In: Proc. of the 11th Brazilian Symposium on Data Bases (SBBD'96), São Carlos, Brazil, Oct. 1996. pp. 58-72.

An Approach to Multi-User KBMS in Workstation/Server Environments

Fernando de Ferreira Rezende and Theo Härder

Department of Computer Science - University of Kaiserslautern
P.O.Box 3049 - 67653 Kaiserslautern - Germany
Phone: ++49 (0631) 205 3274/4030 - Fax: ++49 (0631) 205 3558
E-Mail: {rezende/haerder}@informatik.uni-kl.de

Abstract. *We begin this paper by discussing relevant processing characteristics of Knowledge Base Management Systems (KBMSs). Provided this basis, we present the main organizational forms of workstation/server architectures, discuss their main features, and appoint the most feasible one to KBMSs. As an implementation platform for our techniques for multi-user KBMSs, we introduce an existing single-user KBMS and its architecture. Thereafter, we approach aspects of transaction facilities for such systems, and discuss a general architecture offering transaction services. Following, we detail it and introduce multi-user architectural components of such systems. Finally, we highlight the most important characteristics of each one of these components and discuss their most distinguishing qualities.*

Keywords: *knowledge base management systems, database management systems, transaction processing, concurrency control, recovery, cache coherency, deadlock management.*

1 Introduction

Knowledge Base Management Systems (KBMSs), the core of the *intelligent information systems* of the future [Br86], aim at the effective and efficient management of Knowledge Bases (KBs), and were originated from knowledge management demands. However, most advanced, knowledge-based systems nowadays have no adequate facilities for allowing the KB to be shared among several applications. Typically, each user has his own private copy of the KB in 'multi-user' environments. This leads to a considerable redundancy in stored knowledge with many severe consequences: Waste of storage space, consistency problems, inefficiency, unsuitability, inapplicability, and the like. Consequently, KBMSs must be designed/extended to support real multi-user environments. Moreover, this is imperative for both efficiency reasons as well as the complete success and establishment of those systems in the market.

Without indulging in a discussion of the different (and sometimes contrary) ways of interpreting the term *knowledge base*, we briefly highlight the features of KBMSs which are important for our purposes. On the one hand, from the Artificial Intelligence community's point of view, the most important characteristics of such systems for our purposes are the explicit structures representing the knowledge as well as the organizational principles used to organize the knowledge. On the other hand, from the Database (DB) community's point of view, it is important to our purposes that such systems must provide what a Database Management System (DBMS) provides, i.e., transaction management, shared access, recovery facilities, etc. All in all, any data model which (1) explicitly represents the knowledge and, therefore, explicitly encodes the knowledge and the semantic structure of an application domain, and which (2) needs transaction facilities' support, can use the results we present in this paper. Therefore, unless otherwise noted, we use in this paper the terms *databases* and *knowledge bases* interchangeably, meaning that the results here presented are general and applicable to a broad class of applications.

The processing of knowledge in KBMSs can be characterized through different aspects. In our view, a relevant aspect is that the modeling of knowledge in KBs allows for the specification of user-defined relationships between objects (in a sense with 'poor' semantics) as well as frequently occurring relationships with system-controlled semantics, e.g. the so-called *abstraction concepts* like generalization/classification, association, and aggregation. Rules and constraints may be applied to refine these modeling concepts. From an operational point of view, an abstraction relationship can be viewed as a very large, meshed data structure forming a Directed Acyclic Graph (DAG). Furthermore, since the different kinds of relationships share common data (objects), all DAGs are overlapping.

An additional aspect is that declarative query languages are used to select and manipulate the KB objects. In such languages, query statements may be issued to *dynamically* define and access sets of KB objects. Hence, the KB access requests are typically set-oriented, and the resulting reference behavior is largely directed by the existing KB structures. Since there is a rich variety of such requests by multiple applications, KB access behavior and storage structures are hard to pre-plan. In particular, object clustering according to these multiple dynamic object views seems to be impossible.

In this paper, we present an overview of our techniques for multi-user KBMSs. In Sect. 2, we present the basic organizational forms of workstation/server architectures (the most promising ones for KBMSs), and appoint the most feasible approach given the processing characteristics of KBMSs. As a vehicle for the appliance of our multi-user KBMS techniques, we use an existing single-user KBMS, which is presented in Sect. 3. Thereafter, we approach aspects of transaction facilities for such systems, present a general architecture offering transaction services, and finally we detail it and highlight the most distinguishing qualities of our techniques for multi-user KBMSs in Sect. 4. Lastly, Sect. 5 summarizes our discussions.

2 Architectural Aspects of KBMSs in Workstation/Server Environments

We have shortly discussed previously the specific processing characteristics of KBMS applications. Typically, object-oriented systems, and so knowledge-based systems, are tailored to workstation/server environments. In such an environment, the objects have to be fetched to the workstation memory (object buffer) for further evaluation, transformation or visualization. Thus, the processing of knowledge generally takes place in the workstation. Since under transactions' logic performance dictates main memory operations, the knowledge is then temporarily stored in the workstation to the end of effectively exploiting the principle of *near-by-the-application locality* [Ma91, HS93, HMNR95]. In this context, a server acts as both a persistent knowledge reservoir as well as a source of knowledge for all the participating workstations. For these reasons, query shipping to the use of stored procedures at the server is not feasible. In contrast, data shipping is needed to provide the data supply of the KBMS applications at the workstation side. Note that large sets of data may temporarily migrate to the reach of the application. As a consequence, main-memory-based query processing may become desirable in the workstation, thereby relieving the server and the communication network. Obviously, the more tasks are assumed by the workstations, the more workstations can be efficiently supported by a server. Additionally, since great amounts of data are to be transferred between workstations and server, the organization of the communication is very important in such environments. The communication channel must be alleviated, on the one hand, and measures must be taken to provide isolation from failures, on the other.

Workstation/server architectures may be differentiated accordingly to the way the data is transferred and hence the distribution of tasks is organized. The three most important forms are illustrated in Fig. 1 [HMNR95, DMFV90]. Each one of the approaches puts available to the applications the same navigational and object-oriented interface by means of the object manager, and each server possesses as a minimal functionality an storage system for the management of the data in the secondary storage, and a page buffer for the minimization of I/O operations (shadowed boxes in Fig. 1). Additionally, the server, in the role of the central component, must take care of the synchronization of the workstations' requests. Further on, preventive measures against DB failures and server crashes (logging) as well as the handling of these situations (recovery) are tasks of this central component. In the following, we discuss the characteristics of these organizational forms to the end of appointing the most feasible one to KBMSs.

