SFB-Bericht 26/88, SFB 124, University of Kaiserslautern, 1988

Key Concepts of the PRIMA Access System

Andrea Sikeler
University Kaiserslautern

Abstract

PRIMA, as the kernel of a non-standard database management system (NDBMS), is divided into multiple layers representing different levels of abstraction. The access system is an internal layer implementing an atom-oriented interface which allows for direct and navigational retrieval as well as manipulation of single atoms. Different kinds of scan operations support sequential access to either homogeneous or heterogeneous atom sets. Some scan operations, however, depend on certain storage structures (namely access path structures, sort orders, and atom clusters) which introduce redundancy to be maintained by the access system.

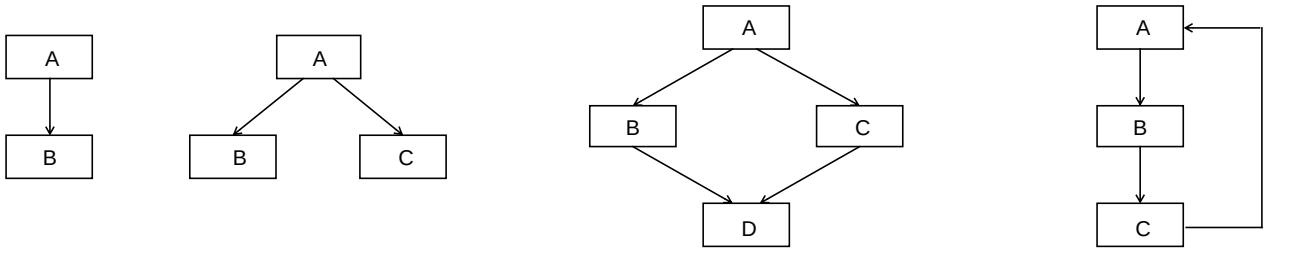
This paper describes the current state of the access system and its embedding into the overall PRIMA system (especially transaction concept). It gives an overview of our design decisions and of some implementation aspects.

1. Introduction

The design of PRIMA (prototype implementation of the molecule-atom data model, [Hä87]) is strongly influenced by the investigation of four different application areas for which we have implemented and evaluated sizable prototype systems: VLSI circuit design, construction of solids in 3D modeling [HHLM87], and map handling in geographic information systems as well as diagnostic expert systems [HMP87]. The observations made during these investigations motivated some important design guidelines:

- The data model supported by PRIMA has to be object-oriented in that it allows for the manipulation of sets of complex objects [BB84, Mi85]. Complex objects have to be constructed dynamically out of basic objects and relationships among these. All kinds of relationship types, 1:1, 1:n as well as n:m, should be represented in a direct way thus allowing for symmetric traversal and use of complex objects.
- PRIMA as the implementation of the data model should support object processing by a variety of storage structures, use of tuning mechanisms, and performance enhancements transparent at the data model interface. Therefore, a clean break-up of PRIMA into different layers with appropriate tasks is mandatory.

In the molecule-atom data model (MAD model, [Mi88]), the objects the user has to deal with are called *molecules* corresponding to the above mentioned complex objects. Each molecule consists of more primitive molecules or atoms and belongs to a certain molecule type. This type is defined in terms of its component types and the relationships among these types (see Fig. 1 for the three generic types of molecule structures). Each *atom* is composed of attributes of various types, has an identifier, and belongs to a corresponding atom type. The atom type is put together by the constituent attribute types to be cho-



a) hierarchical molecule types b) network-like molecule type c) recursive molecule type

Figure 1: Three generic types of molecule structures

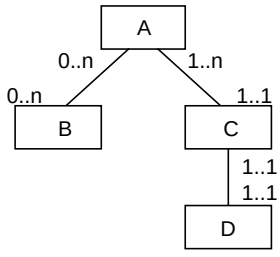
sen from a richer selection than in conventional data models. With respect to molecule management the most important of these types are the IDENTIFIER type and the REFERENCE type. The IDENTIFIER type serves as a surrogate [ML83] which allows for the identification of each atom. Based on this IDENTIFIER type it is easy to define the REFERENCE type as a typed reference to another atom of the same or of different type. However, each reference requires a corresponding back-reference in order to support symmetric processing. Hence, the REFERENCE type together with the repeating-group type SET can be used to express all kinds of relationship types between two atom types. The whole set of attribute types offered by MAD is summarized in Fig. 2. Additionally, MAD supports user-defined keys, in that unambiguity is guaranteed for the corresponding key values.

The MAD query language (or molecule query language, MQL) distinguishes three sublanguages in order to define and modify a database schema, to manipulate the database contents, and to establish tuning mechanisms:

- The *data definition language* (DDL) comprises of means to create or drop an atom type as well as to expand or shrink an atom type, i.e. to add or remove some attributes, and to rename an atom type. Additionally, molecule types may be defined and released. Fig. 3 shows a simple example for a MAD schema and corresponding molecule type definitions.
- The query capabilities of the *data manipulation language* (DML) allow for vertical access, i.e. retrieving selected molecules as a whole or parts of them, as well as horizontal access, i.e. retrieving all mole-

attribute types		description
basic types :	BOOLEAN	
	CHAR (n)	fixed length character string
	BYTE (n)	fixed length byte string
	BIT (n)	fixed length bit string
	INTEGER	2 byte integer
	LONGINT	4 byte integer
	REAL	
	TIME (n)	timestamp
	HULL_DIM (n)	rectangular box of dimension n
special types :	IDENTIFIER	surrogate
	REF_TO	single reference
	CHAR_VAR	variable length character string
	BYTE_VAR	variable length byte string
	CODE	
composed types :	ARRAY	of basic type or composed type
	RECORD	of basic type or composed type
repeating-group types :	SET_OF	of basic type or reference type or composed type
	LIST_OF	of basic type or reference type or composed type

Figure 2: Attribute types offered by MAD



a) MAD schema

```
CREATE ATOM_TYPE A
      ( a_id        :  IDENTIFIER,
        a1          :  INTEGER,
        a_to_b      :  SET_OF (REF_TO (B.b_to_a)) (0, VAR),
        a_to_c:     :  SET_OF (REF_TO (C.c_to_a)) (1, VAR)      )

CREATE ATOM_TYPE B
      ( b_id        :  IDENTIFIER,
        b1          :  CHAR (3),
        b_to_a      :  SET_OF (REF_TO (A.a_to_b)) (0, VAR  )  )

CREATE ATOM_TYPE C
      ( c_id        :  IDENTIFIER,
        c1          :  BYTE (7),
        c_to_a      :  REF_TO (A. a_to_c),
        c_to_d      :  REF_TO (D. d_to_c)                       )

CREATE ATOM_TYPE D
      ( d_id        :  IDENTIFIER,
        d1          :  CHAR_VAR,
        d_to_c      :  REF_TO (C. c_to_d)                       )
```

b) atom type definitions

```
DEFINE    MOLECULE_TYPE    ABC
FROM      A - (B - C)

DEFINE    MOLECULE_TYPE    ABCD
FROM      A - (B, C - D)
```

c) molecule type definitions

Figure 3: Example for the MAD-DDL

cules, of a certain type, perhaps satisfying certain selection criteria. Both kinds of queries are illustrated in Fig. 4 a and b, respectively. The FROM-clause establishes the structure of the desired molecules, in that all essential atom types and the corresponding REFERENCE attributes are named (briefly "-" when unambiguous). The (optional) WHERE-clause restricts the appropriate result set by evaluating the specified qualification criteria (including quantifiers). The SELECT-clause allows for a projection either unqualified or qualified. The unqualified projection selects atoms and attributes of the result molecules on type level, whereas the qualified projection selects atoms and attributes of all result molecules satisfying an additional condition. Additional features, such as sorting, aggregation, etc. are also supported. In a similar way to these query capabilities the INSERT, MODIFY, UPDATE, and DELETE operations of the DML deal with integral molecules as well as with its components (see Fig. 4 c and d). Again, the FROM-clause and WHERE-clause are used to determine the desired molecules, whereas the projection selects the corresponding components (attributes, atoms) to be manipulated.

- The third sublanguage is the *load definition language* (LDL) used by the database administrator to enhance performance. This language comprises of means to define or release various types of storage structures:
  - several access methods for one or more attributes permitting multidimensional access
  - (vertical) partitioning of atoms to improve clustering of frequently accessed attributes
  - sort orders to speed up sequential processing according to a given sort criterion
  - "physical clusters", called static molecules, to provide physical contiguity for atoms belonging to frequently requested molecules.

  However, the MAD model itself, i.e. the DML, makes no reference to these storage structures.

Based on this short description of the MAD model and its transparent support by additional tuning mechanisms, the following chapter gives an overview of the concepts and ideas used in PRIMA for its imple-

```
SELECT   A, B, C, D :=    SELECT    D
                          FROM      RESULT
                          WHERE     D.d1 = 'example'
FROM     A - (B, C - D)
WHERE    A.a1 = 4711
```

a) vertical access combined with a qualified projection

```
SELECT   ALL
FROM     C
```

b) horizontal access (the most simple case)

```
INSERT   A.a1        :=  1024
         A.a_to_b    :=  VALUE  ( SELECT   B.b_id
                                  FROM     B
                                  WHERE    B.b1 = 'ex '   )
FROM     A
```

c) insertion

```
UPDATE       D.d1   :=  'modified'
FROM         D
WHERE        D.d1 = 'example'
```

d) modification

Figure 4: Examples for the MAD-DML

mentation only as far as they are necessary for the comprehension of a detailed description of the access system. Subsequently, a basic version of the access system is described (chapter 3). This version offers all concepts needed for the support of the MAD query language. Optimizing of query evaluation, however, is not well enough supported. Therefore, additional storage structures (access path structures, sort orders, atom-cluster types, and partitions) are introduced in order to support efficient query processing (chapter 4). According to these additional storage structures, the access system has to cope with redundancy (chapter 5). Dynamic aspects, i.e. the transaction concept, will be shortly discussed in chapter 6, whereas chapter 7 addresses the current state of the implementation. The paper concludes with a short summary and an outlook as to current and future work.

