Towards Effective Support of Engineering Information Systems

Christoph Hübel, Bernhard Mitschang

University Kaiserslautern
Erwin-Schrödinger-Straße
D - 6750 Kaiserslautern
Federal Republic of Germany

Abstract

Existing interfaces to nowadays database systems show to be increasingly unsuitable for the evolving wide spectrum of engineering applications (e.g. CAD/CAM, geographic information management, knowledge-based systems for planning and design, etc.). This is further intensified due to the workstation-oriented processing scheme prevailing in the engineering area.

Starting with an architectural approach tailored to this distributed processing concept, we propose the PRIMA-NDBS and its most important interfaces, i.e. the data model and the application/user interface offering effective support of engineering information systems.

1. Introduction

Some of the main characteristics of engineering information systems comprise

- handling the design process,
- cooperation of different users and their design activities, and
- demands for generally available services in a local area network.

In the engineering application area workstations are predominant. They are tailored to the applications' or the users' requirements exploiting novel techniques, e.g. graphical interfaces and multi-window techniques. This workstation concept easily allows for an extensive local and efficient processing of the user's operations. All services requested are mainly restricted to data management, recovery facilities, integrity control, as well as synchronization and autorization (i.e. organization of multi-user processing). These services coincide with the capabilities offered by database systems (DBS) which have to be installed on a host computer, in the context of this workstation-oriented scenario.

However, the practical use of DBS in such a workstation-host environment raises some major problems. Firstly, conventional DBS and their interfaces offer only poor support for engineering applications. The reason for this lies in the inadequacy of their data modeling constructs as well as in their inappropriate and inefficient operational support [BB84, Mi85, HHLM87]. The performance deficiency primarily stems from the fact that conventional DB processing is not locality preserving and that it does not take into account the distributed workstation-host processing.

To solve these problems, we come up with an architecture for so-called non-standard database systems (NDBS), which is tailored to the workstation-host environment. The two most important interfaces of our NDBS approach are explained and their assistance towards effective support of engineering informations systems in highlighted.

2. The Architectural Approach

The kernel architecture is a promising candidate for an NDBS architecture which also has been exploited in other similar projects [DA86,PSSWD87]. As a key idea, the underlying NDBS architecture (fig. 1) is divided into two parts:

- the kernel offers neutral data management functions while
- the application layer (AL) provides application specific support.

The advantages of this approach are on one hand the application orientation of the AL offering the so-called application model interface, i.e. the desired application objects and operations, on the other hand the kernel unites all neutral data representation and access facilities in an efficient manner. Thus, the kernel realizes a neutral data model (or data model implementation) as a basis for the semantically enriched application-specific models within the AL.

An important criteria towards a model's suitability for a 'kernel data model' is the degree of 'object-orientation' offered by the data model. Underset by several detailed analyses [Hä88], the following requirements are most important:

- direct definition of the desired object granules (i.e. structured heterogeneous record sets) to work with, and
- adequate object processing supporting both vertical access to heterogeneous record sets (complex object) as well as horizontal access to sets of (complex) objects.

These neutral concepts embody structural object orientations [Di86]. Supplementary to this, it is the task of the to AL provide application-specific support, i.e. behavioral object orientation [Di86]. AL maps a concrete application model (that is the application objects and their associated operations) onto the data model interface of the kernel. Since application orientation is obtained by the AL, different types of application layers are designed for different application classes. Hence, the interface between kernel and AL seems appropriate to decompose the NDBS when distributed processing has to be supported in an engineering environment. The AL together with the true engineering application is assigned to a workstation whereas the central server carries the kernel and handles all kernel requests [HHMM87].

This homogeneity between hardware and software architecture offers some important advantages:

- The set orientation together with the structural object orientation of the kernel data model allows for requesting sets of complex objects, thus minimizing the communications overhead between workstation and host.
- The locality of data references during major processing steps could be kept and exploited within the workstation site.
- Distributed processing allows for mutual failure masking that yields to an increased application autonomy; a failure on the workstation should not bother the server and vice versa.
- Increasing the workstation capacity promises true performance gains.

This architectural approach provides an application-oriented modeling tool simultaneously rising to the performance demands of engineering applications.

In the following chapters we give an overview of the PRIMA project, which is a major endeavor towards an NDBS realization done at the University Kaiserslautern. Firstly, the kernel data model, here termed Molecule Atom Data model (MAD) and its kernel implementation called PRIMA (Prototype Implementation of the MAD model) are introduced. Then we focus on the concepts inherent to the AL of our PRIMA-NDBS. More detailed information concerning the PRIMA project can be found in [Hä88]. Here, we only want to stress the major concepts of the PRIMA-NDBS and how they fit together yielding to an NDBS that effectively supports engineering information systems.