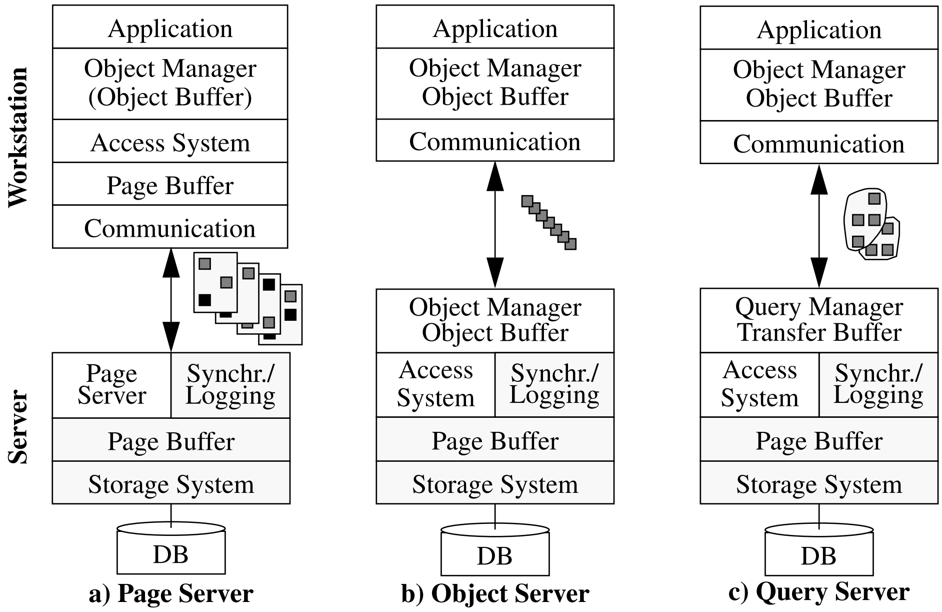

Figure 1: Basic organizational forms of workstation/server architectures.

2.1. Page Server

In the *Page Server* approach (Fig. 1a), a page request is triggered in the workstation whenever a page fault is detected, in which case the corresponding page is fetched from the server. The server is unaware of the object concept, it manages just pages, which in general also are the granule of synchronization and logging. Due to that, the buffering of objects in the workstation can only be done by means of buffering the corresponding whole pages. All object-related accesses and evaluations take place in the workstation. Due to its simplicity, the page server approach was implemented in many systems (e.g., ObjectStore [LLOW91] and O_2 [De91]).

Since the unit of transfer is a page, the communication overhead is strongly influenced by the number of pages to be transferred and specially by their size. In order to make this scheme efficient, *cluster* techniques should be employed, so that the objects to be accessed by an application are stored in as the minimal number of pages as possible. Such an approach works fine as long as complex objects can be allocated to separated physical containers (segments, files), and only a simple view and a single access path are provided for the application. Unfortunately, in KBMS applications, multiple views to sets of objects, preferably defined at run time, usually prevent to take advantage from such static clusters. Furthermore, such cluster formations are in general done at the time the objects are created, and being so following applications cannot change them and they may then become unfavorable. Additionally, as we approach the era of *4T Machines* [Gr95] (**t**era-op processing rate, **t**erabytes of RAM storage, many **t**erabytes of disc storage, and **t**erabits-per-second of communications bandwidth), the page size is being significantly enlarged (e.g., from 2 or 4 to 32 Kbytes). Thus, with ill-formed clusters things can go from bad to worse, because the few the required objects are placed together, the more other objects are unnecessarily transferred to the buffer. This has negative consequences to parallel transactions, since the granule of locking is the whole page.

Another deficiency of this scheme is the impossibility to realize attribute projections. The page content and representation are managed by the server and remains unchanged in the workstation. Thus, it is always only possible to load all the attributes of an object, although generally just a subset of them is necessary. Further on, to realize selections of objects on the basis of their contents or object scans is only possible at the workstation side, which must then fetch into its buffer all the corresponding pages. These aspects lead to waste of storage, unnecessary transfer operations, and lower levels of concurrency. Due to all these reasons, the page server approach would often lead to unfavorable performance behavior in KBMSs.

2.2. Object Server

In its basic form, an *Object Server* (Fig. 1b) provides on request a single object (through an OID - object identifier).[1] Workstation and server are now aware of objects. On trying to access an object which is not in the workstation's buffer, an object request is sent to the server. The server then checks the corresponding page into its page buffer, copies from it the required object to its object buffer, locks the object, and sends it to the workstation. In this approach, the object can be used as the granule of synchronization and logging. This allows for higher levels of parallelism (object locking) and efficient logging (entry logging). In turn, this scheme requires more complex algorithms than the page server one, and was followed by, for example, Ontos [On91], Versant [Ve90], and Itasca (Orion) [Ki90].

In this approach, the communication overhead is determined by the number of objects (the unit of transfer) to be transferred and their size. Hence, single object requests from the workstation strongly increase the communication costs. In turn, this approach is insensitive with respect to cluster formation. Here, clusters may be used to reduce the I/O operations between the server's page buffer and DB. However, as before, the workstation receives all the objects it references with all their attributes (no projection is possible). Further on, object selections and scans may only be performed in the workstation's object buffer. Hence, the before mentioned drawbacks implied by these aspects remain in this approach, and therefore it is not the most appropriate one for KBMSs either.

2.3. Query Server

At last, a *Query Server* (Fig. 1c), on request, delivers to the application sets of (complex) objects, which are the unit of transfer, and stored in the object buffer. The most important enhancement of this approach is a powerful query interface, which is put available to the application by the object manager, and by means of which the application can formulate its object requests. In this scheme, object faults are declaratively recognized, and sets of objects are put available by the server upon request. Also here, synchronization and logging can be performed on the basis of objects. This approach is a big research challenge, and was up to now followed only by prototype systems. Examples are: AIM-P [KDG87], PRIMA [HHMM88], ADMS-EWS [RD91], XNF [MPPLS93], Starburt's Coexistence Approach [AGKLP93], Persistence [KJA93], KRISYS [TMMD93], and PENGUIN [LW94].

Essentially, a query server works on the basis of *contexts*. A context comprises a set of complex objects, and may be specified for example by means of MQL or SQL/XNF statements [MPPLS93]. The evaluation of such statements takes place now in the server, where the complex objects are derived and stored in the transfer buffer. By means of operations like projections, selections, and joins, the objects' views can be tailored to the necessities of the applications. Hence, the volume of information to be transferred and checked into the

1. Extensions of this approach allow for simple queries by means of *Simple Search Arguments*.

workstation's buffer is significantly reduced and optimized. Additionally, as before, this approach is insensitive to cluster formations. Therefore, due to all its distinguishing qualities, we hold the opinion that the query server approach mostly fulfils the necessities of KBMSs.

Before passing on to the presentation of our techniques for multi-user KBMSs, we present in the next section the single-user KBMS we have chosen for applying our research results, namely KRISYS. Hence, we have used KRISYS as a practical vehicle, an implementation platform for our techniques, simply because we were not intended to develop all the functionalities provided by the upper layers of KBMSs but to support them offering transaction services.

3 An Overview of the Single-User KBMS KRISYS

KRISYS integrates concepts from DB systems and knowledge representation in order to support the effective and efficient modeling, manipulation, and maintenance of knowledge for complex, advanced applications in workstation/server environments [Ma91]. The three orthogonal ways of looking at KBMSs from a conceptual point of view [BL86, BM86] are reflected in KRISYS through its three different layers, denoted, respectively, *application*, *engineering*, and *implementation layers* (Fig. 2):

- The *application layer* addresses all issues related with the needs of the applications, and hence it pays attention to the information content of a KB;
- The *engineering layer* considers the modeling aspects of an application expressed as KB contents; and finally
- The *implementation layer* views a KB in terms of data structures and access algorithms.

