MOOD: A Knowledgebase System with Objectoriented Deduction

Rudolf BAYER
Bayerisches Forschungszentrum für Wissensbasierte Systeme
(F O R W I S S)
Institut für Informatik, Technische Universität München

Abstract: Object-oriented Database Systems offer certain advantages over Deductive Database Systems by providing techniques for dealing with complex objects and hierarchies of object classes. Often, however, such systems fall back onto procedural or even navigational techniques for querying and manipulating object bases. The prime goal of the MOOD project is to integrate concepts of Object-oriented and Deductive Databases by:
- providing powerful knowledge modelling techniques via class hierarchies of complex objects
- dealing with objectbases primarily in a nonprocedural, declarative way
- handling large objectbases efficiently

The talk covers the essential concepts of the MOOD project and the overall architecture of the MOOD system. A modelling example from automobile logistics is discussed in some detail, which requires both object-oriented and deductive specification techniques. The paper also describes the implementation technology based on the Deductive Database System LOLA and a parallel version of the Relational Database System TransBase.

1. Introduction

Various manifestoes [2, 3] and position papers [1, 5] have been launched to spell out the requirements for and the benefits offered by the next generation of database systems. Some of these papers sound like promising every- thing to everybody, but little is said how to fulfill all these promises.

Well in advance of the arrival of the manifestoes the MOOD-project was already on the way as a serious effort to achieve a fair number of goals projected by the manifesto-visionaries into the future.

Attempting to achieve so many goals in a single system may seem unrealistic, especially if that system, like MOOD, is being built in an academic environment. Our experience shows, however, that this is possible and that the key to success lies in exploiting carefully designed and matured database systems of previous generations as subsystems. Such previous systems are in particular distributed relational database systems, deductive database systems, fulltext database systems and image handling systems. There are, however, two prerequisites for the successful integration of such subsystems:
- the subsystem must be open and expose the important internal interfaces, like optimization passes of SQL and logic language compilers, the resulting operator trees of an extended relational algebra and the two-phase commit protocol of a relational database system [9].
- the subsystems must have a very clean and fine structured modular design in order to be able to explore several architectural variants in an easy and flexible way.

DATABASE SYSTEMS FOR ADVANCED APPLICATIONS '91
Ed. A. Makinouchi

2. The MOOD-Architecture

The MOOD-Architecture is shown in Fig. 2.1 together with some applications which are presently being developped on top of MOOD. It relies heavily on the high performance, distributed relational database system TransBase™. TransBase is an open system exposing several interfaces like SQL, Operator-Trees, B-Trees, prepare-state of the two-phase commit protocol. In addition, a massively parallel version of TransBase will be available in the future in order to cope with potential performance bottlenecks. For details see section 5.