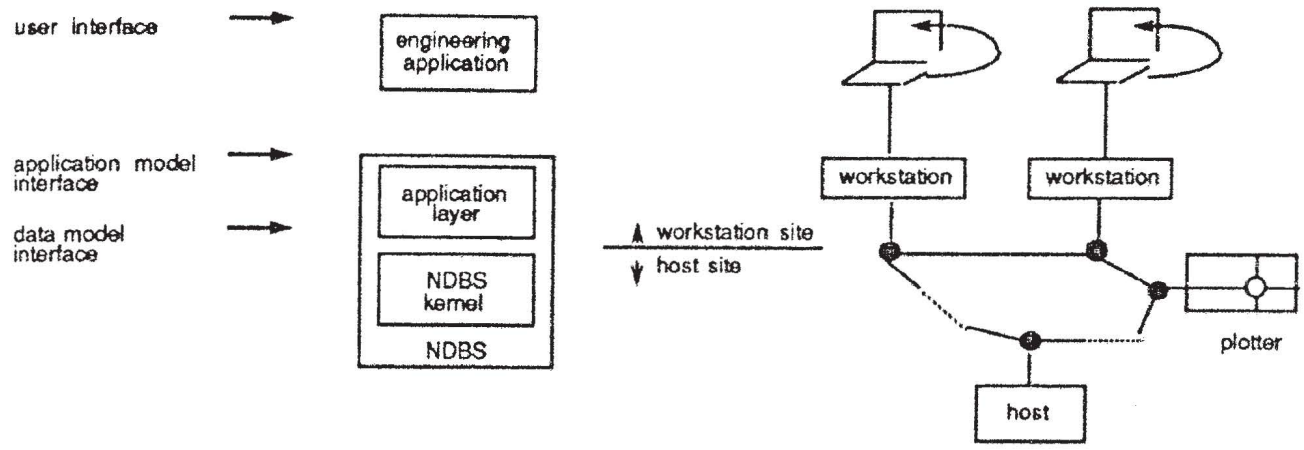


Fig. 1: Architecture of NDBS-based engineering information systems

3. The Molecule-Atom Data Model

In the following, we present an introduction to the MAD model and its molecule query language MQL providing structural object orientation (a detailed description of the MAD model resp. MQL can be found in [Mi88a]). MQL is embedded in a host programming language and can be directly used in an application programming environment; additionally, interactive operation is supported. Similar to SQL [X3H286], MQL is subdivided into three parts reflecting data definition (DDL), load definition (LDL), and data manipulation (DML). Here, we focus on the latter, that is, on query (i.e. retrieval) and manipulation (i.e. insertion, deletion, and modification) capabilities for complex object management.

Underlying General Concepts

The basic elements (building blocks) of the MAD model are called atoms. They play a similar role to tuples in the relational model. Each atom is composed of attributes of various types, is uniquely identifiable, and belongs to its corresponding atom type. The attributes' data type can be chosen from a richer selection than in conventional data models.

The type concept has been extended by RECORD, ARRAY, and the repeating-group types SET and LIST to yield a powerful structuring capability at the attribute level. For identification and connection of atoms, we have introduced two special types. The IDENTIFIER type serves as a surrogate [ML83], which allows for the identification of each atom. Based on this type, it is easy to define the REFERENCE type allowing for typed references (that is, logical pointers) to other atoms of the same or of different type (similar to foreign/primary-key connections). This basic mechanism for connecting atoms is called the link concept. Organized as repeating-group attributes, these links may be used to efficiently map n:m relationships and recursions. A link is always symmetric in that the referenced record must contain a 'back reference' that can be used in exactly the same way.

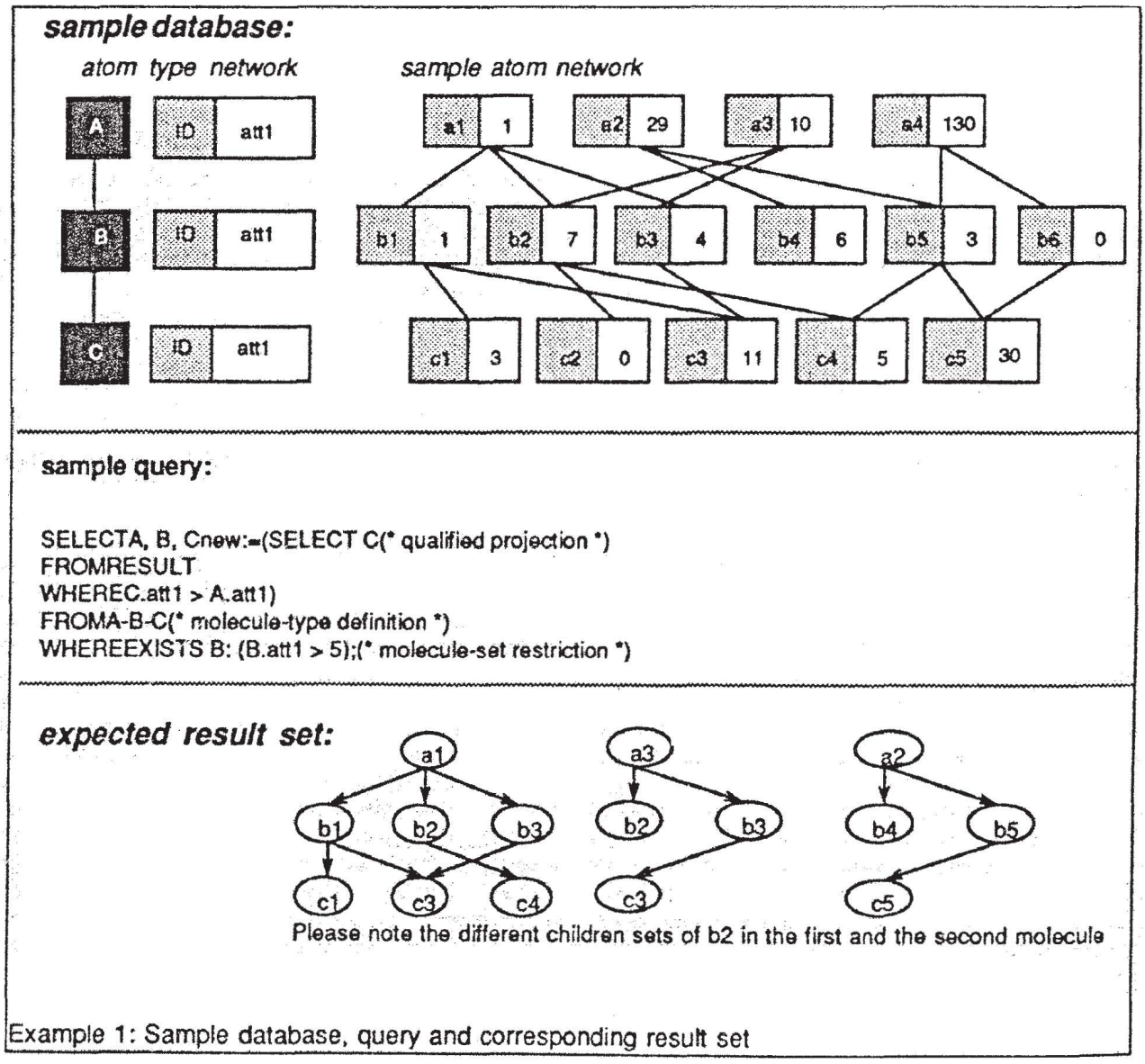
Based on the link concept, it is feasible to dynamically construct molecules using atoms as elementary building blocks. Each molecule belongs to its corresponding molecule type. The molecule type is defined (in the query language, not in the schema) by naming the atom types and link types. For the purpose of molecule construction, a direction is assigned to all link types specified. Each molecule type determines both the molecule structure and the corresponding molecule set, grouping all the molecules with the same structure. The molecule structure is superimposed dynamically on the atom network consisting of sets of atoms linked by references. Thus, the concept of dynamic molecules is introduced, yielding the required complex object notion of the MAD model.

