Object Orientation within the PRIMA-NDBS

C.Hübel, B.Mitschang
University Kaiserslautern
P.O. Box 3049
6750 Kaiserslautern
West-Germany

1. Overview

In the following we wish to highlight the design and implementation concepts of a non-standard database system (NDBS) called PRIMA and its provision for object orientation.

Non-standard database applications such as CAD/CAM, VLSI design, and knowledge-based systems require adequate modeling facilities for their application objects for various reasons [Hä88b]. Data models dedicated to supporting such applications embody some degree of object orientation to the application objects [DD86]. In this context, the notion of complex object has emerged to indicate that such an object holds an internal structure and that access to the entire object as well as to its components (which may again be of a complex type) is provided by means of generic operators - in this sense we speak of *structural object orientation.* To enhance integrity control and semantic as well as operational expressiveness, more object properties (beyond the above introduced structural relationships) have to be specified and preserved - this extended notion of object orientation is also referred to as *behavioral object orientation.* By this means appropriate forms of data abstraction and operational support are provided, thus relieving the application from the burden of

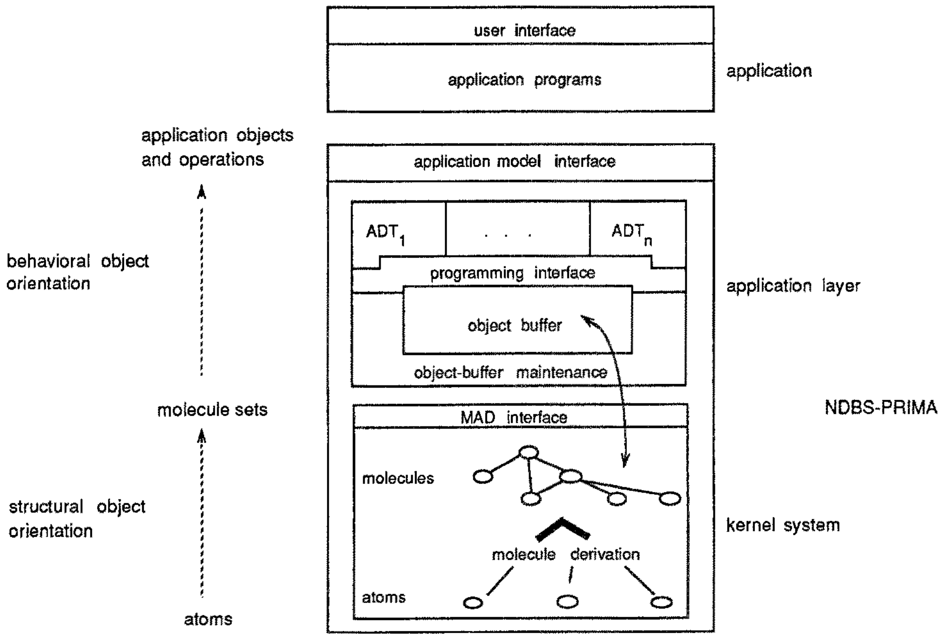


Fig.1: The PRIMA architecture and its provision of object orientation

maintaining and manipulating intricate object representations as well as checking complex integrity constraints.

The Kernel Architecture

The PRIMA-NDBS is a database management system tailored to the support of non-standard applications. As a key idea, its underlying architecture is divided into two parts (Fig. 1):

- The so-called NDBS-kernel offers neutral data management functions embodying structural object orientation. The kernel interface is determined by the MAD model (molecule atom data model) and its language called MQL (molecule query language), which offers composition and decomposition of complex objects, here called molecules [Mi88].
- The application layer provides application-specific support, i.e. behavioral object orientation offered by ADTs. Complex objects offered by the kernel are used and tailored to objects according to the application model of a given application. This mapping is specific for each particular application area. Hence, different application layers exist which offer tailored interfaces in the form of a set of ADT operations for the corresponding application.

In the following, we want to concentrate on the essential characteristics of the MAD model and the most important objectives of the application layer. Both topics are refined with special emphasis on their provision for structural and behavioral object orientation.