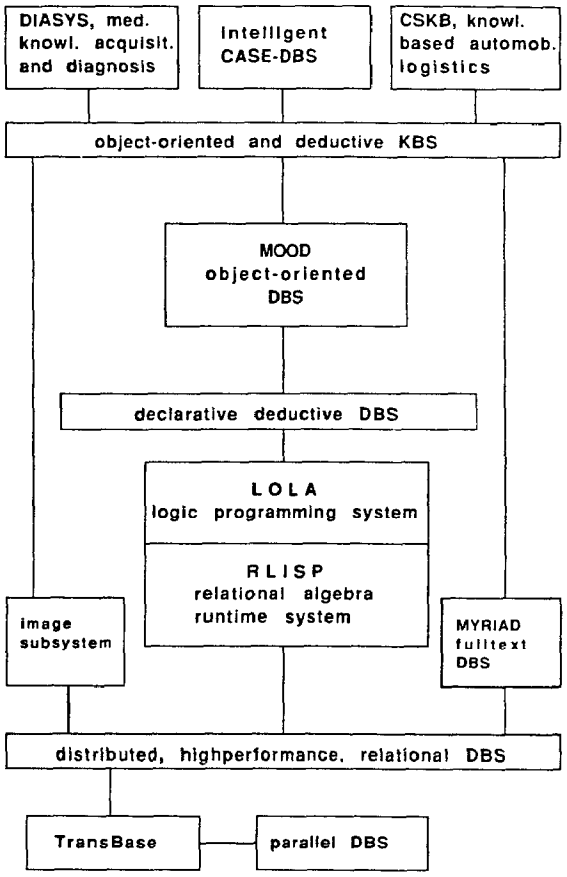

Fig. 2.1

On top of TransBase we built the Deductive Database System LOLA [6], which is also being presented at this conference. For the purpose of rapid prototyping LOLA was written in LISP. LOLA has an extended relational algebra runtime system RLISP and a high performance connection to TransBase. Thus LOLA can exploit TransBase as a workhorse to store and to process massive data efficiently. In addition LOLA utilizes the special iteration and transitive closure operators of TransBase to process recursive queries efficiently. LOLA also relies on TransBase for storing logic programs, optimized queries - which tend to become extremely large - the facts of LOLA programs and stored intermediate results - in parti- cular for common subexpressions - in a secure, reliable and crashproof way.

LOLA also relies on TransBase to provide the multiuser capability via the classical ACID transaction concept. Future versions of MOOD and LOLA resp. will provide more general transaction concepts [9], which are presently under investigation. Through LOLA, MOOD can benefit from all the optimization techniques of logic programming, e.g. Δ-iteration [10] and magic sets [12] and further advances in logic programming.

On top of LOLA finally an object-oriented layer completes the MOOD- system. This layer introduces class-hierarchies, object-identity, multiple inheritance, lazy evaluation and many of the other properties postulated e.g. by the OO-DBS manifesto [3] and 3rd-generation- DBS manifesto [2].

Object-orientation, however, is just a parti- cular way of looking at problems, it does not really solve any problems yet. Treating texts as objects e.g. does not solve the problem of searching large textbases quickly according to general patterns. For this reason we are integrating powerful, specialized subsystems like the fulltext Database System MYRIAD™ [8], which can archive texts and perform a fulltext search on a Gigabyte of text in less than 5 seconds according to very general text patterns and phrase structures.

Similarly, we are integrating a specialized image-subsystem, which can compress raster images, store them in TransBase, retrieve them and decompress them. Fig. 2.2 gives performance data of our image system for a representative set of images. This system utilizes new image representation techniques and new compression and decompression algorithms. The performance data are based on very high level implementations of the algorithms. The compression ratios are representative for pages of technical documents containing arbitrary mixtures of printed text and graphical material like figures and charts. Decompression time is faster than the compression time by roughly the compression factor, i.e. it takes about 0.1 seconds or less. At this point the retrieval time for an image from the database and the CPU-time needed for compression or decompression are well balanced on a SUN4 providing subsecond response times.

One of the central capabilities of MOOD is logical deduction over an object-oriented database. Chapter 3 will demonstrate this need using a specific practical modelling example. Object-oriented databases for practical applications tend to grow very large and therefore the typical navigational interface of object-oriented systems is rather useless. Set-oriented processing in the style of relational systems using a logic based query language is an absolute must. The syntactic form of such a language - more like SQL or more like Horn-Clauses - is a question of taste and of the targeted users. We are presently still experimenting with several syntactic variants of the query language.

Object-oriented deduction needs techniques, which are similar, but much more general than the techniques used in Deductive Database Systems: The role of logic variables is taken over by classes, the range of variables is no longer the whole term-universe, but the subclasses and the instances of the class replacing the logic variable. Term unification must be generalized substantially to take into account the constraints of the class hierarchies. For more details see chapter 4.

LOLA prepared the ground for object-oriented deduction by introducing a flat type system [11] and by developing fast unification algorithms for unifying large sets of terms. This problem arises in large deductive databases, if the deduction rules require joins over relations (real or virtual) with term-valued attributes and complex join conditions.

All these developments and integrated subsystems led to the object-oriented and deductive interface of MOOD, which is presently being used to develop a number of ambitions applications:

- knowledge-based logistics system for the automobile industry
- intelligent CASE-database system for migrating software from the mainframe to the UNIX world.
- expert system for representing and acquiring knowledge about illnesses in the throat, nose, ear area and for using that knowledge for diagnostic purposes.

