problem which need to be solved before a reasonable DBMS support may be expected in such application areas.

To attack the modeling problem we have advocated the use of a data model with structural object orientation. Performance requirements dictate its efficient operation by exploiting cluster and caching techniques as well as specialized access paths. The data model implemented by the DBMS kernel is complemented by mechanisms to achieve behavioral object orientation in the application layer.

The processing problem is approached by a number of architectural issues: Effective data supply of the workstation, perfect locality of reference by keeping all objects under design in an object buffer as well as local failure handling are the most important measures. These operational concepts are tied together by design transactions which are tailored to the needs of an interactive environment.

First practical experiences have confirmed our design decisions concerning data model and NDBS architecture. We have reimplemented the 3D-CAD as well as the 2D-CAD applications (see sect. 2.2 and 2.3). First observations indicate that only a few DML operations are necessary to fetch and to propagate the objects of a design transaction while the bulk of the workload is executed in the object buffer. As a consequence, the elapsed time was reduced by a factor of 50-100 as compared to the initial approach. Optimization and adjustment of architectural concepts may further improve the performance.

Acknowledgements

The requirement analysis of advanced DBMS applications was a combined effort of the members of the PRIMA project. Many project discussions clarified and shaped the proposed data model and architecture concepts. B. Mitschang has carefully read a draft of this paper. The comments of the referees are greatfully acknowledged; their suggestions greatly improved the presentation of the paper.

6. References

[1] Härder,T., Reuter, A.: Database Systems for Non-Standard Applications, in: Proc. Int. Computing Symposium 1983, H.J. Schneider (Ed.), B.G. Teubner Stuttgart, Nürnberg 1983, pp. 452-466.

[2] Dittrich, K.R., Dayal, U. (eds.): Proc. Int. Workshop on Object-Oriented Database System, Pacific Grove, 1986.

[3] Special Issue on Engineering Design Databases, IEEE Database Engineering, Vol. 7, No. 2, June 1984.

[4] Banerjee, J. et al.: Data Model Issues for Object-Oriented Applications, in: ACM TOOIS, Vol. 5, No. 1, 1987, pp. 3-26.

[5] Requicha, A.A.G., Voelcker, H.B.: Solid Modelling: A Historical Summary and Contemporary Assessment, in: IEEE Computer Graphics and Applications, Vol. 2, March, 1982, pp. 9-24.

[6] Chen, P.P.: The Entity-Relationship-Model - Toward a Unified View of Data, in: Proc. ACM TODS, Vol. 1, No. 1, 1976, pp. 9-36.

[7] 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.

[8] 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.

[9] Haas, L.M. et al.: An Extensible Processor for an Extended Relational Query Language, IBM Research Report RJ 6182, San Jose, CA, April 1988.

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

[11] Wedekind, H., Zörntlein, G.: A conceptual basis for database applications in flexible manufacturing systems, in: Proc. 1987 IEEE Int. Conf. on Robotics and Automation, Raleigh, NC, March 1987, pp. 551-557.

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

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

[14] Maier, D. et al.: Development of an object-oriented DBMS, in: Proc. ACM Conf. on Object-Oriented Programming Systems, Languages, and Applications, Portland, Oregon, 1986.

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

[16] Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - a DBMS Prototype Supporting Engineering Applications, Forschungsbericht Nr. 22/87 des SFB 124, Universität Kaiserslautern, 1987, in: Proc. VLDB 87, S. 433-442.

[17] Rosenthal, A. et al.: Query Facilities for Part Hierarchies: Graph Traversal, Spatial Data, and Knowledge-Based Detail Supression, Research Report, CCA, Cambridge, Ma. 1987.

[18] Mitschang, B.: Towards a unified view of design data and knowledge representation, in: Proc. Second Int. Conf. on Expert Database Systems, Tysons Corner, Virginia, April 1988, pp. 33-49.

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

[20] Bancilhon, F., Kim, W., Korth, H.F.: A Model of CAD Transactions, in: Proc. 11th Int. Conf. on VLDB, Stockholm, Aug. 1985, pp. 25-33.

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

[22] Härder, T., Hübel, Ch., Meyer-Wegener, K., Mitschang, B.: Processing and Transaction Concepts for Cooperation of Engineering Workstations and a Database Server, in: Data and Knowledge Engineering, 1988.

[23] 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.

[24] Härder, T., Rothermel, K.: Concepts for Transaction Recovery in Nested Transactions, in: Proc. ACM SIGMOD'87 Conf., San Francisco, May 1987, S. 239-248.

[25] Wilms, P.F., Schwarz, P.M., Schek, H.-J., Haas, L.M.: Incorporating Data Types in an Extensible Database Architecture, in: Proc. 3rd Int. Conf on Data and Knowledge Bases, Jerusalem, 1988.

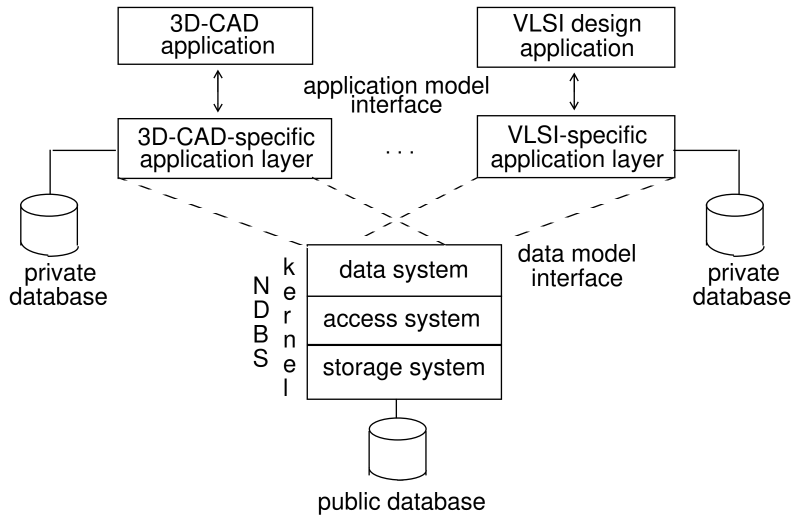


Fig. 6: Framework of an NDBS architecture

4.3 Processing in the application layer

In order to obtain 'near by the application' locality, a carefully designed processing model has to be provided for the application layer. With the characteristic referencing behavior of enhanced applications in mind, a design transaction should be processed as follows [22]:

- If existing design data stored in the public DB is required, it issues one or more subsequent checkout requests. The kernel selects the corresponding design objects, protects them against concurrent access from other applications and transfers them to the workstation.
- The design objects are temporarily stored in the object buffer (a special main memory structure) and maintained by the application layer. To allow for fast operational access, the object representations are mapped to efficient storage structures (e.g. hash based) which enable a pointer-like mechanism for all object references.
- Object manipulation is always local when a sufficiently large object buffer is available. Updates of the object data are accumulated, i.e. no intermediate update propagation to the public DB takes place. Hence, perfect locality of references within the workstation is accomplished.
- Local recovery measures are performed in the private database to approximate perfect failure isolation. By use of explicit commands, the current design state can be saved (SAVE) or a previously saved design state can be reestablished (RESTORE).
- Design activities may be interrupted by the user (SUSPEND) and continued later on (RESUME). If an implicit SAVE is executed as part of the SUSPEND action, subsequent processing can be guaranteed without loss of information.
- The end of a design transaction is constituted by a single checkin. All accumulated updates of the design objects are propagated to the public DB thereby releasing the corresponding protection mechanisms (e.g. locks) and making the new object state visible to the community of users.