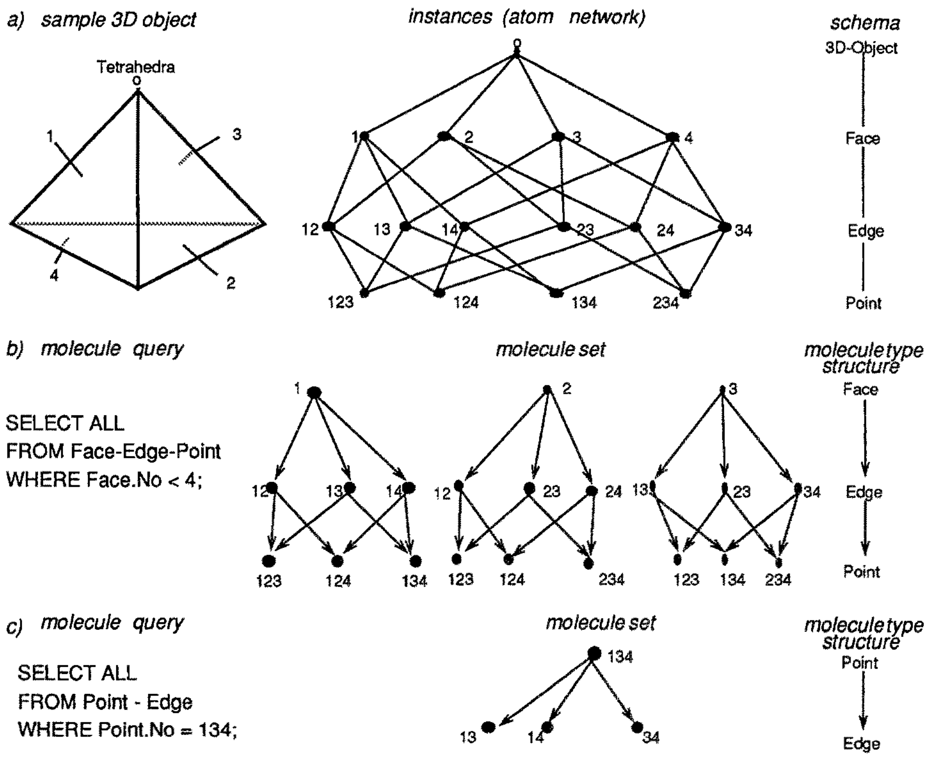
2. Structural Object Orientation enabled by the MAD Model

The most important objectives of the MAD model can be characterized as follows:

- complex objects viewed as structured sets of elementary building blocks
- they are dynamically defined and derived
- all relationships among elementary building blocks are represented in a direct and symmetrical way allowing for shared subobjects
- set-oriented processing is offered by the descriptive SQL-like query language MQL.

Atoms and Links as Basic Building Blocks

Atoms are the basic elements of the MAD model used to represent the real world entities. They play a similar role to tuples in the relational model. Each atom is composed of attributes of various types, is uniquely identifiable, and belongs to its corresponding atom type. The attributes' data types can be chosen from a richer selection than in conventional models yielding a more powerful structuring capability at the attribute level. Relationships between atoms (entities) are expressed by so-called links that are defined as link types between atom types. Links are used to efficiently map all types of relationships and recursions. This direct mapping and the consideration of bidirectional yet symmetric links (represented via reference/back-reference pairs) establishes the basis of the MAD model's flexibility. Hence, in the database all atoms connected by links form meshed structures as depicted in Fig. 2a. Based on this atom network, it is feasible to dynamically construct molecules using atoms as elementary building blocks.



```
SELECT ALL
FROM Face-Edge-Point
WHERE Face.No < 4;
```

```
SELECT ALL
FROM Point - Edge
WHERE Point.No = 134;
```