TM: TransBase and MYRIAD are trademarks of TransAction Software GmbH, Munich

3. Modelling Applications with MOOD

In this section we present a highly simplified example of modelling with MOOD. The example is taken from the automobile-logistics problem and models engines, bodies and assembly procedures of BMW cars. We start with three class hierarchies, which are pure tree-structured taxonomies.

MOTOR to classify BMW engines, Fig. 3.1
KAROSSE to classify BMW bodies, Fig. 3.2
BasisBMW to classify all models of BMW cars, Fig. 3.3

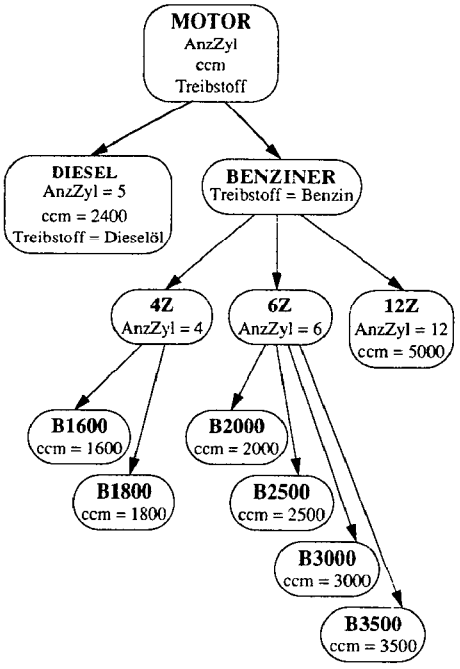

Fig. 3.1

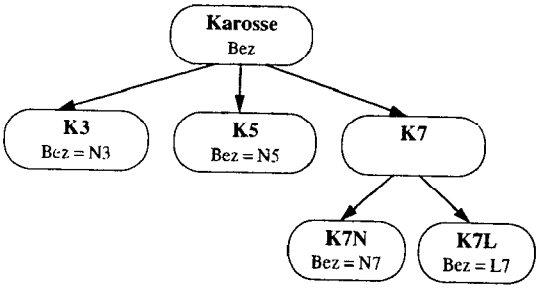

Fig. 3.2

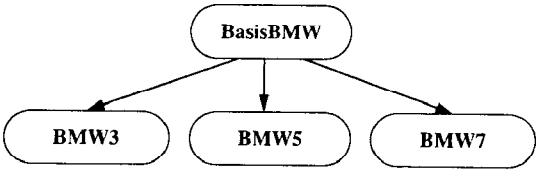

Fig. 3.3

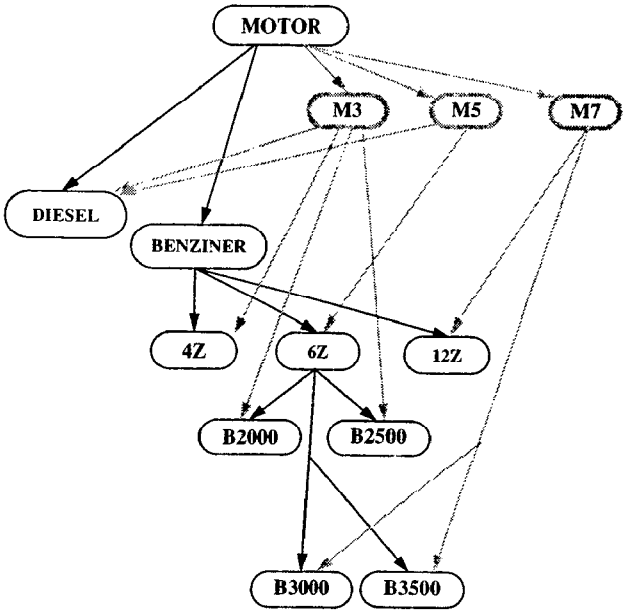

Fig. 3.4

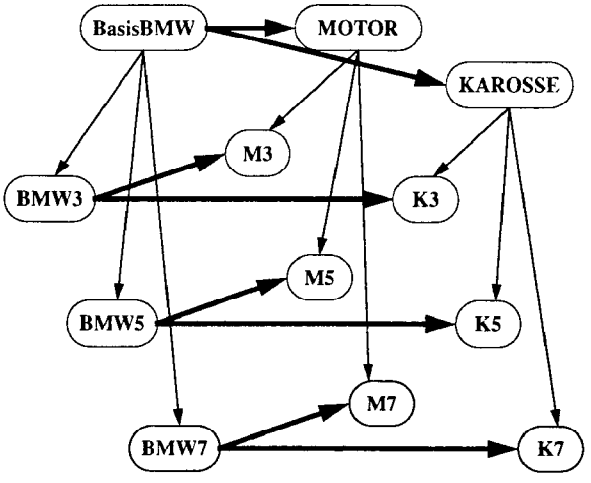

Fig. 3.5

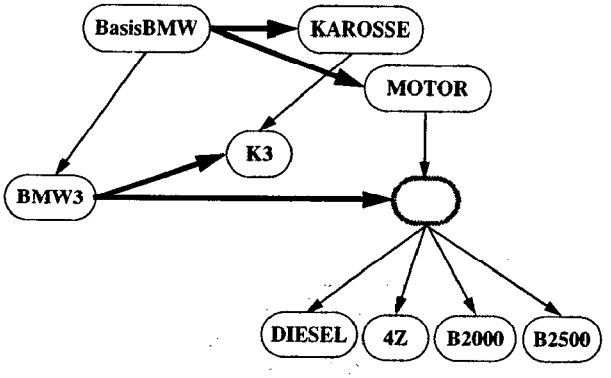

Fig. 3.6

In reality, the objects in those classes have a very complex structure, of course. We will investigate this structure in some more detail for the class BasisBMW, assuming for the moment that cars are simply assembled from engines and bodies.

3.1 Refinement of Class Hierarchy and Multiple Inheritance

At this point the hierarchies become intertwined in a very complex way. This reflects the observation, that certain standardized subsystems are used within several other systems, e.g. a variety of BMW-engines is used to make both the BMW3 series and the BMW5 series.

To reflect this situation we must refine the engine hierarchy MOTOR by introducing the engine classes M3, M5, M7 which are used to build the series BMW3, BMW5, BMW7, Fig. 3.4.