These three layers provide, respectively, the needs of knowledge system (KS) applications, knowledge engineering support, and suitable resources and implementation aspects [Ma91].

Fig. 2, in addition, relates this conceptual KBMS architecture to KRISYS' overall system architecture. The application layer of KRISYS, which is responsible for the realization of its application interface, is resembled by the *KOALA processing system*, whose task is to prepare and control the processing of KOALA statements. Further, KRISYS provides the KB designer with a mixed knowledge representation framework, defined by the *KOBRA knowledge model*, which corresponds to the engineering layer. Additionally, the *Constraint Manager* appears as an extra component besides KOBRA. It performs all activities related to checking or processing the constraints of the KB [De93]. Finally, the implementation layer is divided into two main parts. The server part, which is resembled by an *Advanced DBMS*, concentrates on an efficient and reliable KB management. It provides application independent data management functions at its interface. The workstation part is partitioned into three components: The *Working Memory* (WM) embodies the *near-by-the-application locality* principle, and is seen as a passive application buffer controlled by the *Context Manager*. The context manager keeps a declarative description of the WM contents and is responsible for loading and unloading sets of objects into or from the WM. The replacement of these objects is carried out in accordance with processing characteristics of the KB, so that calls to the advanced DBMS, accesses to disk, as well as transfer overhead are substantially reduced. In order to transfer objects between server and workstation, the context manager interacts with the *Mapping System*, which transforms objects from the advanced DBMS' structures to KOBRA's structures and vice versa. It is also responsible for generating an appropriate mapping scheme for the processing phase of an application. To implement a multi-user KBMS prototype as a research vehicle, we did not want to start from the scratch and to code already existing components anew. Therefore, we have used components such as context management, query processing, object mapping, etc. from single-user

KRISYS to rebuild a multi-user system. In this way, we could focus on the various aspects of transaction management, cache coherency control, and communication.

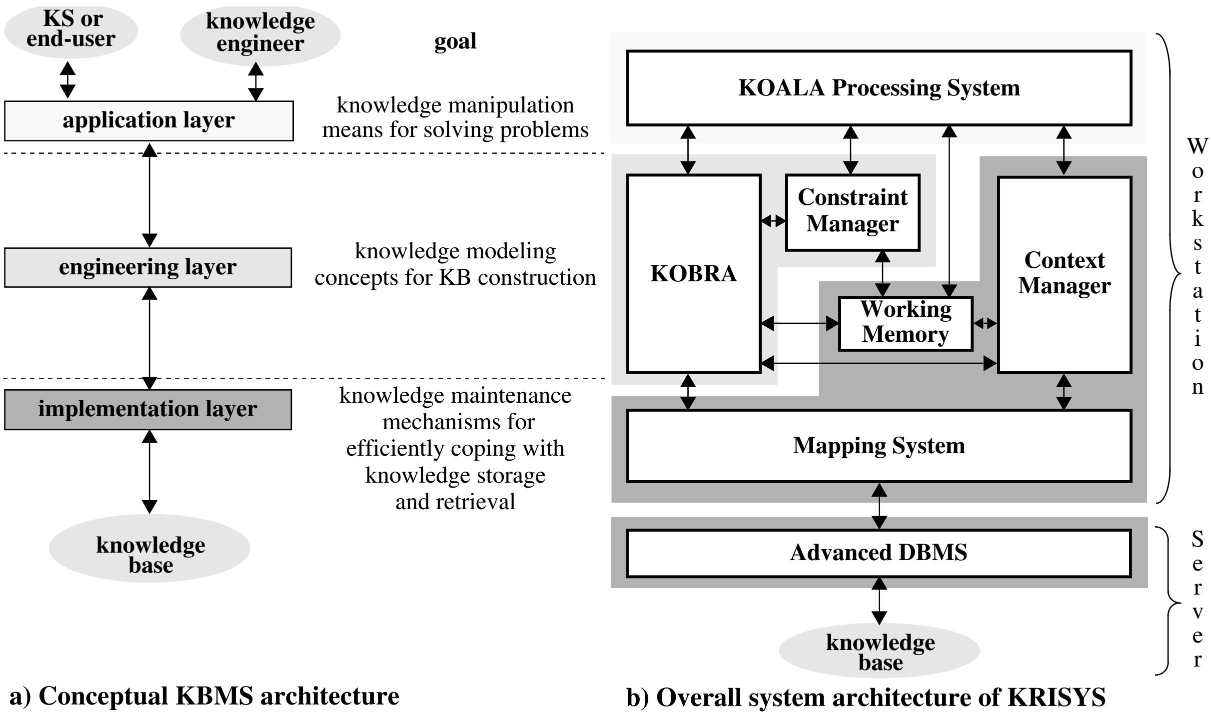

Figure 2: The KRISYS approach towards KBMSs.

4 Architectural Components of Multi-User KBMSs

In order to be capable of providing correct, concurrent access to the KB by many users at once, KBMSs must possess the distinguishing quality of *transaction management*, the ability to manage simultaneously large number of *transactions*. The concept of *transaction* is established in the DB community for over two decades [EGLT76], and provides a basis for automatically enforcing DB consistency. This concept reflects the idea that the activities of a particular user, i.e. the sequence of interactions with the DB, must be *isolated* from all concurrent activities, and *atomic* in the sense that the activities inside a transaction are indivisible (the particular actions belong together) as well as uninterruptible (either all actions are properly reflected in the DB as a whole or none of them). Härder and Reuter have coined the widely-used term ACID [HR83], an acronym for the four fundamental properties of transactions which describe the major highlights of the transaction paradigm:

- ***A**tomicity*: A transaction's changes to the state of the DB must be of the 'all-or-nothing' type described above: Either all happen or none happen.
- ***C**onsistency*: A transaction is a correct transformation of the DB state, and must therefore preserve the DB consistency. It brings the DB from a consistent state to another consistent state.
- ***I**solation*: The actions within a transaction must be hidden from other concurrently running transactions. Thus, it must appear to each transaction that other concurrent transactions are executed either before or after it, but not both.
- ***D**urability*: Once a transaction completes successfully (i.e. commits), it has the guarantee that its changes to the state of the DB survive any subsequent failures.

The consistency property of transactions requires that each transaction must be a correct program. The KBMS can, of course, watch for keeping the consistency of the KB by means of checking defined integrity constraints, and taking the necessary actions when the KB consistency is to be injured (as seen before, the constraint manager is the component who takes care of that). But, fortunately, the user is responsible for defining and writing correct programs. Hence, as a general consensus, it is assumed by definition that each transaction is correct, and the effects in the DB of an inevitable incorrect transaction can only be removed by counter-transactions, thus. Therefore, we will no longer give further attention to the consistency property of transactions along this paper.[2] In turn, the techniques that achieve the isolation property of transactions are known as *synchronization* [GLPT76], and concurrency control protocols take care of it [Ko81]. On the other hand, the atomicity and durability properties of transactions are achieved by *recovery* techniques.

