Towards a unified view of design data and knowledge representation

Bernhard Mitschang
University of Kaiserslautern
Erwin-Schrödinger Straße
D-6750 Kaiserslautern
West-Germany

0. Abstract

Adequate information modeling in non-standard application areas (e.g. engineering applications such as CAD/CAM, VLSI design or knowledge-based applications) requires the abstraction concepts of classification, aggregation, generalization, and association. The Molecule-Atom Data model (MAD) designed for the effective support of such an information model is justified and described with its essential properties and features. MAD offers dynamic object definition and object handling, based on direct and symmetric management of network structures and recursiveness. These generic mechanisms can be used to map the above mentioned abstraction concepts in a straight-forward manner. Thus, the mapping of a wide variety of semantic and object-oriented modeling constructs, including complex objects with shared subobjects, becomes feasible. All these concepts are illustrated by means of some vivid examples taken from the areas of CAD/CAM and knowledge-based applications.

1. Introduction

Recently, the development of a new generation of database systems capable of supporting non-standard application areas such as engineering applications for CAD/CAM and VLSI, scientific and statistical applications, knowledge-based applications, image and office applications has emerged as an important direction in database system research. These advanced applications differ from conventional (business) applications in a number of critical aspects, including data modeling and processing, concurrency control and recovery mechanisms, as well as access methods and storage structures. Of course, the chosen data model and its specific properties play the dominant role in all these design and implementation approaches; most of them can be classified in the following manner:

- They focus on the flat relational model with only few selected enhancements [CD86,LMP86, SR86].
- They concentrate on integrating and superimposing hierarchical structures on relations [Da86,DKM86,LK84, PSSWD87,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 the specific needs, we have thoroughly investigated four different application areas (their structures and algorithms) by implementing and evaluating sizable prototype systems: VLSI circuit design, construction of solids in 3D-modeling, DB-based expert system, and map handling in geographic information systems [Hä86,HHLM87,HMP86].

Based on this empirical data we present some key observations and the consequences thereof concerning information modeling and management of the application objects, as well as its mapping to the data model interface. In particular, we come up with

- essential application-object characteristics triggering
- new proposals for a data model which allow for straight-forward mapping of a wide variety of semantic and object-oriented constructs, including complex objects with shared sub-objects.

The remainder of the paper describes and illustrates the Molecule-Atom Data model and is organized as follows. Chapter 2 presents two major application areas, that is, engineering disciplines and knowledge-based systems. The main characteristics of the existing application objects reveal the essential data model requirements for accurate and efficient mapping to data objects, guiding our data-model design. Chapter 3 describes the general concepts underlying the MAD model, its data and load definition, as well as its query and manipulation facilities. Finally, chapter 4 compares the model with other data models and chapter 5 summarizes the paper and concludes with an outlook covering current and future investigations.

2. Information Modeling in Non-Standard Application Areas

One of the most demanding requirements in non-standard 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. Here we use the Entity-Relationship model [Ch76] for explanatory purposes. By modeling application objects, it becomes evident that the ER model has to be extended by some kind of object-orientation, thus yielding an appropriate information model. Firstly, the whole application has to be modeled by means of this ER model and then mapped to the data-model interface of the underlying database.

2.1 Modeling and Managing Application Objects

For our purpose, the best reference is [BB84] where a thoroughgoing analysis and characterization of the application objects, specifically in engineering disciplines revealed the general concept of **molecular objects** - in [LK84] synonymously 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. Furthermore, 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.

Modeling a complex object by means of the ER model leads to an entity type containing the attributes of the complex object, an entity type for each component, and a relationship type between them meaning 'consists of'. Hence, the complex object is represented by all entities related in this way. They are said to form a 'molecule' with the entities resembling 'atoms', respectively.

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. These four cases are summarized and visualized in fig. 2.1 using ER diagrams. In fig. 2.1 the molecular objects are represented by the entity types named MOL_OBJ, whereas the primitive parts are modeled using the Px_OBJ entity types. All represented relationship types hold the property 'consists-of'.

A more quantitative examination of our prototype application systems reveals some further important object characteristics (cf. fig. 2.2): Firstly, in most cases we have to deal with network structures, i.e. non-disjointness. Secondly, there are no static molecular objects, i.e., we have to cope with dynamically changing molecular objects, depending on the actual view of the application, that is, the level of abstraction and the way of processing. Thus, it seems much more appropriate to define the molecules dynamically instead of predefining molecules statically. Refering to 3D modeling, these dynamics are apparent: One important representation of solids, especially for graphical output, is the boundary representation (BREP) depicted in fig. 2.2a (geometric model). It consists of faces which are in turn composed of their borderlines (edges) limited by endpoints. 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. Table 2.1 expresses these dynamics showing some kinds of molecular objects which are subject to manipulation during application processing. The first and the last column (name and characteristics) of table 2.1 depict the higher level view of molecular objects, i.e. the objects as atomic units. This abstraction concept is often called aggregation since the higher level object is aggregated from its component objects. The remaining two columns reveal the lower level view, that is, the components. Refering to other engineering application areas it looks quite similar. For example, geographic applications (fig. 2.2.b) have to deal primarily with maps, while VLSI-design systems (fig. 2.2.c) use the generic concept of cells for the description of functional, structural, and physical domains of the circuits to be designed. In fact, these application objects also represent non-disjoint and recursive molecules.