Observe that we suddenly no longer have a tree-structured taxonomy of engines, but a complicated system of overlapping classes. To specify such a system requires the concept of *multiple inheritance*. Some object-oriented languages exclude multiple inheritance for reasons of simplicity. This, however, severely limits the modelling power of such languages, particularly for technical applications. For these and several other reasons, MOOD supports multiple inheritance.

3.2 Complex Objects

Now the three hierarchies must be joined properly to define the detailed structure of the cars in the classes BMW3, BMW5, BMW7. The result is Fig. 3.5. This figure shows two kinds of edges: thin edges depict the isa-hierarchy of classes and subclasses, bold edges depict the part-of relationship between classes. If the classes used in the structure definition are shown in more detail, a very complex picture arises. Fig. 3.6 shows only the subpicture related to the definition of the class BMW3. The interested reader is invited to complete this picture by overlaying all the complete classes involved.

3.3 Class Specification by Predicates

Even this extremely simplified example shows clearly, that a purely object-oriented approach to modelling becomes unmanageable very quickly. Creating all the resulting classes and edges explicitely and drawing them by hand (or mouse clicks) becomes cumbersome and errorprone, the result is confusing and hard to understand. For these reasons, MOOD integrates the object-oriented paradigm with the logic programming paradigm. This allows also a set oriented approach to specification, using predicates (or rules) over already existing classes to define the new classes M3, M5, M7.

M3 is specified by the predicate **condition** MOTOR.ccm ≤ 2500 within the construct:

```
(create-class :class-name 'M3
  :general-list '(MOTOR)
  :special-list (condition 'MOTOR.ccm ≤2500)).
```

Observe that this predicate is satisfied by the classes DIESEL, 4Z and by the instances B2000 and B2500 within the class 6Z.