2. The PRIMA Architecture

PRIMA is divided into multiple layers representing different levels of abstraction (Fig. 5). Each of these layers makes available at its interface a number of objects and operations fulfilling certain requirements of the non-standard applications needed at the chosen level of abstraction:

- The DBS-kernel, i.e. the data system as the highest layer, makes available at its interface the molecule-atom data model which we have described in the introduction.
- The interface of the access system is atom-oriented. Its expressiveness is similar to the interface of the Research Storage System (RSS, [As81]) of SYSTEM R [As76].
- The storage system implements a set of "infinite" linear address spaces which allow for virtual addressing by the higher layers. Hence, the storage system of PRIMA represents a lower level of ab-

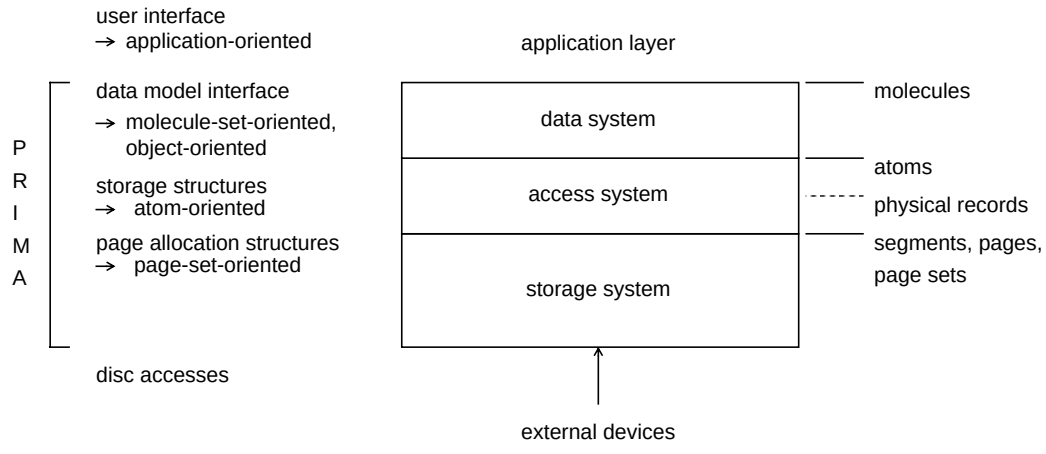


Figure 5: Architecture of a non-standard database system

straction than RSS [As81], the Wisconsin Storage System [Ch85] or the Storage System of the DASDBS [DPS86].

In the following sections, the data system and the storage system are described in more detail, whereas the access system is treated throughout the rest of the paper.

2.1 The Data System

The main task of the data system [Sch88] is to transform the molecule-set oriented MQL interface into lower level programs as well as their subsequent execution. This is done by first transforming the user-submitted MQL statements into valid, semantically equivalent, but not necessarily optimal query evaluation plans (QEP). These are stored within so-called access modules (compilation phase). In an optimization phase, these QEPs are rearranged according to different heuristics in order to speed up their processing. Subsequently, these QEPs are evaluated yielding the desired result (evaluation phase). Thus, compilation and optimization are separated from execution, allowing for repeated execution without repeated compilation. This is an important aspect with respect to the execution of DML statements which are expected to be used more than once, since they are normally embedded in application programs.

A QEP consists of an operator tree describing the execution sequence within the data system. In the case of DDL or LDL statements, the operator tree is simple: Description information is generated and stored using the meta-data system (MDS [We87]), and the access system is invoked to establish the corresponding storage structure. Considering DML statements, however, the operator tree may become rather complex.

The compilation phase first checks the syntactical correctness of the retrieval statement. Following this, a couple of query transformation steps (e.g. macro expansion, molecule-structure simplification, etc.) is performed in order to achieve standardization and simplification of the queries to be dealt with. The nodes of the resulting QEP may be divided into two classes. The leaf nodes are used to construct the molecules, whereas the inner nodes operate on the derived molecules (projection, recursion, aggregation, etc.). Thus, only leaf nodes invoke the access system in order to retrieve the desired atoms. Leaf nodes are always of the type "construction of simple molecules", i.e. they deliver simple molecules of a non-recursive, hierarchical structure, according to a query of the following form:

SELECT <unqualified projection> <optionally: order specification>
FROM <one non-recursive, hierarchical molecule type>
WHERE <molecule qualification>

In the optimization phase, the resulting QEP is first rearranged guided by a set of rules and heuristics leading to an improved QEP. Subsequently, evaluation strategies are chosen for the different nodes, e.g. nested loop or merging scan join. For the "construction of simple molecules", for example, the most obvious strategy (called "top down") is to call the root atom first (since exactly one root atom is present in each molecule). Starting with this root atom all children, grandchildren and so on may be called, terminating after all leaves of the molecule structure are reached. Calling children means performing the hierarchical join which is supported by attributes of the REFERENCE type. However, sometimes a "bottom up" strategy might be more efficient, i.e. starting with a child atom, executing the hierarchical join in the direction of the root atom, and constructing the whole molecule as in the "top down" approach. However, the strategy, which should be chosen, strongly depends on the existing storage structures and on the operations allowed for these structures. Thus, the optimization phase also determines the storage structures to be used for the "construction of simple molecules".

During the execution phase, the QEP operator tree is interpreted node by node, in that an agent (e.g. a process or a procedure) is provided for each operator type executing the operator's computation.

2.2 The Storage System

The storage system [Si88] as the lowest layer of PRIMA pursues two major tasks: It manages the database buffer and organizes the external storage devices, thus being responsible for the data exchange between main storage and disk storage. By managing the database buffer, it provides a number of "infinite" linear address spaces with visible page boundaries. Therefore, the database is divided into various segments consisting of a set of logically ordered pages. All pages of a segment are of equal size, which can be chosen to be 1/2, 1, 2, 4, or 8 kbytes and which is kept fixed during the lifetime of a segment. Segments are dynamic in that the segment-internal free place administration, which is integrated into the storage system, automatically allocates new pages when space is required which cannot be satisfied by existing pages. Similarly, the FPA automatically deletes empty pages when they reside at the end of the segment. Otherwise, empty pages are reused as soon as possible when space is required.

Hence, two different page types are distinguished within a segment. FPA pages, used only for free place administration, are managed by the storage system. Therefore, they are not visible outside the storage system. Data pages, however, are used by the access system to map the different storage structures. They are requested by the FIX-PAGE operator indicating whether the page is needed for read or write purposes and they are released by the UNFIX-PAGE operator indicating whether or not the page has actually been modified.

The five page sizes, however, are not sufficient with respect to the mapping process performed by the access system. The restriction to a certain page size, even 8 kbytes, is too stringent, especially regarding arbitrary length objects which may grow up to some Mbytes. Therefore, *page sequences* are introduced as predefined page sets supported by physical clustering. A page sequence is a set of logical consecutive pages of a segment which contain (from the viewpoint of the access system) one single object spanning these pages [DPS86]. A page sequence consists of a so-called header page and one or more component pages. Page sequences are dynamic since component pages may be added and removed arbitrarily depending on the current length of an object. Requesting a page sequence (FIX-PAGE-SEQUENCE operator) causes the storage system to fix all pages in the database buffer belonging to that

page sequence. These pages, however, are distributed over the buffer, i.e. the page sequence does not build one contiguous linear address space within the buffer, and the access system must be aware of the page boundaries. Correspondingly, the UNFIX-PAGE-SEQUENCE operator releases the whole page sequence, indicating for each page whether or not it has actually been modified.

Additionally, the storage system offers three operators to handle not only such predefined page sets but also arbitrary page sets. The (UN)FIX-PAGE-SET operator is just a shorthand for a number of subsequent (UN)FIX-PAGE/(UN)FIX-PAGE-SEQUENCE operators. In other words, all specified pages and page sequences are fixed (unfixed) in the buffer. On the other hand, the FIX-ONE-PAGE operator supports access to replicated data fixing only one of the specified pages or page sequences, selecting that page or page sequence with minimum cost, i.e. the storage system performs a "page contest" along certain selection criteria (e.g. "already in buffer", page size, lock conflicts).

Fig. 6 summarizes the operations offered by the storage system.

3. The Basic Version of the Access System

Although the access system supports quite different storage structures, most of these structures are intended to increase performance. Therefore, a basic version of the access system, comprising means to support data definition, update, and "simple" retrieval, is sufficient in order to guarantee the functional behaviour of MQL. However, before we go into particulars concerning these operations, we first describe the mapping of atoms onto the different "containers" offered by the storage system.

3.1 The Mapping of Atom Types and Atoms

In the basic version of the access system, each atom type is assigned to a single segment which includes all atoms of the corresponding type. This decision essentially simplifies some of the operations, especially deleting an atom type, although it is also possible to store atoms of different types in the same segment (or even in the same page) due to the addressing concept described in section 5.1.

Atoms are initially mapped onto so-called physical records. A physical record is a byte string of variable length, which represents, in the case of the basic version of the access system, exactly one atom (in contrast to other storage structures, where physical records represent either a part of an atom or a set of atoms, see chapter 4). The composition of a physical record considers two important aspects:

1. When inserting an atom, values may be assigned either to all or only to a few selected attributes.
2. Shrinking an atom type happens in a deferred manner, i.e. there may exist physical records which still contain deleted attributes.