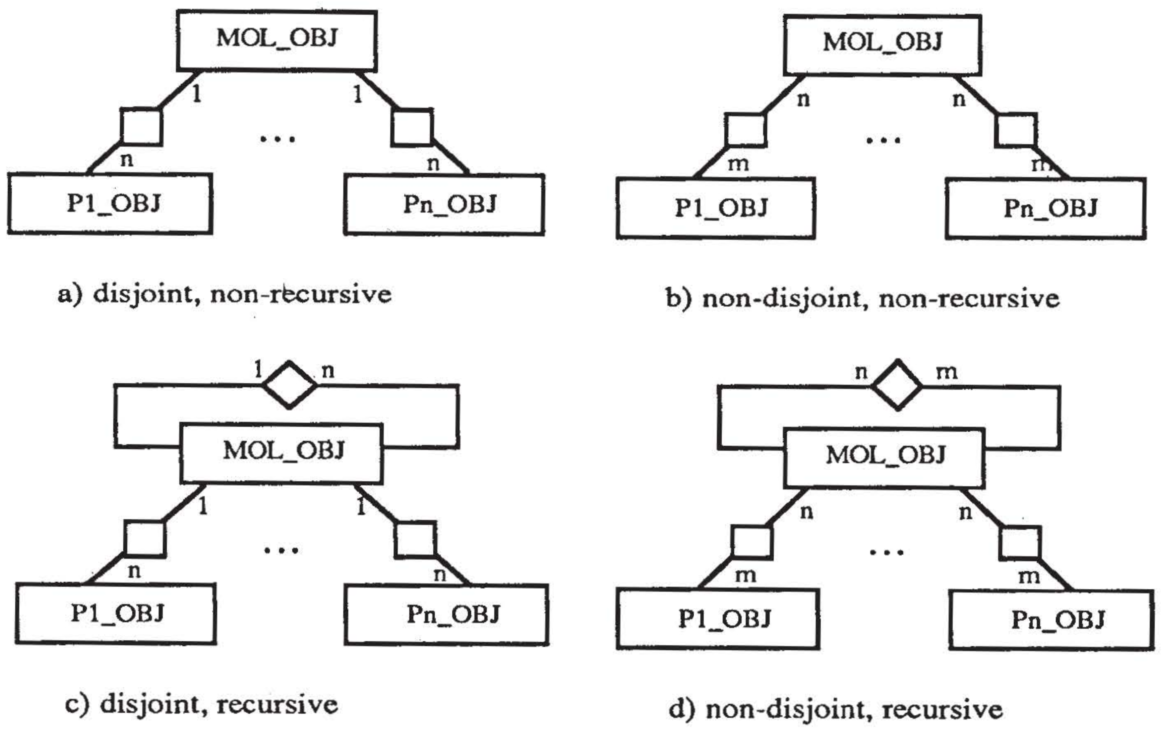


Figure 2.1: Four types of molecular objects

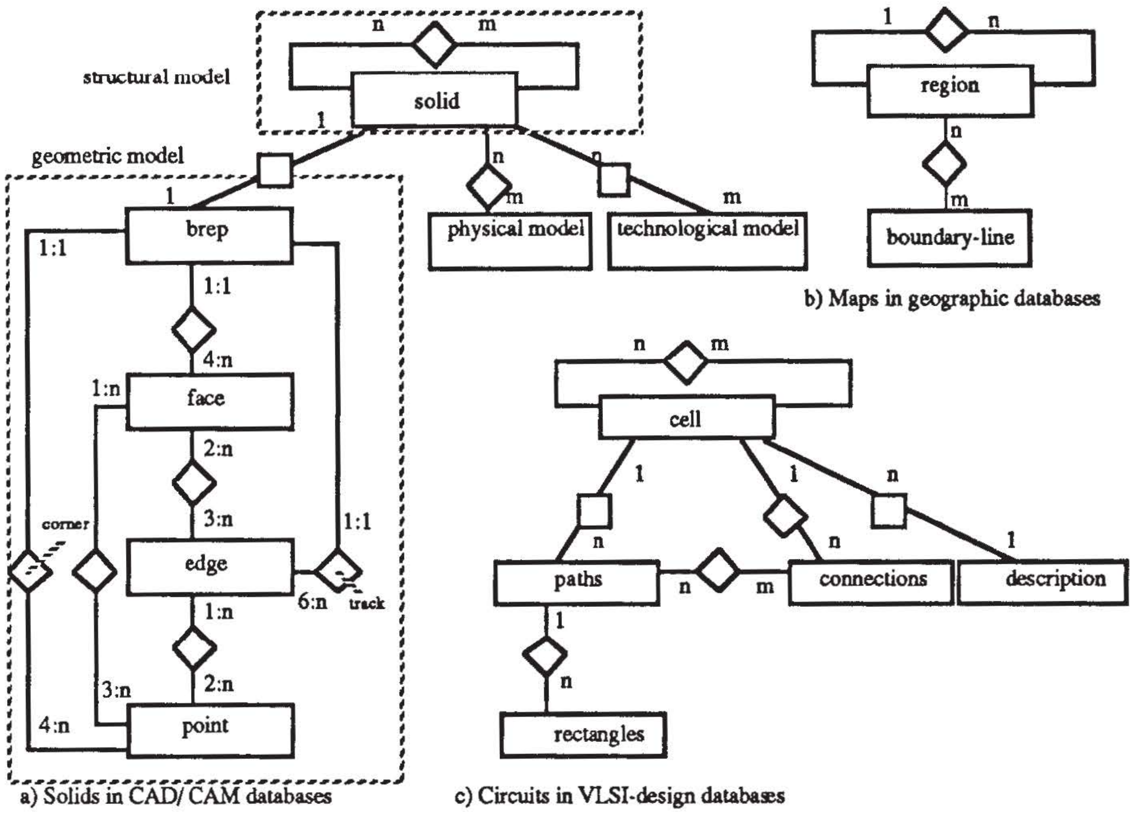


Figure 2.2: Examples of molecular objects in engineering disciplines

Modeling knowledge-based applications reveals some further important abstraction concepts. Knowledge representation techniques incorporate the concepts of generalization and association hierarchies, in addition to aggregation and classification. The main characteristics of these abstraction concepts may be summarized as follows:

- The concept of **classification** collects objects (entities) with common properties (attributes) to form object types (entity types). Between an object and an object type there exists an 'instance-of' relationship.
- **Aggregation** refers to an abstraction in which all component objects are aggregated to build a higher level object. Between the component objects and their aggregate object there exists a 'consists-of' relationship (cf. fig. 2.1).
- The **generalization** concept refers to a set of similar object types which is regarded as a generic object type. This concept ignores some individual differences of the specialized types. There is an 'is-a' relationship between the specialized and the generalized object types. Repeated use of the generalization concept yields a generalization hierarchy.
- Finally, the **association** concept is a kind of abstraction in which a relationship between elements is considered as a higher level object

known as set object. Between the elements and its set object there exists a 'member-of' relationship with repeated use of this concept creating an association hierarchy.

name	subtypes	object structure	characteristics
point_obj	point	point	non-disjoint and non-recursive
point_nbhd	point, edge, face	point - edge - face	non-disjoint and non-recursive
edge_obj	edge, point	edge - point	non-disjoint and non-recursive
edge-nbhd	edge, point, face	edge - (point,face)	non-disjoint and non-recursive
face_obj	face, **edge_obj** or face, edge, point	face - **edge_obj** or face - edge - point	non-disjoint and non-recursive
brep_obj	brep, **face_obj** or brep, face, edge, point	brep - **face_obj** or brep - face - edge - point	disjoint and non-recursive
piece_list	solid	solid(recursive:solid)	non-disjoint and recursive

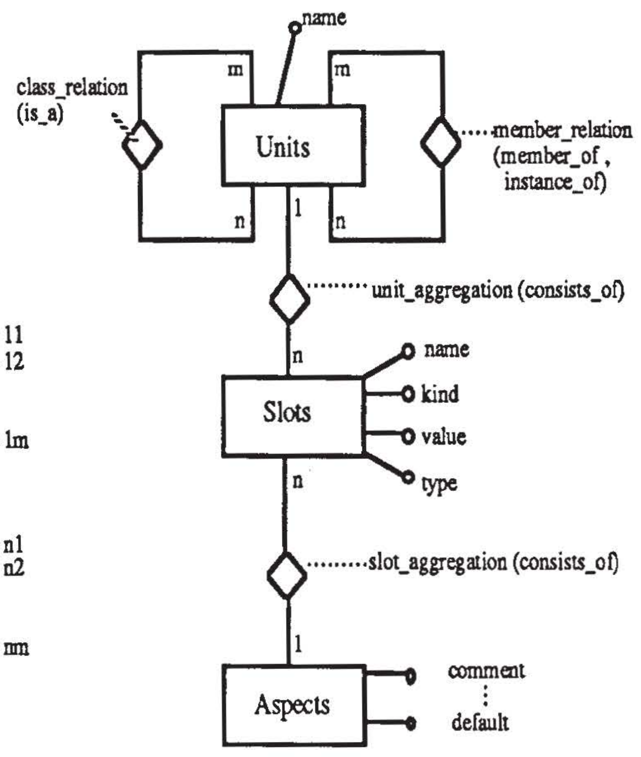
Table 2.1: Dynamics in molecular objects

As a reference example, we have chosen the Frame model [Mi75,FK85, BS83], which is one of the essential knowledge representation techniques used in artificial intelligence, especially in expert-system applications. Fig. 2.3 contrasts the structure of a generic frame (here called unit) with its corresponding ER modeling.