The classes M5 and M7 can be defined analogously. Here MOTOR.ccm plays the role of a logic variable, whose range is the ccm-attribute of all subclasses and instances in the class MOTOR. This example was chosen to demonstrate two principles of MOOD:

- MOOD gives the most general possible answers to predicate-queries, e.g. 4Z instead of the set of instances {B1600, B1800} of 4Z.
- MOOD completes the class hierarchy automatically and therefore MOOD can draw the class-nodes and the edges if the user wants to have them visualized graphically as in Fig. 3.5.

3.4 Classes as Views and Expressions

Here M3, M5, M7 are created as named classes, which are then generally available and can be used for further definitions. There are obvious similarities to the definition of views in relational database systems, and one might call named classes also class-views as generalizations of relation-views.

To avoid explicite naming MOOD allows class expressions in addition to named classes in specifying further classes. The class BMW3 could also be introduced without the named-class M3 as follows:

```
(create-class
   :class-name 'BMW3
   :general-list '(BasisBMW))
(create-structure
  (list (create-class
   :general-list '(MOTOR)
   :special-list (condition 'MOTOR.ccm≤2500))
   'K3) 'BMW3)
```

Introducing classes (named or unnamed) which are only needed for the definition of other classes may lead to an excessive number of classes and may clutter the class hierarchy. Therefore MOOD allows general structure conditions in the specification of classes with the goal to avoid classes like M3, M5, M7, K3, K5, K7 altogether. Thus, one obtains a much simpler and clearer class-hierarchy and part-of structure as in Fig. 3.7.

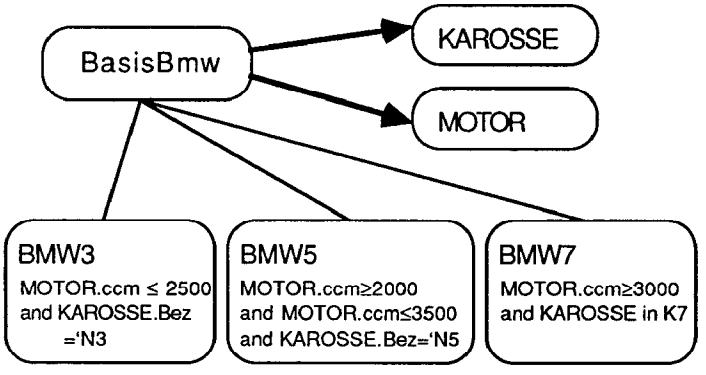

Fig. 3.7

In this figure we used a mixture of graphical and predicate specification. A purely textual MOOD specification of the class BMW3 is:

```
(create-class
   :class-name 'BMW3
   :general-list '(BasisBMW)
   :structure condition 'MOTOR.ccm ≤ 2500 and
                KAROSSE.Bez = 'N3)
```

Note that BMW3 inherits the structure-components MOTOR and KAROSSE from its general class BasisBMW and therefore the attributes MOTOR.ccm and KAROSSE.Bez can be used for formulating the predicate specifying the subclasses of MOTOR and KAROSSE resp.

3.5 Inheritance of Predicates

We now want to make our example slightly more complicated and also model the assembly procedure. Our simple assumption is that a class **MONTAGE** of assembly procedures is already defined which only depend on the number of cylinders (attribute AnzZyl) of the engine to be assembled with the body.

This adds the structural component MONTAGE to the definition of BasisBMW. It also adds the condition

MOTOR.AnzZyl = MONTAGE.AnzZyl

to the definition of BMW3, BMW5, BMW7 to make sure, that the proper assembly procedure is used in making cars.

Observe now, that this condition is exactly the same for BMW3, BMW5, BMW7 (and indeed for all conceivable cars) and therefore should be specified at the more abstract level of the class BasisBMW. It then should be inherited automatically by the special classes. This would result in the specification of Fig. 3.8, in which we used a semi-graphical technique. The reader is invited to try this specification in a pure object-oriented technique without taking recourse to predicates and logic programming.