The dynamics of design activities is sketched by a sequence of actions in Fig. 7. The checkout/checkin mechanism establishes a design transaction [19] which guarantees concurrency isolation for the design objects. Long transactions, however, need additional mechanisms to provide an inner structure (e.g. savepoints for failure recovery). Obviously, activities in server and workstation are decoupled as far as possible facilitating failure isolation. A viable technical solution for controling transactions in a distributed environment is provided by the concept of nested transactions [23]. Refinements and adjustments of nested transactions as well as design transactions to a workstation-server environment are discussed in the literature [20,24,19].

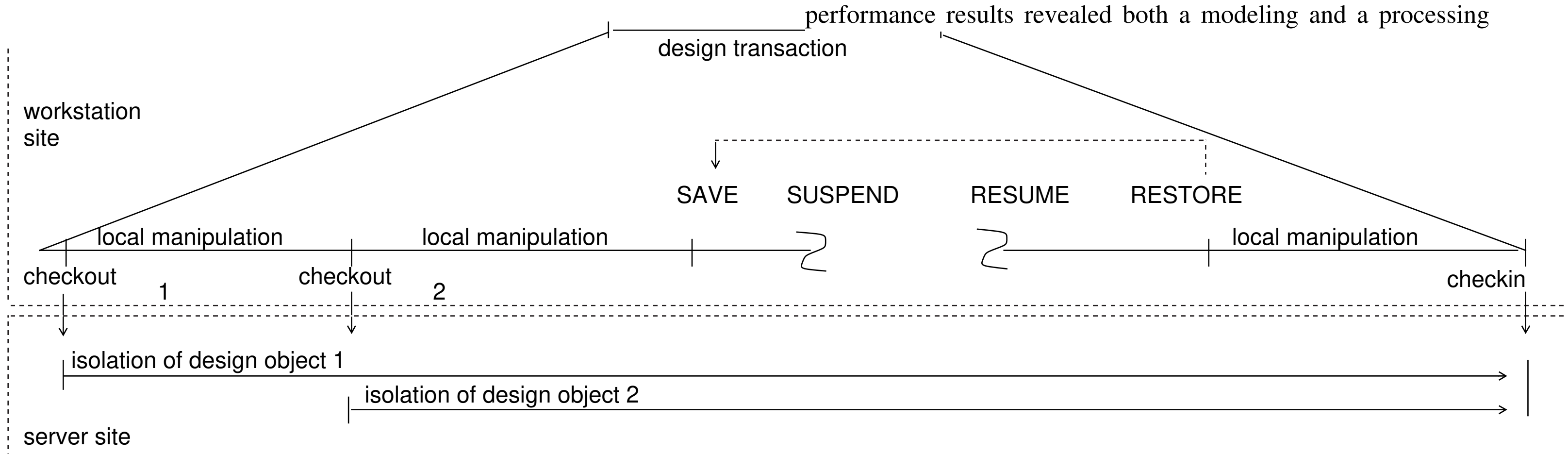


Fig. 7: Sequence of actions in the processing model

4.4 Properties of the application model

Finally, let us make some remarks concerning the application model to be implemented by the application layer. Currently, many research efforts are directed towards this interface and its linkage to the programming language environment of the application [25]. Although several issues are still controversial, some essential properties of the application model are commonly accepted:

- Application objects have an identity; they can be handled as integral entities at the application model interface. Since such objects have an internal structure, reference to structured subcomponents is possible.
- Data abstraction and encapsulation is provided to relieve the application from the complexities of internal object representation. Furthermore, complex integrity constraints can be guaranteed when a reliable mechanism for application linkage is used (ADT's, user-defined functions).The ADT implementation refers to the data structures in the object buffer and, hence, achieves the required performance of pointer-like references for object access.
- Application objects are defined as persistent objects i.e. the properties of the transaction paradigm are maintained by the NDBS.

These brief remarks may suffice to outline the current direction of designing an appropriate application model interface.

5. Conclusions

We have described various prototype experiments for supporting advanced applications by conventional DBMS. The extremely bad performance results revealed both a modeling and a processing

Emerging applications will often run on powerful workstations, sometimes equipped with special functional hardware support. Our key observation concerning object buffer use nicely fits into such architectural considerations. By allocating the application layer to the workstation we divide the DBMS architecture into an application-independent part, running as the DBMS kernel on a server processor, and into an application-specific subsystem closely attached to the application.

Data supply

References to the server require inter-processor communication. Therefore, the volume of data transfer and the number of communications between workstations and server have a strong impact on performance. We have already discussed the role of the data model in this interaction. By requesting/ propagating sets of heterogeneous record structures at a time, communication costs can be greatly reduced compared to record-oriented fetches or writes. A transaction model discussed later governs such a checkout/checkin processing [19]. The delivered data is then kept in the workstation until the application explicitly starts the checkin procedure.

Normally, data are kept in the workstation for a long span of time (hours, days, or weeks) until a major design step is reached or the current design task is finished. A private database, that is, a disk allocated to the workstation serves to save the contents of the object buffer or to keep backup copies of the user's design objects. When the design work is resumed, the object buffer can be rapidly reloaded from the private database without burdening the local area network (LAN). Note, the concept of a private database also offers substantial advantages for handling local failure recovery.

Failure isolation

As a large number of users may be affected by any kind of failure occuring in such an NDBS environment, failure isolation is a very critical design objective. This is especially true because interactive design transactions are typically very long [20]. Workstations and server may fail independently; due to loose coupling failure isolation is greatly facilitated. Thus, an essential design goal for the treatment of all failures is mutual masking: an error on the workstation should not bother the server and vice versa. Local failure recovery and a suitable transaction concept are the key mechanisms for achieving this design goal.

4.2 An architectural framework

The specific nature of advanced applications leads to a workstation-based hardware architecture with decentralized and autonomous processors. Each application processor equipped with sufficient 'horse power', main memory and private disk space serves a fixed number of users (mainly one user) and provides special functions for the application (e.g. graphics). Coupled via LAN, the workstations have access to a central server which carries the DB.

