystem integration.

X–INFO: Integration of Foreign Objects by Extensibility

A foreign object in the simplest case is an ADT for the DBMS the realization of which is implemented in a foreign type system, outside the DBMS (i.e., as an EDT). The connection to the DBMS (i.e., persistence) is established by the IN and OUT functions which produce legal instances of some type in the DBMS and vice versa. The code of these functions is generated automatically in order to avoid serious malfunctions at this critical subsystem interface. In addition to a foreign realization, also a DBMS realization is useful in order to utilize generic DBMS functions better. This may be a "pure" DBMS realization in rare cases, or more frequently one that uses a more primitive EDT in combination with data model structures.

In the ideal case, it must be possible that several realizations of an object can co–exist, so that the one with the most convenient or most efficient realization can be selected to perform an operation. Note that several realizations also occur if several programming languages must be supported and cooperation is needed between these and the database. Therefore, the impedance mismatch is attacked by studying different representations of objects and the transformations from one represention to another. Program code must be generated automatically whenever possible, based on type definitions and high–level specifications of the user.

PROMISE: Processing Methods on Index and Storage Structures for External Objects

As foreign objects bring their own predicates to be used in query selection expressions and their own operations for deriving new objects from one or several input objects, index support for these foreign objects seems difficult in the most general case. Solutions proposed so far include generic indexes and index attachments [WSSH88,WWH88,DSW90]. We try to generalize these even further by utilizing the observation that any known index can be regarded as 1) a set of precomputed queries and 2) a special storage structure in order to quickly retrieve a precomputed query. It is obvious that (1) is independent of the application. If the DBMS is able to perform foreign operations in queries, it also can precompute queries. The main problem is (2), i.e., the storage organization of the precomputed queries, and related to it, the selection of the sort of queries that should be precomputed. We hope that concepts such as dominant predicates, disjoint or overlapping partitions, hierarchical decompositions, partial indexes, and object approximations help in finding a higher level of abstraction allowing the implementation of more generic access mechanism code (see also the DASH project below).

COEXIST: Cooperation with External Information Systems

This subproject investigates the spectrum from loose to tight cooperation. While the other two subprojects aim at a rather tight cooperation, in this subproject we want to know what the minimal requirements are to facilitate the most primitive cooperation and how much autonomy must be given up when a federated cooperation is gradually turned into a tighter one. Stand–alone data dictionaries and directories facilitate the loosest kind of cooperation in that they contain information on data structures and programs for application programmers. We study how this passive role of a directory can be turned into a more active one. For example, multiple subsystems could be automatically controlled and supervised by the directory, and the directory could be automatically kept consistent by the sub-

systems. Cooperative transaction management, versions and version-derivation book-keeping, and workflow monitoring must be looked at (see also the OPERA project above).

2.4 HYDRA – Database Service for Hypermedia Documents

HYDRA stands for HYpermedia in Databases: Requirements and Architecture. In this subproject, we investigate the cooperation between database systems and hypermedia document servers as front-ends. Office applications serve as example applications to study the needs of such front-ends. Therefore, the HYDRA project complements investigations in the COAX project with its orientation towards geographical and engineering information systems (GIS, CIM). The HYDRA project is also part of a larger, joint project with the information retrieval and document management group at ETH.

HOMER: Hypermedia Documents in Object Models – Extensibility & Representation

Given the many proposals for document models we want to investigate how they fit with the proposals for semantic data models or object oriented concepts in databases. The following list of questions are addressed: How can nonlinear documents (i.e., graph structures) be supported by the COCOON object model? How does an "unstructured" document collection evolve into a generalization lattice supported by object models? What external data types and operations must be added in order to reflect the multimedia aspect? How are multiple representations of documents kept consistent. Are there specific requirements in the document area that are different from similar requirements in the engineering field? How do extensibility approaches fit the object model in order to support flexible integration of new types of information representation?