Since we are considering KBMSs in a workstation/server environment, these properties must be achieved by components offering transaction services distributed along such an environment. As seen, the server acts as a reservoir and source of knowledge. Thus, the server must synchronize the object requests coming from the workstations. In addition, the server must guarantee the atomicity and durability of workstations' object manipulations performed against the KB. In order to do that, the server must then now provide *global transaction services* (Fig. 3). On the other hand, a transaction is started at the workstation side, and, further, the application starting it must be informed about its fate. Additionally, object requests are specified in a declarative way by the workstation, and the corresponding formulas are used for concurrency control. Furthermore, in-transaction backout would enable minimal fine-grained recovery for workstation transactions. Thus, the workstations must be equipped now with *local transaction services* (Fig. 3).

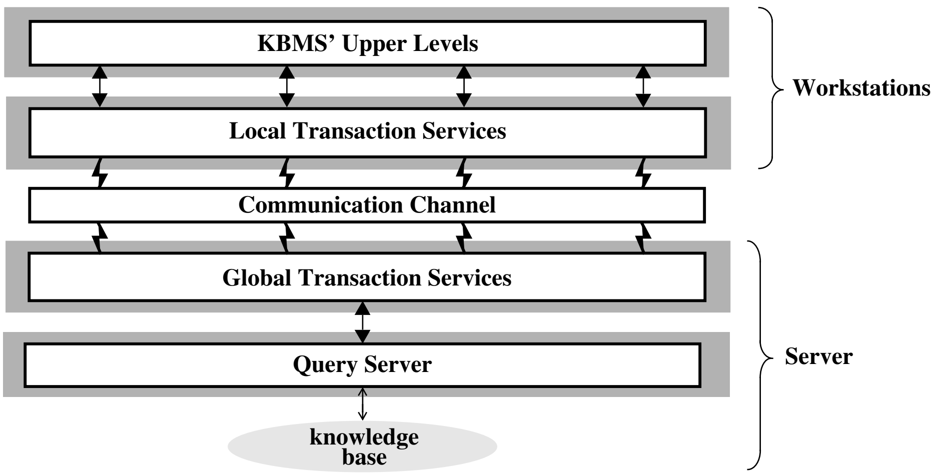

Figure 3: General transaction services in multi-user, workstation/server KBMSs.

4.1. An Architecture for a Multi-User KBMS

Fig. 4 extends the general architecture presented in Fig. 3. The activities of the user, expressed as statements of the knowledge language, must be seen as a unit of work, and are then considered a transaction, to which, as explained before, the ACID properties apply. To control such activities, a *Transaction Manager* was introduced. Further, to take care of the transaction properties, a *Lock Manager* (isolation) and a *Recovery Manager* (atomicity and durability) were introduced. Additionally, since now objects may be checked into several workstations' buffers at the same time, a *Cache Manager* was introduced to watch out for the coherency of the objects. Due to the fact that (1) transactions are started by the applications at the workstation side, (2)

2. The reader is referred to [De93] for an approach to maintain the semantic integrity in advanced DBMSs.

the contexts must stay in the workstation buffer for normally long periods of time, (3) the object requests are generated in the workstation, and (4) the applications should be able to control the fate of their own transactions, all these new components work closely related to each other, both at the workstation side, as local managers, as well as at the server side, as global managers.

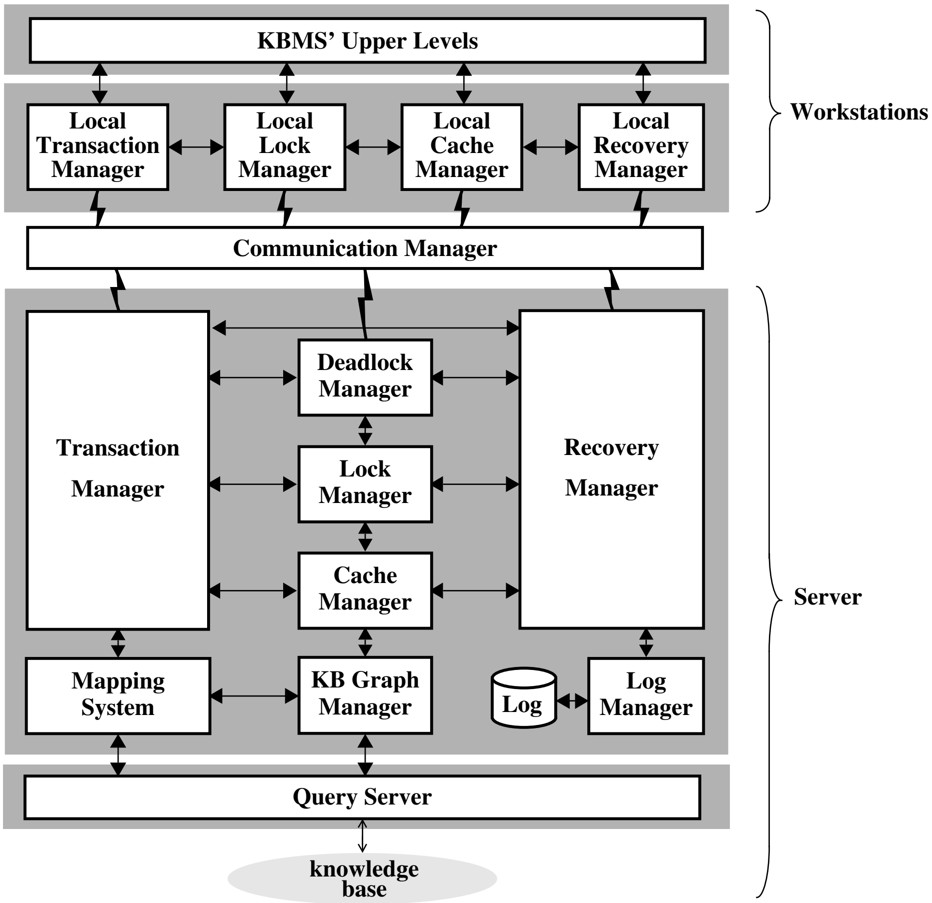

Figure 4: Architectural components of multi-user KBMSs.

When objects are to be accessed, transactions require locks on those for avoiding inconsistency anomalies due to the concurrent access. By requiring locks on objects, transactions may sometimes have to wait for other transactions. At this time, there may appear cyclic waiting situations between transactions. These situations characterize *deadlocks*, whose detection and resolution are tasks of the *Deadlock Manager*. Additionally, in order to control the locks and the graph structure built by the objects in a KB, a *KB Graph Manager* was introduced. Finally, a *Log Manager* assists the recovery manager in recording and accessing the log records of transactions in a *Log File*. These components work on behalf of all the workstations, and are thus located at the server side. At last, a *Communication Manager* controls the traffic of information through the communication network among the several workstations and the server.

We have seen in the architecture of single-user KRISYS (Fig. 2) that the *Mapping System* was located at the workstation side. The mapping system is activated whenever objects are checked out from the KB into the buffer (as KRISYS' objects), and the other way round. Hence, this component is able to exactly reflect the KB structure with all its abstraction relationships, i.e., to build the KB graph. Since our synchronization protocol works on the basis of the KB graph by exploiting the semantics of the abstraction relationships, it is assisted by the mapping system in building and maintaining the KB graph up-to-date as changes go by. Due to this reason, the mapping system was placed now in the server side. However, its tasks remain the same as before, just its component interfaces were adapted and expanded. Finally, the advanced DBMS still provides application independent data management functions at its interface. However, it is seen now purely as a *Query Server* in the new architecture, by means of which sets of objects can be stored and retrieved as necessary.

