PRIMA - a DBMS Prototype Supporting Engineering Applications

T. Härder, K. Meyer-Wegener, B. Mitschang, A. Sikeler

University of Kaiserslautern
Erwin-Schrödinger Straße, D-6750 Kaiserslautern, West-Germany

Abstract
The design of the Molecule-Atom Data model, aimed at the effective support of engineering applications, is justified and described with its essential properties and features. MAD offers direct and symmetric management of network structures and recursiveness, dynamic object definition and object handling allowing for both vertical and horizontal access. Its prototype implementation PRIMA is discussed using a multi-level model for the DBMS architecture. Our DBMS kernel provides a variety of access path structures, tuning mechanisms, and performance enhancements transparent at the data model interface. PRIMA is assumed to be used in different run-time environments including workstation coupling and multi-processor systems. In particular, it serves as a research vehicle to investigate the exploitation of 'semantic parallelism' in single user operations.

1. Introduction

Conventional DBMS have failed to provide appropriate support and satisfactory performance for a wide variety of engineering applications [HL82, DB83]. Therefore, considerable research efforts are directed to the design and implementation of a new generation of DBMS architectures including data models, extensible implementations, storage structures, transaction concepts and so on. Of course, the chosen data model and its specific properties play the dominant role in all these approaches; most of them can be classified in the following manner:

- They focus on the flat relational model with a few selected enhancements [SR86, LMP86, CD86].
- They concentrate on integrating and superimposing hierarchical structures on relations [LK84, PSSWD87, Da86, RKB85].

Apparently, the provision of genuine and symmetric support of network structures or even recursive structures has drawn much less attraction, although it is urgently needed in many application areas for natural and accurate modeling and efficient processing of their objects. To identify their specific needs, we have thoroughly investigated three different application areas - their structures and algorithms - by implementing and evaluating sizable prototype systems: VLSI circuit design, construction of solids in 3D modeling, and map handling in geographic information systems [HHLM87]. Our observations may be summarized as follows:

- There is a considerable share of meshed (non-hierarchical) structures due to extensive occurrence of n:m relationships.
- No general preference of execution direction (traversal of data structures) is found, that is, efficient support for symmetric traversal is required.
- Locality of reference strongly depends on the algorithms used; they typically perform non-uniform references to subobjects. As a consequence, selective access to large objects is desirable.
- To enhance integrity control and semantic expressiveness, all essential relationships (beyond hierarchical ones) should be preserved by the data model.

These observations have motivated our research efforts and have strongly influenced many features and functions of our engineering DBMS. The rationale of our approach may best be characterized by the following design guidelines:

- object-orientation of the data model allowing for manipulation of complex objects at the data model interface
- direct representation of n:m relationships along with symmetric traversal and use of objects
- dynamic construction of complex objects as opposed to their static representation
- set-orientation at the data model interface and in internal layers where appropriate
- support of object processing by a variety of storage structures, use of tuning mechanisms, and performance enhancements transparent at the data model interface.

We have designed and are implementing a DBMS kernel for so-called non-standard applications, in particular for those applied in various engineering disciplines: Numerous papers

were published justifying the kernel architecture approach, its benefits, and its key properties [Da86, HR85, PSSWD87]. Therefore, we can start immediately with the description of our own way, which resulted in substantial differences with other DBMS kernel designs. We feel that the above guidelines have provoked quite a number of novel ideas, which are incorporated in

- the design of the 'Molecule-Atom Data model', called MAD model
- its implementation by a database kernel named PRIMA (PRototype Implementation of the MAD model)
- the processing model for PRIMA to take advantage of 'semantic parallelism' in powerful operations used in engineering applications.

The results of our system design are described in the following.

2. The Molecule-Atom Data Model

One of the most demanding requirements in engineering applications is accurate modeling and efficient management of application objects. Starting with an analysis and characterization of the application objects, we point out the essential requirements to facilitate application modeling. The shortcomings of existing data models gave rise to the development of the MAD model as an object-oriented data model. We describe and exemplify both its modeling and processing concepts. Additionally, we introduce the **Molecule Query Language** (MQL) by illustrating its query facilities as well as its data manipulation and data definition capability.

2.1 Data Model Requirements for Engineering Applications

For our purpose, the best reference is [BB84] where a thoroughgoing analysis and characterization of the application objects in engineering disciplines revealed the general concept of **molecular objects** - in [LK 84] and in the introduction above called complex objects. These objects are seen and manipulated on different levels of abstraction. At higher levels, they are treated as atomic units of data, e.g. moved or copied as a whole. Futhermore, each entire entity is described by several attributes. At lower levels, they reveal their internal structure. Their components may again be complex objects, or just primitive objects without internal structure. Complex objects are of the same type, if all their attributes and components have the same type.

Mapping complex objects to data objects leads to a record data type containing the attributes of the complex object, a record data type for each component, and a relationship between them meaning 'consists of'. Hence, the complex object is represented by all data elements related in this way. They are said to form a 'molecule' with the data elements (records) resembling 'atoms', respectively.

Two different complex objects may share components, for instance, 3D solids share the face where they are 'glued' together. In this case, according to [BB84] they are called **non-disjoint**, and their molecules overlap; hence, the consists-of relationship must be of type many-to-many (n:m). Otherwise, they are called **disjoint**, building non-overlapping molecules and a one-to-many (1:n) relationship. Additionally, complex objects are called **recursive**, if they are composed of objects of the same type; otherwise, they are called **non-recursive**. For example, solids in 3D modeling are 'constructed' using previously defined solids, thus forming a recursive consists-of relationship.