Query and Manipulation Facilities

The operational power of the MAD model lies in adequate means for molecule processing provided by MQL. Analogously to SQL, there are three basic language constructs:

- The FROM clause specifies the molecule type to work with.
- The WHERE clause allows for the restriction of the corresponding molecule set.
- The projection clause (i.e. the SELECT clause in the case of retrieval statements) defines the final molecule structure and is responsible for proper molecule projection.

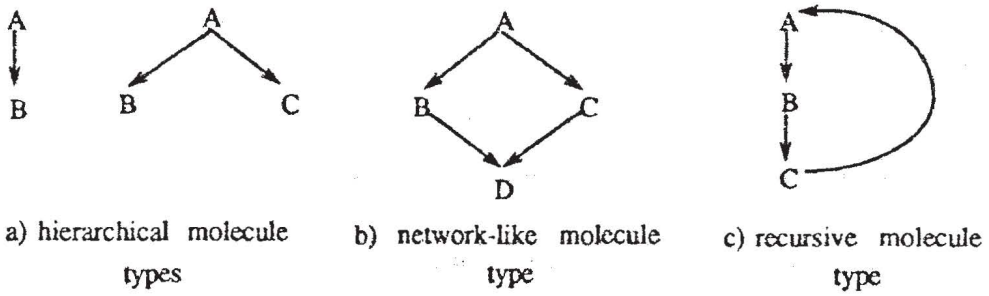
Compared to SQL, these constructs exhibit extended semantics and syntax according to the now more complex objects which have to be dealt with. They form the basis of all DML-statements offered. The result of each molecule operation is defined by a molecule type. It can be shown [Mi88b] that the closure of the MAD model under its molecule operations is guaranteed. This is a very important fact, which allows for the nesting of molecule queries (cf. query in example 1); each molecule-type specification (e.g. A-B-C in example 1) can be replaced by a molecule query.



Example 1: Sample database, query and corresponding result set

In the following, we wish to illustrate the descriptive and operational power of the MAD-DML in more detail, thereby refining the above introduced basic clauses, which are depicted within example 1. The upper part of example 1 shows a sample database consisting of some atom type specifications in combination with link type specifications and corresponding occurrences. The sample query demonstrates important language features of MQL. The result of this query applied to our database is shown in the lower part.

The FROM clause of each given DML-statement determines the molecule type to operate on. The molecule structure is explicitly specified (in our case A-B-C), and the corresponding molecule set is determined using a simplified, but in this case sufficient abstract model: Starting with the atoms of the 'root' atom type, all component atoms referenced via specified link types are used for molecule construction. Referring to our example, the first atom a1 of the root atom type A is selected. Afterwards, all atoms of type B linked to a1 (i.e. b1, b2, b3) are accessed and then the referenced C atoms (c1, c2, c3, c4) are selected. This procedure is repeated for all remaining A atoms. Once a molecule has been built, qualification and projection are applied.



a) hierarchical molecule types

b) network-like molecule type

c) recursive molecule type

Example 2: Three generic types of molecule structures

There are three generic molecule types distinguishable according to their structure (cf. example 2):

- Hierarchical molecule types (cf. example 2a) define a hierarchical graph. Example 1 depicts such a sample molecule type having atom type A as the root and the atom types B and C as component types.
- The molecule structure of network-like molecule types (cf. example 2b) resembles a meshed graph. In this case, there are component types with more than one referencing link type. In graphic terms, this fact is expressed by nodes with more than one incoming edge.
- Recursive molecule types (cf. example 2c) use a hierarchical or network-like component molecule type combined with a recursion-defining link type expressed in a special RECURSIVE clause (e.g. C-A in example 2c). The resulting molecule structure is the recursively continued molecule structure of its component molecule type. The derivation of the corresponding recursive molecules has to be performed step by step in an iterative manner, going from one level (i.e. component molecule) to the next subordinate level using the recursion-defining references. Here, the transitive closure has to be computed, which could be additionally cut off by an optional restriction clause (UNTIL clause).

Although molecule types are generally defined as part of a query, it is possible to predefine frequently used molecule types and to assign to them a name.

The optional WHERE clause restricts the molecule set (determined by the molecule type of the FROM clause) to those molecules satisfying the given qualification condition. Since molecules normally comprise of a structured and heterogeneous set of atoms, it is necessary to extend the qualification facilities of the language. That is, the molecule structure has to be included, yielding quantified qualification terms. Hence, testing for the existence (EXISTS-quantifier) of atoms of a given component type or using the ALL-quantifier as an alternative quantification construct is allowed. The standard presetting used in MQL is the existential quantifier, so that the use of quantifiers is optional. The query of example 1 depicts an explicitly quantified qualification condition.

The well-known projection expressed by simply listing the components (atom types, attribute types) to be retrieved is also valid in MQL. The complementary ALL_BUT construct allows the expression of those components, that are not to

be retrieved. To retrieve the whole result set unchanged ALL may be used. Furthermore, for more selective specification of the resulting molecules, MQL introduces the so-called qualified projection (opposed to the above mentioned unqualified projection). Qualified projection is expressed as a 'SELECT...FROM...WHERE ' expression within the projection clause. This nesting allows for a supplementary restriction of the components of the result-set molecules by evaluating the qualification condition of the WHERE clause within the qualified projection. The scope of this qualification comprises the whole molecule; therefore, we use the presetting RESULT for the corresponding FROM clause. Referring to our example, only those C atoms are finally retrieved, which satisfy the qualification term stated. Exploiting these two projection capabilities, we are able to retrieve only those components (sub-molecules) of the 'surrounding' result-set molecules we are interested in. Hence, the projection clause determines the final structure of all molecules in the result set. In the case of retrieval, the SELECT clause may be extended by an order specification. Furthermore, aggregation functions like SUM and AVG can be applied. For the conversion of a molecule set into a list, the special aggregation function VALUE is offered; however, the molecule structure is restricted to one atom type consisting of one attribute type.

Without discussing all concepts in detail, we wish here to summarize the above introduced and important concepts. The operational support of the MAD model (i.e. MQL) for adequate complex object management comprises of:

- dynamic object definition by means of dynamic molecules expressed in the FROM clause by naming the atom types and their link types.
- powerful molecule restrictions within the WHERE clause exploiting quantified qualification
- molecule-component specification by means of an extended projection concept (qualified resp. unqualified projections)
- set-orientation, expressiveness and simplicity of the language.