Fig. 2: Dynamic construction of molecules from a sample geometric object

Dynamic Derivation of Molecules

Molecules are defined by MQL statements and have to be derived at run-time (Fig. 2). Each molecule belongs to a molecule type specified in the FROM clause. This type description establishes a connected, directed, and acyclic type graph as subgraph of the database schema. This graph becomes cyclic only when recursive types are involved. Thus, each type description specifies a starting point (i.e. root atom type) and all participating atom and link types (abbreviated by '-'). A simple example depicted in Fig. 2b is the molecule type Face-Edge-Point. Each molecule type determines both the molecule structure as well as the molecule set which groups all molecules with the same structure. At least at the conceptual level, the dynamic derivation of molecules proceeds in a straight-forward way using the type graph as a kind of template: For each atom of the root atom type all children, grandchildren etc. are connected according to the molecule type structure terminating after all leaves have been reached. The derivation of the children atoms means performing the hierarchical join [LK84]. Fig. 2b shows the resulting molecule set for the Face-Edge-Point molecule type, where the set of molecules was restricted by means of the WHERE clause. The SELECT clause defines the final molecule (type) structure and allows a proper molecule projection. The symmetric use of the atom network is shown with the help of the example in Fig. 2c depicting a Point-Edge molecule query and its corresponding result.

In toto, the MAD model offers adequate concepts gaining structural object orientation. The prototype implementation of the MAD model, i.e. the PRIMA kernel [HMMS87], supports efficient molecule processing which is additionally enhanced by a variety of storage structures and tuning mechanisms (e.g. molecule materialization or molecule caching instead of its dynamic derivation). All these performance enhancements are transparent at the data model interface. An in-depth description of the design and implementation concepts of PRIMA can be found in [Hä88a].

3. Issues of the Application Layer

The application layer (AL) constitutes the topmost layer of our NDBS architecture as illustrated in Fig. 1. The main task of the AL lies in the tailoring of the NDBS towards a specific application to achieve the desired behavioral object orientation. Hence, the AL constitutes the linkage between the application-independent NDBS-kernel system and the application at hand. Its interface is also called the application model interface emphazising that the AL is carrying out application-oriented objects and their associated operations. The desired application model is achieved by mapping the application's objects to the molecules offered by the underlying MAD model, and by including special algorithms for the application specific molecule processing. These algorithms describe implicitly the application objects' behavior including their integrity constraints, since they determine the object modification initialized via actions in the application. This encapsulation of data (molecules) and algorithms results in a kind of abstract data type (ADT). Therefore, the AL is organized as a collection of ADTs. AL programs constitute the ADT operations, and ADT instances are represented or defined by references to the molecules.

Embedding of Molecules

First of all, we have to discuss the question of how to embed the molecules (and atoms), which are specified by MQL-queries and delivered by the NDBS-kernel system, into the ADT environment. Hence, embedding means that MAD objects are made directly accessible for AL programs. For this reason, the query's result is stored in a specific part of main memory, the so-called object buffer. This buffer enables the utilization of the access locality of AL programs. To further enhance the processing performance, fast operational access is achieved via a pointer-like reference mechanism. To support the imperative/procedural processing of the complex and network-like data structures stored in the object buffer, we introduce a cursor concept which allows us to define cursors explicitly, to bind them to parts of the atom network, and to move them from one atom to the other in a navigational manner. Since navigational processing of hierarchical molecule components are often required, we allow for the definition of dependent 'flat' cursors yielding a hierarchical cursor concept. Thus, navigation via one 'flat' cursor automatically affects the dependent cursors on subordinate levels.

The object buffer and the cursor concept deliver access to preselected molecule structures. A further question, which concerns the embedding of MAD objects deals with data structure adaptation. That is, how do the AL programs, written in conventional programming languages, 'assimilate' atoms which usually contain attributes of MAD specific data types. This problem is well known as the problem of language binding in the conventional DBMS environment, and multiple solutions are given in the literature. Our own proposal provides the use of a special precompiler, which handles the inclusion of MAD statements, enriches the underlying host language in order to handle MAD specific data types, and additionally integrates particular language constructs to support the navigational and cursor-oriented access concepts mentioned above.

Language Binding Via Precompiler

The use of precompiler statements is sketched in Fig. 3. It depicts the scheme of the sample AL program 'solid_geometry' in a PASCAL-like programming language. We distinguish between the definition of types (query types which contain a MAD statement specification, and cursor types) and the declaration of multiple corresponding instances. These instances are handled like host language variables. The query variable represents the query's result stored in the associated object buffer. The linkage between such a variable and the result of a particular query is done via the built-in function EVAL. EVAL directs actual query parameters to the NDBS-kernel interface and triggers the query evaluation and the result's preparation. The inverse semantics is carried out by the PROPAGATE function. This function propagates all modifications on buffered molecules back to the kernel system. Hence, EVAL and PROPAGATE constitute a kind of check-out/check-in mechanism controlling the contents of the object buffer. Similar to the definition of object buffers (query variables), the ADT programmer can generate cursor instances. The binding of such a cursor variable to an object buffer is provided by the ATTACH function.