The primitive elements, i.e. the objects of our Frame model, are called units. Each unit has a name for identification purposes and consists of a number of attributes, called slots, which, in turn, consist of a list of aspects. Slots are used for describing the unit they belong to. Aspects, in turn, are needed for proper specification of the slot and its value. Our ER diagram depicts only one aspect occurrence for each slot, because all specifications for one slot are grouped within one aspect occurrence. So, it is possible to share this information, preventing redundancy. In terms of the above introduced abstraction concepts, we could say that units are formed by an aggregation of slots and aspects (this is explicitly depicted in our ER diagram of fig. 2.3 using semantically enriched relationship types). There are two types of units distinguishable: a unit is called a class-object if it represents either an object type or a set object, and its instance objects or element objects are called member-objects. Additionally, there are two quite different relationships between member- and class-objects: the member_relation relates member-objects to class-objects and therefore corresponds to the concept of association and classification; whereas, the class_relation associates only class-objects with each other, thus forming a generalization hierarchy. The first four slots in fig. 2.3 are used for modeling these two relations: is_subclasss_of and has_subclasses form the class_relation whilst is_member_of and has_members express the member_relation.

unit_name			
is_subclass_of :		list of units	
has_subclasses:		list of units	
is_member_of:		list of units	
has_members:		list of units	
slot 1:	type 1	kind 1	value 1
		aspect 11	aspect_value 11
		aspect 12	aspect_value 12
		.	.
		aspect 1m	aspect_value 1m
.			
slot n:	type n	kind n	value n
		aspect n1	aspect_value n1
		aspect n2	aspect_value n2
		.	.
		aspect nm	aspect_value nm

structure of a unit



ER diagram

Figure 2.3: Modeling the Frame model

Inherent to all frame models is the concept of inheritance, that is, information about the units is being transported along the generalization hierarchy. Therefore, we have to distinguish class-slots describing properties of the corresponding unit from member-slots which are used as attributes of all associated member-objects. Thus, it follows that member-slots have to be inherited along the subclass hierarchy as member-slots and along the member_relation as class-slots, whereas class-slots are never inherited. This enforces the attachment of a type attribute (declaring the slot a member- or class-slot) and of a kind attribute (stating whether it is an owned or an inherited slot).

In managing application 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. For example, fetching the boundary representation of a complex 3D object, fetching all maps within a clipping section, or fetching the generalization hierarchy of a specified frame object. 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 much more frequent than compared to **horizontal access**. The latter derives all objects of a common type, i.e., all stored maps, circuits, solids or units, perhaps satisfying some special qualification criteria.

Summarizing all above mentioned object characteristics, we can argue that the essential modeling requirements of an adequate information model that offers a unified view to design data and knowledge representation should comprise object abstractions and object dynamics, as well as operational support for vertical and horizontal access, which together define some kind of object orientation. Here, we neither want to concentrate on the definition of an appropriate ER model nor on the definition of the notion 'object orientation'. Instead, we focus on data models capable of accurate and efficient modeling of this sketched information model.

2.2 Mapping Complex Objects to Data Objects

Existing data models do not match the above mentioned requirements properly. When modeling in an hierarchical manner, one has to cope with redundancy. This holds for the classical data models like IMS [Mc77] as well as for novel ones such as non-first-normal-form models [SS86, RKB85] and the so-called complex-object model [LK84]. Fig. 2.4 illustrates such a consequence using our BREP schema. A first observation is that the hierarchical schema is semantically not equivalent to the network schema, because the hierarchical one is not symmetric; there is a loss of information concerning the bottom-to-top access. Secondly, 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 of fig. 2.4. On the right-hand side of fig. 2.4, we have shown the desired modeling approach, referred to as direct and symmetric modeling, thus avoiding the above mentioned problems, which are even more valid in the case of recursive objects (frame nets, map hierarchies, etc.).

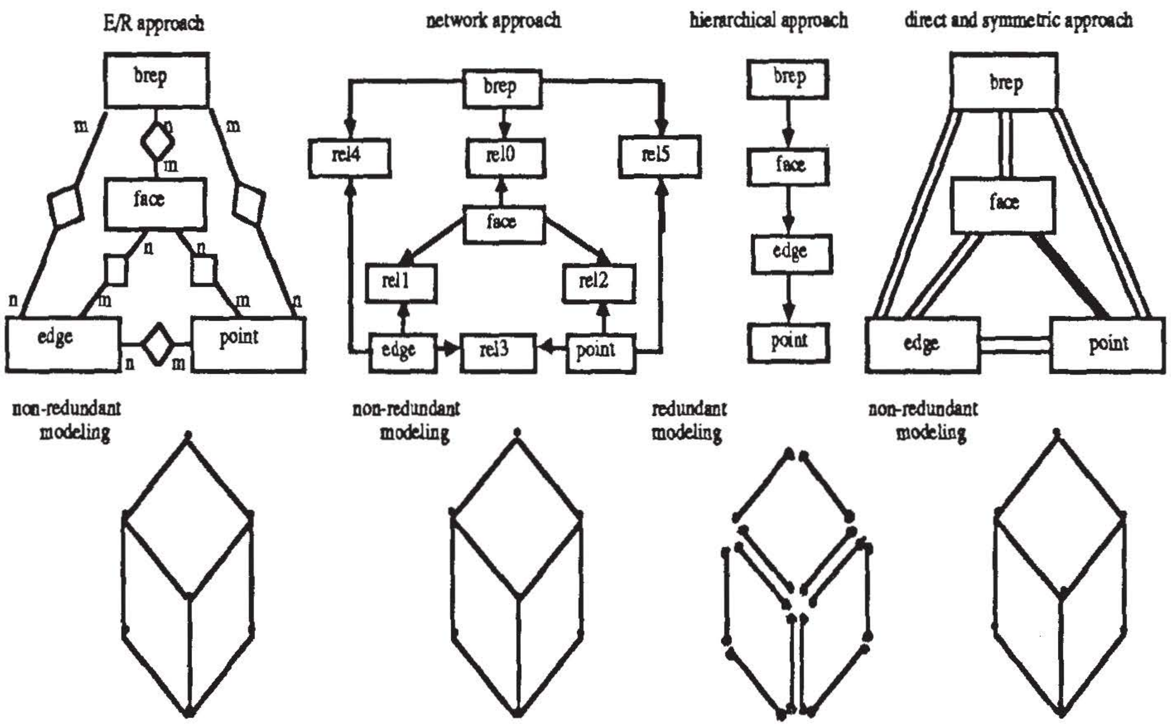


Figure 2.4: Modeling approaches to boundary representation

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

- direct and symmetric management of network structures and recursiveness
- mapping and operational support for all abstraction concepts
- 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 complex objects, i.e. some kind of object-orientation comprising
  - modeling techniques for 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 and 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 (record) sets

- 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 (Molecule-Atom Data model), that will be described in the next chapter.