Load Definition for "Transparent" Performance Enhancements

The main characteristics of the MAD model are set-orientation, dynamic object definition, and processing of heterogeneous record sets. These concepts offer a broad spectrum for query optimization with a number of novel optimization aspects (e.g. pipelined or parallel derivation of all molecules within the result set [Schö88]). Effective processing support could be accomplished by an appropriate set of storage structures. However, the MAD model itself makes no reference to such "physical" objects to preserve data independence. Therefore, we need a separate mechanism for the specification of the various storage structures supporting a given application.

For this purpose, we have defined a load definition language used by the database administrator. The main concepts for performance control are

- several access methods for one or more attributes permitting multidimensional access
- (vertical) partitioning of records to improve clustering of frequently accessed attributes
- sort orders to speed up sequential processing according to given sort criteria
- "physical clusters" to provide physical contiguity for atoms belonging to frequently requested molecules, i.e. materialization of molecules.

4. PRIMA - a Prototype Implementation of the MAD Model

So far, we have outlined the features of the MAD model and its transparent support by application-dependent turning mechanisms. In the following, we present an overview of the concepts and ideas used for its implementation - a detailed description concerning our implementation endeavor can be found in [Hä88].

A multi-layer DBMS architecture [As76, HR85] with well-defined internal interfaces is a prerequisite to modularity, data independence and extensibility in the various layers. Our implementation model for PRIMA illustrated in fig. 2 distinguishes three different layers for mapping molecules visible at the MAD interface onto blocks stored on external devices.

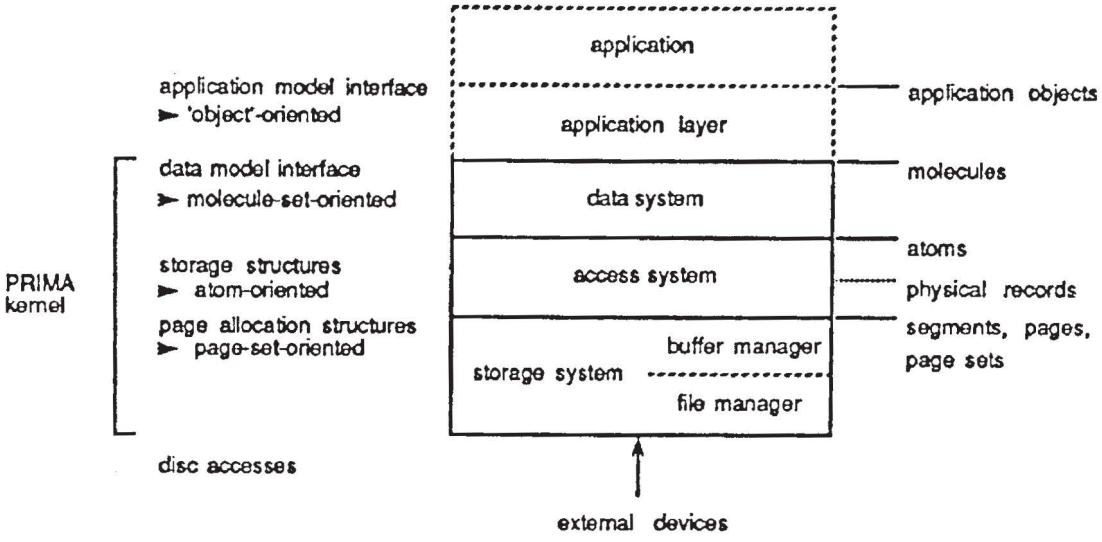


Fig. 2.: Architectural model of the PRIMA-NDBS and its application

The main task of the data system is to perform the complex mapping of the molecule-set-oriented interface onto the atom-oriented interface of the underlying access system. This is done by translating and optimizing the user-submitted MQL statements into an executable form (in terms of access system calls), while preserving their original meaning. For example, execution of an MQL statement means that the data system fetches single atoms via access systems calls, creates one result molecule (at a time) and delivers the whole result set across the kernel interface. Apparently, the data system is therefore responsible for the dynamic construction of molecules by means of the access system interface. For a detailed view to the PRIMA data system and its query processing of molecules we refer to [Schö88].

The access system (extensively described in [Si88a]) offers - comparable to the Research Storage System (RSS) of the System R prototype [As76] - an atom-oriented interface which allows for retrieval and update of single atoms. To satisfy the retrieval requirements of the data system, it supports direct access to atoms as well as access to atom sets. Performing update operations, it is responsible for the automatic maintenance of referential integrity defined by reference attributes (system-enforced integrity). Thus, an update operation on a reference attribute comprises implicit update operations on other atoms to adjust the appropriate back-reference attributes. Effective processing of data system operations critically depends on the availability of powerful navigational capabilities. This includes the notion of a "position" in a set of atoms, that is, a current position has to be maintained under traversal and modification operations. For that purpose, scans are introduced as a concept to control a dynamically defined set of atoms, to hold a current position in such a set, and to successively deliver single atoms (NEXT/PRIOR) for further processing.

All tuning mechanisms specified by an LDL statement - atom clusters as well as access paths, sort orders, and partitions - generate additional storage structures which materialize homogeneous or heterogeneous result sets. For example, a physical cluster serves to materialize molecules, whereas partitions collect the results of projections. The underlying idea is to make storage redundancy available to speed up molecule processing. To manage redundancy in the access system, physical records are introduced as byte strings of variable length. They are stored consecutively in the "containers" offered by the storage system. Storage redundancy may introduce substantial overhead when an atom is modified (and necessarily all its allocated physical records). To limit the amount of immediate overhead, deferred update is used, i.e., during an update operation only one physical record is modified whereas all others are modified later. The advantage of the redundancy becomes obvious when accessing an atom, since any physical record can be used. The one with minimum access cost should be selected. This has to be supported by the storage system.

The storage system is responsible for the management of the database system buffer and for the mapping of its contents to external storage. It provides segments devided into pages of uniform size at its interface to the access sys-

tem. To meet the most important requirement of the access system concerning containers of arbitrary length the storage system offers at its interface page sequences as additional objects or containers. A page sequence treats an arbitrary number of pages as a whole. It is supported by a cluster mechanism of the underlying file manager enabling an optimal transfer of the whole page sequence, e.g. by chained I/O. Moreover, the storage system provides the concepts of "page set" and "page contest" supporting set-oriented processing of pages to optimize I/O behavior. More information about the storage system can be found in [Si88b].