From a hardware point of view, extensibility of the application system may be achieved easily by attaching additional workstations to the server complex. However, when a certain number is exceeded, bottlenecks given by the central database server and the LAN will occur and degrade the system's performance. The closer the coupling between server and workstation becomes, the more they mutually influence one another (communication delays, various kinds of failures). On the one hand, the use of workstations is the key to solving many aspects of extensibility (available "horse power', number of users), but on the other hand, it requires careful architectural design to overcome the illustrated problems. The only reasonable approach seems to be the use of a loosely coupled system of components which have only a limited need for database access and communication. These issues must be solved by suitable software structures.

Every DBMS architecture should consist of a number of hierarchically ordered layers [21]. Such an architecture with well defined internal interfaces is a prerequisite for modularity, data independence and extensibility in the various layers. The architectural framework we propose is composed of a kernel with 'neutral' data management functions and an application layer where application-specific support is provided. Fig. 6 illustrates such an NDBS architecture where the kernel is refined by the creation of three layers (e.g. like System R [21]):

- The storage system is responsible for the management of the DB buffer and for the mapping of its contents onto external storage. It provides segments divided into pages of uniform size (within a segment) at its upper interface.
- The access system typically offers a 'record at a time' interface for direct access as well as for sequential access along various access paths (scan operations). Special storage structures such as record clusters, containers for long objects, or spatial access paths should be made available to support performance-critical operations at the physical level.
- The data system maps the objects and operations of the data model interface to the primitives available at the access system interface thereby translating, optimizing, and executing the DML requests. In case of the MAD model, it is responsible for the dynamic construction of molecules and for the set-oriented delivery of result sets to the requesting application layer.

The natural 'breakpoint' in such an architecture is the data model interface when the corresponding data model is appropriately designed along the guidelines of chapter 3. It provides neutral, yet powerful mechanisms for simultaneously supporting a multiplicity of different applications. By allocating the kernel to the server complex and the application layers to the workstations we obtain a clear division of the architecture necessary for the objective of loose coupling. Note, the properties of the data model play a key role in making such a decision useful. To extend it towards multiple application support, different application models have to be realized in the various application layers while referring to the structural object orientation and the basic integrity features of the kernel. As a consequence, an application layer provides a kind of tailoring mechanism for a specific application area; this implies that behavioral object orientation as well as the handling of most semantic issues have to be necessarily dealt with in the corresponding application layer. On the other hand, the set-oriented data model interface is a prerequisite for the separation of kernel and application layer in a loosely coupled environment. For example, MQL allows the request of sets of molecules at a time; hence, data supply of the application layer in a workstation may be effectively optimized.

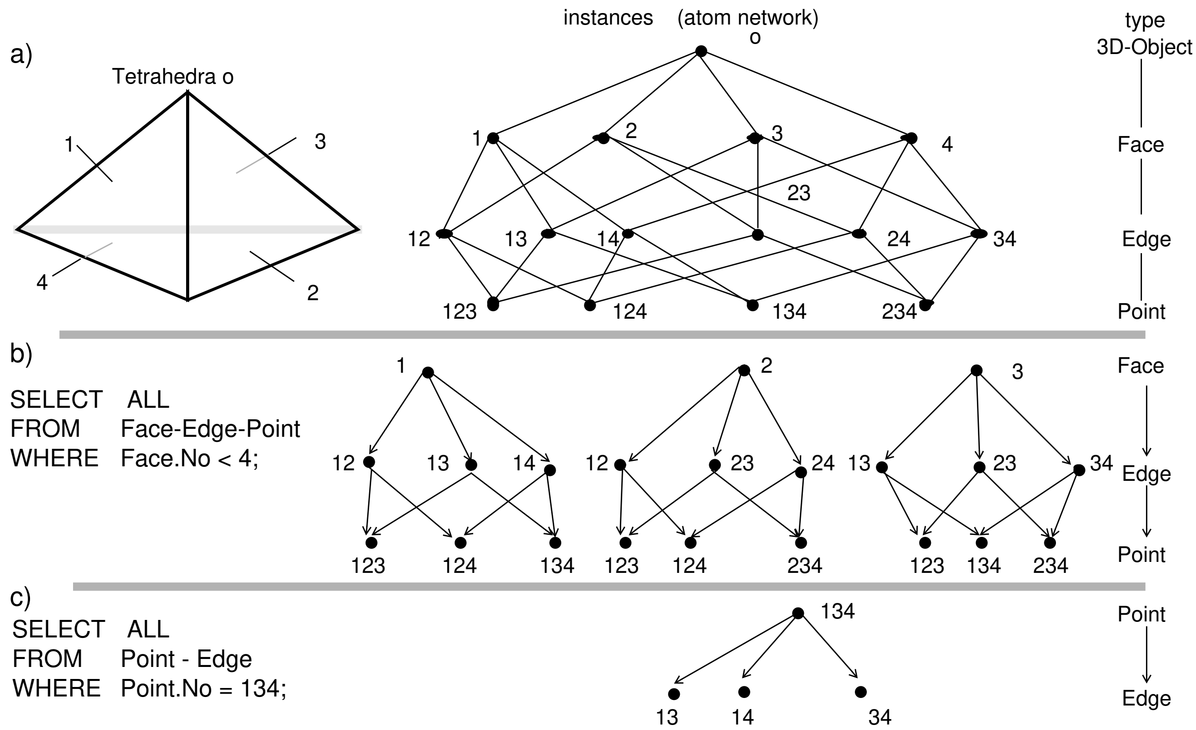


Fig. 5: Dynamic construction of molecules from a sample geometric object

4. Architectural issues

In the previous chapter, we have identified the major concepts underlying advanced data models and have sketched as an example the MAD model which satisfies many of the essential requirements. The question we would like to discuss now is how such a data model is embedded in a system architecture and how different applications with specific object views have access to the shared DB. In a typical engineering environment we have to deal with data and I/O intensive applications (that is, one reference at the application level may produce a great number of references at the data model level or trigger the output of a geometric object at a graphic screen). Furthermore, handling and manipulation of application objects are carried out by powerful algorithms which often consume a great deal of processor power. Moreover, a certain responsiveness is a prerequisite of the system to be accepted by the interactive user.

To refine the problem, the following issues have to be considered in more detail:

- How can we most effectively support a varying number of different types of application by a DBMS architecture?
- How can we provide locality of reference 'near by the application' in order to efficiently execute the huge number of references to the object representations?
- Given optimal locality preservation how do we obtain fast data supply in data intensive applications?
- How much failure isolation is desirable for the application and the DBMS?

To answer these fundamental questions, we discuss important architectural issues and their influences on the overall system.

4.1 General requirements

Multiple application support

The key idea of DBMS is the integration of all data structures necessary for a broad application area. Hence, the design of tailored systems for advanced applications would violate the original DB principle and would prohibit shared use of (non-redundant) data. Therefore, an NDBS should support multiple applications to be used in a sufficiently broad range of data processing. Obvious examples are integrated areas of design and manufacturing data or all phases of the VLSI design (from behavior specification to mask-layout) in order to guarantee a single, consistent, and non-redundant representation of application objects. For this reason, stepwise extensions of the application domain are highly desirable, e.g. the combining of production planning with CAD/CAM.