DASH: Document Access Structures for Hypermedia

In order to support similarity searches and (non-Boolean) queries for relevant documents including relevance feedback mechanisms, different access techniques have already been developed in the field of textual databases. Some are based on inversion of keywords or descriptors, some use these for a special encoding of a document, often called a document signature [Dep89]. These techniques aim at an approximation of a document by its main features. The objective of DASH is to store these structures inside the DBS without knowing the details of methods that assign descriptors (often called automatic indexing) or extract features suitable for object approximations. We will not try to invent new automatic indexing or feature extraction techniques, because these are application-specific. We rather investigate what generic components can be developed for an efficient storage organization of descriptors or features.

References

[ACM1] ACM Computing Surveys, Vol. 22, No. 3, 1991, *Special Issue on Heterogeneous Databases*

[BSW88] Beeri, C., Schek, H.-J., and Weikum, G., *Multi-Level Transaction Management, Theoretical Art or Practical Need?*, 1st Int. Conf. on Extending Database Technology, Venice, 1988, Springer, LNCS 303

[Dep89] Deppisch, U., *Signatures in Database Systems*, Ph.D. thesis, Technical University of Darmstadt, 1989 (in German).

[DSW90] Dröge, G., Schek, H.-J., Wolf, A., *Extensibility in DASDBS* (in German), Informatik Forschung und Entwicklung, Vol. 5, 1990

[HS91] Heuer, A., Scholl, M.H., *Foundations of Object-Oriented Query Languages*, Proc. GI Conf. on Database Systems for Office, Engineering, and Scientific Applications (BTW), Kaiserslautern, March 1991, Springer IFB

[HW91] Hasse, C., Weikum, G., *Multi-Level Transaction Management for Complex Objects: Implementation, Performance, Parallelism*, Technical Report, ETH Zurich, 1991

[Jia90a] Jiang, B., *A Suitable Algorithm for Computing Partial Transitive Closures in Databases*, IEEE Data Engineering Conf., Los Angeles, 1990

[Jia90b] Jiang, B., *Design, Analysis, and Evaluation of Algorithms for Computing Partial Transitive Closures in Databases*, Technical Report 132, Dept. of Computer Science, ETH Zurich, 1990

[Jia91] Jiang, B., *Traversing Graphs in a Paging Environment – BFS or DFS?*, Information Processing Letters, Vol. 37, No. 3, February 1991

[MW91] Mönkeberg, A., Weikum, G., *Conflict–Driven Load Control for the Avoidance of Data–Contention Thrashing*, Proc. IEEE Data Engineering Conf., Kobe, April 1991

[Pau88] Paul, H.–B., *The DASDBS Database Kernel System for Standard and Non–standard Applications – Architecture, Implementation, Applications* (in German), Ph.D. thesis, Technical University of Darmstadt, 1988

[PSSWD87] Paul, H.–B., Schek, H.–J., Scholl, M., Weikum, G., Deppisch, U., Architecture and Implementation of the Darmstadt Database Kernel System, ACM SIGMOD 1987

[Sch90] Scheuermann, P., et al., *Report of the Workshop on Heterogeneous Database Systems*, Northwestern University, Evanston, December 1989, ACM SIGMOD Record, 1990

[Sche85] Schek, H.–J., *Towards a Basic Relational NF^2 Algebra Processor*, Int. Conf. on Foundations of Data Organization, 1985

[Scho86] Scholl, M.H., *Theoretical Foundation of Algebraic Optimization Utilizing Unnormalized Relations*, 1st Int. Conf. on Database Theory, Springer LNCS 243, 1986

[Scho88] Scholl, M.H., *The Nested Relational Model – Efficient Support for a Relational Database Interface* (in German), Ph.D. Thesis, Technical University of Darmstadt, 1988

[SLT90] Scholl, M.H., Laasch, C., Tresch, M., *Updatable Views in Object–Oriented Databases*, Technical Report 150, ETH Zurich, 1990

[SPS87] Scholl, M.H., Paul, H.–B., Schek, H.–J., *Supporting Flat Relations by a Nested Relational Kernel*, Int. Conf. on Very Large Data Bases, Brighton, 1987

[SPSW90] Schek, H.–J., Paul, H.–B., Scholl, M.H., Weikum, G., *The DASDBS Project: Objectives, Experiences, and Future Prospects*, IEEE Transactions on Knowledge and Data Engineering Vol.2 No.1, 1990

[SS86] Schek, H.–J., Scholl, M.H., *The Relational Model with Relation–Valued Attributes*, Information Systems Vol.11 No.2, 1986

[SS89] Schek, H.–J., Scholl, M.H., *The Two Roles of Nested Relations in the DASDBS Project*, in: S. Abiteboul, P. C. Fischer, and H.–J. Schek (eds.), Nested Relations and Complex Objects in Databases, Springer LNCS 361, 1989.