Therefore, a physical record is composed of an administration part and a data part (Fig. 7):

- The first entry of each physical record corresponds to the value of the IDENTIFIER attribute of the appropriate atom. Since this value is unique, it is used to locate a physical record within a segment, i.e. within a page.
- The second entry delivers the total length of the physical record (administration part as well as data part). This entry is restricted to 4 bytes, so that the maximum length of a physical record is 2^{32}-1 bytes.

operation	description
INIT_SEGMENT	creates a new segment, generates an appropriate number of empty pages and initializes the segment-internal FPA.
DELETE_SEGMENT	deletes a segment
GET_FREE_PLACE	determines a set of pages which together satisfy the requested free place, perhaps allocates new pages, and delivers the corresponding page numbers.
MODIFY_FPA INCREMENT_FPA DECREMENT_FPA	modify the FPA entry of a page by either overwriting or altering the existing value.
FIX_PAGE	locates a page in the database buffer
UNFIX_PAGE	releases a page
FIX_PAGE_SEQUENCE	locates a page sequence in the database buffer
UNFIX_PAGE_SEQUENCE	releases a page sequence
FIX_PAGE_SET	locates a number of pages and/or page sequences
UNFIX_PAGE_SET	releases a number of pages and/or page sequences
FIX_ONE_PAGE	locates a page or a page sequence out of a number of pages and/or page sequences

Figure 6: Operations offered by the storage system

- The third entry encodes the state of the physical record with respect to a not yet finished shrink operation.
- The fourth entry consists of a variable length bitmap and a leading length field (4 byte). This bitmap indicates for each attribute whether or not a corresponding value exists. Hence, the absence of an attribute value is represented indirectly by a NULL bit instead of an explicit NULL value. As a consequence, expanding an atom type requires expanding the bitmap of each physical record belonging to the corresponding atom type. However, this is also done in a deferred manner, i.e. the bitmap is modified along with the next update operation on the atom. Thus, read access to a bit, i.e. attribute, behind the bitmap is interpreted as NULL bit.

These four entries form the administration part, whereas the fifth entry contains the primary data, i.e. the attribute values. Attribute values are stored without an identification code of the appropriate attribute, although such a code would allow for more flexibility in arranging attribute values. Such a code, however, would also require additional storage thus leading to fairly large overhead especially in the case of small attribute values. Therefore, the arrangement of attribute values within a physical record of a certain atom type is determined by description information available from the MDS.

The way attribute values are stored depends on their attribute type. For this purpose, the attribute types supported by MAD (Fig. 2) are divided into three groups:

- Group 1 comprises of all attribute types of fixed length, i.e. all basic types, composed types (all components have to be of fixed length) as well as the special types IDENTIFIER and REF_TO. Appropriate attribute values are stored without additional information as a byte string of fixed length depending on the attribute type. The corresponding length information is derived from the MDS.
- Group 2 are the special types except IDENTIFIER and REF_TO. They are stored with a leading length field of 4 bytes indicating the current length (in byte). Hence, attribute values are restricted to a maximum length of 2^{32}-1 bytes.

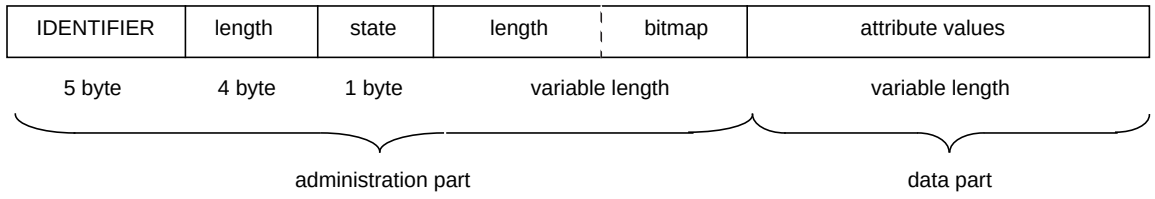


Figure 7: Structure of a physical record

- Group 3 includes the repeating-group types. They are also stored with a leading length field of 4 bytes indicating now the current number of elements. The length of a single element, however, is derived from the MDS, since the element type has to correspond to an attribute type of group 1.

Physical records are, in turn, mapped onto pages and page sequences according to their current length. This means, if a record goes beyond the page size of the segment assigned to the corresponding atom type, this record is mapped onto a page sequence. Otherwise, the record is mapped onto a page. Within a page sequence only one record is stored, whereas within a page multiple records may be stored subsequently without intermediate space. As a consequence, modifying a record may cause movement of other records within the page. However, only the record to be modified may move to another page or page sequence.

In order to locate a physical record, a physical address indicating the appropriate segment and page or page sequence is assigned to each physical record. However, this physical address cannot be used as IDENTIFIER value, since with respect to the additional storage structures multiple physical records may be assigned to a single atom. Therefore, we use a special address structure maintaining all physical addresses assigned to an IDENTIFIER value. The organization of this address structure will be described in section 5.1 in more detail. However, for the following description of the operations offered by the basic version of the access system it is sufficient to know that such an address structure exists.

3.2 The Interface

As already mentioned, the basic version of the access system comprises of means to support data definition, update, and "simple" retrieval. The data definition operations directly correspond to appropriate atom-oriented MQL-DDL statements, whereas the update and retrieval operations, in contrast to MQL-DML statements, are restricted to a single atom.

Data Definition

The creation of a new atom type requires little effort within the access system. A new segment as well as a corresponding address structure are initialized. User-defined keys, however, raise some problems. The overhead to check for duplicates may become enormous without appropriate access path structures, such as B-trees, allowing for fast value-dependent access. Therefore, an access path structure should be initialized for each user-defined key. This, however, is only possible when the current configuration of the access system includes appropriate components implementing the desired structures (see chapter 4). Information about these access path structures are stored in the MDS.

In order to delete an atom type, the appropriate segment and address structure as well as all dependent storage structures are deleted.

Both expanding and shrinking an atom type are processed in a deferred manner. While expanding an atom type it is sufficient to defer the adjustment of the bitmap till the next update operation on an atom.

While shrinking an atom type, however, it appears useful, due to storage utilization, to initialize a process performing the required updates subsequently on each atom instead of waiting for the next update operation on an atom. To indicate a running shrink operation the state information maintained by the MDS is altered. As a consequence, it has to be checked for each access to a physical record whether the state information stored within the physical record corresponds to this state information. If not, the record still contains the deleted attributes. However, only one shrink operation may run in parallel since the state information only encodes two states.

Update

Update operations are restricted to single atoms identified by their logical address. This logical address corresponds, as already mentioned, to the IDENTIFIER attribute of an atom. The appropriate value is assigned by the access system when the atom is inserted and it is released when the atom is deleted. Modification of a logical address, i.e. of the IDENTIFIER attribute, is not allowed.

When inserting an atom (INSERT_ATOM operation) values may be assigned to all or only selected attributes. The access system has to check whether or not values are assigned to all attributes belonging to a user-defined key and whether or not the corresponding values are unique. Hence, either the above mentioned access path structures are utilized, if such structures or the appropriate access path components exist, or all atoms of the atom type are searched sequentially. Additionally, for each REFERENCE attribute it has to be checked whether or not the referenced atoms exist. For this purpose, the address structure of the referenced atom types is used. Afterwards, the logical address is determined and the corresponding physical record is constructed and stored either in a page or a page sequence.

Accordingly, it is allowed to modify single attribute values within an atom, either as a whole or selected parts, depending on the corresponding attribute type (MODIFY_ATOM operation). The latter holds for structured types (i.e. ARRAY and RECORD) as well as for the repeating-group types (i.e. SET and LIST). Concerning the structured types, the appropriate part is specified by an offset and a length, whereas in the case of the repeating-group types special operations exist which enable the modification of single elements (Fig. 8). If these elements are of type ARRAY or RECORD, offset and length may be specified additionally. Again the access system has to check for user defined keys and REFERENCE attributes before the corresponding physical record is modified. If the size of a physical record changes due to the modification, other records within the same page have to be rearranged. Moreover, the record itself has to be stored in another page or even in a page sequence if it no longer fits in its original page.

Deleting an atom is quite simple (DELETE_ATOM operation). The corresponding physical record is erased and the other records within the same page are rearranged. The logical address assigned to a deleted atom may be reused when a new atom is inserted.

Performing any update operation, the access system is responsible for the automatic maintenance of the referential integrity defined by the REFERENCE attributes (system-enforced integrity). Therefore, each update operation on a REFERENCE attribute includes implicit update operations on other atoms to adjust the appropriate back REFERENCE attributes.

Moreover, each update operation requires maintenance of all additional storage structures defined for the corresponding atom type. The actions to be performed, however, will be described in chapter 4, separately for each storage structure.

Retrieval

Retrieval operations of the basic version of the access system include direct access to a single atom identified by its logical address, as well as sequential access to all atoms of a certain type, either in sys-

tem-defined order or sorted following a certain sort criterion. In accordance to the update operations, it is possible to read an atom as a whole or to select single attribute values. Again offset and length for structured types and special operations (Fig. 8) for repeating-group types may be specified in order to project parts of an attribute.

When directly accessing an atom (READ_ATOM operation), the address structure of the appropriate atom type is utilized to determine all physical addresses assigned to the corresponding logical address. These physical addresses are passed to the storage system by a FIX_ONE_PAGE operation. Thus, the storage system delivers a page or a page sequence which contains one of the physical records assigned to the requested atom. This physical record has to be interpreted according to the underlying storage structure (see chapter 4) in order to return the atom in a form described by the so-called projection list. The projection list determines the (parts of the) attributes to be retrieved as well as the ordering of the attributes within the result atom.

Sequential access requires an OPEN_SCAN operation which defines the characteristics of the result set of the appropriate scan, subsequent READ_NEXT operations in order to retrieve the atoms of the result set step by step, and a CLOSE_SCAN operation which explicitly ends the scan. The characteristics of the result set are determined by different parameters depending on the corresponding scan type, i.e. atom type scan or sort scan. Again a projection list describes the desired attributes as well as their ordering within a result atom. A simple search argument (SSA) restricts the result set to atoms satisfying the specified condition. A sort criterion defines the sequence in which the atoms of the result set are delivered by subsequent READ_NEXT operations. Additionally, a start/stop condition may limit the result set to a certain range within the sort sequence defined by the sort criterion. However, specifying a sort criterion and an additional start/stop condition is restricted to a sort scan, whereas specifying a projection list as well as an SSA is allowed for both atom-type scan and sort scan.