What makes up a complex object depends on the actual view of the application, that is the level of abstraction and the way of processing. Thus, it seems much more adequate to define the molecules dynamically instead of predefining molecules statically. Referring to 3D modeling, these dynamics are apparent: One important representation of solids, especially for graphical output, is the boundary representation (BREP) depicted in Fig. 2.1. It consists of faces, which are, in turn, composed of its borderlines (edges) limited by endpoints (cf. the Entity-Relationship diagram of Fig. 2.1). Some applications need face objects with their edges and points, while in another processing state it may be necessary to handle just the inverse object nesting, that is a point object with its neighboring faces forming the point-edge-face hierarchy.

Existing data models do not match these requirements properly. When modeling in an hierarchic manner, one has to cope with redundancy. This holds for the classical models like IMS [Mc77], as well as for novel ones such as non-first-normal-form models [SS87, RKB85] and the so-called complex-object model [LK84]. Fig. 2.1 illustrates the consequence using our BREP schema. A first observation is, that the hierarchical schema is not equivalent to the network schema. Second, a substantial portion of redundancy is introduced. There are several independent representations for every edge and every point. Since the DBMS is not aware of this redundancy, it must be handled by the application (or at least above the data model interface). This may lead to problems concerning integrity (no gap between faces), preservation of topology, update, etc.

The network approach avoids redundancy, but at the cost of introducing a number of 'relation records' that represent n:m relationships. The mentioned data models only support non-recursive, disjoint objects referring to a static object type in a non-symmetric manner, e.g., looking from points to all corresponding edges and faces is not possible in the hierarchical example in Fig. 2.1. On the right-hand side of Fig. 2.1 we have shown the desired modeling approach, referred to as direct and symmetric modeling, thus avoiding the above mentioned problems.

In managing engineering objects, we have to cope with two different kinds of access. The most challenging is called **vertical access**. It is characterized by accessing the object as a whole, i.e., fetching all constituting (more primitive) components. In addition, vertical access may select only some components of an object that fulfill given qualification criteria. This kind of access is expected to be

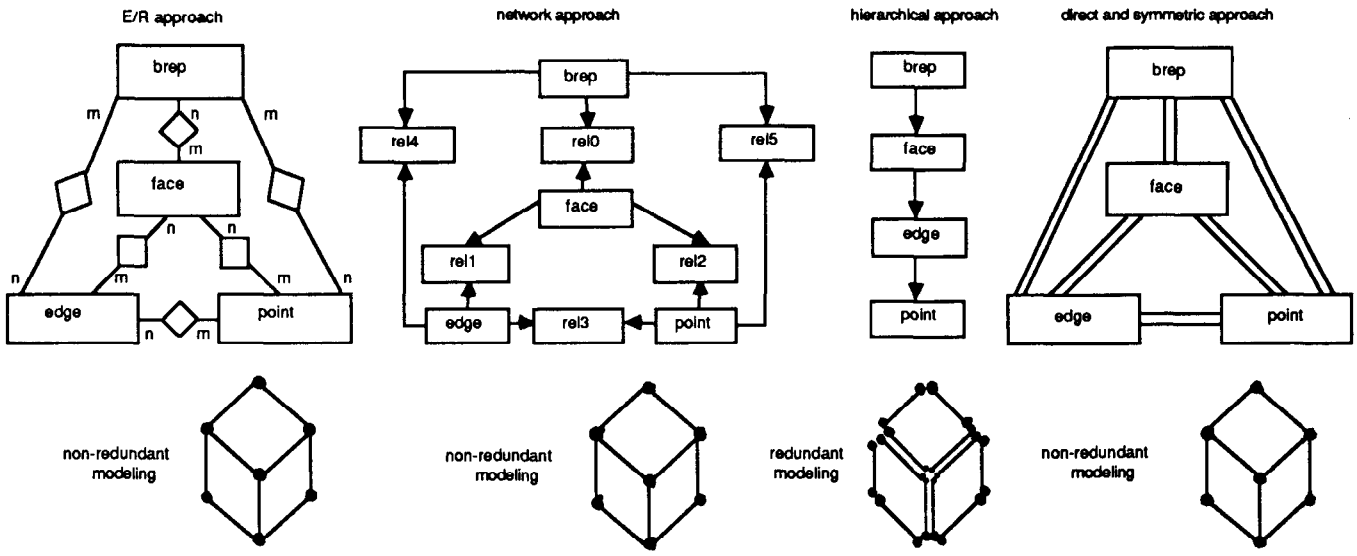


Figure 2.1: Modeling approaches to boundary representation

much more frequent compared to **horizontal access**. The latter derives all objects of a common type, i.e., accessing all stored maps, circuits or solids, perhaps satisfying some special qualification criteria.

Summarizing the arguments and considerations, we can definitely argue for the following essential data model requirements:

- direct and symmetric management of network structures and recursiveness
- dynamic object definition
- adequate object handling supporting both vertical as well as horizontal access.

Due to these requirements, an adequate data model has to provide

- support for molecular/complex objects, i.e. some kind of object-orientation comprising
  - modeling techniques describing the structure of an object as well as the object as an integral entity
  - operational semantic including object management
  - appropriate granularization of data and operations due to the composition/decomposition concept inherent to dynamic object definition and management (dynamic object handling)
  - support for more structural integrity (consistency in case of non-disjointness)
- in particular
  - support for vertical access with efficient derivation and assembling of the corresponding heterogeneous data or record sets, i.e. efficient record-type crossing operations (operative foreign/primary-key connections) in both directions (symmetry)
  - a descriptive language allowing for the processing of sets of heterogeneous records
  - a set-oriented embedding into the application program.