[SS90a] Scholl, M.H., Schek, H.–J., *A Synthesis of Complex Objects and Object–Orientation*, Proc. IFIP TC 2 Conf. on Object–Oriented Databases, Windermere, 1990, North–Holland

[SS90b] Schek, H.–J., Scholl, M.H., *Evolution of Data Models*, in A. Blaser (ed.): Database Systems for the 90's, Springer LNCS 466, 1990

[SS90c] Scholl, M.H., Schek, H.–J., *A Relational Object Model*, Proc. 3rd Int. Conf. on Database Theory, Paris, 1990, Springer LNCS 470

[SSW90] Schek, H.–J., Scholl, M.H., Weikum, G., *From the KERNEL to the COSMOS – The Database Research Group at ETH Zurich*, Technical Report 136, Dept. of Computer Science, ETH Zurich, 1990

[SW86] Schek, H.–J., Waterfeld, W., *A Database Kernel System for Geoscientific Applications*, Symp. on Spatial Data Handling, 1986

[SW91] Schek, H.–J., Weikum, G., *Extensibility, Cooperation, and Federation of Database Systems*, Proc. GI Conf. on Database Systems for Office, Engineering, and Scientific Applications (BTW), Kaiserslautern, March 1991, Springer IFB (in German)

[SWS91] Schek, H.–J., Weikum, G., Schaad, W., *A Multi–Level Transaction Approach to Federated DBMS Transaction Management*, Proc. 1st Int'l Workshop on Intreoperability in Multidatabase Systems, Kyoto, 1991

[Tre91] Tresch, M., *A Framework for Dynamic Schema Evolution in COCOON*, Draft, ETH Zurich, 1991

[Wei86] Weikum, G., *A Theoretical Foundation of Multi–Level Concurrency Control*, ACM PODS 1986

[Wei87a] Weikum, G., *Enhancing Concurrency in Layered Systems*, 2nd Int. Workshop on High Performance Transaction Systems, 1987, Springer LNCS 359

[Wei87b] Weikum, G., *Principles and Realization Strategies of Multi-Level Transaction Management*, Technical Report, TH Darmstadt, 1987, accepted for ACM TODS

[Wei89] Weikum, G., *Set-Oriented Disk Access to Large Complex Objects*, IEEE Data Engineering Conf., 1989

[WHBM90] Weikum, G., Hasse, C., Broessler, P., Muth, P., *Multi-Level Recovery*, ACM PODS 1990

[WHMZ90] Weikum, G., Hasse, C., Mönkeberg, A., Zabback, P., *The COMFORT Project: A Comfortable Way to Better Performance*, Technical Report 137, Dept. of Computer Science, ETH Zurich, 1990

[WS91] Weikum, G., Schek, H.-J., *Multi-Level Transactions and Open Nested Transactions*, in IEEE Data Engineering Bulletin, Vol. 14, No. 1, March 1991

[WSSH88] Wilms, P.F., Schwarz, P.M., Schek, H.-J., Haas, L.M., *Incorporating Data Types in an Extensible Database Architecture*, 3rd Int. Conf. on Data and Knowledge Bases, Jerusalem, 1988

[Wol89] Wolf, A., *The DASDBS Geo-Kernel: Concepts, Experiences, and the Second Step*, Int. Symp. on Design and Implementation of Large Spatial Databases, Santa Barbara, 1989, Springer LNCS 409

[WWH88] Waterfeld, W., Wolf, A., Horn, D., *How to Make Spatial Access Methods Extensible*, 3rd Int. Symp. on Spatial Data Handling, 1988

[WZS90] Weikum, G., Zabback, P., Scheuermann, P., *Dynamic File Allocation in Disk Arrays*, Technical Report 147, Dept. of Computer Science, ETH Zurich, 1990