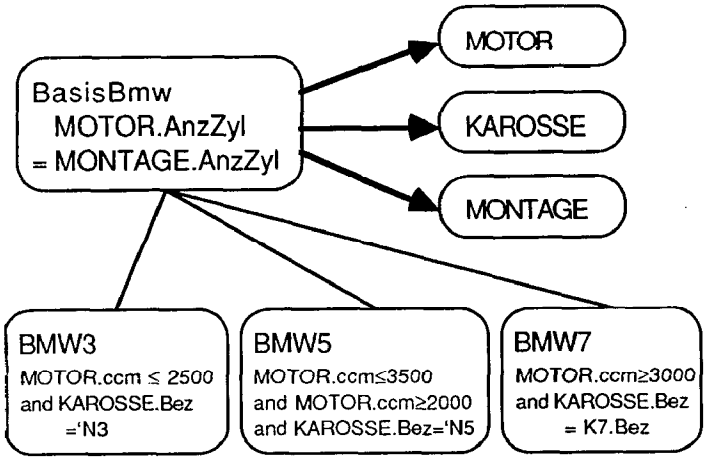

Fig 3.8

The simple example just shown is a demonstration of the general phenomenon, that the inheritance of restriction predicates is an important and widespred principle in object-oriented modelling.
Primarily, however, the examples in this chapter demonstrate clearly, that a careful integration of object-oriented specification with logic programming is necessary in order to provide the modelling power that is needed for advanced, realistic applications.

4. Object-Oriented Deduction

Object-oriented deduction (OOD) has, of course, very deep similarities with ordinary deduction of logic programming. In this section we will briefly discuss some aspects which are different from conventional logic programming.

4.1 Variables and Class Names

Let us reconsider the simple predicate MOTOR.ccm ≤ 2500 used in the definition of the classes BMW3 and M3 in chapter 3. We actually used the class-name MOTOR like a logic variable, whose range are all the subclasses and instances within the MOTOR hierarchy but not the whole universe of discourse. This resembles the usage of a relation name R as a tuple variable and simultaneously as a range-restriction for that tuple variable to the tuples that presently make up the relation R. Formally this is not very clean, but it is convenient and seems to correspond well to natural thinking and language. Thus, in MOOD class names play the role of logic variables in predicates, rules, open facts, queries and answers to queries.

4.2 Most General Class Names and Unifiers

Answers to queries should be the most general answers that can be given within the defined class hierarchies. In our previous example the answer to the query

condition MOTOR.ccm ≤ 2500

should be the set

{DIESEL, 4Z, B2000, B2500 }.

Giving an answer

{DIESEL, B1600, B 1800, B2000, B2500}

in which the class 4Z is replaced by its extension, could also be considered as a correct answer, but it is not an answer which the user expects. Giving sets of most general classes as answers corresponds to giving the most general unifiers (mgu) as answers in logic programs. Therefore the well established concept of mgu must be generalized considerably for object-oriented deduction.

4.3 Terms

Terms are formed using constants, class names as variables with selectors for the components of classes of structured objects, like MOTOR.AnzZyl or MOTOR.ccm. Terms are recursively combined to form more complex terms as usual. In MOOD terms are evaluated like in the join predicate

MOTOR.AnzZyl = MONTAGE.AnzZyl

used in the previous section. Since we are dealing with classes of complex structure, the terms can become very complicated, too.

4.4 Indexes

Accessing and joining large classes of

structured objects via general join conditions leads to complicated and compute-intensive algorithms. To lower the complexity of such algorithms clever indexing techniques are needed. Some work has been done to build tree structured indexes for large sets of complex terms as they arise in theorem proving. Generalizations of such indexing techniques are still needed to be usable for class hierarchies of structured objects.

4.5 Object-oriented Deduction and Join Algorithms

Many algorithms have been developed for joining flat relations. OOD immediately leads to the problem of joining class hierarchies, but little is known about algorithms for solving this problem efficiently.

A special subproblem of join-algorithms is efficient class-unification for large sets of terms involving structured objects.

4.6 Query-Optimization

In this paper we used predicates only for the specification of the class hierarchy, i.e. as part of the DDL-equivalent of a relational database system. Predicates play an even more important role in the query language and in the object-manipulation language. Independently of the syntactic form of the query language - more like SQL or more like Horn-clauses - query optimization is an important issue.