4.2. Transaction Manager

The main goal of the *transaction manager* is to orchestrate the execution of transactions (their fate - commit/abort) as well as the recovery of objects, resource managers, or sites after failures. By accomplishing these tasks, the transaction manager, governed by a transaction model, provides a rigorous basis for automatically enforcing a criterion for DB consistency for a set of multiple concurrent read/write accesses to the DB in the presence of potential system failures.

As a matter of fact, when processing complex applications and thereby executing more complex transactions, it turns out that single-level transactions do not achieve optimal flexibility and performance. As a solution, the concept of nested transactions was popularized by Moss [Mo85], where single-level transactions are enriched by an inner control structure. Such a mechanism allows for the dynamic decomposition of a transaction into a hierarchy of subtransactions thereby preserving all properties of a transaction as a unit and assuring *atomicity* and *isolated execution* for every individual subtransaction. These aspects lead to advantages in a computing system like: Intra-transaction parallelism, intra-transaction recovery control, explicit control structure, system modularity, and distribution of implementation [Mo85, HR93]. Additionally, nested transactions lead to some more significant advantages in KBMSs. Among others, they provide adequate control structures for modeling: Methods inside methods, complex functions, and virtual rule processing [RH95]. Further on, nested transactions offer suitable control structures for the realization of main-memory-based query processing.

Due to these inherent functionalities and to the very end of achieving a high degree on intra-transaction parallelism, we have chosen to implement a nested transaction model in our architecture [RH95, Zi95]. In our model, we allow the transactions in all levels to be executed simultaneously, achieving by this means the maximum degree of parallelism in as well as among transaction hierarchies. Further, besides supporting the basic concept of *upward inheritance of locks* [Mo85], our model allows for *controlled downward inheritance of locks* [HR93], in order to enable a transaction to restrict the access mode of its inferiors for an object [RH95].

4.3. Lock Manager

To guarantee isolation, the transactions set *locks* on the objects when accessing them. The main task of the *lock manager* is to control such locks on objects, thus regulating the concurrent access to objects. By this means, it prevents a transaction from both seeing uncommitted updates of as well as altering the data read or written by concurrently running transactions. Among the most important classes of concurrency control mechanisms are *locking*, *timestamps*, and *serialization graphs* [BHG87]. In particular, the class of locking-based algorithms has shown its practicality and performance. Additionally, locking-based algorithms have special solutions for graph structures, the abstractions for KBs that appear to be the most appealing [CHM92, RH94]. Consequently, we have chosen to develop our concurrency control technique for KBs based on locking.

As seen, the key feature of KBs is the presence of several semantic relationships. The basic idea which has originated our lock mechanism tailored for KBMSs (*LARS - Locks using Abstraction Relationships' Semantics* [RH94, RH96, Lu96]) is based on this feature. The most important characteristic of LARS is the partition of the KB graph into several logical ones, allowing by this way transactions to concurrently access such partitions through different points of view.

Thereafter, to each one of these partitions, LARS applies granular locks, providing thus many different lock types and taking the necessary precautions with respect to the dynamism of the KB graph. In this manner, LARS captures more of the semantics contained in the KB graph in the sense that it does not consider descendants of an object as being simply descendants of it, but, on the contrary, descendants with special characteristics and significance, which are based on the abstraction relationships of *classification*, *generalization*, *association*, and *aggregation*. This is the most important point of LARS, by means of which it can obtain a high degree of concurrency, exploiting the inherent parallelism in a knowledge representation approach.

4.4. Deadlock Manager

As briefly explained, *deadlock* situations may appear between transactions in their process of acquiring locks on objects. Such situations happen whenever there is a cyclical sequence of transactions (T) each waiting for the next to release a lock it must acquire ($T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_1$), and hence no one in the cycle can make any progress. To detect and above all resolve such situations are the main tasks of the *deadlock manager*. The literature reflects different opinions about deadlock management. Gray and Reuter [GR93] advocate that deadlocks are very, very rare events (rare^2), and Härder [Hä84] states that there are no general purpose deadlock-free locking protocols that always provide a high degree of concurrency. On the other hand, Yannakakis [Ya82] holds the opinion that it is very easy to construct scenarios where deadlocks arise, and Silberschatz and Kedem [SK80] state that deadlock detection and recovery in general is an expensive task and should be avoided whenever possible. Anyway, our lock technique does not avoid deadlocks, and so we must detect and resolve them.

There are a lot of strategies to detect deadlocks. One of them is *timeout*, where the system, finding that a transaction is waiting too long for a lock, just guesses that there may be a deadlock involving this transaction and simply aborts it and restarts it later again (although imprecise in the detection of deadlocks, it works). *Waits-for-graph* is another strategy, where the system maintains a graph showing which transactions are waiting for other ones. When a cycle is found in this graph, it precisely means that the transactions in the cycle are deadlocked. The system then chooses one of them as a *victim*, aborts it, obliterating its effects from the DB, and restarts it later again. Particularly, we feel that timeout does not always offer an optimal solution to deadlocks. Although being very easy to implement, the number of transactions that may be unnecessarily aborted and restarted again may be unacceptably high, due to the impreciseness of this technique. On the other hand, waits-for-graph shows a very good precision for all kinds of transactions, independently of their duration.

The deadlock manager must be aware of the transactions' nesting. In the nested transaction model, deadlocks may occur among transactions belonging to various transaction hierarchies and even among subtransactions within a single transaction hierarchy. In contrast to single-level transactions where *direct-waits-for-lock* relations are sufficient to search for waiting cycles among transactions, detection of all deadlocks in nested transactions further requires the maintenance of *waits-for-commit* relations. If deadlocks are frequently anticipated, opening-up deadlocks, which may span transaction trees, should be detected as early as possible to save transaction work [HR93]. For this purpose, our deadlock manager maintains further information to detect opening-up deadlocks as early as possible (*indirect-waits-for-lock* relations). Finally, our strategy additionally represents in the waits-for-graph *detection arcs*, by means of them, and only them, cycles in the graph can be found out. They represent an abstraction of the other waiting relations, and serve the only purpose of detecting deadlocks' occurrences. Hence, they significantly alleviates the search for cycles' process, since the analysis of the whole waiting relations in the graph every time a waiting situation occurs is no longer necessary [RG96, Gl96].

4.5. Recovery and Log Managers

As a matter of fact, the DB must contain all of the effects of committed transactions and none of the aborted ones. Essentially, the system must provide for the safety of the data stored despite failures. This is the primary responsibility of the *recovery manager*. The recovery manager relies on a *log manager*, which implements a sequential file of all the updates of transactions to objects, so that a consistent version of all objects can be reconstructed in case of failure.[3]

Most of the commercially available DBMSs employ the Write Ahead Logging (WAL [Gr78]) protocol. Traditionally, recovery strategies following the WAL protocol and doing entry logging employ a unique *log sequence number* (LSN) per page in order to track the page's state with respect to logged actions to the page. This is usually so, no matter whether a locking granule finer than a page is supported. Notwithstanding, if we think of such large pages of the 4T machines' era (e.g. 32 KB), it turns out that one LSN per page is insufficient for precisely tracking what is really going on inside a page. This is a fact because such large pages may store hundreds/thousands of objects. However, one operation affecting any of these objects must be generalized by such recovery strategies as being an operation performed to a page, simply because a page's LSN disregards the contents of the page. This generalization leads to a lack of information, with hard consequences to the restart processing in such recovery strategies.