3. The Molecule-Atom Data Model

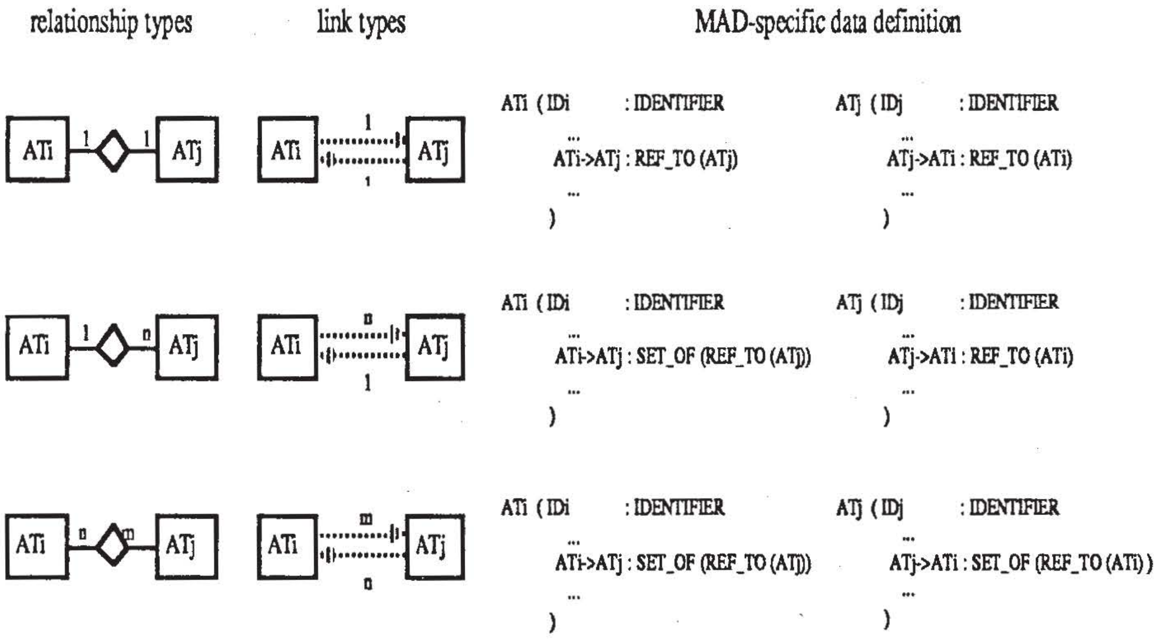
In the following, we present an introduction to modeling, as well as operational aspects of the MAD model. For this purpose, we use the examples of fig. 2.1 and fig. 2.3 and the syntactical simplicity of an SQL-like language as an explanatory vehicle. Some special syntactical expressions used in the **Molecule Query Language (MQL)** are taken from [PA86, RKB85, X3H286]. A formal description of MAD resp. MQL yielding to the so-called Molecule Algebra is unter completion. The focus of this chapter is primarily on discussing and illustrating the major capabilities of the MAD model and its language MQL. All implementation concepts are summarized in another published paper [HMMS87] describing the prototype implementation of MAD, called PRIMA. A detailed view to the query evaluation concepts within PRIMA is under completion and will be submitted for publication.

3.1 Underlying General Concepts

The most primitive elements of the MAD model are called **atoms**. They are comparable to tuples known from the relational model. According to the relational model, each atom is composed of attributes of various types, is uniquely identifiable, 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. Compared to the relational model, we have the following enhancements:
- IDENTIFIER type and REFERENCE type,
- RECORD type and ARRAY type as well as
- the repeating-group types SET and LIST.

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

Combining the attribute type REFERENCE with the repeating-group type SET is sufficient to express all kinds of relationship types between two atom types. This has been sketched in figure 3.1. For example, the declaration of an 1:n relationship type is expressed by two REFERENCE-based attributes (dotted arrows), one in each atom type. Thus, it is obvious that reference attribute and corresponding 'back reference' attribute define the relationship types (i.e., link type) in a direct and symmetric manner, as required.



```
ATi (IDi      : IDENTIFIER              ATj (IDj      : IDENTIFIER
     ...                                     ...
     ATi->ATj : REF_TO (ATj)                 ATj->ATi : REF_TO (ATi)
     ...                                     ...
    )                                       )

ATi (IDi      : IDENTIFIER              ATj (IDj      : IDENTIFIER
     ...                                     ...
     ATi->ATj : SET_OF (REF_TO (ATj))        ATj->ATi : REF_TO (ATi)
     ...                                     ...
    )                                       )

ATi (IDi      : IDENTIFIER              ATj (IDj      : IDENTIFIER
     ...                                     ...
     ATi->ATj : SET_OF (REF_TO (ATj))        ATj->ATi : SET_OF (REF_TO (ATi))
     ...                                     ...
    )                                       )
```

Figure 3.1: Expressing relationship types in terms of link types

Based on the link concept, it is feasable to dynamically construct **molecules** using atoms as elementary building blocks. Each molecule belongs to its corresponding molecule type. The **molecule type** is defined (in the query language, not in the schema) by naming the atom types and link types. Each molecule type determines both the **molecule structure** and the corresponding **molecule set**, grouping all the molecules with the same structure. The molecule structure is superimposed dynamically on sets of atoms 'linked' by references, thus introducing the concept of **dynamic molecules**, i.e., the required object orientation of the MAD model. Its operational power lies in adequate means for molecule processing provided by MQL. The overall design goal was the consistent extension of processing homogeneous to processing heterogeneous record sets, dynamically defined by molecules.

3.2 Modeling Concepts and Data Definition

Transformation of an Entity-Relationship schema to an equivalent MAD schema is straightforward. To exemplify this task, we have modeled the four cases of molecular objects of fig. 2.1 in terms of the MAD model shown in fig. 3.2. Details of such a transformation and two examples of the data definition language (DDL) are illustrated by fig. 3.3 and fig. 3.4. The transformation replaces all entity types by corresponding atom types and all relationship types by link types (which are built between the resp. atom types). If there is a relationship type having some attributes, we additionally have to introduce one atom type representing this relationship and its attributes as well as two link types to model the relations. The usefulness of the extended type concept and the cardinality restrictions associated with the SET type (i.e. exact mapping of relationship types allowing for refined system-enforced structural integrity) are also illustrated in fig. 3.3 and fig. 3.4.