Dynamic extensibility

The system architecture should be extensible to allow for the dynamic adjustment of new types of applications. This requirement is related to the problem of application linkage, the enhancement and adaptation of application model interfaces as well as to the problem of extending the existing DB schemata. Furthermore, the aspect of dynamic extensibility requires a system to be designed for scalability; an application system initially serving one or two users should be extensible to a system for one or two hundred users, without any loss of performance and without any change of the system's interface.

Efficient object manipulation

As has already been sufficiently discussed and evaluated in chapter 2, traditional DBMS processing concepts are of little value when dealing with complex application objects. Lack of locality of reference as well as immediate propagation of the results of each tiny operation back to the DB cause a processing problem which makes interactive work impossible. Assume for example, an algorithm for computing a cut or a special view of a 3D-object consumes up to 10^6-10^7 references to the object representation. DBMS references locating the data in the DB buffer require about 1-10ms (disk references cost 25-100ms). Hence, a simple calculation may convince the reader that such an approach is not feasible.

The solution is to be found in the principle of exploiting locality of reference near by the application, that is, objects to be processed should be kept closer to the application in a so-called object buffer. Ideally, it is desirable to provide a mechanism that enables the application program to reference a DB object directly, for instance by using the pointer concept of a programming language. The concern for consistency dictates a certain isolation of program code and DB structures; thus, hiding data structures kept in the object buffer by means of ADT operations seems to be an acceptable tradeoff. Now, the average speed of a reference may be expected to be in the range of 10-100 μs which represents a substantial performance gain.

Use of workstations

lar application. Complex integrity constraints exhibiting application semantics such as control of geometric properties require special application knowledge and must be treated in a different way (e.g. two solids should not overlap). Effective behavioral object orientation implies the specification of sufficient application semantics in a formal way.

Such a semantically enriched data model enables appropriate forms of data abstraction and encapsulation, which relieve the application from the burden of maintaining intricate object representation and checking of complex integrity constraints. As a consequence, we achieve adequate operational support at the application model interface.

3.2 A proposal for a data model

In order to show how the general concepts can be transformed into a practical data model, we sketch the outcome of our design as an example. It is called molecule-atom data model (MAD model [18]) and serves as the data model of our PRIMA system [16] embodying the interface of the DBMS kernel (e.g. the data model interface in Fig. 1). It is designed along the guidelines discussed to provide powerful concepts for structural object modeling in non-standard applications. Note, more semantic and application-related concepts such as versioning, abstraction, and behavioral object orientation are achieved above the MAD model interface.

The prevalent property of the MAD model is the ability to dynamically build composite objects called molecules starting from basic building blocks called atoms. For this purpose, the language MQL (molecule query language) which is similar to the well-known SQL allows the handling of such molecules, that is, sets of interconnected tuples of different types (see Fig. 5). Here, we cannot introduce this model with all its complex details, but only illustrate the most important concepts to support our discussion. More complete descriptions may be found in [16,18].

In the MAD model, atoms are used as a kind of basic element in order to represent entities of the real world. In a similar way to tuples in the relational model, they consist of an arbitrary number of attributes. The attributes' data types can, however, be chosen from a richer selection than in the relational model, i.e. apart from the conventional types the type concept includes

- the structured types RECORD and ARRAY,
- the repeating group types SET and LIST, both yielding a powerful structuring capability at the attribute level as well as
- the special types IDENTIFIER (serving as surrogates) for identification purposes and REF_TO for the connection of atoms.

Atoms are grouped to atom types. Relationships between atoms are expressed by so-called links and are defined as link types between atom types. Such links are used to directly map all types of binary relationships (1:1, 1:n, n:m) as well as recursion. The flexibility of the data model is greatly increased by this direct mapping and by the provision of bidirectional links (symmetric relationships) in all cases. For example, references and 'back references' may be specified as follows:

- FIDs: SET_OF (REF_TO (Face.EIDs)) in an atom type *Edge*
- EIDs: SET_OF (REF_TO (Edge.FIDs)) in an atom type *Face*.

These definitions establish a symmetric (n:m)-relationship between the atom types Edge and Face. In the database, all atoms connected by links form meshed structures (atom networks) as illustrated in Fig. 5.

Molecules are defined dynamically by using MQL statements and have to be derived at run-time. Each molecule belongs to a molecule type (which is specified in the FROM clause). The type description establishes a connected, directed and acyclic type graph (cycles occur when recursive types are specified), in that a starting point (i.e. root atom type) and all participating atom and link types (for short '-') are specified. A particular example of a molecule type is Face-Edge-Point. Such a molecule type determines both the molecule structure as well as the molecule set which groups all molecules with the same structure. At the conceptual level, the dynamic construction of molecules proceeds in a straight-forward manner using the molecule type description as a template: For each atom belonging to the root atom type all children, grandchildren and so on are connected to the molecule structure, terminating after all leaves of the molecule structure are reached. Connecting children to the molecule structure means performing the hierarchical join which is supported by the link concept. Hence, for each root atom a single molecule is constructed. Fig. 5b shows the result of a molecule construction for Face-Edge-Point molecules, where the set of molecules was restricted using the WHERE clause. The SELECT clause offers proper projection capabilities such that the query result may be reduced to the minimum required molecule information. Furthermore, it illustrates the most important properties of the MAD interface:

- An MQL request handles a set of molecules.
- The molecules as complex objects consist of sets of atoms of different type, i.e., they are embodied by sets of interrelated heterogeneous record structures.
- Molecule construction is dynamic and allows symmetric use of the atom networks (e.g. *Point -Edge* (Fig. 5c)).

These arguments may justify the need for object orientation in the DBMS [2]. If the object representation is adjusted to the needs of a particular application, the intended object handling may be achieved in a natural and smooth way, thereby saving a lot of inadequate operations and ponderous deviation steps. Data models supporting such applications embody some degree of object orientation (towards the application objects).

Structural object orientation

The notion of complex objects is used to indicate that such objects have an internal structure (structured components) maintained by the DBMS and that access is provided to the object as a whole as well as to its components. Assume a complex object described by means of the ER model. A subset of the DB schema could be specified as a complex object type (see Fig. 3 or 4). An entity type containing the attributes of the complex object plays the role of the object-defining entity (root). An entity type is assigned for each of the component types which are connected by relationship types 'consists of'. Hence, a particular complex object is represented by all entities in reach starting from the root entity and observing the type structure.

Disjoint vs non-disjoint object representation

Complex objects may share common subobjects, that is, identical components are part of different objects (non-disjoint [7]). 'consists of'-relationships of type (n:m) are responsible for such a kind of component sharing. In contrast to this, functional relationships (1:1, 1:n) lead to disjoint objects. As illustrated in our examples, a considerable portion of the relationships occurring in emerging applications are of type (n:m). Our notion of object orientation requires that all essential relationships should be known to the DBMS, that is, they must be included in the data model representation.