We have designed a recovery strategy, called WALORS (*WAL-based and Object-oriented Recovery Strategy* [RB96, Ba95]), focusing on such aspects like very large page sizes. When we started designing WALORS, our main goal was to cope with the problem mentioned above, and thus to more precisely track the state of the objects inside a page, rather than the state of the page as a whole. In order to achieve this goal, we have then employed an LSN per object. Hence, each object carries its own LSN in WALORS. On this basis, we can be very precise when analyzing the log records, and track a log record to the specific object that it gives respect to, and not to the set of objects inside a page. The general features of WALORS are:

- WALORS was designed to cope with *transaction, system*, and *media failures*. Additionally, WALORS also supports partial rollbacks.[4]
- In behalf of a better performance during normal processing, since failures should be considered as exceptions, WALORS does *entry logging at the object level.*[5]
- As the acronym itself betrays, WALORS follows the WAL protocol.
- WALORS supports fine-granularity locking, i.e., object-level locking.
- WALORS does selective redo (i.e., it avoids repeating the history [MHLPS89] to gain a unique DB representation after a crash) as well as selective undo passes. Additionally, WALORS dynamically builds at restart time a *forward chain*, in which all log records to be redone take part, and by means of which a very efficient redo processing is possible.
- WALORS is flexible enough and does not make any kind of restrictive assumptions about the buffer management policies being employed (*force/no-force*, *steal/no-steal* [HR83]). In

3. The reader may have noticed that we have not considered log files at the workstation side (Fig. 4). The main reason for that is merely to avoid having to do log merges, and to support a broader class of workstations, especially the ones where no disk is available. However, due to its features, our recovery strategy may be straightforwardly extended in order to cope with several log files and hence log merges.
4. Partial rollbacks are crucial for user-friendly handling integrity constraint violations, application program failures and deadlocks [LS87], problems arising from using obsolete cached data [LHMWY84], etc.
5. Entry logging allows to do logical logging and supports semantically rich lock modes (e.g., based on the commutativity of operations [Ga83, SS84, BR88, We88, FO89, RGN90, CRR91, HH91]). (Notwithstanding, we have strived for maintaining WALORS as simple as possible and hence avoiding complex, and thus error-prone, algorithms.)

our implementation, the no-force/steal policy is in effect.

Finally, we have extended WALORS to work in a client/server environment, and to support nested transactions. In this context, WALORS follows our nested transaction model [RH95], and supports the enhanced lock modes for KBMSs provided by LARS [RH94, RH96].

4.6. Cache Manager

By employing large buffers, *caching* can substantially reduce the amount of expensive and slow disk accesses by utilizing locality of reference [EH84]. As seen, in order to provide then faster access to objects, each workstation has its own *cache*, where the objects are temporarily stored. By such a means, a single object can be stored into different workstations' caches at the same time. If such objects reside in the caches just during the processing of transactions, their coherency is guaranteed by means of the lock manager. Nevertheless, to the end of exploiting even more the near-by-the-application locality principle, such objects should stay in the caches beyond transaction boundaries. Consequently, a generic mechanism must regulate the coherency of the objects in the many caches. This is the responsibility of the *cache manager*.

Hence, a *buffer invalidation* problem arises with caching, since a modification of the object in any buffer invalidates copies of this object in other workstations as well as its copy stored in secondary storage. The basic task of cache coherency is to ensure that transactions always see the most recent version of the objects despite buffer invalidations.

In general, buffer invalidations can be either *detected* or *avoided*, and choosing the best alternative to solve this problem depends on the concurrency control policy (optimistic/pessimistic), the strategy used for update propagation to disk (force/no-force), and the granule for locking (page/object) [Ra91]. Considering the features of both our lock and recovery managers, we have chosen to implement an avoidance scheme similar to *retention locks* [HR85], but based purely on *client locks* at the server side [MN91]. In this approach, for every cached object only a client lock exists in the server. In turn, transaction locks exist in the workstation for the objects accessed by its transactions. Before an object can be modified at any workstation, an exclusive transaction lock must be granted to the transaction, and for that an exclusive client lock must be requested to the server, which may then conflict with other client locks. In addition, a client lock is released only if no corresponding transaction lock exists. Further, on releasing a client lock, the object is purged from the buffer so that its invalidation is avoided. Finally, our scheme follows a *dynamic owner* principle, and allows by this means objects to be exchanged directly among the workstations, alleviating thus the communication with the server [Ha96].

4.7. KB Graph Manager

As discussed, our lock manager sets locks on objects in favor of transactions by using the semantics of the abstraction relationships. In order to do that, it needs a structural representation of the objects in a KB and their abstraction relationships. To build and maintain such a representation, the KB graph, are the main tasks of the *KB graph manager*. This manager maintains a single-rooted directed acyclic graph, where the objects represent the nodes and the abstraction relationships between the objects the edges. It provides interfaces for inserting, removing, and querying the contents of the graph. Additionally, some general management tasks, like representation of object LSNs and storing the lock modes on objects, are also included in the functionality of this manager [Ka95].

4.8. Communication Manager

All these above mentioned managers have their own communication interfaces, both local at the workstation side and local/global at the server side. To control the managers' traffic of information in the workstation as well as between workstations and server is the main purpose of the *communication manager*. Our communication manager provides synchronous (using RPC) and asynchronous (using multi-threads) communication facilities, and ensures safe, reliable information exchanges [He96].

5 Conclusions

We have discussed the relevant processing characteristics of KBMSs, and presented the basic organizational forms of workstation/server architectures. We have appointed the query server approach as the most appropriate one for KBMSs, especially because (1) it provides a powerful query interface by means of which the application can formulate its object requests, (2) object faults are declaratively recognized, (3) sets of objects are delivered by the server upon request, (4) an object can be the granule of synchronization and logging, (5) the objects' views can be tailored to the application's necessities, (6) the volume of information to be fetched is significantly reduced, and finally, (7) ill-formed clusters do not increase the communication overhead.

We have introduced the single-user KBMS KRISYS, the practical vehicle we are currently using to implement our multi-user KBMS techniques. Following, we have introduced the basic principles of transaction management together with the fundamental properties of transactions (ACID properties). In this scope, we have presented a general architecture for multi-user KBMSs where general transaction services are then considered. Thereafter, we have extended this architecture with the main components for providing transaction facilities, and discussed their locations in the architecture and their team work. Finally, we have given an overview of these components' tasks and goals, and highlighted their most distinguishing qualities.