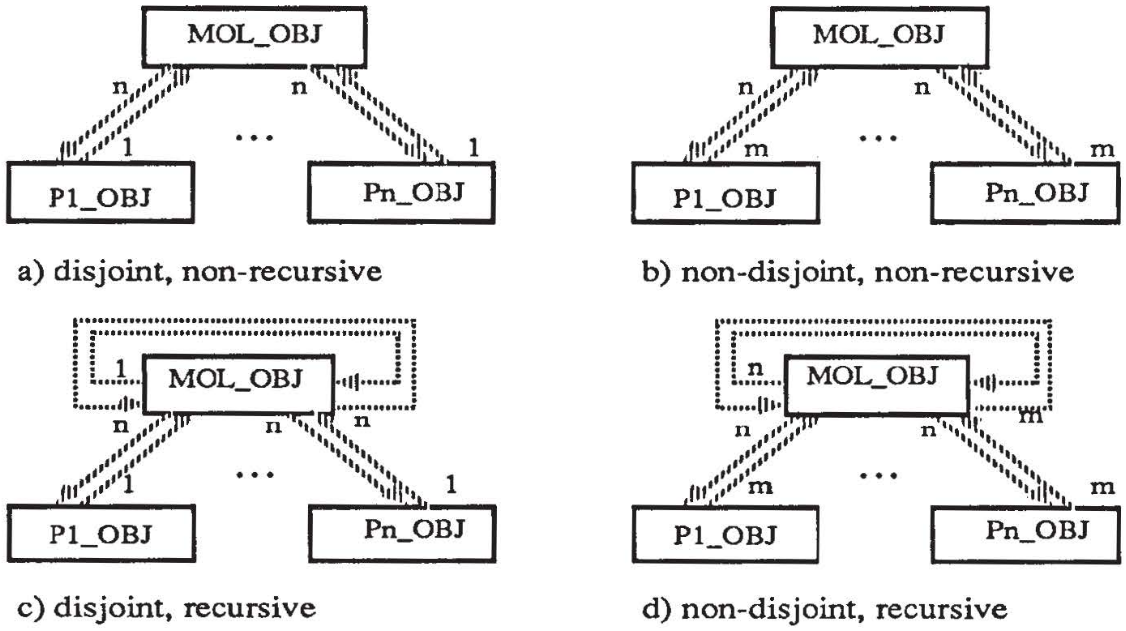


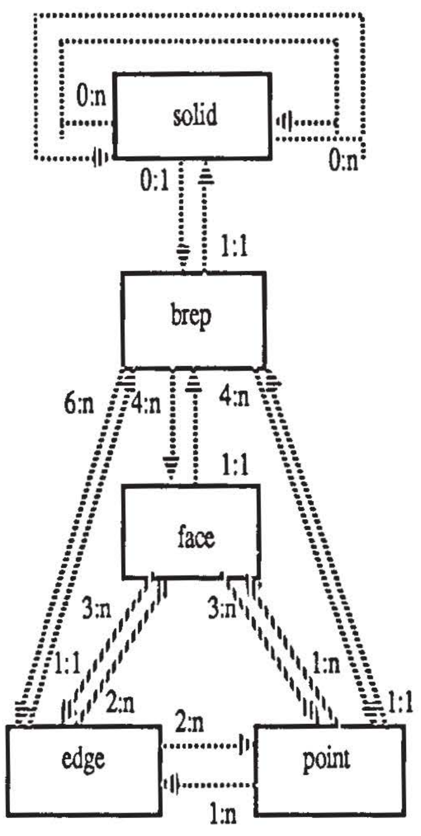
Figure 3.2: Modeling the four cases of molecular objects by means of the MAD model

The atom-type definition consists of three parts: naming, attribute definition, and key definition (fig 3.3b and fig. 3.4b). The atom-type name must be unique within the whole schema definition. Each attribute definition, in turn, consists of the attribute name (unique only within its atom-type) and the attribute data type. For identification and connection of atoms, each atom-type definition must contain exactly one attribute of the

IDENTIFIER type. Key definitions are simply expressed by lists of attribute names.

Although molecules are generally defined as part of a query, it is allowed to give a name to often used molecule types (fig. 3.3c and fig. 3.4c). A molecule-type definition (similar to a view) specifies the molecule-type

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)) (0,VAR),
     super        : SET_OF (REF_TO (solid.sub)) (0,VAR),
     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,
     boundary     : 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 : REAL,
                       y_coord : REAL,
                       z_coord : REAL
                    END
     line         : SET_OF (REF_TO (edge.boundary)) (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   comp_hier(solid)
       (RECURSIVE: solid.sub-solid)
```

Figure 3.3: Solid representation expressed in terms of the MAD-DDL

name, its corresponding structure, and its molecule set. In the FROM-clause, the constituent molecule subtypes connected by the selected link types are listed (in case of ambiguity the reference attribute has to be denoted, otherwise it is optional and could be substituted by some simple syntactical signs (braces or lines)). By means of the RECURSIVE-clause we are also able to define recursive molecule types. Thus, the molecule structure becomes cyclic and the derivation of the corresponding recursive molecules has to be performed stepwise in an iterative manner. The WHERE-clause additionally allows for the restriction of the molecule set. In the second statement of fig. 3.4b we have restricted the molecule type 'subclasses_of_id123' to exactly one recursive molecule, that with IDENTIFIER value '123' of its root atom. FROM-, WHERE- and RECURSIVE-clause are explained in the next chapter in more detail.

Fig. 3.3 and fig. 3.4 have outlined the direct and symmetric mapping of the abstraction concepts by means of the link concept of the MAD model. Here it becomes apparent that the MAD model supports (at least structurally) the above introduced essential characteristics of the complex (application) objects to its full extent. To illustrate the operational support as well, we now focus on the molecule processing, i.e. query and data manipulation facilities.

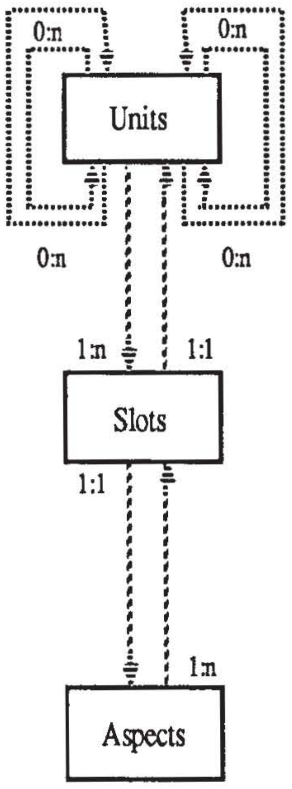
3.3 Query Facilities

The query capabilities of the molecule query language MQL comprise vertical as well as horizontal access (cf. chapter 2). Fig. 3.5 shows some hand-picked query examples as well as the corresponding molecule diagrams showing the molecule structure of the whole query and its finally projected subpart (encircled area). The examples of fig. 3.5 refer to the solid-representation schema of fig. 3.3. All subsequent explanations use a simple mental model underlying the query evaluation:
• Firstly, all molecules defined in the FROM-clause are assembled.
• Then the optional FROM-clause restricts this result set evaluating all qualification terms specified.
• Finally, the projection has to be performed, that fixes the final molecule stucture cutting away all unused molecule components.

Obviously, all three clauses are used in a quite similar sense compared to their equivalents in SQL-like languages. In the following, these evaluation steps are explained in more detail.

Vertical access to a network structure is illustrated in the query of fig. 3.5a. Firstly, all atoms constituting the brep molecules defined in the FROM-clause are assembled. This starts with the brep atoms and uses all specified links to deduce the dependent face, edge, and point atoms. Then

a) MAD-schema diagram



b) atom-type definitions

```
CREATE ATOM_TYPE units
   ( unit_id            : IDENTIFIER,
     name               : CHAR_VAR,
     is_subclass_of     : SET_OF (REF_TO (units.has_subclasses)) (0,VAR),
     has_subclasses     : SET_OF (REF_TO (units.is_subclass_of)) (0,VAR),
     is_member_of       : SET_OF (REF_TO (units.has_members)) (0,VAR),
     has_members        : SET_OF (REF_TO (units.is_member_of)) (0,VAR),
     unit_aggregation   : SET_OF (REF_TO (slots.is_slot_of)) (0,VAR))
   KEYS_ARE (name);

CREATE ATOM_TYPE slots
   ( slot_id            : IDENTIFIER,
     name               : CHAR_VAR,
     type               : (M,C),
     kind               : (O,I),
     value              : BYTE_VAR
     is_slot_of         : REF_TO (units.unit_aggregation),
     slot_aggregation   : REF_TO (aspects.is_aspect_of));