An atom-type scan is performed by scanning the address structure of the corresponding atom type in order to obtain all logical addresses eventually. In accordance to the READ_ATOM operation, for each logical address all physical addresses are passed to the storage system which delivers a single page or page sequence containing an appropriate physical record. It is checked whether or not the physical record satisfies the specified SSA. If not, the next logical address is treated. Otherwise, the result atom is built up according to the projection list.

A sort scan, however, requires much more effort within the access system. All atoms of the appropriate atom type have to be sorted according to the specified sort criterion. Simultaneously, the SSA as well as the start/stop condition have to be evaluated. The result set has to be successively processed by subsequent READ_NEXT operations. Therefore, it is stored in a segment which has to be deleted at the end of the scan. More information about sorting is given in chapter 4.

All operations offered by the basic version of the access system as well as the corresponding parameters are summarized in the appendix.

3.3 Parameter Formats

In order to avoid multiple copying of data within the different layers, the data exchange format between the access system and the data system is strongly influenced by the internal representation of an atom used in the data system. This internal representation consists of a pointer array which includes an entry for each attribute present in the corresponding atom. These entries point to the appropriate attribute values which may be located anywhere. This means, the result of a retrieval operation is an atom structured in the described way (moreover, the pointer array and the desired attribute values are clustered (Fig. 9)).

operation	description
position	determines first position of an element within the list
count_list	counts the number of elements
read_elements	reads one or more elements of the list
read_list	reads all elements of the list
insert_elements	inserts one or more elements to the list
change_elements	exchanges one or more elements of the list by one or more new elements
replace_list	replaces the whole list by a new one
delete_elements	deletes one or more elements of the list
delete_list	deletes the whole list

a) LIST operations

operation	description
check_element	checks whether or not a set contains a value
count_set	counts the number of values
read_set	reads the whole set
insert_value	adds a new value to the set
replace_set	replaces the whole set by a new one
delete_value	removes a value from the set
delete_set	deletes the whole set

b) SET operations

Figure 8: LIST and SET operations

However, this also means in the case of an update operation (insert or modify), the attributes to be inserted or modified are delivered in the same way.

As a consequence, we need an additional parameter which allows for the interpretation of the corresponding data. In the case of retrieval, the projection list determines which attributes have to be retrieved and how these attributes have to be arranged within the result atom. For this purpose, the projection list consists of multiple entries (Fig. 9) describing a single attribute value by a number of different specifications:

- *Length* encodes the total length either of the appropriate attribute or of a single element for attributes of type LIST or SET.
- *Offset* and *offset_length* specify the desired part of an attribute or element of type ARRAY or RECORD.
- *List/set operation* defines the retrieval operation to be performed on an attribute of type LIST or SET (Fig. 8). Additional parameters of this operation are specified using *number of elements* and *position*.

The sequence of these entries within the projection list, however, defines the arrangement of the corresponding attribute values within the result atom. In the same way, the projection list is utilized to describe the input data on an update operation, i.e. offset and offset_length specify the part of an attribute or element to be modified and list/set operation defines an update operation on an attribute of type LIST or SET (Fig. 8).

An SSA as well as a start/stop condition consists of an operator tree describing the expression to be evaluated and a descriptor list specifying the attributes for which the expression has to be evaluated. The

evaluation of an expression itself is performed by a separate component to which the operator tree as well as the appropriate attribute values have to be passed.

The sort criterion defines the sort attributes. If more than one attribute is specified, the atoms are sorted on the basis of the first attribute, and within each of these values on the basis of the second attribute, and so on. Hence, for each attribute a sort order, either ascending or descending, may be defined.

3.4 Meta-Data System

As already mentioned, we utilize a special component called meta-data system (MDS) in order to maintain the description information needed throughout the mapping process within the data system and the access system. This MDS may be compared, with respect to both functional behaviour (objects and operations) and internal mapping mechanisms, to the basic version of the access system, i.e. it maintains record-oriented structures representing meta-data.

The data definition operations offered by the MDS allow for the interactive definition, retrieval, modification, and deletion of the so-called meta-schema (Fig. 10). The meta-schema defines the description information by means of meta-data types (which corresponds to atom types) and relationships among these types. In contrast to the access system, however, the number of attribute types are restricted to IDENTIFIER, REFERENCE, and BYTE string. Fig. 11 shows the currently used meta-schema. This meta-schema only consists of the meta-data types which are needed by the currently implemented components of PRIMA. New components such as the extensions of the access system described in the next chapter, however, may require additional meta-data types. Therefore, it is important that an existing meta-schema is kept extensible.

In order to store information about the meta-data types themselves, the MDS utilizes a "basic schema" which is directly mapped onto segments and pages (Fig. 12). This basic schema has to be interpreted throughout each MDS operation. Therefore, it is a critical point regarding the overall system performance. First experiences have already proven that we should redesign the MDS with respect to this, since meta-data access is too slow.

The data manipulation operations (Fig. 10) offered by the MDS are utilized by all PRIMA components in order to insert, read, modify, and delete meta-data, i.e. description information. As in the case of the access system, the MDS offers a record-oriented interface which only allows for direct and navigational retrieval and manipulation of single meta-data records. Moreover, the MDS does not support a sort scan or even SSAs.

4. Extensions to the Access System

Whereas the design of the basic version of the access system (except the sort scan) is completed and a first prototype is already running, the extensions described in this chapter are still in the design phase.

4.1 Access Path Structures

Access path structures provide appropriate means for fast value-dependent access to records, i.e. atoms, of a certain type. However, depending on different characteristics of the access pattern one access

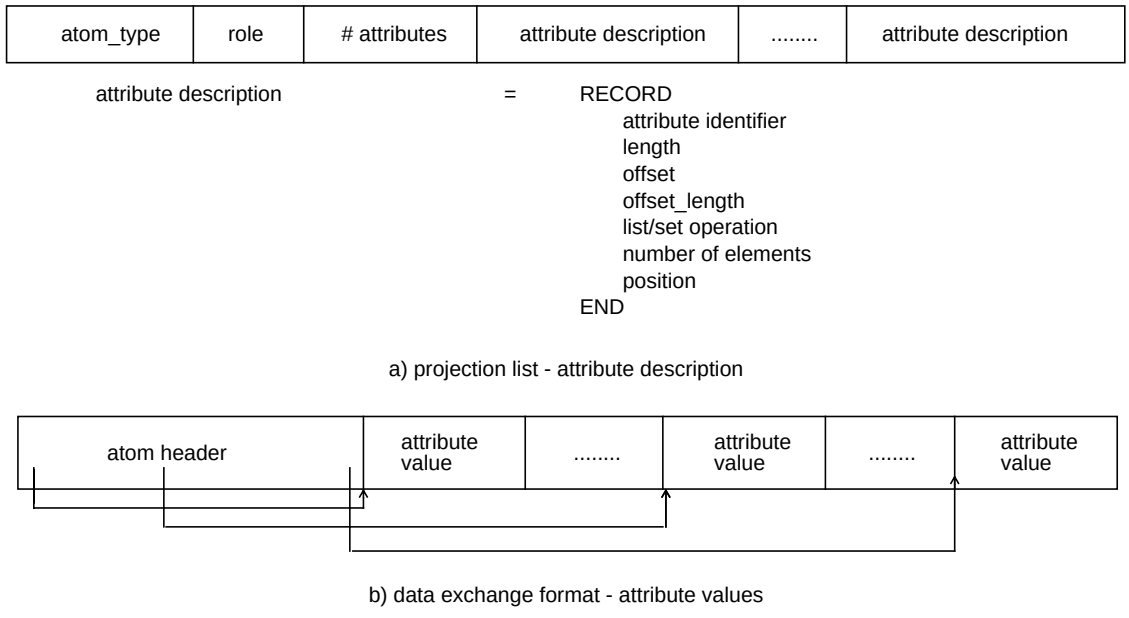


Figure 9: Parameter formats

path structure may be more efficient than another one. Therefore, multiple access path structures should be supported by the access system with regard to the following access pattern:

- exact match as well as exact range queries for a single attribute,
- exact match, partial match, exact range, and partial range queries for multiple attributes,
- spatial access as well as temporal access.

For each of these access patterns (except temporal access) a broad variety of access path structures is described in the literature [Be79, BF74, BM72, CF81, Fr83, Gu84, NHS84, Ta81, Ya84]. We have put in some effort in order to compare these structures in an appropriate way [Si85], and we have decided to utilize the following access path structures:

- B-tree [BM72] for single attribute access,
- grid file [NHS84] for multiple attribute access,
- R-tree [Gu84] for spatial access,
- no special structure for temporal access.

In both tree structures references to the appropriate atoms are maintained. These references correspond to the logical addresses of the atoms. As a consequence, an additional access to the address structure is required in order to obtain the corresponding physical addresses. However, the access path structures are isolated from the redundancy introduced by the different extensions of the access system. Moreover, moving a physical record does not affect any access path structure. On the other hand, the two-disk-access principle is an important characteristic of the grid file structure. Therefore, a grid file directly maintains physical records, although maintenance of logical addresses should be possible. As a result, redundancy is introduced the first time, since multiple grid files may be defined for a single atom type.

An access path structure is constructed either on behalf of the database administrator (DBA) issuing a DEFINE_ACCESS_PATH operation or during a CREATE_ATOM_TYPE operation in order to support checking of user-defined keys. In either cases the corresponding access path structure is determined by the access system according to the following criteria:

data definition:	CREATE_META_DATA_TYPE READ_META_DATA_TYPE MODIFY_META_DATA_TYPE DELETE_META_DATA_TYPE	data manipulation:	INSERT_META_DATA READ_META_DATA MODIFY_META_DATA DELETE_META_DATA OPEN_SCAN READ_NEXT CLOSE_SCAN

Figure 10: Operations of the MDS

- Use a B-tree for a single attribute of a basic type (Fig. 2).
- Use a grid file for multiple attributes each of a basic type.
- Use a R-tree for a single attribute of type HULL.

This means, the access system only supports value-dependent access to a few selected attribute types, whereas value-dependent access to complex attribute types (such as SET or LIST) has to be performed by the data system.