To satisfy these requirements, we have developed the MAD model. An important design goal was the consistent extension of processing homogeneous to processing heterogeneous record sets, defined by molecules. The basic mechanism is the **association** implemented by attributes containing logical pointers or references to other records (of the same or different type). These associations may be used to efficiently map n:m relationships and recursions. An association is symmetric in that the referenced record must contain a back-reference that can be used in exactly the same way.

The concept of **dynamic molecules** is based on such associations. Molecules are defined - in the query language, not in the schema - by naming the atom types and their associations. Any of the associations can be used to construct molecules. The molecule structure is superimposed dynamically on sets of atoms linked by associations, thus introducing the required object-orientation in the MAD model. Its operational power lies in adequate means for molecule processing, provided in MQL, which is similar to SQL [X3H286].

2.2 Key Properties of the MAD Model

In the following, we present a brief introduction of modeling as well as operational aspects of the MAD model. For this purpose, we use the example of Fig. 2.1 and the syntactical simplicity of MQL as an explanatory vehicle.

The objects the user has to deal with are called molecule occurrences, shortly **molecules**. Each molecule consists of more primitive molecules and belongs to its molecule type. This type determines both the molecule structure and the corresponding molecule set, grouping all the molecules with the same structure. Each molecule type is defined in terms of its component types. The most primitive molecules are called **atoms**. Each atom is composed of attributes of various types, has an identifier, and belongs to its corresponding atom type. The atom type is put together by the constituent attribute types to be chosen from a richer selection than in conventional data models. For identification and connection of atoms, we have introduced two special types of attributes, comprising the above mentioned association

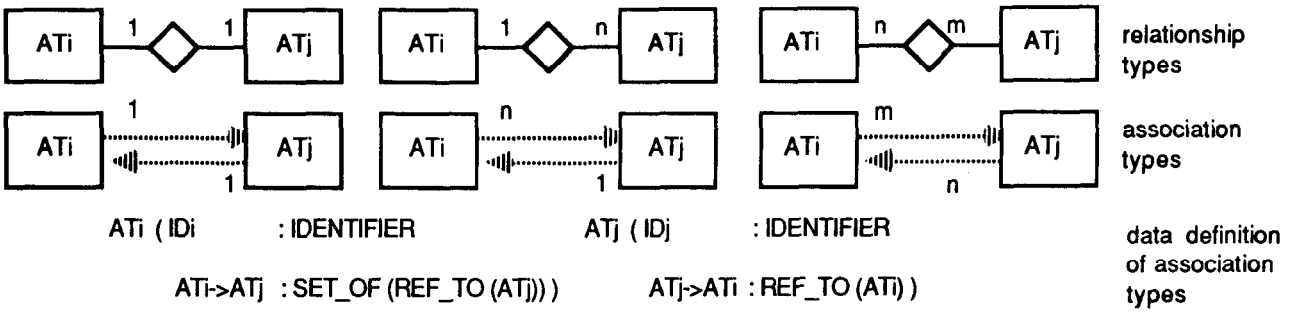


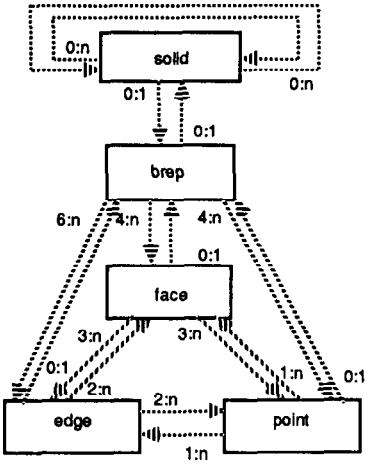
Figure 2.2: Expressing relationship types in terms of association types

concept. The IDENTIFIER type serves as a surrogate [ML83] which allows for the identification of each atom. Based on this attribute type it is easy to define the REFERENCE type allowing for typed references to other atoms. The extended type concept also includes RECORD, ARRAY, and the repeating-group types SET and LIST.

The three binary association types shown in Fig. 2.2. can be used to express all kinds of relationship types between two atom types, combining the attribute type REFERENCE with the repeating-group type SET. As an example, we have added the declaration of an 1:n association type. It is defined by two REFERENCE-based attributes (dotted arrows), one in each atom type. Transformation of an Entity-Relationship schema to an equivalent MAD schema is straightforward. Details of such a transformation and an example of the data definition language (DDL) are illustrated by Fig. 2.3. The transformation replaces all entity types by corresponding atom types and all relationship types by association types (which are built into the resp. atom types). The usefulness of the extended type concept and the cardinality restrictions that can be associated with the SET type (i.e. exact mapping of relationship types allowing for refined structural integrity enforced by the system) are also illustrated in Fig. 2.3.

Although molecules are generally defined as part of a query, it is allowed to give a name to often used molecule types (Fig. 2.3c). A molecule type definition specifies the molecule type name and the corresponding structure. In the