CREATE ATOM_TYPE aspects
   ( aspect_id          : IDENTIFIER,
     name               : CHAR_VAR,
     comment            : CHAR_VAR,
     value_set          : BYTE_VAR,
     cardinality_min    : INTEGER,
     cardinality_max    : INTEGER,
     metric_units       : CHAR_VAR,
     default            : BYTE_VAR,
     is_aspect_of       : SET_OF (REF_TO (slots.slot_aggregation)) (0,VAR))
   KEYS_ARE (name);
```

c) molecule-type definitions

```
DEFINE MOLECULE_TYPE unit_obj
FROM   units - slots- aspects
```

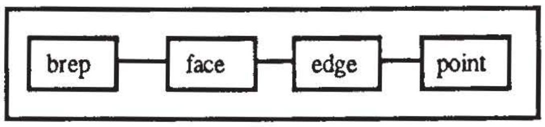
```
DEFINE MOLECULE_TYPE sub_classes_of_id123
FROM   subordinate_classes
       (unit_obj)
       (RECURSIVE: units.has_subclasses-units)
WHERE  subordinate_classes.units(0).id='123
```

Figure 3.4: Frame model expressed in terms of the MAD-DDL

the WHERE-clause restricts this result set by evaluating all existing qualification terms. The result set of this query comprises only those brep molecules holding a brep atom having the value '1713' for attribute 'brep_no'. In fig. 3.5b retrieval of a recursive structure is specified with piece_list as a predefined recursive molecule type (cf. fig. 3.3c). 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, thereby avoiding multiple evaluation of the same recursion path (cf. chapter 3.4 especially fig. 3.8).

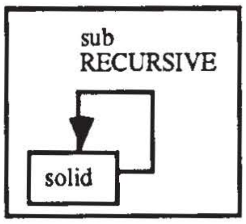
a) vertical access to network molecules

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



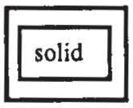
b) vertical access to recursive molecules

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

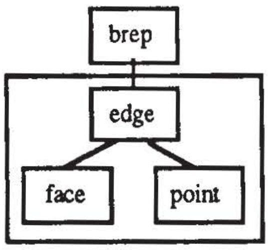


Figure 3.5: Some hand-picked query examples

Finally, fig. 3.5c shows some kind of horizontal access already known from the relational model: Here, we want to retrieve all primitive solids, i.e. solids not having any subpart-hierarchy. This query shows the use of a projection expressed in the SELECT-clause: All atoms resp. attributes listed are being retrieved. In our example the projection consists of the two attributes 'solid_no' and 'description'. Thus the corresponding result set comprises solid atoms, each holding two attribute values, one for attribute 'solid_no' and the other one for attribute 'description'. Some other important features are illustrated in fig. 3.5d. Arbitrarily structured molecule types are definable by means of brace-expressions; branching as well as combination of the graph underlying the molecule structure is done using brace-expressions. In our example, the FROM-clause shows the definition of a tree-like molecule type. The corresponding molecule structure is shown in the molecule diagram on the right-hand side. Another important feature illustrated in this example refers to molecule restriction. Using quantified qualification terms greatly enhances the expressiveness of the WHERE-clause. Our example shows an EXISTS-quantifier testing for the existence of at lèast 2 edges that satisfy the 'length' qualification. The ALL-quantifier could also be used as qualification term. The third special feature shown in the example refers to a finer grained projection capability. For proper specification of the final result set of the whole query, we use the **qualified projection** expressed as just another 'SELECT...FROM...WHERE' expression within the SELECT-clause. This nesting allows for a supplementary restriction of the components of the result-set molecules (not of the result set itself) by evaluating all qualification terms (i.e., WHERE-clauses) stated. Referring to our example, only those face atoms of the projected 'edge(face,point)' molecule are finally retrieved, whose square_dim value satisfies the qualification. Exploiting these capabilities, we are able to retrieve only those components of the 'surrounding' result-set molecules we are interested in.

3.4 Manipulation Facilities

Based on these flexible query facilities, we now sketch the remaining parts of molecule management: Similar to retrieval capabilities, insert, delete, and modify operations enable us to deal with an integral molecule as well as its components. All component operations have to be expressed by means of the previously introduced qualified resp. unqualified projection. 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 inversely for the insert statement. Modification especially supports, apart from update, connection and disconnection of molecule components.

For explanatory purposes, we have expressed in fig. 3.6 and 3.7 two typical Frame operations by means of MAD operations. The 'insertion of a member-slot' shown in fig. 3.6 is a quite complex operation which has been expressed using five distinct MAD operations. The first operation inserts an atom representing the aspect 'wheels'. The second operation delivers the IDENTIFIER value of the previously inserted aspect atom. This value is needed in the subsequent operations for properly setting the reference to the aspect atom, thus guaranteeing shared (aspect) atoms. These operations are quite similar to those known in relational languages. The third operation retrieves the whole recursive molecule type 'unit_hierarchy' composed of the root unit named 'automobile', its subordinate class hierarchy, their associated member units together with all linked slots. Fig. 3.8 shows two expamples of retrieved unit hierarchies. Within the application program this hierarchy is being modified to reflect the addition and inheritance of the member-slot named 'wheel_number', as exemplified in the fourth operation. Now the database has to be actualized. This is done via the modification operation (operation five), which changes the specified molecule or molecule set according to its data contents. That is, all atoms already existing within the database are being modified in accordance to the given data and all atoms not yet stored were inserted and connected to the right parents. Here, only the inherited member-slots and the previously added root member-slot have to be inserted and connected to their corresponding subclasses. (N.B.: When using all existing MQL capabilities it is also possible to express the inheritance of slots directly in one MQL statement).

Example 1: Insertion of the member-slot named 'wheel_number'
to the unit called 'automobile'

1. Insertion of the aspect data

```
INSERT aspect_name := 'wheels',
          comment := 'a solid disk or circular frame turning on a central axis',
          value_set := 1..25
          cardinality_min := 1,
          cardinality_max := 1,
          metric_units := 'none',
          default := 3 : aspects
FROM   aspects
```

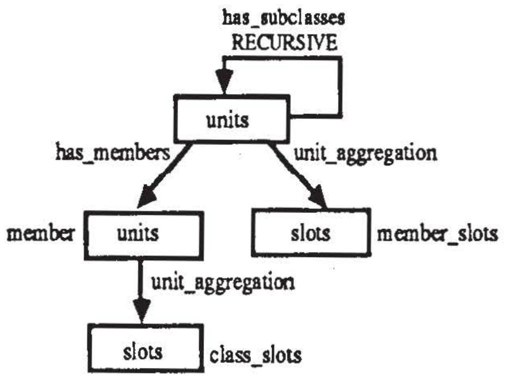
2. Retrieval of the IDENTIFIER value of the previously inserted aspect atom

```
SELECT aspect_id
FROM   aspects
WHERE  name = 'wheels'
```

3. Retrieval of the whole subclass hierarchy and all referenced member units

```
SELECT   ALL
FROM     unit_hierarchy
           (units-(.units_aggregation-member_slots(slots),
                 .has_members-member(units).unit_aggregation-class_slots(slots))
           (RECURSIVE: units.has_subclasses-units)
WHERE    unit_hierarchy.units(.0).name = 'automobile'
```

4. Modification within the application program:
   - . addition of the member-slot to the root unit, that is the unit named 'automobile'
   - . inheritance of member-slots to all subordinate units (class-units)
   - . inheritance of class-slots to all subordinate members



5. Actualization of the database