The illustrated AL-program gives an example of ADT specification, too. Obviously, it is divided into two parts:

```
ADT  solid_geometry;
QUERY TYPE
     solid_type = 'SELECT ALL
                FROM 3D-Object-Face-Edge-Point
                WHERE 3D-Object.No = $no'
                (no: identifier type);
     ...
CURSOR TYPE
     face_curor_type = Face, Edge, Point -> solid_type;
     edge_cursor_tupe = Edge, (Face, Point) -> solid_type;
     ...
VAR
     solid_buffer : solid_type;
     face_cursor : face_cursor_type;
     edge_cursor : edge_cursor_type;
     ...
     volume : integer;
     ...
```

```
OPERATION
...
activate (number, solid)
   BEGIN EVAL   (solid.solid_buffer, ... number ...);
         ATTACH(solid.face_cursor, solid.solid_buffer);
         ATTACH(solid.edge_cursor, solid.solid_buffer);
         ...
         calculation of the solid's volume
         ...
   END;
   ...
move (solid):
...
union (solid1, solid2, solid);
...
END solid geometry
```

Fig. 3: Data part and operation part of a sample ADT for solid geometry

- the data part determines the type of 'abstract data' (e.g. the ADT-instances) by a collection of type definitions.
- the operation part contains application independent operations, such as 'generate', 'activate', 'deactivate', and 'delete', as well as application-specific operations (e.g. 'move', 'rotate', 'union', 'intersect' and 'difference').

4. Summary and Outlook

At first glance, the PRIMA-NDBS looks like a synopsis of quite different aspects. But in fact these system concepts fortify each other in a very fruitful way resulting in an efficient database support for advanced applications. In the following, the main concepts and their achievements are listed:

- the MAD model at the kernel interface provides structural object orientation
- the object buffer exploits 'near by application' locality,
- the cursor concept allows for efficient pointer-like access capabilities, and
- the ADT concept offers behavioral object orientation.

Furthermore, a number of system design decisions were taken with special emphasis on workstation-oriented environments, which in our opinion are the most realistic and prevailing ones. Workstation-oriented processing could be effectively enhanced by delegating the application layer together with the application programs to the workstation and the NDBS-kernel to the host system. This partitioning is further favored by the set-orientation of the kernel interface minimizing workstation-host communication.

At the moment, we are performing a broad 'in-the-field' validation of these system concepts in the areas of VLSI design and 3D modeling. On the other hand, we are looking for further enhancements achieving intelligent CAD, planning and diagnosis support. Here we consider the integration of ideas known from several knowledge representation methods.

Acknowledgements
The design and implementation of the PRIMA-NDBS was a combined effort of the members of the PRIMA project superintended by T. Härder.

Literature

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

Hä88a Härder, T.: The PRIMA Project - Design and Implementation of a Non-Standard Database System, Research Report No. 26/88 SFB124, University Kaiserslautern, 1988.

Hä88b Härder, T.: Non-Standard DBMS for Support of Emerging Applications - Requirement Analysis and Architectural Concepts, submitted for publication.

HMMS87Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - a DBMS prototype Supporting Engineering Applications, Research Report No. 22/87 SFB124, 1987, in: Proc. VLDB 87, pp. 433-442.

HS88 Hübel, Ch., Sutter, B.: Experiences in Supporting an Engineering Application by PRIMA, in: The PRIMA Project - Design and Implementation of a Non-Standard Database System, Härder, T. (ed.), Research Report No. 26/88 SFB124, University Kaiserslautern, 1988.

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

Mi88 Mitschang, B.: Towards a Unified View of Design Data and Knowledge Representation, in: Proc. 2nd. Int. Conf. on Expert Database Systems, Tysons Corner, Virginia, April 1988, pp. 33-49, further publ. by Benjamin/Cummings Publ. Co.