a) MAD schema diagram



b) atom type definitions

```
CREATE ATOM_TYPE solid
   ( solid_id     : IDENTIFIER,
     solid_no     : INTEGER,
     description  : CHAR_VAR,
     sub          : SET_OF (REF_TO (solid.super)),
     super        : SET_OF (REF_TO (solid.sub)) ,
     brep         : REF_TO (brep.solid)  )
     KEYS_ARE (solid_no)

CREATE ATOM_TYPE brep
   ( brep_id      : IDENTIFIER,
     brep_no      : INTEGER,
     hull         : HULL_DIM(3),
     solid        : REF_TO (solid.brep),
     faces        : SET_OF (REF_TO (face.brep)) (4,VAR),
     edges        : SET_OF (REF_TO (edge.brep)) (6,VAR),
     points       : SET_OF (REF_TO (point.brep)) (4,VAR)  )
     KEYS_ARE (brep_no)

CREATE ATOM_TYPE face
   ( face_id      : IDENTIFIER,
     square_dim   : REAL,
     border       : SET_OF (REF_TO (edge.face)) (3,VAR),
     crosspoint   : SET_OF (REF_TO (point.face)) (3,VAR),
     brep         : REF_TO (brep.faces)  )

CREATE ATOM_TYPE edge
   ( edge_id      : IDENTIFIER,
     length       : REAL,
     boundery     : SET_OF (REF_TO (point.line)) (2,VAR),
     face         : SET_OF (REF_TO (face.border)) (2,VAR),
     brep         : REF_TO (brep.edges)  )

CREATE ATOM_TYPE point
   ( point_id     : IDENTIFIER,
     placement    : RECORD
                      x_coord, y_coord, z_coord : REAL,
                    END
     line         : SET_OF (REF_TO (edge.boundery)) (1,VAR),
     face         : SET_OF (REF_TO (face.crosspoint)) (1,VAR)
     brep         : REF_TO (brep.points)  )
```

c) molecule type definitions

```
DEFINE MOLECULE TYPE   edge_obj
FROM   edge - point

DEFINE MOLECULE TYPE   face_obj
FROM   face - edge_obj

DEFINE MOLECULE TYPE   brep_obj
FROM   brep - face_obj

DEFINE MOLECULE TYPE   piece_list
FROM   solid.sub - solid (recursive)
```

Figure 2.3: Solid representation expressed in terms of the MAD-DDL

FROM-clause, the constituent molecule subtypes connected with the selected association types are listed (in case of ambiguity the reference attribute has to be denoted; for example see the definition of molecule type piece_list). Here it becomes apparent, that the MAD model supports - at least structurally - the concept of molecular objects to its full extend. To illustrate the operational support as well, we now focus on the molecule processing, i.e. query and data manipulation facilities. The syntax of MQL follows the examples of SQL [X3H286] and its derivates [PA86, RKB85].

Vertical access to a network structure is illustrated in the query of table 2.1a. First, all atoms constituting the brep molecules defined in the FROM-clause are assembled. This starts with the brep atoms and uses the associations to deduce the dependent face, edge, and point atoms. Then the (optional) WHERE-clause restricts this result set of molecules by evaluating qualification terms. In table 2.1b retrieval of a recursive structure is specified with piece_list as a pre-defined recursive molecule type (cf. Fig. 2.3c). Therefore, we first have to specify all roots of the desired recursive molecules, using the 'seed-qualification' predicate in the WHERE-clause. For all qualified root atoms (there is only one because the qualification of a key attribute is used), we have to evaluate the recursion in a stepwise manner going from one level to the next subordinate level using the solid.sub references.

Table 2.1c shows some kind of horizontal access: Here, we want to retrieve all primitive solids, i.e. solids not having any subpart-hierarchy. This query also shows the use of the projection expressed in the SELECT-clause. Some other important features are illustrated in table 2.1d. First, the FROM-clause shows the definition of a tree-like molecule type; branching (as well as combination, not exemplified here) is done using brace-expressions. The WHERE-clause includes a quantified qualification term testing for the existence of at least 2 edges that satisfy the length qualification. The ALL-quantifier could also be used as qualification term. The SELECT-clause describes the final result set of the whole query. Here, we use the so-called **qualified projection** for proper specification of the result set. Only those faces are finally selected, whose square_dim value satisfies the qualification. Exploiting this capability, we are able to retrieve only those components of a 'surrounding' molecule we are interested in.

Based on these powerful query facilities, we now sketch the remaining parts of molecule management: Analogously to retrieval capabilities, insert, delete, and modify operations allow for dealing with an integral molecule as well as its components. Modification especially supports connection and disconnection of molecule components. The delete statement reflects removal of single components as well as of whole component sets, thereby automatically disconnecting these parts from the specified surrounding molecules. The same holds for the insert statement inversely. Common to all manipulation operations is the system-enforced support for structural integrity, i.e., modifying a REFERENCE attribute implies the automatic maintenance of

a) vertical access to network molecules

```
SELECT  ALL
FROM    brep-face-edge-point
WHERE   brep_no = 1713                                  (* qualification *)
```

b) vertical access to recursive molecules

```
SELECT  ALL
FROM    piece_list                          (* pre-defined molecule type *)
WHERE   piece_list [0].solid_no = 4711        (* seed qualification *)
```

c) horizontal access combined with unqualified projection

```
SELECT  solid_no, description               (* unqualified projection *)
FROM    solid
WHERE   sub = EMPTY
```

d) miscellaneous query

```
SELECT  edge, (point,                       (* unqualified projection p1 *)
              face := SELECT face_id, square_dim
                      FROM   face       (* qualified projection q3, p2 *)
                      WHERE square_dim > 1.9E4)
FROM    brep-edge (face, point)
WHERE   brep_no = 1713                                  (* qualification q1 *)
        AND
        EXISTS_AT_LEAST (2) edge : edge.length > 1.0E2)
                                                  (* quantified restriction q2 *)
```

Table 2.1: Some hand-picked query examples

the corresponding back-reference.

Due to space limitations we cannot give a more detailed language discussion. An in-depth description of the MQL can be found in [Mi87]. It is apparent that the relational as well as the extended relational model [LK84] are just special cases of the MAD model. Equivalence or inclusion of the NF^2 model [SS86] is subject to further investigations.