Analogously, access path structures are deleted on behalf of a RELEASE_ACCESS_PATH operation, during a DELETE_ATOM_TYPE operation for the appropriate atom type, or when removing a corresponding attribute by a SHRINK_ATOM_TYPE operation.

We have decided to provide a uniform access path scan for all access path structures supported by the access system. Thus, any access path structure may be added or removed without affecting the data system (except the query optimizer). As a consequence, the access path scan itself becomes more complex. Start/stop conditions and directions may be specified individually for every attribute or even for every dimension of the spatial HULL attribute, thus extending the OPEN_SCAN operation.

In the case of the update operations, the access path structures have to be maintained in an appropriate way, i.e. inserting, modifying, or deleting an access path entry (in the tree structures) or a physical record (in the grid file).

4.2 Sort Orders

A sort scan is expected to be a rather frequent operation, since sorting is required with respect to the ORDER_BY clause of the SELECT statement, and moreover sorting considerably speeds up internal processing of the data system, for example when performing a merging-scan join. However, sorting an entire atom type, repeatedly for each sort scan, is expensive and time consuming. Therefore, a sort scan may be supported by a redundant storage structure, the sort order.

A sort order is generated on behalf of the DBA (DEFINE_SORT_ORDER operation), whereas it is deleted either on behalf of the DBA (RELEASE_SORT_ORDER operation) or when the appropriate atom type is deleted (DELETE_ATOM_TYPE operation).

The storage structure generated for a sort order consists of a sorted list of physical records, one for each atom of the corresponding atom type. Initially, these physical records are stored in subsequent pages according to the sort criterion. Update operations, however, would require a reorganization of the whole structure. Therefore, the corresponding pages are chained. As a consequence, physical records may be added or removed at arbitrary points as pages are split or merged.

Sorting itself is performed in a straight-forward manner. This means, starting with an empty sort order an atom-type scan is initialized and each atom, i.e. physical record, delivered by the scan is successively

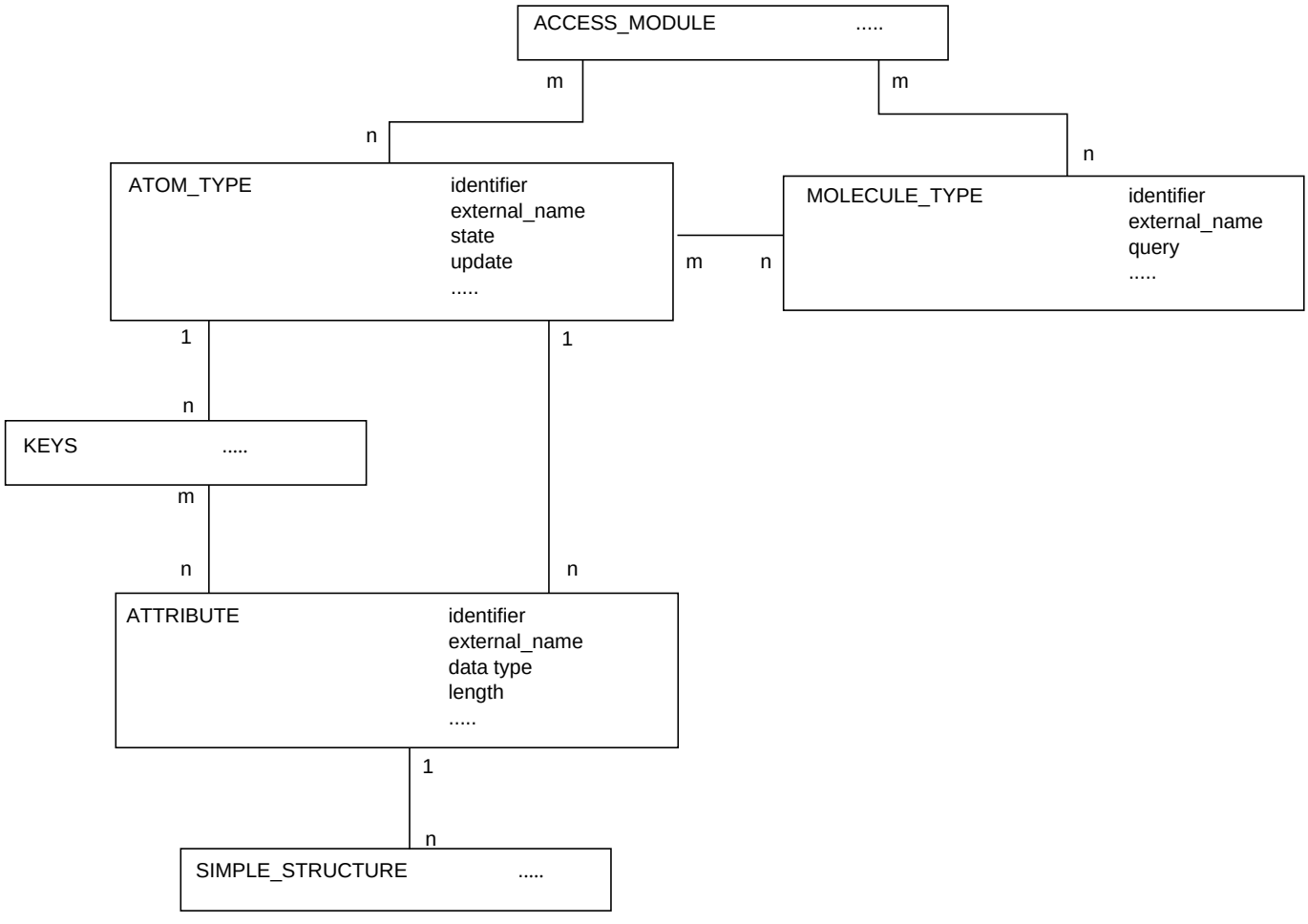


Figure 11: Current meta-schema

inserted into the sort order at the appropriate point. This procedure, however, is rather inefficient. Therefore, different sort techniques (such as quicksort, [Hä77]) have to be investigated with respect to their applicability in the access system. Moreover, the extent in which existing access path structures or sort orders may be utilized has to be considered.

4.3 Atom-Cluster Types and Atom Clusters

The concept of atom clusters has been introduced in order to speed up construction of frequently used molecules, by allocating all atoms of a corresponding molecule in physical contiguity. Atom clusters, however, are restricted to simple molecules of a non-recursive, hierarchical structure according to a query of the following form:

SELECT ALL
FROM <one non-recursive, hierarchical molecule type>

Such an atom cluster corresponds either to a heterogeneous or to a homogenous atom set described by a so-called characteristic atom. This characteristic atom simply contains references to all atoms, grouped by atom types, belonging to the atom cluster (Fig. 13a). Each atom cluster is mapped onto one physical record containing the characteristic atom as well as all atoms referenced by the corresponding characteristic atom (Fig. 13b). The physical record is, in turn, mapped onto a page or a page sequence depending on its current length. Thereby, the mapping of a physical record onto a page sequence is performed

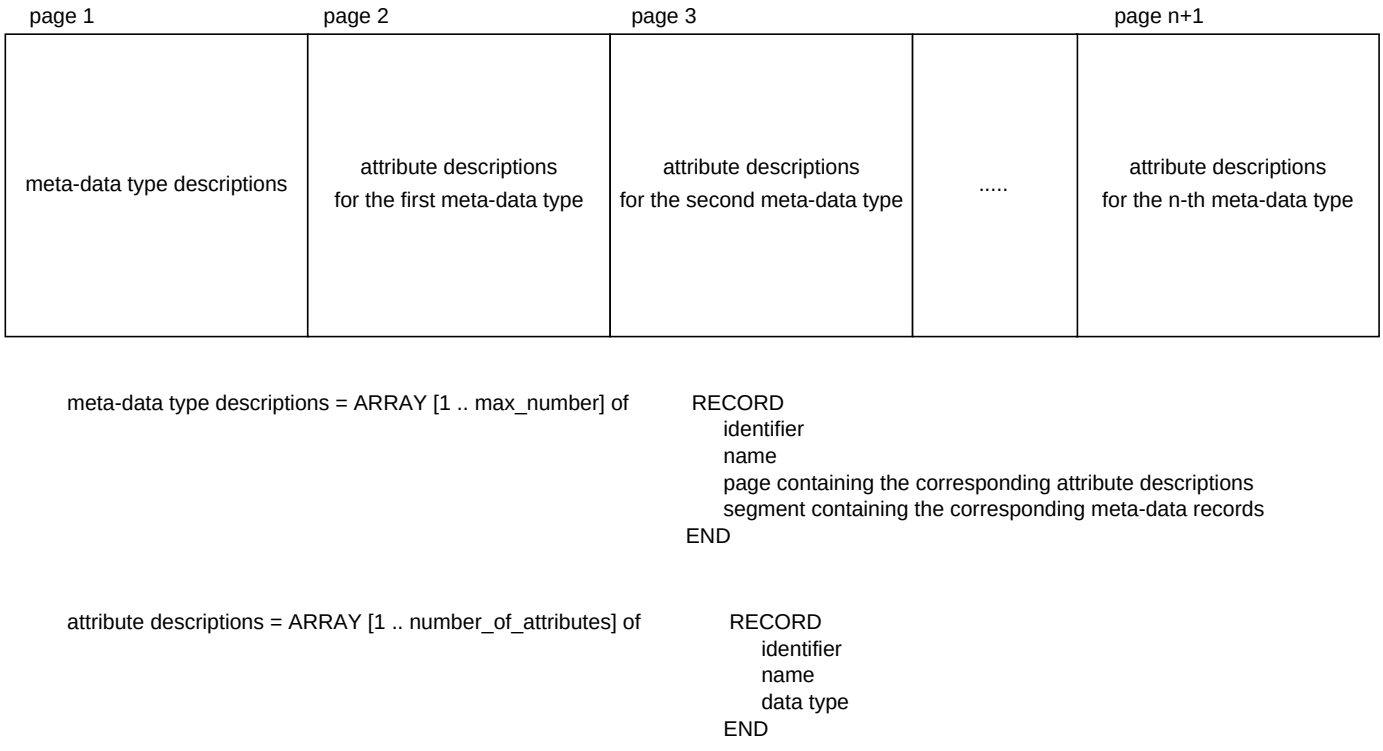


Figure 12: "Basic schema" of the MDS

as follows (Fig. 13c): All atoms of a single atom type are placed into a subrecord. All subrecords are subsequently mapped onto pages. If a subrecord exceeds the free space available within a page, a new page is allocated. If a subrecord requires multiple pages, these pages are exclusively used by the subrecord. However, in order to locate an atom within an atom cluster, i.e. within a page sequence, an additional address structure (described in the next chapter) is required.