```
MODIFY unit_hierarchy
FROM   unit_hierarchy
```

Figure 3.6: Insertion of a member-slot expressed in terms of the MAD-DML

The 'deletion of a member-slot' shown in fig. 3.7 is just the inverse operation of the one discussed. In contrast to the above, this Frame operation is formulated by one single MQL statement to exhibit the expressiveness of MQL. In fig. 3.7 we use, for explanatory purposes only, a kind of macro technique to construct the final MQL statement. The first part of fig. 3.7 predefines the molecule type to work with, by means of the DEFINE MOLECULE_TYPE statement (cf. section 3.2). A first subquery is needed to pick up the IDENTIFIER values of all member-slots (the 'owned' slot at the root of the hierarchy is included). All class-slots were retrieved in a similar subquery. Finally, the fourth part of fig. 3.7 depicts the resulting MQL statement. All slot molecules (i.e. slot atoms and linked aspect atoms) named 'wheel_number' whose slot IDENTIFIER value is contained within the union of the previously mentioned two subqueries (i.e. only slots contained within the unit_hierarchy are considered) are deleted from the corresponding atom types (i.e., removed from the database) and disconnected from their units. Thus, the member-slot named 'wheel_number' belonging to the unit called 'automobile' and all its inherited member- and class-slots together with their aspect atoms were deleted. This fact is depicted in fig. 3.8 showing a typical scenario. For sake of simplicity no aspect atoms are shown. All unit atoms belonging to the unit_hierarchy of unit u1 are marked with bold rectangles and all slot atoms inherited from slot s4 are drawn in dotted circles. These are the candidates for deletion. All 'has_subclasses' references are marked with 's' and all 'has_members' references are marked with 'm'. The pre- and post-state diagrams of the two above mentioned Frame operations are depicted.

Common to all manipulation operations is the system-enforced support for structural integrity, i.e. modifying a REFERENCE attribute implies the automatic maintenance of the corresponding 'back-reference'.

Summarizing the above introduced and illustrated concepts, the operational support of the MAD model (i.e. MQL) for adequate object orientation comprises:

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

These concepts are prerequisites for

- molecule processing (i.e. insertion, deletion, modification, and retrieval of integral molecules) and
- molecule-component management (i.e. component insertion, deletion, modification, and retrieval using the FROM- and WHERE-clause to specify the surrounding molecules).

Example 2: Deletion of the member-slot 'wheel_number'
from the unit named 'automobile'

```
1. Definition of the molecule type to work with
   DEFINE MOLECULE_TYPE unit_hierarchy
   FROM   units_rec
            (units - (.units_aggregation-member_slots(slots),
                    .has_members-member(units).unit_aggregation-
                              class_slots(slots)))
            (RECURSIVE: units.has_subclasses-units)

2. Picking up the IDENTIFIER values of all member-slots
   SM ::= SELECT  member_slots.(ALL).slot_id
          FROM    unit_hierarchy
          WHERE   unit_hierarchy.units.(0).name = 'automobile'

3. Picking up the IDENTIFIER values of all class_slots
   SC ::=  SELECT  class_slots.(all).slot_id
           FROM    unit_hierarchy
           WHERE   unit_hierarchy.units.(0).name = 'automobile'

4. Deletion of the member-slot 'wheel_number' and
   all its inherited member- and class-slots
   together with the deletion of the linked aspect atom
   DELETE ALL
   FROM   slots.slot_aggregation-aspect
   WHERE  slot_name = 'wheel_number' AND
          slot_id ELMT (SM UNION SC)
```

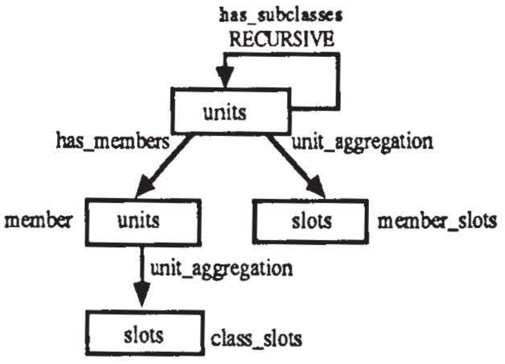


Figure 3.7: Deletion of a member-slot expressed in MAD-DML

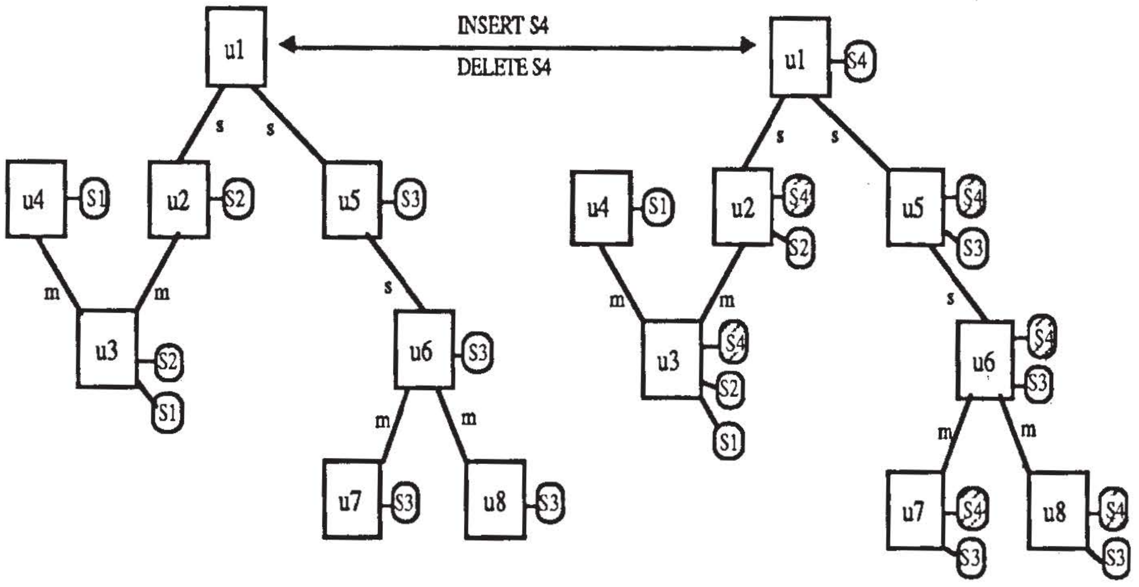


Figure 3.8: Two state diagrams of a unit hierarchy

Furthermore, it is apparent that the MAD model and its language MQL support vertical as well as horizontal access to its full extent. Moreover, it offers adequate operational capabilities for handling recursion, i.e. recursive molecules. The MAD model with its language MQL is not only designed for an interactive environment. Additionally, it is embedded in a host programming language. This application programming environment is used only by system engineers and not by novice users. Due to space limitations we cannot give a more detailed language discussion. An in-depth description of MQL can be found in [Mi86].

3.5 Load Definition for 'Transparent' Performance Enhancements

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

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

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

In the following, we show some examples sketching the expressive power of the LDL. All examples refer to the solid-representation schema of fig. 3.3 (in this context, these examples are not necessarily meaningful):

- Definition of physical clusters