Due to its clear layered architecture MOOD benefits automatically from the present query optimization in LOLA and TransBase and from future improvements in those subsystems. Query optimization must be particularly effective for recursive predicates, since they arise naturally in object-oriented systems, the most obvious being inheritance rules like:

Has Attribute (C1,A) ← Isa (C1,C2)
Has Attribute (C2,A)

to describe inheritance of attribute A from class C2 to subclass C1 or:

HasValue (C1,A,V) ← Isa (C1,C2)
HasValue (C2,A,V)

to describe the inheritance of value V for attribute A from class C2 to the subclass C1.

Another recursive problem is parts-explosion which is typical for the kind of applications modelled in chapter 3 of this paper.

Therefore optimization techniques for recursive logic programs like Δ-iteration [12] and magic sets [10], which are built into LOLA, are essential. But in addition powerful optimization techniques including meta-level semantic optimization [15] is needed for the object-oriented layer of MOOD. This is an area of active research.

5. Parallel Database Support for MOOD

Database applications tend to evolve to surprisingly complex schemata and queries. This happens even in simple applications if excessive use is made of the pure normalization theory. Queries involving 10 to 15 joins are not unusual [13]. This phenomenon becomes even more severe with added logic-programming or object-oriented abstraction layers and interfaces. It is not at all unusual, to observe operator trees with 500 to 1000 operator-nodes (at the database or RLISP levels of our system) for ambitious applications like AMOS [14], an expert system for the analysis of ancient Hebrew language. With the addition of the MOOD layer we expect to see even more complex operator trees.

Such applications open the perspective for massive parallelism. For this reason a research and development project was started in 1990 (founded by Deutsche Forschungsgemeinschaft DFG) to investigate massively parallel database systems, using TransBase as a research vehicle. In this project at least five (according to recent plans) orthogonal methods will be exploited to arrive at a massively parallel database system.

5.1 Replicated TransBase Kernels

TransBase is a highly modular distributed database system designed for many TransBase kernels to work in parallel on a database, which is distributed over a loosely coupled network of workstations. If these kernels run on a system of tightly coupled multi-processors, their performance is much better. The TransBase database-cache architecture is ideally suited to work well even on multi-processors without a common main memory: Each kernel has a local cache for storing data needed for query-processing; there is a global cache which is only used for sharing common data, in order to minimize access conflicts.

Kernel replication seems to emerge as the standard and fairly straight-forward technique for other commercial database systems to run on tightly coupled multiprocessors. As opposed to multiplexing one processor between several transactions, this technique provides truely parallel execution of as many transactions as there are processors available. It does not speed up the processing of a single complex query, however.

5.2 Partitioning of the TransBase Kernel

Since TransBase is a highly modular system, its kernel can be partitioned into several processes. Candidates for such processes are the SQL-translator, several passes of the query-optimizer, several low-level modules like the B-tree manager and the segment manager, the log-manager, the lock-manager and the operator-tree interpreter. There are, of course, many challenging technical problems like sharing of information, high-performance communication protocols, synchronization and coordination of processes which are beyond the scope of this paper.
From kernel-partitioning we expect to gain about an order of magnitude increase of the number of processes that can run in parallel.

5.3 Parallel Query Execution

As explained before, we expect to see very large operator trees in MOOD. These trees have the potential for massively parallel interpretation, especially since the operators of TransBase are rather fine-grained, resulting in about 80 different, highly specialized and therefore rather small operators, most of them having less than 10 KB of code.
One clearly advantageous property of operator trees is the fact that every mode must communicate with only a few neighbours. The communication topology is fixed and known in advance after the query optimization phase. Even the structure of the data to be communicated is known. This opens up many optimization and load-balancing possibilities at the interprocess communication level. It is clear that query-optimization and communication-optimization are not independent and cause challenging research questions.
In many cases. especially in the near future, there will be not nearly as many processors in a typical workstation or server as there are potential processes resulting from a large operator tree. Therefore an important question is how to cut an operator tree into subtrees to be made into processes. Again, this problem is not independent of the classical query-optimization in relational or deductive database systems.