Summarizing the above introduced concepts, the operational power of the MAD model, i.e. MQL, comprises:
- molecule insertion, deletion, modification, and retrieval by optionally using the FROM- and WHERE-clause
- component management, i.e. component insertion, deletion, modification, and retrieval by mandatory use of the FROM- and WHERE-clause to specify the surrounding molecules.

2.3 The LDL for 'Transparent' Performance Enhancements

Molecules specified at the MAD interface are built dynamically from atoms which are represented by record structures. To effectively support this dynamics, a variety of storage and access path structures are provided by PRIMA. 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 an appropriate set of storage structures supporting a given application.

For this purpose, we have defined a **load definition language** (LDL) used by the database administrator to provide

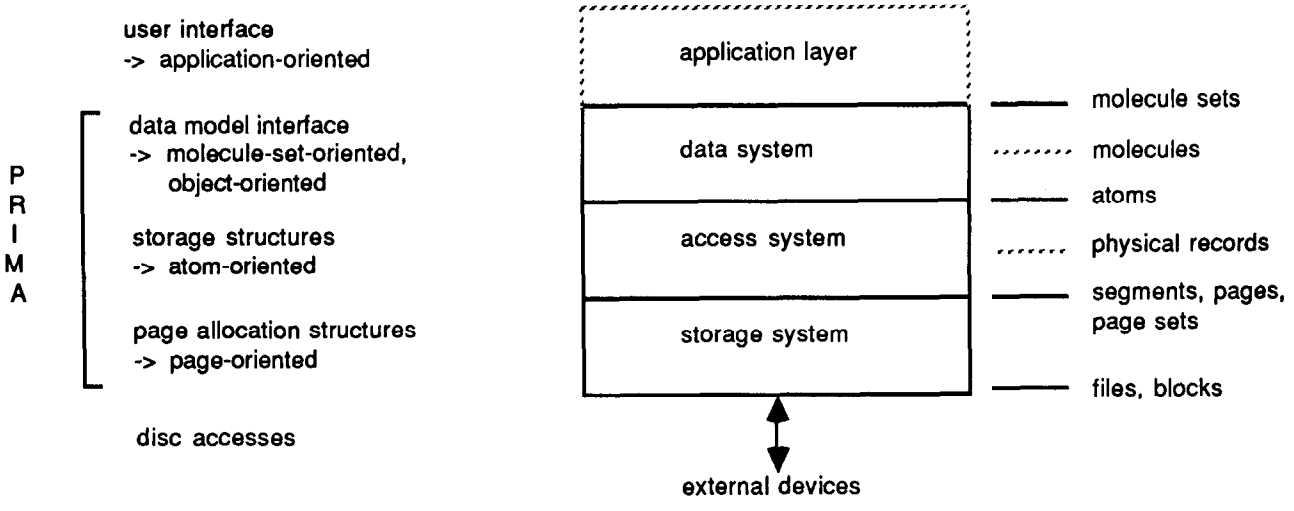


Figure 3.1 : Implementation model of PRIMA

some 'hints' for the access system (fig 3.1) which is responsible for the creation of appropriate storage structures, tailored access paths, and special tuning mechanisms. The main concepts for performance control are

- several access methods for one or more attributes permitting multidimensional access
- partitioning of physical records to improve clustering of frequently accessed attributes
- sort orders to speed up sequential processing according to given sort criteria
- 'physical cluster' to provide physical contiguity for atoms belonging to frequently requested molecules.

Depending on the processing needs, LDL may be used to introduce controlled redundancy in the storage structures (e.g. two different sort orders for the same object or physical clusters for objects in n:m relationships). Such measures only serve to improve performance - they are controlled by the access system and are not visible to the application referencing the MAD interface.

3. The Implementation Model

So far, we have outlined the features of the MAD model and its transparent support by application-dependent tuning mechanisms. In the following, we present an overview of the concepts and ideas used for its implementation.

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. 3.1 distinguishes three different layers for mapping molecules visible at the MAD interface onto blocks stored on external devices. In the following, we want to present the abstraction as well as processing capabilities of each layer, as far as the consecutive steps of data mapping in the multi-level hierarchy are concerned.

3.1 The Data System

The main task of the data system is to perform the complex mapping of the molecule-oriented interface onto the atom-oriented interface of the access system. This is done by translating the user-submitted MQL statements into an executable form (in terms of access system calls), while preserving their original meaning. The design of query translation and optimization is guided by extensibility and maintenance requirements, yielding a so-called modular data system [Fr86].

The **query validation and modification** checks the initial query for syntactic and semantic correctness, performs the resolution of predefined molecule types as well as the resolution of a meshed molecule type into an equivalent hierarchical one which is easier to cope with. Finally, it generates some internal representation of the query, i.e. the processing plan. The **query simplification** transforms qualified projections and nested query blocks into a symmetric query structure, if possible, and uses query partitioning otherwise [Ki82]. Finally, **query preparation** creates a finer grained processing plan adding functional descriptors for sorting, duplicate elimination, evaluation of qualified projection, molecule join as well as recursion. Here, we have to deal with the optimization of molecule join and recursion thereby exploiting information from the meta-data, join strategies, and different strategies solving recursion.

A one-molecule-at-a-time interface is provided by the **molecule management**. It delivers all molecules of a specified 'simple' (non-recursive) molecule type by offering a molecule-type-scan facility. A molecule-type-specific optimization has to be aware of access methods, sort orders, partitions of atom types, and physical clusters. Finally, molecule processing has to cope with cursor management and cluster management, hiding the underlying access system interface. It deals with searching the qualified parts of the desired molecule and combining these parts, while performing 'simple' projections and

qualifications 'pushed down' for efficiency reasons. Cursor management and cluster management allow for projected and qualified scanning the resp. types.