Restrictions introduced by hierarchical data model (extended relational model [13], NF^2 model [15]) would either lead to poor modeling and loss of information or to hidden redundancies:

- Dropping relationships (e.g. corner or neighbourhood information in Fig. 3) dictated by the mapping process to hierarchical structures may not always be a good design decision.
- Non-disjoint objects could be remodeled by using (1:n)-relationships only, that is, they could be transformed into disjoint object representations at the data model interface (DB schema). Depending on the degree of overlapping, a large amount of hidden redundancy may result (e.g. the same edge is represented several times). Such a redundancy cannot be controled by the DBMS since it is not described in the DB schema. Hence, it must be handled by a layer above the data model interface.
- Loss of information occurs because only one direction of relationships is incorporated by a hierarchy (e.g. Solid -> Face -> Edge -> Point).

In other words, we postulate the direct and non-redundant mapping of all important relationships (including n:m) in the DB schema.

Recursive vs. non-recursive objects

Complex objects are called recursive if they may be composed of objects of the same type; otherwise, they are called non-recursive. For example, a module recursively consists of submodules as shown in Fig. 4. Besides (n:m)-relationships, recursiveness seems to be a distinct characteristic of the advanced DBMS applications discussed: solids are built using previously constructed solids, cells of a chip contain cells, or rules are organized in a rule hierarchy or precedence network. As a consequence, the lack of recursive object handling causes tedious and ineffective object manipulation (e.g. large number of level-to-level joins). Hence, recursive object definition as well as object manipulation is considered to be very helpful in our framework.

Static vs. dynamic object definition

Static objects are rather infrequent, i.e., a variety of views of the object should be supported, depending on the current need of the application, that is, the level of abstraction and the manner of processing. Let us once again refer to Fig. 3. Some construction algorithms require face objects along with their edges and points (Face-Edge-Point), while in another processing state it may turn out that just the inverse object nesting has to be manipulated, that is, a point object with its adjacent faces and edges (Point-Edge-Face). Static objects (predefined at DB creation time in the DB schema) do not offer a great deal of flexibility for the dynamics of a construction environment and may sacrifice many advantages inherent in the concept of object orientation. Hence, dynamic object definition support enables us to effectively cope with the dynamically changing 'object views' of DBMS applications.

Symmetry of relationship representation

Dynamic object definition requires flexible relationship representation and derivation (either at run time or supported by caching techniques). A critical issue for achieving such a dynamic definition facility is related to the handling of relationships. Symmetrical representation of all relationships captured in the DB schema allows for the desired freedom in building up dynamically defined complex objects. For reasons of efficiency, even (n:m)-relationships should be represented directly (without special connection record) and should support bidirectional traversal and use of objects.

Based on our experiences and the empirical results of our requirement analysis, we have discussed the desirable concepts which should be incorporated into a 'new' data model. All static design decisions should be removed from the DB schema. Of course, a performance penalty may arise compared to the case where a hierarchical structure is allocated statically and its elements are fetched one at a time. Since performance is an important issue, the system implementation should react properly to meet this challenge. At the access path and storage structure level, one could provide special clustering techniques [16] to speed up the aggregation of complex objects. Redundant allocation of objects is another idea whose usefulness has to be examined. Furthermore, various caching techniques have to be proposed for the discussed problem [10].

In our view, the data model should not be biased towards a particular application domain. Based on such a neutral model (implemented by the DBMS kernel), object orientation for application objects of a specific domain is gained by mechanisms which are provided by a so-called application layer (see sect. 4.2). For this kind of tailoring, we need some additional concepts whose realizations may greatly vary according to the application requirements.

Versioning

Temporal aspects have to be modeled in advanced applications. Sometimes the precise history of objects is required which implies the maintenance of time relationships. In engineering applications, it is often sufficient to provide the ordering of versions belonging to the same design object [4].

Abstraction properties

Structural object orientation facilitates the integration of abstraction concepts such as generalization or aggregation. Since the 'syntactic' relationships are provided by the neutral data model, these forms of abstraction can be viewed as semantic enhancements. As a consequence, the concept of inheritance may be utilized along generalization/ classification paths [14]. Moreover, it is possible to exploit implied predicates in aggregation hierarchies [17].

Behavioral object orientation

To enhance integrity control and semantic expressiveness, more object properties beyond the structural relationships have to be specified. For example, referential integrity is part of object structure preservation, which is independent of the needs of a particuments in other application domains. In the area of VLSI design we implemented a DB-based chip planner which receives a module list as well as a net list (wiring) as input information for the 'cell under design'. The result is an optimal placing of modules and wiring areas on the chip. Here, we only want to illustrate the structural complexity of the design data (DB view) in Fig. 4. The key entity type is Module. Each instance is either a standard module or is composed of submodules; hence, by use of a recursive relationship on Module the inner structure of modules can be described (structure hierarchy). The external relationships among modules at each level of decomposition are recorded by the entity type Net and the corresponding relationship type (n:m). Pin is used to represent I/O-connections to other modules and Channel to model the wiring area attached to each module. Furthermore, a particular part of the design information guides the stepwise placement and optimization of the module structure: Graphnode is required to represent the so-called slicing tree. Curve and Point map the sizing functions used for the allocation of the inner nodes and the evaluation of the slicing tree.

DBGEO was dedicated to a particular type of land surveyor application where geographic information (e.g. plots, streets, etc.) and some representative functions helped in planing tasks and for handling maps. Geometric as well as string data had to be managed jointly. DBMS support for expert systems was investigated by coupling a diagnose expert system (MED1) with the relational system INGRES. In a number of consecutive optimization steps, the performance of system coupling was improved drastically, primarily due to the use of a working memory in close proximity to the application MED1. These experiments have proven the importance of exploiting the 'near by the application locality' to minimize expensive DBMS calls for repeated data references.

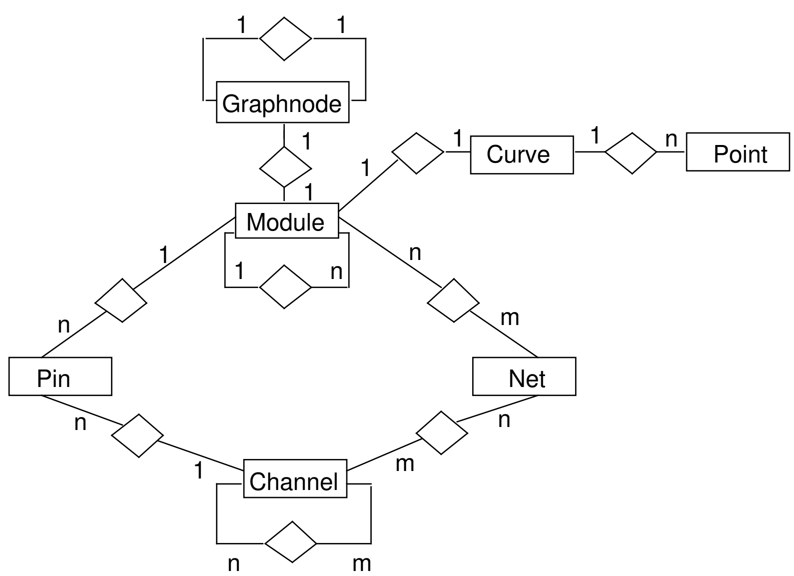


Fig. 4: ER representation of information for the VLSI chip design (subset)