Our transaction manager implements a nested transaction model, and supports upward as well as controlled downward inheritance of locks. Our lock manager implements LARS, which sets granular locks using the semantics of the abstraction relationships. The introduction of the detection arcs in our deadlock management policy has enormously facilitated the deadlock manager's work of detecting the occurrence of a deadlock. Our recovery strategy, WALORS, employs an LSN per object, and by such a means it can very precisely track the state of each particular object with respect to logged actions to it, a distinguishing feature as long as the page sizes are being enlarged, and which enables it to do selective undo as well as selective redo passes during system restart. Our cache manager avoids buffer invalidations by means of client locks, and implements a dynamic owner policy, enabling the workstations to exchange objects between themselves, and alleviating thus the communication with the server. Our KB graph manager represents the KB objects as nodes in a DAG, and the abstraction relationships as edges between the nodes, therefore giving support to the other components. Finally, the communication manager provides synchronous as well as asynchronous communication facilities to the other components, ensuring a safe, reliable information exchange.

At last, as a point for future work and further optimization, we are currently interested mainly in two issues. On one hand, our architecture, as any centralized DB architecture, has its Achilles's heel: A single point of failure and which may become a bottleneck when the system grows. To optimize this, we intend to investigate architectures containing clusters of servers, which jointly manage a single (distributed) DB. By such a means, we can achieve high levels of availability, resilience, as well as scalability. On the other hand, we intend to enter the area of distribution, and research aspects of providing the transaction facilities as much as possible locally, e.g., allowing transactions to independently commit locally, and handling client crashes by the own clients. This is particularly important to give support for the ever increasing area of mobile computing.

Acknowledgments

The research project here presented would perhaps not have reached its actual stage of development and implementation without our team work with Jörg Lutze, Christian Kasparek, Jan Zielinski, Thomas Baier, Andreas Gloeckner, Ulrich Hermsen, and Victoria Hall. We would like to acknowledge all of them.

References

[AGKLP93] Ananthanarayanan, R., Gottemukkala, V., Käfer, W., Lehman, T.J., Pirahesh, H.: Using the Coexistence Approach to Achieve Combined Functionality of Object-Oriented and Relational Systems. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Washington, DC, USA, May 1993. pp. 109-118

[Ba95] Baier, T.: *WALORS - A WAL-Based and Object-Oriented Recovery Strategy* (in German). Diploma Work, Univ. of Kaiserslautern, Germany, Nov. 1995.

[BHG87] Bernstein, P.A., Hadzilacos, V., Goodman, N.: *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, USA, 1987.

[BL86] Brachman, R., Levesque, H.: *The Knowledge Level of KBMS*. In: [BM86]. pp. 9-12.

[BM86] Brodie, M., Mylopoulos, J. (Eds.): *On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies*. Springer-Verlag, New York, USA, 1986. (Topics in Information Systems).

[Br86] Brodie, M.L.: Future Intelligent Information Systems: AI and Database Technologies Working Together. In: Mylopoulos, J., Brodie, M.L. (Eds.), *Artificial Intelligence and Databases*, Morgan Kaufmann, USA, 1986.

[BR88] Badrinath, B.R., Ramamritham, K.: Synchronizing Transactions on Objects. *IEEE Transactions on Computers*, Vol. 37, No. 5, May 1988. pp. 541-547.

[CHM92] Chaudhri, V.K., Hadzilacos, V., Mylopoulos, J.: Concurrency Control for Knowledge Bases. In: *Proc. of the 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, Cambridge, USA, 1992. pp. 762-773.

[CRR91] Chrysanthis, P.K., Raghuram, S., Ramamritham, K.: Extracting Concurrency from Objects: A Methodology. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Denver, USA, May 1991. pp. 108-117.

[De91] Deux, O. et al.: The O_2 System. *Communications of the ACM*, Vol. 34, No. 10, 1991. pp. 34-49.

[De93] Dessloch, S.: *Semantic Integrity in Advanced Database Management Systems.* Doctor Thesis, Univ. of Kaiserslautern, Germany, Sept. 1993.

[DMFV90] DeWitt, D.J., Maier, D., Futtersack, P., Velez, F.: A Study of Three Alternative Workstation/Server Architectures for Object-Oriented Databases. In: *Proc. of the VLDB*, Brisbane, Australia, 1990. pp. 107-121.

[EGLT76] Eswaran, K.P., Gray, J.N., Lorie, R.A., Traiger, I.L.: The Notions of Consistency and Predicate Locks in a Database Systems. *Communications of the ACM*, Vol. 19, No. 11, Nov. 1976. pp. 624-633.

[EH84] Effelsberg, W., Härder, T.: Principles of Database Buffer Management. *ACM TODS*, Vol. 9, No. 4, 1984.

[FO89] Farrag, A.A., Ozsu, M.T.: Using Semantic Knowledge of Transactions to Increase Concurrency. *ACM TODS*, Vol. 14, No. 4, Dec. 1989. pp. 503-525.

[Ga83] Garcia-Molina, H.: Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM TODS*, Vol. 8, No. 2, Jun. 1983. pp. 186-213.

[Gl96] Gloeckner, A.: *Deadlock Management in Nested Transactions in KRISYS* (in German). Project Work, Univ. of Kaiserslautern, Germany, Aug. 1996.

[GLPT76] Gray, J.N., Lorie, R., Putzolu, F., Traiger, I.L.: Granularity of Locks and Degrees of Consistency in a Shared Data Base. In: *Proc. of the IFIP Working Conf. on Modelling in DBMS*, Freudenstadt, Germany, Jan. 1976. pp. 365-394.

[Gr78] Gray, J.N.: Notes on Database Operating Systems. In: Bayer, R., Graham, R.M., Seegmueller, G. (Eds.), *Operating Systems: An Advanced Course*, Springer-Verlag, Berlin, Germany, 1978. pp. 393-481. (LNCS 60).

[Gr95] Gray, J.N.: Super-Servers: Commodity Computer Clusters Pose a Software Challenge. In: *Proc. of the BTW*, Dresden, Germany, Mar. 1995. pp. 30-47.

[GR93] Gray, J.N., Reuter, A.: *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, USA, 1993.

[Ha96] Hall, V.: *Cache Coherency in a Client/Server Knowledge Base Management System* (in German). Diploma Work, Univ. of Kaiserslautern, Germany, July 1996.

[Hä84] Härder, T.: Observations on Optimistic Concurrency Control Schemes. *Information Systems*, Vol. 9, No. 2, 1984.

[He96] Hermsen, U.: *Communication Management in a Client-Server/Multi-User KBMS* (in German). Project Work, Univ. of Kaiserslautern, Germany, March 1996.

[HH91] Hadzilacos, T., Hadzilacos, V.: Transaction Synchronization in Object Bases. *Journal of Computer and Systems Sciences*, Vol. 43, No. 1, Aug. 1991. pp. 2-24.

[HHMM88] Härder, T., Hübel, C., Meyer-Wegener, K., Mitschang, B.: Processing and Transaction Concepts for Cooperation of Engineering Workstations and a Database Server. *Data and Knowledge Engineering*, Vol. 3, 1988. pp. 87-107.

[HMNR95] Härder, T., Mitschang, B., Nink, U., Ritter, N.: Workstation/Server Architectures for Database-Based Engineering Applications (in German). *Informatik Forschung & Entwicklung*, 1995.

[HR83] Härder, T., Reuter, A.: Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys*, Vol. 15, No. 4, Dec. 1983. pp. 287-317.