3.2 The Access System

The access system offers - like the Research Storage System (RSS) of the System R prototype [As76] - an atom-oriented interface which allows for retrieval and update of single atoms [Si87]. 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). An update operation on a reference attribute thus includes implicit update operations on other atoms to adjust the appropriate back-reference attributes.

Update operations and direct access are restricted to atoms identified by their logical address. A **logical address** (or surrogate [ML83]) is used to implement the IDENTIFIER attribute as well as the REFERENCE attributes. It is generated by the access system when an atom is inserted, and it is released when the atom is deleted.

When inserting an atom, values are assigned to all or only selected attributes. Accordingly, it is allowed to modify only some attributes of an atom (excluding the logical address) and to select attributes when reading an atom. The projection of frequently used attributes may be supported by means of partitions, i.e. separate storage of attribute combinations. This is one of the tuning mechanisms triggered by the LDL.

The other tuning mechanisms - access paths, sort orders, and clusters - are implemented in the access system, too. While access paths and sort orders are known from conventional DBMS, the concept of clustering to support molecule processing shows new aspects. In order to speed up construction of frequently used molecules, we introduce the concept of **atom clusters**. They serve to allocate in physical contiguity all atoms of the 'main lanes' to be traversed during molecule derivation. These clusters may be installed via LDL commands; the access system is in charge of maintaining this selective redundancy and of guaranteeing all related issues of consistency.

An atom-cluster type is declared by naming the atom types whose atoms are to be clustered. Such an atom cluster corresponds mostly to a heterogeneous and sometimes to a homogeneous atom set defined 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. 3.2a). Inserting a characteristic atom generates a new atom cluster consisting of the characteristic atom and all atoms referenced by it. Modifying a characteristic atom adds new atoms to an atom cluster and deletes old ones whereas deleting a characteristic atom deletes a whole atom cluster.

All tuning mechanisms - atom clusters as well as access paths, sort orders, and partitions - generate additional storage structures which materialize homogeneous or heterogeneous result sets. For example, an atom 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. Such a redundant structure - specified by an LDL statement - may be generated and dropped at any time.

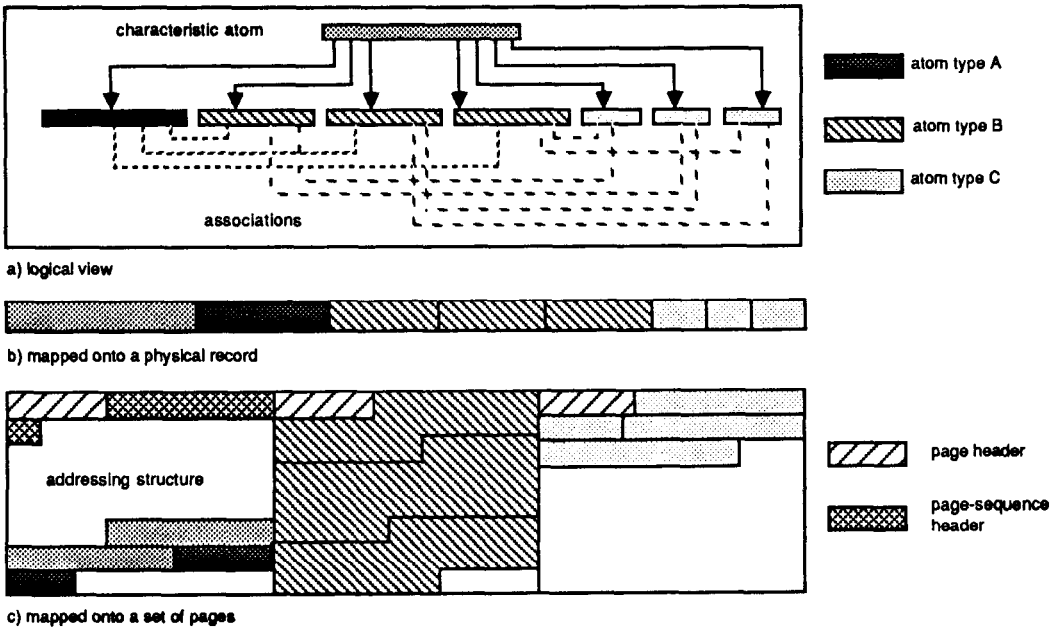


Figure 3.2: Atom cluster

To manage redundancy in the access system, **physical records** are introduced as byte strings of variable length. They are stored consecutively in 'containers' offered by the storage system. Depending on the storage structure, a physical record corresponds to either a part of an atom (a partition), an entire atom (in a sort order) or an atom cluster (Fig. 3.2b). This establishes an n:m relationship between atoms and physical records, whereas the usual mapping of conceptual to internal schema is built on a 1:1 relationship. A sophisticated addressing structure is required to manage such n:m relationships [Si87].

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.

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 accept single atoms (NEXT/PRIOR) for further processing.

The simplest of these scans is the **atom-type scan**. It successively reads all atoms of one atom type in a system-defined order - either as a whole or only selected attributes. In addition, the result set of the scan can be restricted by a simple search argument decidable on each atom. Hence, the atom-type scan corresponds to the relation scan of the RSS [As76].

Unlike the atom-type scan, the **sort scan** serves to read all atoms of one atom type in a 'user'- defined order according to a specified sort criterion. In this case, the result set can be restricted by a simple search argument as well as a start/stop condition. Since sorting an entire atom type is expensive and time consuming, the sort scan may be supported by a redundant storage structure, the **sort order**. It consists of a sorted list of physical records, one for each atom of the resp. type. But the sort scan also works without such a sort order. It may engage an access path if available, or has to perform the sort explicitly creating a (temporary) sort order.