5. Structure of the Application Layer

So far, we have described the static aspects of PRIMA, that is, of mapping MAD constructs to blocks and files by referring to a multi-level hierarchical model. PRIMA is considered a research vehicle for a variety of DBMS applications in possibly distributed engineering environments. Therefore, it is intended to run as a "generic" kernel in different kinds of either centralized or multi-processor environments leading from a "kernel-only" DBMS to a tool for building multi-processor DBMS [HHM86], and a base system for workstation-host coupling [HHMM87]. In the following, we want to concentrate on the latter theme that is the PRIMA-NDBS and its support for engineering information systems in a workstation-host environment.

Interactive manipulation of complex engineering objects requires the use of effective communication protocols between kernel and AL as well as a large share of local DBMS processing within the AL in order to guarantee satisfactory response times. On demand, complex objects have to be efficiently extracted and transferred from the public DB (managed by the kernel on a server) to the workstation. Then, the AL takes care of these objects - usually for a long time; for temporary storage, it may use a private DB on an own disk. To refine the problem, the following questions have to be considered in more detail:

- How does the workstation (and the application program) get its data?
- How does the application program at the workstation manipulate these data?
- How should the changes performed at the workstation be propagated back to the server?
- How should the server database system reflect these changes?

To answer these questions, we introduce the so-called processing model of the AL and some implementation concepts for local buffer management.

5.1 Processing Model of the Application Layer

The overall model describing the DBMS activities in the workstation is called the processing model of the AL. Its prime purpose is to provide a framework for the exploitation of locality. Ideally, it is desirable to make a mechanism available that enables the application to reference an object directly, for instance using the pointer concept of a programming language.

With such a typical referencing behavior in mind, we propose a processing model aimed at high locality of object references. Extraction of data from the public DB is similar to the approach described in [LP83b]. A design transaction issues a checkout request if existing design data is required. Such a request is used to fetch a design object from the public DB. More checkouts may follow when additional data is required by the application. All checked out data is protected by the kernel against concurrent access. The design objects are temporarily stored in the workstation, they are organized in a special main memory structure called object buffer which offers fast operational access and a pointer-like reference mechanism. For recovery purposes and for saving particular design states, copies of the design objects may be preserved in the private DB. A design object is committed to the public DB by a checkin request. Since commit implies giving up the right of unilateral rollback, the separation of checkin and end of design transaction is meaningless. Hence, we argue for the delay of all checkins to the end of the design transaction.

Summarizing the design transaction, we can identify the following characteristics:

- isolation against concurrent design transactions (provided by the synchronization capabilities of the server DBMS);
- design cooperation only via already checked in (committed) data;
- possibly n checkout requests (n>0) in combination with no or only one checkin request;
- in between there is local manipulation accompanied with the accumulation of design data changes.

Fig. 3 depicts the scheme of such a design transaction following the proposed processing model. After the start of the design transaction it is allowed to checkout the design data needed, using possibly several checkout requests. Then local manipulation is performed on the design objects allocated in the object buffer. It can be structured by issuing one of the following requests

- SAVE, saving the current design stage;
- RESTORE, backing out to a previously saved design state;
- SUSPEND, interrupting the manipulation activities (implies a SAVE);
- RESUME, continuing an interrupted design transaction.

Thus, SAVE and RESTORE provide a user-controlled recovery concept for the design process, i.e. saving a consistent design stage or wiping out the latest actions, while SUSPEND and RESUME support design interruption guaranteeing subsequent processing without loss of information.

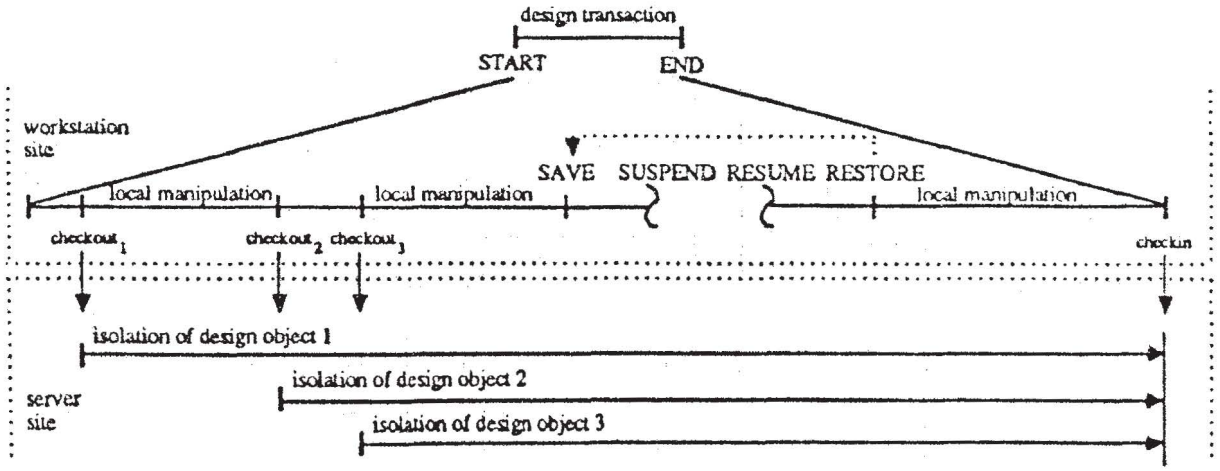


Fig. 3: Sequence of actions in the processing model

In the following, we want to describe an adequate implementation concept for the above introduced general processing model. Obviously we have to be aware of the following optimization criteria:

- minimal number of workstation-server communications
- minimal volume of data transfer
- distribution of the work to do among workstations and server, avoiding duplicated work.

They are supposed to yield high degree of site autonomy and optimized workstation-server cooperation.