5.4. Intra-Operator Parallelism

Some operators needed in an extended relational algebra for a logic programming system are quite compute intensive. Our preliminary analysis indicates that they will require at least 500 MIPS to keep up in real time with the data stream delivered by a single winchester disk, if it is driven by a raw-I/0 file system. Such operators are prime candidates to be parallelized themselves, since 500 MIPS will not be delivered by workstations for some time.
Intra-operator parallelism can be used in addition to the techniques discussed before,

but from an optimization point of view it is not independent of these techniques. MIPS demands of different operators are vastly different. Therefore it will probably be useful to exploit intra-operator parallelism even before the potential for intra-operator parallelism has been exhausted.

5.5. Database Partitioning

For ambitious applications - like the ones discussed in this paper - database and logic programming systems become heaviliy CPU-bound. But with increasing processor-speed and massive parallelism, an I/O bottleneck must be reached, of course. At this point partitioning a database across several disks becomes the obvious next step to improve performance and to hand the neck of the bottle back to the processors. The question, how to partition a database and how to distribute the partitions across a collection of disks in the context of massive parallelism is difficult. It depends - of course - on the population and dynamics of the queries to be posed in a particular application, and therefore we have again a fascinating optimization problem. But it will not be until sometime in the future that we have to face this problem.

References

[1] M.L. Brodie, F. Bancilhon, C. Harris, M. Kifer, Y. Masunaga, E.D. Sacerdoti, K. Tanaka: *Next Generation Database Management Systems Technology*. In W. Kim, J-M. Nicolas, S. Nishio (eds): Deductive and Object-Oriented Databases, Elesevere Science Publishers, Amsterdam, The Netherland, 1990, pp. 1-13

[2] M. Stonebraker, L.A. Rowe, B. Lindsay, J. Gray, M. Carey, M. Brodie, P. Bernstein, D. Beech: *Third Generation Data-Base System Manifesto.*

[3] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, S. Zdonick: *The Object-Oriented Database Systems Manifesto.* In W. Kim, J-M. Nicolas, S. Nishio (eds): Deductive and Object-Oriented Databases, Elesevere Science Publishers, Amsterdam, The Netherland, 1990

[4] F. Manola: *An Evaluation of Object-Oriented DBMS Developments.*Technical Report TR-0066-10-89-165. GTE Laboratories, Waltham, MA, 1989.

[5] A. Silberschatz, M. Stonebreaker, J.D. Ullmann (eds.): *Database Systems: Achievements and Opportunities ("Lagunita Report").* Technical Report TR-90-22, Dept. of Computer Sciences, The Univ. of Texas at Austin, Austin, Texas,1990.

[6] B. Freitag, H. Schütz, G. Specht: *LOLA - A Logic Language for Deductive Databases and its Implementation.* Technical Report TUM-I9043, Institut für Informatik, Technische Universität München, Nov. 1990. Also DASFAA '91 Conference, Kogakuin University Tokyo, Japan

[7] *TransBase Relational Database System, System Guide, Version 3.3*, Manual, TransAction Software GmbH, Munich 1989

[8] *MYRIAD: An Innovative Fulltext Database System*. TransAction Software GmbH, Munich 1990

[9] R. Bayer: *Nondeterministic Computing, Heterogeneous Transactions and Recursive Atomicity*. Invited Paper, CompEuro 91, Bologna, Italy, May 1991

[10] J.D. Ullmann: *Principles of Database and Knowledge-Base Systems, Vol. II: The New Technologies.* Computer Science Press, Rockville 1989

[11] Th. Huber: *Typüberprüfung für die Logiksprache LOLA* . Diploma Thesis, Technische Universität München, Munich 1990

[12] R. Bayer: *Query Evaluation and Recursion in Deductive Database Systems.* Technical Report TUM-I8503, Institut für Informatik, Technische Universität München, 1985

[13] D. Haderle: *Private Communication*

[14] G. Specht: *Wissensbasierte Althebräischer Morphosyntax: Das Expertensystem AMOS.* EOS Verlag, 1990

[15] H.Schmidt: An Expert Deductive Database System. Dissertation, Institut für Informatik, Technische Universität München, 1989.