A main usage of scans is on access paths where start and stop conditions conveniently provide access to value ranges and where value orders may be exploited for free (**access-path scan**). Since we offer multi-dimensional access path structures, the effect of key-sequential accesses needs some explanation. Linear orders based on B*-trees only allow sequential NEXT/PRIOR traversal. With n keys, navigation has much more degrees of freedom. Therefore, start/stop conditions and directions may be specified individually for every key involved in the scan; hence, the user - the data system - determines the selection path for elements in an n-dimensional space.

Whereas the first three scan operations support only horizontal access to a homogeneous atom set belonging to one atom type, the last two scan operations allow for the vertical access to a heterogeneous atom set across several atom types. The **atom-cluster-type scan** reads all characteristic atoms of an atom-cluster type in a system-defined order, possibly restricted by a simple search argument which now has to 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.

3.3 The Storage System

As in conventional systems the objects, i.e. containers, offered by the storage system [Si87] are segments divided into pages of equal size. In contrast to them, the storage system of PRIMA supports pages of different length. The page size of each segment can be chosen to be $^1/_2$, 1, 2, 4 or 8 Kbyte. The number of page sizes is restricted to these five values for two reasons. The first one is due to the file manager of the underlying operating system [Ne87], it supports exactly these block sizes. Hence, mapping between blocks and pages is very simple. The second reason refers to the problems arising from the management of the database buffer. As the existing replacement algorithms (LRU, etc. [EH82]) are only tailored to one page size, new approaches concerning the management of different page sizes within one buffer are necessary. One way is the division of the buffer into several independent parts, each of which managed by a dedicated replacement algorithm. Such a static partitioning is not very flexible when reference patterns change. Another possibility is to develop or modify a replacement algorithm in such a way that it can handle different page sizes. This idea has been pursued in the storage system, i.e., the well-known LRU algorithm was altered in an appropriate way [Si87]. Hence, we provide at least limited set-orientation when accessing blocks on disks.

The five page sizes, however, do not meet the most important requirement of the access system concerning containers of arbitrary length. The restriction to a certain page size, say 8 Kbyte, is too stringent, especially considering atom clusters and strings like texts and images. Therefore, the storage system offers at its interface **page sequences** as additional containers. A page sequence treats an arbitrary number of pages as a whole. One of these pages is the so-called header page, all others are component pages. The header page contains the

usual page header used for identification, description, and fault tolerance, and a page sequence header, i.e. a list of all pages belonging to the appropriate page sequence. A page sequence 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.

The mapping of an atom cluster onto such a page sequence is shown in Fig. 3.2c. An auxilliary addressing structure [Si87] - together with the page-sequence header - provides relative addressing within the page sequence thereby achieving faster access to single atoms of the atom cluster.

4. Conclusions and Future Plans

We have presented our design of the MAD model and its implementation by a DBMS kernel. The focus of the paper has primarily been on justifying the design decisions and on discussing the major features instead of providing a detailed and complete description of the data model and the kernel system.

For the data model, we have advocated a symmetric and neutral approach allowing for more powerful and complex constructs as the flat relational model, but avoiding the bias on static data structuring and top-down traversal of hierarchical models. Its main philosophy is the ability to dynamically construct molecules using atoms as elementary building blocks which may be conceived as a dynamic view mechanism for complex objects. Symmetric representation of all relationships (including n:m) and derivation of complex objects at run time are considered prime prerequisites to accurate and effective modeling in engineering applications where the 'view' of the object frequently changes.

A number of concepts used in the PRIMA implementation pays attention to DBMS performance requirements. Most important are the facilities of the load definition language which are transparent at the MAD interface. They provide a variety of access paths, redundant sort orders, partitioning of records, and physical clustering to support efficient molecule construction. Kernel controlled redundancy may be introduced to further improve frequent types of operation.

Currently, the single-user version of PRIMA is being finished. For our purpose, 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. Here, we can only present a brief overview of its prospective usages:

- The conceptually simplest system structure would be obtained by using PRIMA without additional components as a 'complete' DBMS. The services at the MAD interface are directly made available to its users. Hence, the particular application itself has to refer to its 'neutral' object-oriented interface - providing access to molecules (sets of heterogeneous records) - to construct (more) application-specific objects in order to facilitate object management for its higher program levels.
- Since application objects require quite complex mapping functions identical or similar for an entire class of applications, e.g. 3D-CAD, it might be a good idea to extract such mapping functions from each particular application program and to provide a 'standardized' interface for the entire application class. As already indicated in Fig. 3.1, we consider such class-specific extensions as our main concept to derive application-oriented objects under DBMS control. Hence, a variety of application layers (AL) as the top-most DBMS layer may be designed tailoring PRIMA services to application classes to be supported.
- Effective workstation-host coupling is a prime requirement for interactive engineering applications. Hence, the architectural separation of PRIMA and AL may be exploited by allocating AL close to the application in the corresponding workstation. The set-oriented MAD interface is a major prerequisite to reduce communication overhead as far as possible. Locality of reference is enhanced by integrating an application-related object management into AL. Large buffer sizes may help to perform most of the DBMS work locally, after the required molecules are transferred to an 'object buffer' (checkout). Ideally, modified or newly created molecules are moved back to PRIMA at commit time (checkin) [HHMM87, KLMP84].
- In an architecture dedicating only a single processor to PRIMA, DBMS processing may apparently become a system bottleneck as the number of applications increases. Therefore, multi-processor architectures should be investigated to enhance processing power of the DBMS kernel.
- Engineering applications with their 'sizable' operations on complex objects incorporate substantial 'portions of inherent parallelism' [HHM86] which may not be exploited when such operations are synchronously invoked and serially executed - in the traditional manner. To overcome these 'unnecessary' restrictions, we have defined the concept of **semantic decomposition**: units of work decomposed from a single user operation are said to allow for inherent semantic parallelism when they do not conflict with each other at the level of decomposition. Such decomposed units of work (DU's) may be scheduled and executed concurrently by the DBMS - given an appropriate processing architecture. Here, we can only point to this particularly important usage of multi-processor PRIMA which embodies our research vehicle to investigate DBMS parallelism within a single user operation and its exploitation.