Compared to section 2.2, similar observations were made in these applications concerning the number of DB-operations, long-winded object mapping, and run-time overhead. Hence, we can conclude that all applications investigated can be effectively supported by a carefully designed application model interface. Such application-specific models are characterized by the existence of complex objects (sometimes called molecular objects [7]) and powerful operations on these objects. By using conventional DBMS they had to be mapped to a large set of meshed data structures at the data model interface, that is, records connected by (n:m)-relationships. Application operations had to be accomplished by thousands of related and serially executed 'tiny' DB-operations during which only rudimentary integrity constraints were checked. Our observations have revealed the following general properties as well as confirmed present hypotheses:

- There is a considerable portion of meshed (non-disjoint) structures due to the occurrence of (n:m)-relationships.
- No general preference of execution direction (traversal of data structures) is found, that is, efficient support for bidirectional or symmetric traversal is required.
- Recursion is primarily important for knowledge-based applications, however, engineering applications such as CAD also require structures whose 'natural' mapping would greatly benefit from recursiveness.
- To enhance integrity control and semantic expressiveness, more object properties and, in particular all essential relationships among objects (beyond hierarchical ones) should be specified and preserved by the data model.
- 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.
- Locality of reference should be exploited as far as possible; buffering objects close to the application seems to be the only means to achieve efficient object references.

From a DBMS's point of view these properties seem to be the most essential ones which need to be considered when designing more appropriate data models and DBMS architectures.

3. Consequences for data models

So far, we have evaluated the data management aspects of a spectrum of important applications for non-standard DBMS. A major reason for the modeling deficiencies encountered was the inappropriateness of conventional data models and, in turn, the resulting processing overhead of inadequate data manipulation operations. Hence, the data model plays a key role when designing DBMS for advanced applications. Of course, this fact also becomes evident when surveying the numerous approaches to design more adequate data models [2]. Most of them can be classified in the following manner:

- The flat relational model is used as purely as possible. A few selected enhancements such as more powerful data types (procedures) are added. Caching of objects or special distribution techniques are expected to solve performance problems [8,9,10,11].
- 'Enriched' data models are used to represent molecular objects and more complex types. They are equipped with more semantics [4,12,13,14,15].

Although once faithful believers in the relational model, we were converted more and more to the second kind of religion, at least as far as advanced applications are concerned.

3.1 General concepts

Let us first identify the data model properties most useful for the support of complex objects. In our application examples, the lack of object orientation was the 'root of the evil'. Note, depending on the richness of modeling, the DB schema for the boundary representation of a solid contains ten or more relations (record types). As a consequence, the subobjects of a solid (points, edges, faces, subsolids) as well as their relationships to each other have to be described by 'independent' tuples of different types. Accordingly, at the data model interface a complex object is embodied by a heterogeneous set of tuples which are connected in various ways by value equality of attributes (primary key - foreign key) or set membership. Such a specification of the application object only provides an 'atomised' view for the DBMS (at the data model interface); the view of the solid as an integral object including its representation, handling, and integrity preservation has been lost. Object access (by sequences of joins), control of integrity constraints, object manipulation must be performed by the application in an extremely expensive way (see Tab. 1).

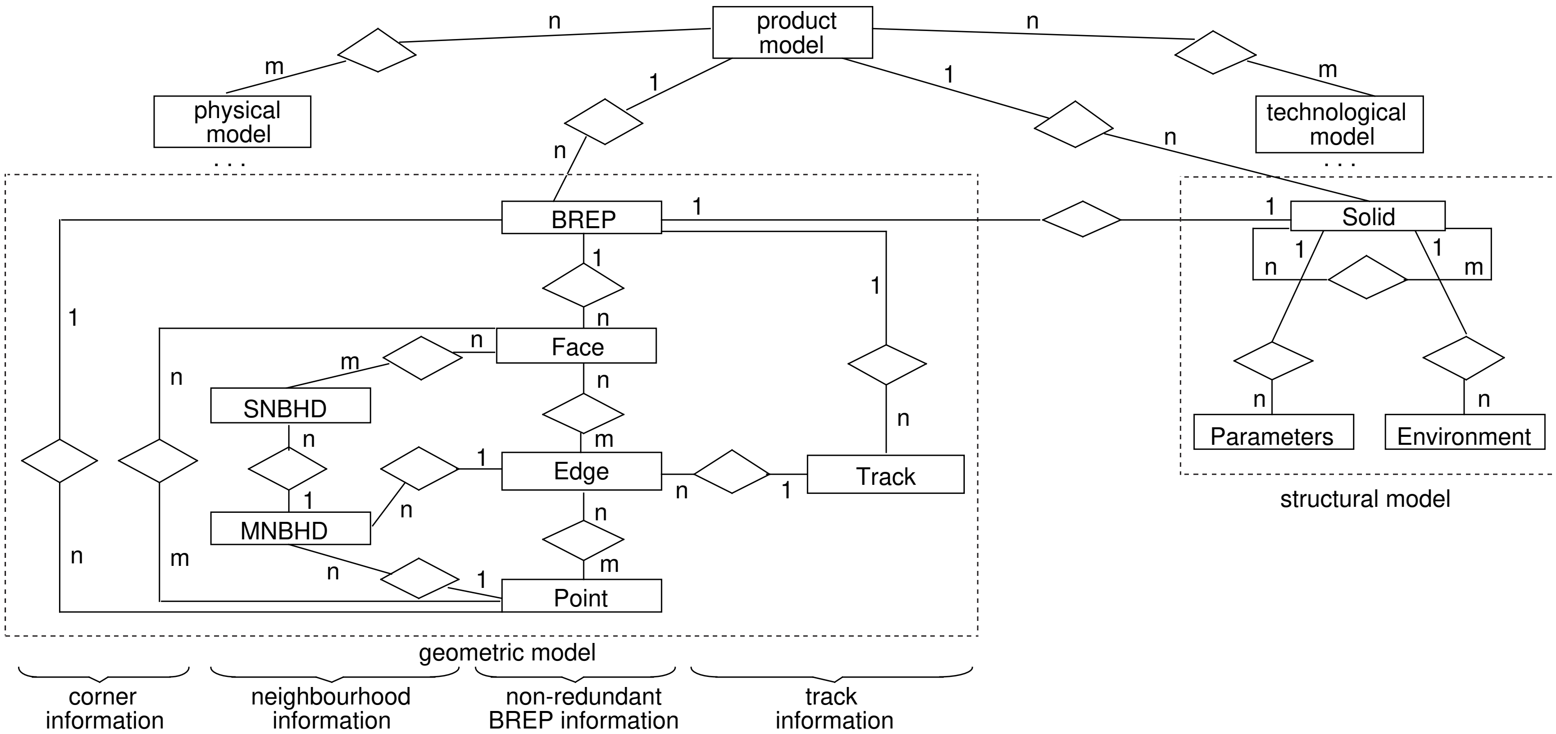


Fig. 3: ER representation of the product model for workpieces

responding manipulation language (CODASYL DML) is a 'one record at a time' language; it provides direct access as well as navigational operations which frequently require previous cursor positioning (e.g. FIND NEXT Point WITHIN Edge-Point). As a consequence, an extreme number of operations is necessary to map even simple geometric objects.