Performing an update operation on an atom, the access system is responsible for maintaining the appropriate atom clusters. This means, the access system has to automatically include atoms into an atom cluster, move them from one cluster into another, and delete them from a cluster depending on the corresponding update operation. This is fairly simple for a non-recursive, hierarchical molecule structure:

- When inserting a root atom, all "down references" have to be evaluated and the referenced atoms have to be incorporated into the atom cluster. For these atoms, again all "down references" have to be evaluated, and so on.
- When inserting a child atom, all "up references" have to be evaluated in order to obtain all predecessors of the atom. For each predecessor, the atom clusters it belongs to are revealed using the address structure of the corresponding atom type. Now, the new atom as well as all its descendants have to be included into the revealed clusters.
- When deleting a root atom, the whole cluster has to be deleted.
- When deleting a child atom, this atom as well as all its descendants have to be removed from all atom clusters it belongs to.
- When modifying "down references" within an atom, all atoms which are no longer referenced, as well as their descendants, have to be removed from all atom clusters the modified atom belongs to. On the other hand, all atoms which are now referenced, as well as their descendants, have to be included into those clusters.

- When modifying "up references" within an atom, the modified atom as well as its descendants have to be removed from those atom clusters which are determined by the predecessors no longer referenced, and they have to be included into those clusters which are determined by those predecessors now referenced.

For recursive and/or network-like structures, however, this is much more complicated as it depends on the semantics of the structures. Therefore, atom-cluster types are restricted to simple molecule types.

Whereas the scans, described up until now, only support horizontal access to a homogeneous atom set belonging to one atom type, the two scans defined for atom-cluster types allow for access to a heterogeneous atom set across several atom types. The atom-cluster type scan delivers all characteristic atoms of an atom-cluster type in a system-defined order. As for all other scans, the result set of the atom-cluster type scan may be restricted by a complex search argument (CSA), which must be decidable in one pass through a single atom cluster (single scan property [DPS86]). Subsequently, direct access to all atoms belonging to an atom cluster is possible as each characteristic atom contains the corresponding logical addresses. The atom cluster scan, however, offers another possibility for accessing the atoms of an atom cluster. It reads all atoms of a certain atom type within one single atom cluster in a system-defined order, again with the possible restriction by a simple search argument.

4.4 Partitions

A further extension to the access system are partitions. A partition allows for a vertical partitioning of an atom type. Thus, frequently used attributes of an atom type may be clustered and stored independently from other attributes clustered in a similar way. As a consequence, multiple physical records, one for each attribute cluster, are assigned to a single atom. Each of these physical records consists of the appropriate attribute values as well as the IDENTIFIER attribute in order to locate the physical record within a page. A partition is automatically utilized by the access system. Therefore, no explicit retrieval operation referring to a partition is required.

5. Maintaining Redundancy

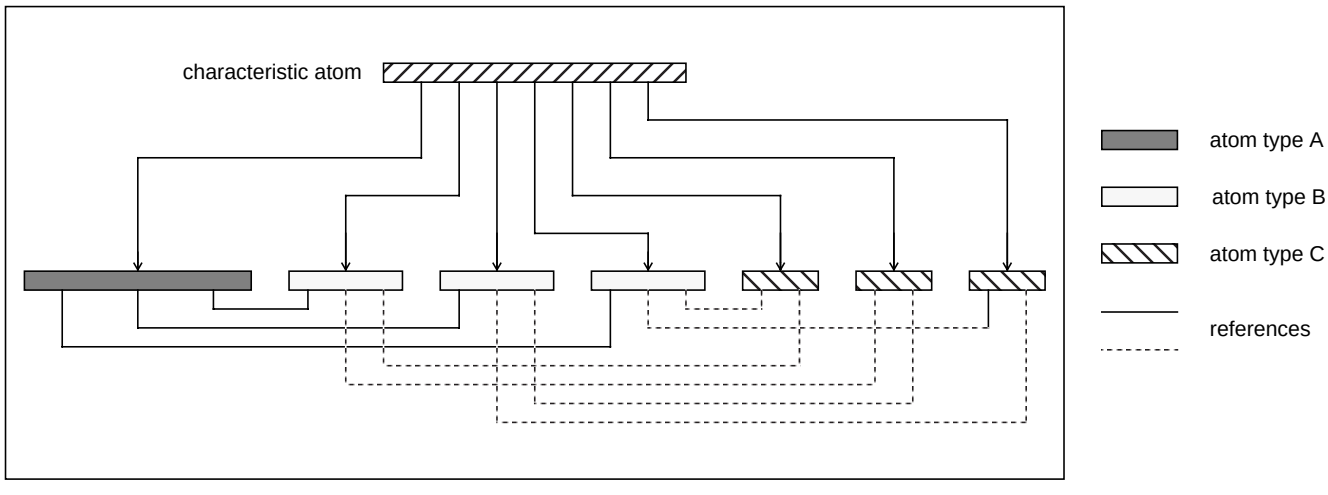
All extensions introduced in the above chapter generate additional storage structures which embody either homogeneous or heterogeneous result sets. For example, an atom cluster serves to embody molecules, whereas partitions collect the results of projections. The underlying concept is to make storage redundancy available outside the access system by offering appropriate retrieval operations (i.e. scans), whereas in the case of update operations storage redundancy has to be concealed by the access system. As a consequence, maintaining storage redundancy in an efficient way is a major task of the access system. Especially in the case of update operations, new concepts have to be investigated in order to speed up a single update operation. However, before concentrating on this problem, the already mentioned address structure has to be described in more detail.

5.1 The Addressing Concept

The addressing concept of PRIMA [Si87a] establishes the way in which the different objects handled within the access system, i.e. atoms, atom clusters, and physical records, are addressed and the way in which the different addresses are mapped onto each other.

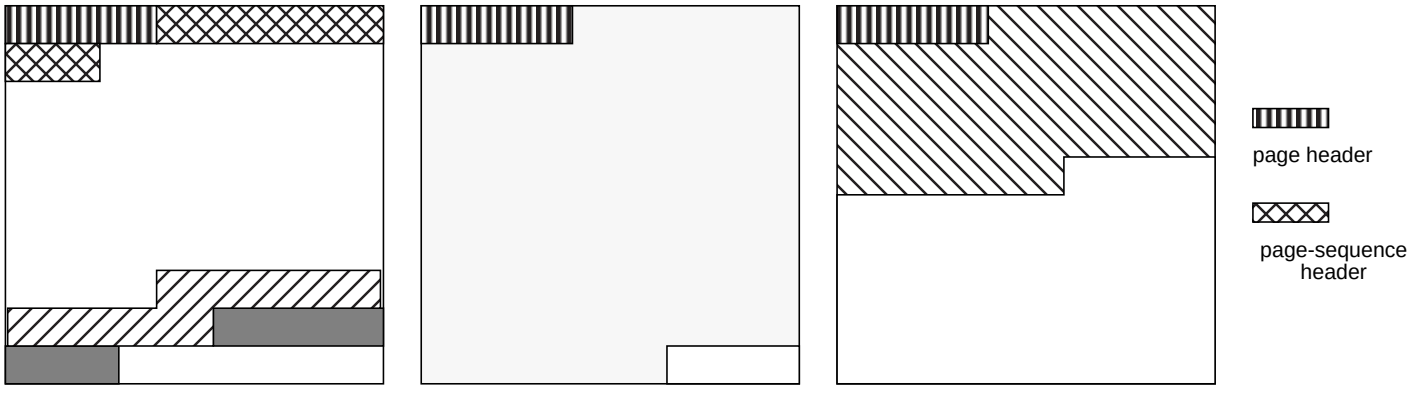
Addressing Atoms

Each atom is, as already mentioned, uniquely identified by its logical address which is assigned when the atom is inserted. The structure of a logical address is very similar to that of the IDENTIFIER type introduced in [Lo84], i.e. a logical address consists of an atom-type identifier (unique within the whole schema) and of an atom identifier (unique within a single atom type):



a) logical view

b) mapping onto a physical record



c) mapping onto a set of pages

Figure 13: Atom cluster

As a consequence, a logical address is unique system-wide, easy to reuse, and independent of every storage structure.

Addressing Atom Clusters

Since each atom cluster is described by a characteristic atom (see section 4.3), the logical address of this characteristic atom may be utilized in order to access the corresponding atom cluster. That is, the concept of logical addresses is also sufficient for the addressing of atom clusters.

Addressing Physical Records

In order to locate a physical record within the "containers" offered by the storage system, the physical address assigned to each physical record consists of a segment number and of a page number or page-sequence number indicating the corresponding page or page sequence in which the record is stored:

segment number	information part	page number / page-sequence number	structure identifier

However, an information part is required in order to distinguish between a page number and a page-sequence number. Additionally, a physical address contains an identification of the storage structure the physical record belongs to. Thereby, the type of the storage structure (sort order, partition etc.) is also encoded in the information part, whereas the structure identifier determines the actual storage structure.

Mapping Logical Addresses onto Physical Addresses

Mapping a logical address onto the appropriate physical addresses requires a flexible address structure, since depending on the additional storage structures a variable number of physical records may exist for each atom. Therefore, a so-called address list is initially assigned to each atom. This address list consists of the corresponding logical address, of a length field indicating the number of physical addresses assigned to the logical address, and of the physical addresses themselves:

logical address	length	physical address	.....	physical address

The access to such an address list is performed by means of an appropriate address translation method. We have investigated different methods (Fig. 14) with respect to the number of page accesses required for the access to a single address list and with respect to the overhead which arises when physical addresses are either added or removed [Si85]. Although these investigations are not yet finished, the most promising method seems to be a dynamic hash method [Si87b], e.g. linear virtual hashing [Li80].