5.2 Implementation Issues of the Application Layer

Describing the implementation aspects of our processing model, we first introduce the basic software architecture. The functionalism of the NDBS kernel interface, which is called Object-Supporting Interface (providing the above, introduced structural object orientation), is determined by the MAD model introduced in chapter 3. On top of this interface, we have designed a component, called Object Buffer Manager (OBM). The main task of the OBM is local handling and organization of all object-related information required by the application. Hence, the OBM consists of the preparation component and the object buffer. The preparation component is responsible for fetching and transferring of data from the NDBS kernel to the object buffer and vice versa. The object buffer is a large main memory buffer, that realizes the 'nearby application locality' and supports the representation of the molecules, which are supposed to carry all information that describes one design object. In order to support efficient main memory organization and direct molecule-oriented access, we require special algorithms for memory management and address calculation. A further component of the OBM is the cursor maintenance component which supports the processing by a structure-oriented cursor management. Supporting the processing primitives, introduced in the previous section (SAVE, RESTORE, SUSPEND, RESUME), an additional component becomes necessary to deposit/reload molecules to/from the private DB. Hence, the OBM establishes a powerful data handling interface (called object supporting programming interface) at the workstation site. Together with the application dependent program modules it forms the application layer. Here, behavioral object orientation is achieved by the abstract data types within this top-most conponent. Fig. 4 shows the basic software architecture; in addition, it indicates their allocation to the associated hardware components. Furthermore, it illustrates that the interface between workstation and server lies inside the OBM layer, that is, our design provides an agent of the OBM at the server site.

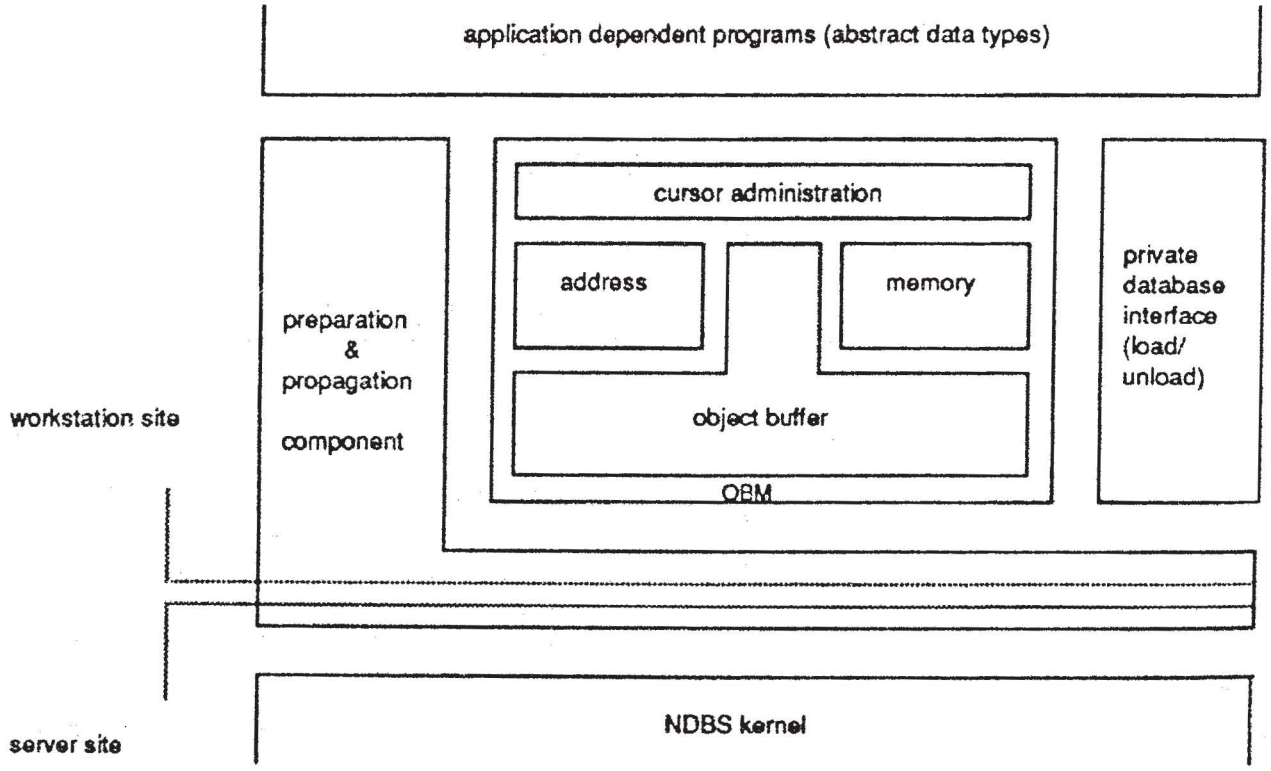


Fig. 4 : Interface architecture supporting adequate molecule processing

After describing the architectural aspects, we now want to characterize the information necessary for workstation-server cooperation in our processing model. First, we have the query, defined and later activated with the actual query parameters by a program module in the application layer. The power for query definition is given by the molecule

query language (MQL). Second, we have the answer information including the query result data aggregated by the NDBS kernel. This information is structured as a set of molecules. Third, there is the modification information which comprises all insertions, updates, and deletions made by the application layer. It is encoded as an atom list enhanced by modification flags and specific information about the modification environment. It seems to be clear that the molecule set of the answer information is associated with the checkout operation, and the atom list of the modification information corresponds to the checkin operation. Hence, we have a high level of abstraction to formulate the query and to represent the result, and we have a low level to propagate the modifications minimizing the amount of data to be transfered and finally checked in.

The Object Buffer Data Structure

In the following, we concentrate our discussion on the answer and modification information, because these carry the more interesting issues. Especially, the organization and the internal data structures of the object buffer will be introduced. The answer information consists of a molecule set. Each molecule is composed of a structured set of atoms. Each of them is represented by a list of attributes and is identified by a special attribute, called atom identifier. Molecule identification is done by its root atom. Fig. 5 shows the essential aspects of the data structures to represent answer information in the object buffer.

Since the data loaded into the object buffer are usually kept on the workstation site for a long span of time, it will therefore become necessary to move the objects from the main memory to the private DB existing on secondary storage (local for this workstation). Thus, we organize the object buffer using so-called main memory areas. The required relocation can be easily managed, if all logical references among molecules or atoms, included in the object buffer, are substituted by some kind of area relative addresses. The relocation can then be achieved without the need of recalculation by simply moving whole areas.