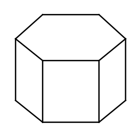
A major design objective of the application model interface (AMI) was the direct and natural support of the geometric construction algorithms executed by the application. To indicate the increase in abstraction achieved by the application layer we illustrate some indicative operations as examples:

- 'Get Points of Edge': delivers a list of records
- 'Define Edge by Points and Edge': creation of a new edge
- 'Confaces': construction of neighborhood information of two edges.

The geometric modeling system was designed based on a rich set of such ADT operations. At the dialogue interface (DI), suitable parameterized operations on application objects are available for the graphic-supported interactive construction.

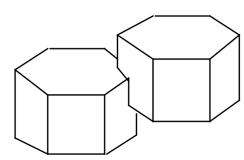
A first example is intended to evaluate the appropriateness of the interfaces in our prototype system. To create the boundary representation of a cylinder approximated by a polyeder, an operation POLYCYL(n) is provided at the dialogue interface, where n determines the number of lateral faces as an actual parameter. Tab. 1a summarizes the frequency of operations at the chosen interfaces for POLYCYL(n) - subject to various degrees of approximation. Hence, the DMI row indicates the number of DML operations. The costs at the AMI clearly rise linearly with n indicating that we have chosen a 'natural' interface for the given application. The overhead at the DMI, however, reveals a strong non-linearity. To be specific, a cy linder generation with 5 resp. 50 lateral faces consumed 1958 resp. 41828 operations at the DMI. This 'catastrophic' behavior of the DMI was caused by the so-called subschema concept used in CODASYL systems: the application has control of, at most, one occurrence per record type and applies 'tiny' DML operations. Hence, locality of reference cannot be used. This means that rereference typically leads to repeated DBMS calls fetching the same data over and over again.

A second example is used to demonstrate the processing overhead during construction, that is, the costs for modifying and maintaining existing structures. Here, the generic operation is 'UNION(SOLID1,SOLID2)' where SOLID2 is identical to SOLID1. Tab. 1b sketches graphically the kind of operation. To facilitate comparison, the chosen solids always produce the same topological effects; the cost of the UNION operation is listed for a few parameters (POLYCYL is abbreviated by CYL). Again, the performance figures indicate a tremendous mapping overhead.



POLYCYL(n)	5	25	50
DI	1	1	1
AMI	64	304	604
DMI	1958	15078	41828

a) generation of a cylinder approximated by a polyeder



Operation	#DMLs
UNION(CYL(3), CYL(3))	16051
UNION(CYL(4), CYL(4))	20658
UNION(CYL(6), CYL(6))	31432

b) UNION of two parameterized solids

Tab. 1: Evaluation of object creation and manipulation

We have applied various processing refinements and have reduced the modeling redundancy as well, that is, we have run our experiments with different degrees of modeling redundancy without confining the overhead to a reasonable limit. Although a fraction of the costs is attributable to the properties of the CODASYL data model (cursor positioning, access path navigation, etc.), the situation is not fundamentally improved by the other classical data model. The relational model, for example, maps each (n:m)-relationship to a separate relation. Operations spanning multiple relations have to be performed by sequences of expensive joins (compared to cheap FIND OWNER/FIND MEMBER navigations). Hence, a complex operation such as POLYCYL or UNION may require several thousand relational operations and therefore causes costs comparable to those of the CODASYL model.

2.3 Observations on prototype experiments

We have sketched in some detail our approach in the area of 3D-CAD. Due to space limitations we can only mention our experi-

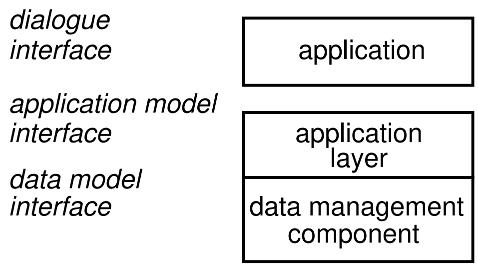


Fig. 1: Architectural framework for prototype systems

this additional layer, which we called application layer, was aimed at alleviating the modeling problem for complex objects and their manipulation by the applications (processing problem). Of course, the application model interface was separately designed and tailored to the requirements of the various application classes (according to our preliminary problem understanding). The applications were programmed in realistic environments with a limited number of operations to perform at least some typical application object transformations. A dialogue interface enabled simple handling of the prototype systems.

An important goal of our investigations was the identification and analysis of DB-related problems and the search for processing bottlenecks within the different applications. To observe call frequencies and execution times of external and internal requests, a flexible measurement and analysis tool was developed to obtain specific and accurate information concerning the internal system behavior, and, in particular, the 'traffic' at the data model and application model interfaces.

For the selected applications, we illustrate our requirement analysis as follows: A brief description of the application is given, and some indicative operations may help to characterize the properties of the desired application interface. In order to sketch the DB view of the application, we choose a representation of the entities and relationships independent of a particular data model. (Note, our examples are always a subset of the complete DB schemata of the real applications). Finally, we present some quantitative results for typical user operations which reveal critical performance problems in the corresponding applications.

2.2 3D-CAD application

Our prototype system to be discussed in more detail was a 3D geometric modeler for workpieces. By using this system the CAD application is able to perform the construction of solids upon user requests from a graphical I/O interface. This construction process is governed by a product model including structural and geometrical as well as technological, physical, and organizational submodels. In the practical design work, additional relationships spanning application domains have to be utilized. It is important to note that design steps are tightly connected with manufacturing steps in the sense that a workpiece under design has frequently to be controlled whether it can be manufactured (using existing machines and standard parts available) and whether it meets the specific restrictions and rules of the manufacturer. Hence, the information structures required are even more complex than those in Fig. 3.

For our investigations, geometry and structure of the solids to be created were of particular interest. Therefore, the kernel part of our system is based on a volume-oriented scheme called Constructive Solid Geometry (CSG [5]) which allows the construction of composite objects starting with a set of primitive objects (Fig. 2).