[HR85] Härder, T., Rahm, E.: Quantitative Analysis of a Synchronization Protocol for DB-Sharing (in German). In: *Proc. of the 3rd Conf. on Measurement, Modeling, and Evaluation of Computer Systems*, 1985. pp. 186-201.

[HR93] Härder, T., Rothermel, K.: Concurrency Control Issues in Nested Transactions. *VLDB Journal*, 2 (1), Jan. 1993.

[HS93] Hübel, C., Sutter, B.: *DB-Integration of Engineering Applications - Models, Tools, Controls* (in German). Database Systems Series, Vieweg, Germany, 1993.

[Ka95] Kasparek, C.: *The KB Graph Management in the Multi-User KBMS KRISYS* (in German). Internal Report, Univ. of Kaiserslautern, Germany, 1995.

[KDG87] Küspert, K., Dadam, P., Günauer, J.: Cooperative Object Buffer Management in the Advanced Information Management Prototype. In : *Proc. of the 13th VLDB*, Brighton, UK, Sept. 1987. pp. 483-492.

[Ki90] Kim, W.: *Introduction to Object-Oriented Databases*. MIT Press, USA, 1990. (Series in Computer Systems).

[KJA93] Keller, A., Jensen, R., Agrawal, S.: Persistence Software: Bridging Object-Oriented Programming and Relational Database. In: *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Washington, DC, USA, May 1993. pp. 523-528.

[Ko81] Kohler, W.H.: A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems. *ACM Computing Surveys*, Vol. 13, No. 2, June 1981. pp. 149-183.

[LHMWY84] Lindsay, B.G., Haas, L.M., Mohan, C., Wilms, P.F., Yost, R.A.: Computation and Communication in R*: A Distributed Database Manager. *ACM Transactions on Computer Systems*, Vol. 2, No. 1, Feb. 1984. pp. 24-38.

[LLOW91] Lamb, C., Landis, G., Orenstein, J., Weinreb, D.: The ObjectStore Database System. *Communications of the ACM*, Vol. 34, No. 10, Oct. 1991. pp. 50-63.

[LS87] Lockemann, P.C., Schmidt, J.W. (Eds.): *Databases Handbook* (in German). Springer-Verlag, Germany, 1987.

[Lu96] Lutze, J.: *Lock Management in the KBMS KRISYS - An Implementation of the LARS Protocol for Nested Transactions* (in German). Project Work, Univ. of Kaiserslautern, Germany, Jan. 1996.

[LW94] Lee, B., Wiederhold, G.: Outer Joins and Filters for Instantiating Objects from Relational Databases Through Views. *IEEE Knowledge and Data Engineering*, Vol. 6, No. 1, 1994. pp. 108-119

[Ma91] Mattos, N.M.: *An Approach to Knowledge Base Management*. Springer-Verlag, Germany, 1991. (LNAI 513).

[MHLPS89] Mohan, C., Haderle, D., Lindsay, B.G., Pirahesh, H., Schwarz, P.: *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging*. IBM ARC Research Report RJ6649, San Jose, USA, Jan. 1989.

[MN91] Mohan, C., Narang, I.: Recovery and Coherency-Control Protocols for Fast Intersystem Page Transfer and Fine-Granularity Locking in a Shared Disks Transaction Environment. In: *Proc. of the VLDB*, Barcelona, Spain, 1991.

[Mo85] Moss J.E.B.: *Nested Transactions: An Approach to Reliable Distributed Computing*. MIT Press, USA, 1985.

[MPPLS93] Mitschang, B., Pirahesh, H., Pistor, P., Lindsay, B., Südkamp, S.: SQL/XNF - Processing Composite Objects as Abstractions over Relational Data. In: *Proc. of the Int. Conf. on Data Engineering*, Vienna, Austria, April 1993.

[On91] Ontologic Inc.: *ONTOS Developer's Guide*. Billerica, USA, 1991.

[Ra91] Rahm, E.: *Concurrency and Coherency Control in Database Sharing Systems*. ZRI Research Report No. 3/91, Univ. of Kaiserslautern, Germany, Dec. 1991.

[RB96] Rezende, F.F., Baier, T.: A WAL-Based and Object-Oriented Recovery Strategy (in German). In: *Proc. of the 7th Workshop on Transaction Concepts*, Nieheim, Germany, Jan. 1996. GI Datenbank Rundbrief 17, May 1996.

[RD91] Roussopoulos, N., Delis, A.: Modern Client-Server DBMS Architectures. *ACM SIGMOD RECORD*, Vol. 20, No. 3, Sept. 1991. pp. 52-61.

[RG96] Rezende, F.F., Gloeckner, A.: *Deadlock Management in Nested Transactions Using Detection Arcs*. Internal Report, Univ. of Kaiserslautern, Germany, 1996.

[RGN90] Rakow, T.C., Gu, J., Neuhold, E.J.: Serializability in Object-Oriented Database Systems. In: *Proc. of the 6th Int. Conf. on Data Engineering*, Los Angeles, USA, Feb. 1990. pp. 112-120.

[RH94] Rezende, F.F., Härder, T.: A Lock Method for KBMSs Using Abstraction Relationships' Semantics. In: *Proc. of the 3rd Int. Conf. on Information and Knowledge Management*, Gaithersburg, USA, Nov. 1994. pp. 112-121.

[RH95] Rezende, F.F., Härder, T.: Concurrency Control in Nested Transactions with Enhanced Lock Modes for KBMSs. In: *Proc. of the 6th Int. Conf. on Database and Expert Systems Applications*, London, UK, Sept. 1995. pp. 604-613.

[RH96] Rezende, F.F., Härder, T.: Multiple Granularity Locks for the KBMS Environment. In: Fong, J., Siu, B. (Eds.), *Multimedia, Knowledge-Based & Object-Oriented Databases*, Springer Verlag, Singapore, 1996. pp. 126-148.

[SK80] Silberschatz, A., Kedem, Z.: Consistency in Hierarchical Database Systems. *Journal of the ACM*, Vol. 27, No. 1, Jan. 1980. pp. 72-80.

[SS84] Schwarz, P.M., Spector, A.Z.: Synchronizing Shared Abstract Types. *ACM TOCS*, Vol. 2, No. 3, Aug. 1984.

[TMMD93] Thomas, J., Mitschang, B., Mattos, N.M., Deßloch, S.: Enhancing Knowledge Processing in Client/Server Environments. In: *Proc. of the 2nd CIKM*, Washington, DC, USA, Nov. 1993. pp. 324-334.

[Ve90] Versant Object Technologies Inc.: *VERSANT Technical Overview*. Menlo Park, USA, 1990.

[We88] Weihl, W.E.: Commutativity-Based Concurrency Control for Abstract Data Types. *IEEE Transactions on Computers*, Vol. 37, No. 12, Dec. 1988. pp. 1488-1505.

[Ya82] Yannakakis, M.: A Theory of Safe Locking Policies in Database Systems. *Journal of the ACM*, Vol. 29, No. 3, 1982. pp. 718-740.

[Zi95] Zielinski, J.: *An Approach to Transaction Management in a Knowledge Base Management System* (in German). Diploma Work, Univ. of Kaiserslautern, Germany, Oct. 1995.