However, we have designed a special address component [Wi87] which conceals the realization of the address translation method in order to allow for an easy exchange of the different methods.

Mapping of Physical Addresses onto Logical Addresses

The mapping of a physical address onto the appropriate logical addresses is implicitly contained in each physical record. If a physical record corresponds to either an attribute cluster or a complete atom, the IDENTIFIER attribute, i.e. the logical address, resides at the beginning of the corresponding physical record. Otherwise, if a physical record corresponds to an atom cluster, the appropriate characteristic atom (at the beginning of the physical record) includes all logical addresses of the atoms belonging to the atom cluster (Fig. 13).

Moreover, each physical record assigned to an atom cluster requires an additional address structure which allows for the fast location of a single atom within the corresponding physical record. This address structure, however, strongly depends on the current size of the physical record. If the record fits into a single page, no additional address structure is necessary, since within a page a sequential search is always performed. If the record is spread over a page sequence, the address structure initially consists of a simple table indicating for each atom type the (first) page to which the appropriate subrecord is mapped (Fig. 13). In addition, each subrecord which requires multiple pages contains a further table indicating for each atom the page in which it is stored. However, this procedure has to be investigated in more detail when the concept of atom clusters is really implemented.

5.2 Updating Storage Redundancy

Introducing storage redundancy serves to speed up retrieval. On the other hand, however, it slows down update, since during a single update operation on an atom multiple physical records and access paths have to be altered in order to achieve consistent storage structures. Sequential update of all physical records and access path structures results in a lack of efficiency which is not acceptable. Therefore, new concepts such as deferred update and parallel update have to be investigated in more detail with respect to their applicability in PRIMA.

Deferred Update

Deferred update means that during an update operation on an atom initially only one of the appropriate physical records is altered. All other physical records as well as the corresponding access paths are marked as invalid. Finally, a "process" is initialized which alters the invalid structures in a deferred manner, whereas the update operation itself is finished.

In order to mark a physical record as invalid the information part of the appropriate physical address is used, i.e. a bit within the information part indicates whether or not the corresponding physical record is valid. Therefore, all operations which utilize the address structure in order to locate a physical record may determine the valid records. Most of the scan operations (except the atom type scan), however, do not utilize the address structure. Hence, the corresponding storage structures themselves (access paths, sort orders, and atom clusters) have to be marked as invalid. Consequently, when performing a scan operation on such an invalid structure, each physical record has to be checked as to whether or not it is valid. This, however, requires an additional access to the appropriate address structure, i.e. the scan operation becomes slower, since each access to the address structure may result in an external disk access. In order to avoid this, all invalid atoms (or their logical addresses), may be collected in a number of special pages assigned to each storage structure. These pages may be kept in the database buffer throughout the whole scan operation thus avoiding extra disk accesses. Nevertheless, each physical record has to be compared with the atoms collected in these pages. In this context, some problems arise, especially in the case of access paths and sort orders, which still have to be investigated in more detail (e.g. when a modification alters a sort sequence). Moreover, deferred update is related to the underlying transaction concept described in the next chapter.

Parallel Update

The problem of maintaining invalid storage structures, however, is avoided by parallel update. Parallel update means that each update operation on an atom invokes a number of processes which alter the appropriate physical records and access paths in parallel. The update operation is finished when all pro-

cesses are finished. Depending on how the access system is structured, there are different ways to perform a parallel update:

- Each update operation is passed directly to all components maintaining a single storage structure type (i.e. sort component, etc.). Each component checks which storage structures of the appropriate type are affected by the update operation, and the corresponding storage structures are modified either sequentially or again in parallel.
- On the other hand, a general component may check which storage structures are affected by the update operation. For each of the affected storage structures the appropriate component is invoked in order to perform the modification.

However, additional investigations are still necessary in order to determine the best way (e.g. with respect to extensibility or performance).

Again, parallel update is strongly related to the underlying transaction concept which will be described in the next chapter.

6. Dynamic Aspects

The requirement for long transactions (e.g. design transactions) and the desire for executing suboperations of complex operations concurrently introduces a new quality of dynamically controlling parallelism. Therefore, we have decided to combine the concept of multilevel transactions [WS84] and the concept of nested transactions [Mo81] in order to improve concurrency among suboperations and to obtain a finer granule of recovery [CP88].

As far as the access system is concerned two transaction levels have to be considered:

- The storage system supports transactions on the page level, i.e. the granule of synchronization and recovery are pages.
- The data system and the access system support hierarchically nested transactions on the atom level.

Hence, each operation at the access-system interface (direct access, scan, update) corresponds to a subtransaction on the atom level and on the page level. As a consequence, the access system has to invoke an appropriate lock and log component in order to request locks or to write log information.

For synchronization purposes, we have chosen hierarchical locks on atom types and on atoms. However, locks are not only requested for the atom type or atom which are affected by the current operation, but also for all atom types or atoms which are referenced by this atom type or atom. That is:

- Each direct access to an atom requests S-locks on the corresponding atom and on all atoms whose references are selected. The appropriate IS-locks on the atom types are implicitly set.
- Each update operation requests X-locks on the corresponding atom and on all atoms whose back-references have to be adapted. Again, the appropriate IX-locks on the atom types are implicitly set.
- Each scan operation requests SIX-locks on the corresponding atom type and on all atom types whose references are selected.

However, this procedure may result in a number of fictitious synchronization conflicts which dramatically decreases parallel processing. Especially, in the case of scans and locking of referenced atoms the cho-

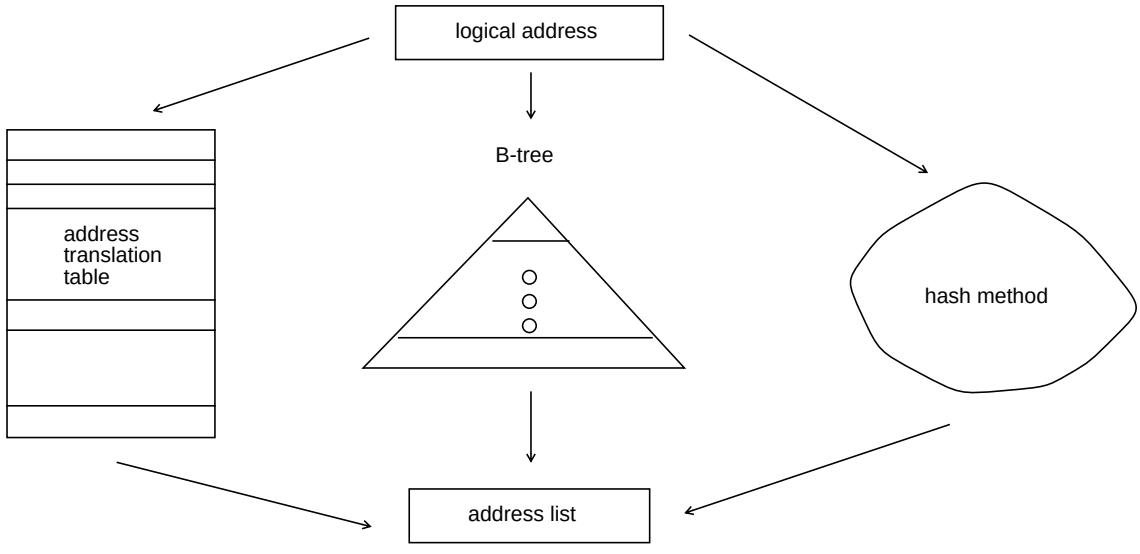


Figure 14: Different address translation methods

sen lock granule may be too coarse. Therefore, the synchronization concept should be revised in order to avoid such conflicts.

The concept for logging and recovery on the atom level is still in the design phase. Especially, the DDL operations SHRINK_ATOM_TYPE and DELETE_ATOM_TYPE cause some problems whose solution probably may require a redesign of the corresponding operation.

Moreover, the transaction concept has to be reflected with respect to deferred update as well as parallel update.

7. Implementation of the Access System

The implementation language of PRIMA is LADY [WM85], a language for distributed systems designed at the University of Kaiserslautern. LADY has three distinguishable language layers:

- A system consists of a set of teams interconnected via ports. The interconnection structure is given by either directed port-to-port channels or by logical busses (multicast, broadcast).
- Each team (more precisely: each team type) is decomposed into a set of processes and monitors. Monitors serve for intra-team-communication.
- At the module level the algorithmic behavior of each module type (process, monitor, class, procedure) is specified. Modules may be compiled separately.

The access system is part of a team type which comprises of a number of processes and monitors needed to implement PRIMA. The first prototype of the access system consists of

- a monitor representing the interface of the access system,
- a process implementing the basic version of the access system except the sort scan,
- a process maintaining the address structure described in section 5.1, and

- a monitor representing the interface to the address structure.

The implementation of this prototype is finished [Sch87, Wi87]. A B-tree component is already implemented [De88], has still to be integrated into the overall system.

8. Conclusions

The design of PRIMA is characterized by a clean break-up into three different layers with appropriate tasks. The main tasks of the access system comprise of

- mapping a number of different storage structures (access paths, sort orders, atom clusters, and partitions) onto the "containers" offered by the storage system,
- maintaining the redundant storage structures according to the update operations, and
- supporting different access types based on them.

For this purpose, the access system offers an atom-oriented interface which allows for direct and navigational retrieval and manipulation of single atoms. To satisfy the retrieval requirements of the data system in the case of "construction of simple molecules", it supports direct access to a single atom as well as sequential access to either homogeneous or heterogeneous atom sets. Performing update operations it is responsible for the automatic maintenance of the referential integrity defined by REFERENCE attributes. Update operations and direct access operate on atoms identified by their logical address. Concerning access to atom sets, scans (atom type scan, sort scan, access path scan, atom-cluster type scan, and atom cluster scan) are introduced as a concept to control such an atom set, to hold a current position in it, and to successively deliver single atoms. The result set of a scan may be restricted by simple or complex search arguments (SSA, CSA) and perhaps by some additional start/stop conditions and directions. SSA and CSA as well as start/stop conditions have to be decidable on a single atom or atom cluster. Some scan operations, however, depend on a corresponding storage structure such as an access path, a sort order, or an atom cluster.