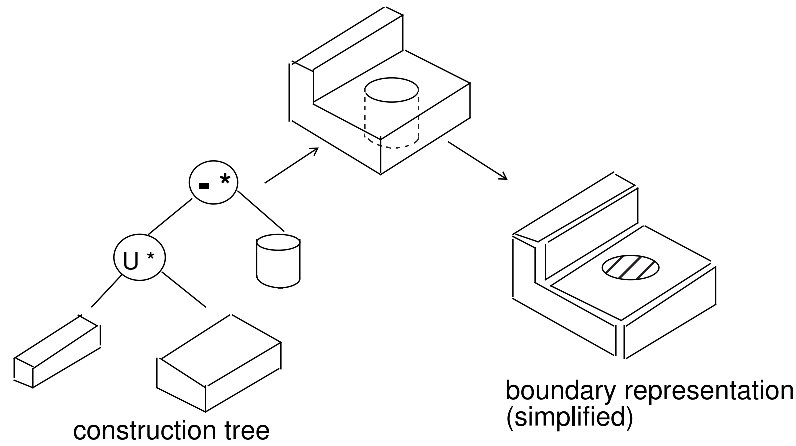


Fig. 2: Construction of composite solids

The user, guided by menues, selects from predefined parametric base volumes and regular operators (union, difference, translation, etc.) and, thus, composes his workpiece in an interactive manner.

The data structures forming the product model are sketched in Fig. 3. Its major purpose is to illustrate the 'typical' complexity and the interleaved relationships inherent in the data structures of such engineering applications. For comparison reasons, we have chosen a 'neutral' representation as an ER-diagram [6] which allows a simple transformation in a relational or network data model. Hence, the DB view of the solids is easily gained from (part of) Fig. 3. They are represented by CSG trees which describe their internal structure (or history of construction). The important relationship among solids is 'part_of' (or 'contains') which is expressed in the structural model. A dual representation called boundary representation is automatically derived and maintained for objects under construction, to facilitate graphical I/O and special geometric operations [5]. The key part of the geometric model consists of the entity types BREP, Face, Edge, Point and the corresponding relationships (most of them are n:m). BREP (boundary representation) serves as the root entity and describes the 3D-object under design as a whole. It is connected to Face by a (1:n)-relationship which allows us to model the individual faces belonging to the object. These are related to their edges which, in turn, are limited by points. This kind of modeling gives a rather spartan, but non-redundant type description of construction objects; the resulting representations, however, may lead to tedious and poor modeling algorithms since implicit information, e.g. all edges belonging to a particular track, has to be derived over and over again. To avoid such ineffective processing it may be advantageous to provide useful redundancy. To simplify the modeling algorithms we added the entity types Track as well as SNBHD and MNBHD (single and multiple neighbourhood). Track is used to partition the set of edges of a solid: an instance of Track is related to all edges on a distinct straight line which improves access to such edge sequences. SNBHD and MNBHD serve for a more accurate description of the geometric environment of a point resp. edge with respect to a solid. This information is especially important when solids come into contact with each other. Additional relationship types represent the corner information necessary to derive the wire frame model of the solid under design. The use of this specific neighborhood and corner information was motivated by the tailored support of our modeling algorithms. For our purpose, it seems sufficient to refer to the geometric model of Fig. 3.

The view given by the DB schema (or equivalently by the ER-diagram of Fig. 3) is that of the data model interface (DMI). The cor-

Non-Standard DBMS for Support of Emerging Applications - Requirement Analysis and Architectural Concepts

T. Härder

In: Proc. 22nd Annual Hawaii Int. Conf. on System Sciences, Hawaii, 1989,. pp. 549-558.

University Kaiserslautern, Department of Computer Science,
P.O. Box 3049 , D-6750 Kaiserslautern, West Germany

Abstract

Today's DBMS are unable to meet the increasing requirements of emerging applications that would like to use a DBMS. To improve this situation a new generation of DBMS architectures adjusted to the demands of such enhanced applications have to be developed. For this purpose, we have analysed the data management needs of a spectrum consisting mainly of engineering applications and have encountered both a modeling and a processing problem in today's DBMS. We try to explain which data model concepts and architectural decisions are necessary and why they can succeed in providing the anticipated data management support. Although the focal point of the paper is a tutorial-like description of application requirements and their consequences on DBMS design, we illustrate the recommended concepts by a practical data model and a DBMS architecture implemented at the University of Kaiserslautern.

1. Introduction

Conventional database management systems (DBMS) are becoming increasingly successful in business and administration applications; however, they are hardly used in emerging application areas such as CAD/CAM, geographic information management, knowledge-based systems, etc. which are often termed non-standard applications. Although these applications urgently need database support for a variety of reasons, DBMS have so far failed to satisfy their requirements. The inherent causes of this situation were illustrated as pointedly as possible by a hypothesis proposed in [1]:

'The general purpose DBMS are general only for the purpose of commercial applications, and their data models are general only for modeling commercial views of the world'.

When trying to use commercial DBMS in emerging application areas, many research groups or developers in the DBMS field have observed the validity of the above hypothesis - at least from a performance point of view [2]. But to describe and analyze the observed deficiences and to draw the right conclusions requires a lot of effort to understand the requirements of the corresponding application as well as its impact on the DBMS behavior. Hence, only a thorough DBMS performance evaluation can reveal the system's response under a workload modeled as realistically as possible.

It is important to know precisely quite a number of properties of such emerging applications which are to be supported by 'improved' DBMS, termed database systems for non-standard applications (NDBS). The desired properties as described in the literature are often too fuzzy and ill-defined, since only qualitative properties are typically referenced and mutual interferences of functions are neglected. Hence, an empirical requirement analysis of such applications is necessary to gain relevant and quantitative data about their essential properties. Then, the results of the requirement analysis may give valuable hints for the design of an 'adapted' NDBS architecture.

To illustrate typical workloads for future NDBS we explored and evaluated their most important requirements and their implications on data management in detail. To cover a wide spectrum of prospective uses we have chosen four different applications in the areas of geometric objects and rule-based processing:

- geometric modeling of 3D objects (KUNICAD system)
- support of chip planning for VLSI design (DBCHIP system)
- management of geographic information (DBGEO system)
- diagnostic expert system coupled to a relational DBMS.

Our conclusions from these investigations confirm results obtained by other researchers in individual application areas [2,3]. They allow for a number of distinct proposals to improve the architecture of DBMS. Our architectural concepts provide solutions to the **modeling problem** and the **processing problem** of so-called complex objects handled in non-standard applications.

2. Requirement analysis

2.1 Architecture for prototype experiments

For our requirement analysis, we have chosen a uniform overall architecture to facilitate comparison of results and to derive general design objectives in a similar framework. Fig. 1 illustrates the major interfaces important for our architectural investigation. Obviously, the data model interface was embodied by conventional DBMS (UDS, a CODASYL-type system, and INGRES, a relational system).

In order to support the application by a more abstract and more suitable interface, an additional system layer was implemented whose upper interface is termed application model interface. Compared to conventional applications, the relevant entities of emerging applications are much more complex: they are internally structured and consist of heterogeneous components [4]. Their pure description by conventional data models (i.e. by means of records, sets, tuples, etc. only) does not provide the view desired by the application, where application objects should be referred to as a whole or as components of aggregate (application) objects. Hence,