Due to space restrictions it is not possible to discuss other important aspects of dynamic system behavior in sufficient depth. Since transaction execution is distributed across AL and potentially multiple PRIMA processors, we need a distributed control structure to keep track of transaction dynamics and to ensure transaction atomicity. Therefore, a flexible transaction concept is mandatory which should also focus on fine grained intra-transaction parallelism and selective in-transaction recovery in various failure events. We have decided to refine the concept of **nested transactions** [Mo81] as a generic mechanism for all

proposed uses of PRIMA. A detailed description of the transaction concept including application-structuring for long transactions [KLMP84] and dynamic control structures across AL-PRIMA processes as well as its distributed implementation is prepared in a subsequent paper.

5. 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.

CD87 Carey, M.J., DeWitt, D.J., et al.: The Architecture of the EXODUS Extensible DBMS, in: Proc. Int. Workshop on Object-Oriented Database Systems, Pacific Grove, 1986, pp. 52-65.

Da86 Dadam, P., et al.: A DBMS Prototype to Support Extended NF^2-Relations: An Integrated View on Flat Tables and Hierarchies, in: Proc. ACM SIGMOD Conf., Washington, D.C., 1986, pp. 356-367.

DB83 several papers in: Proc. of the Engineering Design Applications at the Data Base Week, 1983.

DPS86 Deppisch, U., Paul, H.-B., Schek, H.-J.: A Storage System for Complex Objects, in: Proc. Int. Workshop on Object Oriented Database Systems, Pacific Grove, 1986, pp. 183-195.

EH82 Effelsberg, W., Härder, T.: Principles of Database Buffer Management, in: ACM TODS, Vol. 9, No. 4, 1984, pp. 560-595.

Fr86 Freytag, J.C.: A Rule-Based View of Query Optimization, IBM Almaden Research Center, San Jose, CA, Sept. 29, 1986.

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, SFB 124, Research Report No. 24/87, Univ. of Kaiserslautern, 1987 (submitted for publication).

HL82 Haskin, R.L., Lorie, R.A.: On Extending the Functions of a Relational Database System, in: Proc. ACM SIGMOD Conf., Orlando, Florida, 1982.

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.

KLMP84 Kim, W., Lorie, R., McNabb, D., Plouffe, W.: Nested Transactions for Engineering Design Databases, in: Proc. 10th VLDB Conf., Singapore, 1984, pp. 355-362

Ki82 Kim, W.: An Optimizing and SQL-like Nested Query, in: ACM TODS, Vol. 7, No. 3, 1982, pp. 443-469.

LK84 Lorie, R., Kim, W., et al.: Supporting Complex Objects in a Relational System for Engineering Databases, IBM Research Laboratory, San José, CA, 1984.

LMP86 Lindsay, B., McPherson, J., Pirahesh, H.: A Data Management Extension Architecture, IBM Almaden Research Center, San José, CA, 1986.

Mc77 McGee, W.C.: The Information Management System IMS/VS, in: IBM Systems Journal, Vol. 16, No. 2, 1977, pp. 84-168.

Mi87 Mitschang, B.: MAD - a Data Model for the Kernel of a Non-Standard Database System (in German), in: Proc. of the GI Conf. on 'Database Systems for Office, Engineering and Science Environments', 1987, pp. 180-195.

ML83 Meier, A., Lorie, R.: A Surrogate Concept for Engineering Databases, in: Proc. 9th VLDB Conf., Florenz, 1983, pp. 30-32.

Mo81 Moss, J. E. B.: Nested Transactions: An Approach to Reliable Computing, M.I.T. Report MIT-LCS-TR-260, M.I.T., Laboratory of Computer Science, 1981.

Ne87 Nehmer, J., et al.: Key Concepts of the INCAS Multicomputer Project, accepted for IEEE Transactions on Software Engineering, 1987.

PA86 Pistor, P., Anderson, F.: Designing a Generalized NF^2 Data Model with a SQL-Type Language Interface, Proc. 12th VLDB Conf., Kyoto, 1986.

PSSWD87 Paul, H.-B., Schek, H.-J., Scholl, M.H., Weikum, G., Deppisch, U.: Architecture and Implementation of the Darmstadt Database Kernel System, accepted for SIGMOD87.

RKB85 Roth, M.A., Korth, H.F., Batory, D.S.: SQL/NF: A Query Language for ¬1NF Relational Databases, Deptm.Comp.Sciences, Univ. of Texas at Austin, TR-85-19, 1985.

Si87 Sikeler, A.: Access and Storage System of PRIMA (in German), SFB 124, Research Report, Univ. of Kaiserslautern, in preparation.

SR86 Stonebraker, M., Rowe, L.A.: The Design of POSTGRES, in: Proc. ACM SIGMOD Conf., Washington, D.C., 1986, pp. 340-355.

SS86 Schek, H.-J., Scholl, M.H.: The Relational Model with Relation-Valued Attributes, in: Information Systems, Vol. 2, No. 2, 1986, pp. 137-147.

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