A first prototype of the access system consisting of the basic version (except sort scan) is running. The next step will be to finish the design of the different extensions, to implement appropriate components, and to integrate them into the overall system. As a prerequisite, however, the redesign of the transaction concept with respect to either parallel or deferred update has to be completed.

9. References

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

As81 Astrahan, M.M., et al.: A History and Evaluation of SYSTEM R, in: Communications of the ACM, Vol. 24, No. 10, October 1981, pp. 632-646.

BB84 Batory, D.S., Buchman, A.P.: Molecular Objects, Abstract Data Types and Data Models: A Framework, in: Proceedings of the 10th International Conference on Very Large Databases, Singapore, 1984, pp. 172-184.

Be79 Bentley, J.L.: Multidimensional Binary Search Trees Used for Associative Searching, in: Communications to the ACM, Vol. 18, No. 9, 1975, pp. 509-517.

BF74 Bentley, J.L., Finkel, R.A.: Quad Trees - A Data Structure for Retrieval on Composite Keys, in: Acta Informatica, Vol. 4, 1974, pp. 1-9.

BM72 Bayer, R., McCreight, E.: Organization and Maintenance of Large Ordered Indexes, in: Acta Informatica, Vol. 1, 1972, S. 173-189.

Ch85 Chou, H.T., et al.: Design and Implementation of the Wisconsin Storage System, in: Software - Practice and Experience, Vol. 15, No. 10, October 1985, pp. 934-962.

CF81 Chang, J.-M., Fu, K.S.: Extended K-d Tree Database Organization: A Dynamic Multiattribute Clustering Method, in: IEEE Transactions on Software Engineering, Vol. SE-7, No. 3, 1981, pp. 284-290.

CP88 Christmann, H., Profit, M.: Transaction Support in PRIMA, in: The PRIMA Project - Design and Implementation of a Non-Standard Database System, T. Härder (ed.), SFB 124 Research Report No. 26/88, University Kaiserslautern, March 1988.

De88 Dehnrich, G.:Entwurf und Implementierung einer Komponente zur Verwaltung von B*-Bäumen auf der Basis paralleler Algorithmen, University Kaiserslautern, 1988.

DPS86 Deppisch, U., Paul, H.-B., Schek, H.-J.: A Storage System for Complex Objects, in: Proceedings of the International Workshop on Object Oriented Database Systems, Asilomar, ed.: K. Dittrich, U. Dayal, 1986, pp. 183-195.

Fr83 Frank, A.: Probleme der Realisierung von Landinformationssystemen - Storage Methods for Space Related Data: The FIELD TREE, Report No. 71, Eidgenössische Technische Hochschule Zürich, Institut für Geodäsie und Photogrammetrie, 1983.

Gu84 Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching, in: Proceedings of the ACM SIGMOD Conference, SIGMOD RECORD, Vol. 14, No. 2, Proceedings of the Annual Meeting, 1984, pp. 47-57.

Hä77 Härder, T.: A Scan-Driven Sort Facility for a Relational Database System, in: Proceedings of the 3rd International Conference on Very Large Data Bases, Kyoto, 1977, pp. 236-243.

Hä87 Härder, T., et al.: PRIMA - a DBMS Prototype Supporting Engineering Applications, in: Proceedings of the 13th International Conference on Very Large Data Bases, Brighton, 1987, pp. 433-442.

HHLM87 Härder, T., Hübel, C., Langenfeld, S., Mitschang, B.: KUNICAD - ein datenbankgestütztes geometrisches Modellierungssystem für Werkstücke, in: Informatik Forschung und Entwicklung, Vol. 2, No. 1, 1987, pp. 1.-18.

HMP87 Härder, T., Mattos, N., Puppe, F.: Zur Kopplung von Datenbank- und Expertensystemen, in: State of the Art, Vol. 1, No. 3, 1987, pp. 23-34.

Li80 Litwin, W.: Linear Hashing - A New Algorithm for Files and Table Addressing, in: Proceedings of the International Conference on Databases, Aberdeen, 1980, pp. 260-276.

Lo84 Lorie, R., et al.: Supporting Complex Objects in a Relational System for Engineering Databases, in: Query Processing in Database Systems, ed.: Kim, W., Reiner, D.S., Batory, D.S., Springer, Berlin Heidelberg New York Tokyo, 1984, S. 145-155.

Mi85 Mitschang, B.: Charakteristiken des Komplex-Objekt-Begriffs und Ansätze zu dessen Realisierung, in: Proceedings of the GI Conference on Database Systems in Office, Engineering,

and Science Environments, Karlsruhe, ed.: A. Blaser, P. Pistor, Informatik-Fachberichte No. 94, Springer, Berlin Heidelberg New York Tokyo, 1985, pp. 382-400.

Mi88 Mitschang, B.: The Molecule-Atom Data Model, in: The PRIMA Project - Design and Implementation of a Non-Standard Database System, T. Härder (ed.), SFB 124 Research Report No. 26/88, University Kaiserslautern, 1988.

ML83 Meier, A., Lorie, R.: A Surrogate Concept for Engineering Databases, in: Proceedings of the 9th International Conference on Very Large Data Bases, Florenz, 1983, pp. 30-32.

Mo81 Moss, J.E.B.: Nested Transactions: An Approach to Reliable Distributed Computing, PhD Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 1981.

NHS84 Nievergelt, J., Hinterberger, H., Sevcik, K.C.: The Grid File: An Adaptable, Symmetric Multikey File Structure, in: ACM Transactions on Database Systems, Vol. 9, No. 1, 1984, pp. 38-71.

Sch87 Schares, R.: Implementierung der Operationen auf permanenten und temporären Atom-Typen sowie auf Atomen, University Kaiserslautern, 1987.

Sch88 Schöning, H.: The PRIMA Data System: Query Processing of Molecules, in: The PRIMA Project - Design and Implementation of a Non-Standard Database System, T. Härder (ed.), SFB 124 Research Report No. 26/88, University Kaiserslautern, 1988.

Si85 Sikeler, A.: Untersuchungen von Speicherungsstrukturen für 3-dimensionale Objekte, SFB 124 Research Report No. 19/85, University Kaiserslautern, 1985.

Si87a Sikeler, A.: Ein Adressierungskonzept zur Unterstützung der objekt-orientierten Verarbeitung in PRIMA, in: Proceedings of the GI Conference on Database Systems in Office, Engineering, and Science Environments, Darmstadt, ed.: H.-J. Schek, G. Schlageter, Informatik-Fachberichte No. 136, Springer, Berlin Heidelberg New York Tokyo, 1985, pp. 487-491.

Si87b Sikeler, A.: Untersuchung von Verfahren zur Adreßabbildung in PRIMA, in: Angewandte Informatik, Vol. 8/9, September 1987, pp. 358-368.

Si88 Sikeler, A.: Buffer Management in a Non-Standard Database System, in: The PRIMA Project - Design and Implementation of a Non-Standard Database System, T. Härder (ed.), SFB 124 Research Report No. 26/88, University Kaiserslautern, March 1988.

Ta81 Tamminen, M.: The EXCELL Method for Efficient Geometric Access to Data, Acta Polytechnica Scandinavica - Mathematics and Computer Science Series No. 34, Technical University Helsinki, 1981.

We87 Weber, B.: Implementierung eines einfachen Verwaltungssystems für die Metadaten in PRIMA, University Kaiserslautern, 1987.

Wi87 Wintzheimer, V.: Zuordnungstabelle und Lineares Virtuelles Hashing - Zwei verschiedene Verfahren zur Adreßabbildung in PRIMA, University Kaiserslautern, 1987.

WM85 Wybranietz, D., Massar, R.: An Overview of LADY - A Language for the Implementation of Distributed Operating Systems, SFB 124 Research Report No. 11/85, University Kaiserslautern, 1985.

WS84 Weikum, G., Schek, H.J.: Architectural Issues of Transaction Management in Multi-Layered Systems, in: Proceedings of the 10th International Conference on Very Large Data Bases, Singapore, 1984, pp. 454-465.

Ya84 Yamaguchi, K. et al.: Octree-Related Data Structures and Algorithms, in: IEEE Computer Graphics and Applications, Vol. 4, No. 1, 1984, pp. 53-59.

Appendix : Operations offered by the basic version of the access system

CREATE_ATOM_TYPE	IN:	transaction identifier
		atom-type identifier
	OUT:	returncode
DELETE_ATOM_TYPE	IN:	transaction identifier
		atom-type identifier
	OUT:	returncode
EXPAND_ATOM_TYPE	IN:	transaction identifier
		atom-type identifier
		attribute-number list
	OUT:	returncode
SHRINK_ATOM_TYPE	IN:	transaction identifier
		atom-type identifier
		attribute-number list
	OUT:	returncode
INSERT_ATOM	IN:	transaction identifier
		atom-type identifier
		atom description
		attribute values
	OUT:	atom identifier
		returncode
READ_ATOM	IN:	transaction identifier
		atom identifier
		atom description
	OUT:	attribute values
		returncode
MODIFY_ATOM	IN:	transaction identifier
		atom identifier
		atom description
		attribute values
	OUT:	returncode
DELETE_ATOM	IN:	transaction identifier
		atom identifier
	OUT:	returncode
OPEN_SCAN	IN:	transaction identifier
		atom-type identifier
		atom description

		simple search argument
		sort criteria
		start/stop condition
	OUT:	scan identifier
		returncode
READ_NEXT	IN:	transaction identifier
		scan identifier
	OUT:	attribute values
		returncode
CLOSE_SCAN	IN:	transaction identifier
		scan identifier
	OUT:	returncode