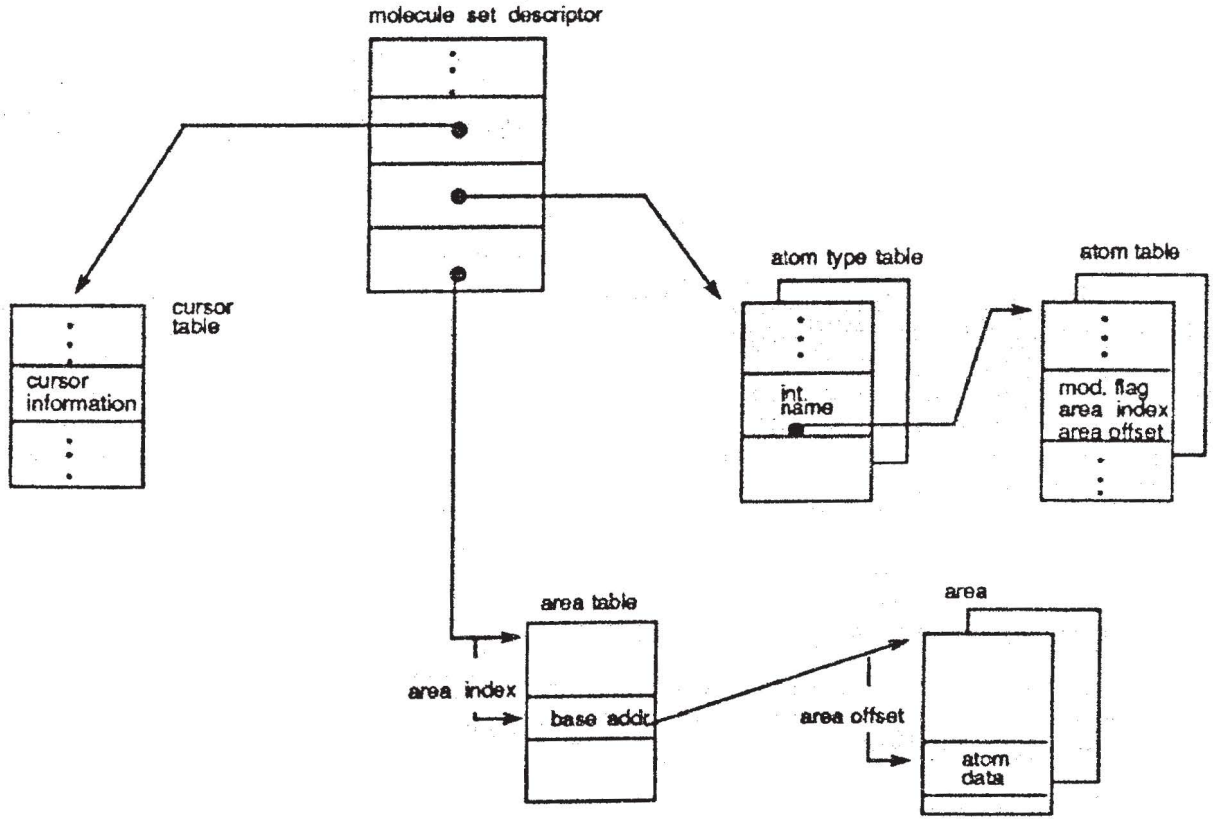


Fig. 5 : Representation of a molecule set within the object buffer

Fig. 5 illustrates the representation of a molecule set within the object buffer. Based on the molecule set descriptor, the overall structure can be divided into three parts:

- The cursor table contains special access information, which are linked with molecules within the considered molecule set.
- The area table and the set of areas obtain structures to support the control of memory which is allocated for atoms. The (main memory) area concept supplies not only relocatability, but also prevents the scattering of main memory by numerous small atoms.
- Furthermore, the atom type table and atom table allow us to calculate atom addresses by using logical atom identifiers. The atom type table separates the atoms by their corresponding atom types and enables efficient type-oriented access. Each entry bears an internal type key and the address of the atom table. This table is organized as a dynamic hash table, since the number of stored atoms can shrink or expand significantly. The atom table entries contain further administrative information. The modification flag indicates the type of modification (insert, delete, update) and is later used for propagating back to the central database. Additionally, there are also the fields area index and area offset. The area index determines an entry of the area table, which finally leads to the area containing the required atom. The area base address incremented by the area offset delivers the atoms main memory address.

The relationships constituting the molecule structure are automatically represented through special reference attributes in the atoms that contain identifiers of other atoms.

The above sketched data structure represents a single result set of just one query. From a logical point of view, it is a snapshot of a database partition. So it seems consequent that an atom is not represented in a redundant manner, if it belongs to more than one molecule within the same result set. On the other hand, an atom is redundantly represented in different result sets. Such multiple occurrences are known to the programs using the object buffer and have to be controlled and managed by them.

The modification information must be collected when downward propagation is initiated by an application program. All changed atoms can be accounted by searching all the atom tables formodification flags set. The collected information is organized as some kind of "delta" information, that is, it contains only the modified attributes and the accumulated changes.

Enbedding the Object Buffer into Applecation Programs

The question we would like to discuss now is how data in the object buffer is manipulated by the ADT's of the application layer. The atom is the smallest unit of data affected by any modification. We need a cursor concept to identify a single atom within the atom set defined by a molecule and within the molecule set given by query's result set. Such a flat cursor points to only one atom at a time. In principle, it is sufficient for reaching all atoms in the result set, because one can navigate via the reference attributes in the atom data. Nevertheless, analysing characteristics of some engineering application prototype systems [HHLM87] has shown us that it is useful in many situations to have a more complex cursor, for example a hierarchical one. Often, there are some hierarchical subunits of processing within a molecule. In our implementation, such a hierarchical cursor is defined by a list of atom type names, which marks the paths for the cursor hierarchy, and by identification of the root atom. The concept of hierarchical cursor may be implemented by a hierarchy of dependent flat cursors. Navigation via one cursor automatically affects the subordinate cursors. The idea to support more descriptive (as opposed to procedural) cursor operations is worth more detailed consideration, but it lies beyond the scope of this discussion.

The next question is how all those queries, result sets, molecules, cursors, and atoms are reflected in the programming language which is used to write application-dependent program modules. In principle, there are four different approaches for language binding [LP83a].

For several reasons we decided for the third approach designing a host-language embedding using a precompiler. The use of precompiler statements is sketched in Fig. 6. It depicts the scheme of an application program in a PASCAL-like programming language. We distinguish between the definition of query types and cursor types, as well as the declaration of corresponding instances. The AL programmers can therefore generate multiple instances of the same type. These are handled like host language variables. The query type declaration is the place for MAD statement specifica-

```
PROGRAM example;
definition of 'normal' types and variables

QUERY TYPEabc_query_type =
'Select A,B,C
FromA-B-C
WhereA.attr1< $max'
(max : integer);

...

CURSOR TYPEabc_cursor_type =
B,C->abc_query_type
...
mcp_cursor:abc_cursor_type;

mcp_buffer:abc_query_type;
...

BEGIN

...
EVAL ( abc_buffer, ... max_value ...);
...
ATTACH (abc_cursor, abc_buffer);
...
DISPLAY (abc_cursor @ B@attr2);
...
END.
```

Fig. 6: Scheme of an application program module

tion. Therefore, all functions of molecule preparation are included in the program by query types. The query variable corresponds to the object buffer introduced above. Note that the instance of a query represents the data which is the parameter of MAD statement's activation. The linkage between variable and data is done via the built-in function eval. Eval directs actual parameters to the kernel interface and triggers the evaluation and preparation activities. Similar to the definition of query instances, the programmers can generate cursor variables supporting the processing of object buffers (query variables). The binding of cursor variable and object buffer is provided by the attach function. The bounded cursor can be moved in molecules around atoms, which are currently in the object buffer, and is also used to specify the required molecule-oriented access (cf. Fig. 6).

6. Conclusions

Database management systems for non-standard applications have emerged as one of the most important directions of nowadays database research. Some of the major challenges in this area are constituted by adequate modelling and effective processing of complex application objects.

As one step towards complex-object management we have designed the MAD model. MAD offers dynamic object definition and object handling based on direct and symmetric management of network structures and recursion. The generic mechanisms can be used to map a wide variety of semantic and object-oriented modelling constructs in a straight forward manner including complex objects with shared subobjects.

Additionally the prototype implementation of MAD (called PRIMA) which is based on the so-called NDBS kernel architecture has been described. The kernel provides the MAD functionalism and contains a variety of tuning mechanisms and performance enhancements transparent at the data model interface. PRIMA is assumed to be used in different

hardware environments including workstation-based multi-processor systems. Located on the workstation, there is an additional system layer, the so-called application layer (AL), which completes the NDBS offering application-oriented abstract data types, i.e. application-oriented data structures and operations. Thus, the NDBS is capable of an orientation towards a specific application.

Furthermore, effective object processing within the AL is provided by the concept of 'nearby application locality'. This is realized in buffering the complex objects, selected by the kernel system, in a main memory resident object buffer on workstation-site.

References

As76 Astrahan, M.M., et. al.: SYSTEM R: A Relational Approach to Database Management, in: ACM TODS, Vol. 1, No. 2, 1976, pp. 97-137.

BB84 Batory, D.S., Buchmann, A.P.: Molecular Objects, Abstract Data Types and Data Models; A Framework, in: Proc. 10th VLDB Conf., Singapore, 1984, pp. 172-184.

Da86 Dadam, P., et. al.: A DBMS Prototype to Support Extended NF2-Relations: An Integrated View on Flat Tables and hierarchies, in: Proc. ACM SIGMOD Conf., Washington, D.C., 1986, pp. 356-367.

Di86 Dittrich, K.R.: Object-Oriented Database Systems - A Workshop Report, in: Proc. 5th ER Conf., 1986, North Holland Publ. Comp., 1986.

Hä88 Härder, T. (ed.): The PRIMA Project, Design and Implementation of a Non-Standard Database System, Report No. 26/88, SFB 124, University Kaiserslautern.

HHLM87 Härder, T., Hübel, C., Langenfeld, S., Mitschang, B.: KUNICAD - A Database System Supported Geometrical Modeling Tool for CAD Applications (in German), in: Informatik Forschung und Entwicklung, Vol. 2, No. 1, 1987, pp. 1-18.

HHM86 Härder, T., Hübel, C., Mitschang, B.: Use of Inherent Parallelism in Database Operations, in: Proc. Conf. on Algorithms and Hardware for Parallel Processing CONPAR 86, Springer Lecture Notes in Computer Sciences, Aachen 1986, pp. 385-392.

HHMM87 Härder, T., Hübel, C., Meyer-Wegener, K., Mitschang, B.: Coupling Engineering Workstations to a Database Server, in: Proc. Conf. on Data and Knowledge Systems for Manufacturing and Engineering, Hartford, Connecticut, 1987, pp. 36-44.

HR85 Härder, T., Reuter, A.: Architecture of Database Systems for Non-Standard Applications (in German), in: Proc. of the GI Conf. on Database Systems for Office, Engineering and Science Environments, 1985, pp. 253-286.

LP83a Lacroix, M., Pirotte, A.: Comparison of Database Interfaces for Application Programming, In: Inf. Systems, Vol. 8, No. 3, 1983, pp. 217-229.

LP83b Lorie, R., Plouffe, W.: Complex Objects and Their Use in Design Transactions, in: Proc. of the Data Base Week: Engineering Design Applications, 1983, pp. 115-121.

Mi85 Mitschang, B.: Characteristics of the complex-object notion and realization approaches (in German), in: Proc. of the GI Conf. on Database Systems for Office, Engineering and Science Environments, 1985, pp. 382-400.

Mi88a Mitschang, B.: Towards a unified view to design data and knowledge representation, in: Proc. 2nd Int. Conf. on Expert Database Systems, Tysons Corner VA, 1988, pp. 33-49, (further publication by Benjamin/Cummings Publ. Co.).

Mi88b Mitschang, B.: A molecule-atom data model for non-standard applications - requirements engineering, data-model design, and implementation concepts (in german), PhD Thesis, University Kaiserslautern, 1988.

ML8a3 Meier, A., Lorie R.: A Surrogate Concept for Engineering Databases, in: Proc. 9th VLDB Conf., Florence, 1983, pp. 30-32.

PSSWD87 Paul, H.-B., Schek, H.-J., Scholl, M.H., Weikum, G., Deppisch, U.: Architecture and Implementation of the Darmstadt Database Kernel System, in: ACM SIGMOD Conf., San Francisco, 1987, pp. 196-207.

Sch088 Schöning, H.: The PRIMA Data System: Query Processing of Molecules, in [Hä88].

Si88a Sikeler, A.: Key Concepts of the PRIMA Access System, in [Hä88].

Si88b Sikeler, A.: Buffer Management in a Non-Standard Database System, in [Hä88].

X3H286 SQL Addendum-2, Doc. ISO/TC97/SC21/WG3/ N143, ANSI X3 H2-86-61, 1986.