```
        DEFINE STATIC_MOLECULE_TYPE brep_cluster
        FROM brep - face - edge - point
```

The language constructs we have chosen for the LDL are similar to those of MAD-DML. Here, we define a physical cluster comprising the previously defined 'logical' molecule type brep_obj (cf. fig. 3.3c). That is, all brep, face, edge and point atoms/records belonging to the same molecule of type brep_obj were clustered in one occurrence of type brep_cluster. This clustering technique is novel, in that it allows us to cluster heterogeneous record sets.

- Definition of access methods

```
DEFINE ACCESS solid_acc ON solid
FOR (solid_no)
USING B_TREE
```

An access path named solid_acc is defined on the 'solid_no' attribute of atom type solid and uses the implementation technique of a b-tree. Multidimensional access is defined in the same way by specifying some additional attributes within the FOR-clause.

- Definition of sort orders

```
DEFINE SEQUENCE edge_seq ON edge
USING (length ASC)
```

This statement defines an ascending sort order named edge_seq on the atom type edge, using its attribute 'length'.

4. Other Data Models

For the MAD model, we have advocated a symmetric and neutral approach allowing for more powerful and complex constructs than the flat relational model, but avoiding the bias on static data structuring and top down traversal of hierarchical models like IMS [Mc77], as well as novel ones as non-first-normal-form models [SS86,RKB85] and the so-called complex-object model [LK84]. As already pointed out (cf. section 2.2), one also has to cope with redundancy when modeling in an hierarchical manner. One special utilization of the powerful link concept of MAD is the construction of hierarchical structures and its support for non-first-normal-form relations. Consequently, the MAD model seems to contain a superset of the capabilities of these proposals. Because of the system-enforced integrity control of reference and 'back- reference' (i.e. link concept) a subset of the MAD model is equivalent to the relational model, including its referential integrity requirement, when all foreign/primary-key relations are modeled by links.

The ability of the MAD model in direct and symmetric management of network and recursive structures is a prerequisite for efficient mapping of all abstraction concepts comprising aggregation, generalization, association, and their hierarchies, as well as classification. Considering its dynamic object definition and management capabilities it seems that the MAD model is even more flexible than semantic and functional data models [Sh81,SS77,HM81,Da85], whilst offering a comparable powerful modeling ability.

Moreover, the MAD model is appropriate for the support of other knowledge representation models [BF81,ML85] such as the Frame model (introduced in section 2.1), that is, semantic nets [BS85] or rule-based representation techniques are conceivable.

Object-oriented models [DD86] are mostly characterized by their facilities comprising modeling and managing of meshed (or recursive) structures that are frequently viewed from different points, depending on the actual processing state. The link concept and the concept of dynamic molecules combined with the expressiveness of MQL seem approximately equal to these characteristics defining object orientation.

Comparing the MAD model to POSTGRES [RS87] is quite difficult. The POSTGRES data model is also a derivation of the relational model which has been enhanced by the ideas of abstract data types, data of type procedure, and inheritance. POSTGRES has been designed as a data model for a next generation extensible DBMS [SR86], whilst the MAD model does not claim to be extensible in that sense. Both models allow for object sharing, recursiveness, and dynamic object definition; both languages offered are SQL-like and very expressive. The concept of data type procedure looks quite powerful and resembles the link concept of MAD when the procedure expressing references to other objects has been evaluated, i.e. all procedure values have been precomputed. Though, differences between or inclusion of these two models concerning object modeling and management is subject to further, more detailed investigations. Moreover, the overall applicability of both models and their languages could only be determined by means of prototype systems, testing the expressiveness, adequateness, and performance of each approach.

Due to space limitation we cannot give a more detailed comparison. An in-depth discussion of the various models and their corresponding languages remains to be subject for a subsequent paper.

5. Conclusions and Future Plans

Adequate information modeling in non-standard areas comprises the abstraction concepts of classification, aggregation, generalization, and association. The MAD model presented in this paper offers generic mechanisms concerning accurate and efficient mapping of such an information model. 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 (probably recursive) objects at run time are considered prime prerequisites for accurate and effective modeling in non-standard applications where the 'view' of the object frequently changes. This symmetric and neutral approach to complex objects allows for the mapping of a wide variety of semantic and object-oriented modeling constructs including objects with shared subobjects.

The focus of the paper has primarily been on justifying the design decisions of MAD and on discussing and illustrating its major features, whilst its implementation concepts were summarized in another paper [HMMS87] describing the database kernel PRIMA (PRototype Implementation of the MAD model). A number of concepts used in the PRIMA implementation pay 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. The exploitation of these novel storage structures in combination with the set-orientation of MAD and its processing of heterogeneous record sets offers new areas for query optimization.

Currently, the single-user version of PRIMA is under completion. 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 leading, from a 'kernel-only' DBMS to a base system for workstation-host coupling [HHMM87], and a tool for building multi-processor DBMS [HHM86].

Acknowledgement
T. Härder, K. Meyer-Wegener, and Nelson Mattos have read an earlier version of this paper and have contributed to clarify and improve the presentation of important issues. Also acknowledged is the help of H. Neu and I. Littler for preparing the manuscript and of H. Molin for preparing the final version. The work of the referees and the program committee is also gratefully acknowledged.

6. References

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.

BF81 Barr, A., Feigenbaum, E.A.: The Handbook of Artificial Intelligence, Vol. 1, William Kaufmann, Inc., Los Altos, CA, 1981, pp. 142-222.

BS83 Bobrow, D., Stefik, M.: The LOOPS Manual, Palo Alto, California, Xerox, 1983.

BS85 Brachman, R.J., Schmolze, J.G.: An Overview of the KL-ONE Knowledge Represen- tation System, Cognitive Science, Vol. 9, 1985.

CD86 Carey, M.J., DeWitt, D.J., et al.: The Architecture of the EXODUS Extensible DBMS, in: [DD86], pp. 52-65.

Ch76 Chen, P.P.: The Entity-Relationship-Model - Toward a Unified View of Data, in: ACM TODS, Vol. 1, No. 1, 1976, pp. 9-36.

Da85 Dayal, U., et al.: A Knowledge-Oriented Database Management System, in: Proc. Islamorada Conf. on Large Scale Knowledge Base and Reasoning Systems, Feb. 1985.

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..

DD86 Dittrich, K.R., Dayal, U. (eds.): Proc. Int. Workshop on Object-Oriented Database Systems, Pacific Grove, 1986.

DKM86 Dittrich, K., Kotz, A., Mülle, J.: Database Support for VLSI Design: The DAMASCUS System, in: CAD interfaces and data transfer formats in electronics, Springer Verlag, 1986.

FK85 Fikes, R., Kehler, T.: The Role of Frame-based Representation in Reasoning, in: Communications of the ACM, Vol. 28, No. 9, Sept. 1985, pp. 904-920.

Hä86 Härder, T.: New Approaches to Object Processing in Engineering Databases, in: [DD86], p. 217.

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 Comp. Sciences, Aachen, 1986, pp. 385-392.

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

HM81 Hammer, M., McLeod, D.: Database Description with SDM, in: ACM Trans. on Database Systems, Vol. 6, No. 3, 1981, pp. 351-386.

HMMS87 Härder, T., Meyer-Wegener, K., Mitschang, B., Sikeler, A.: PRIMA - A DBMS Prototype Supporting Engineering

	Applications, in: Proc. 13th VLDB Conf., Brighton, UK, 1987, pp. 433-442.
HMP86	Härder, T., Mattos, N., Puppe, F.: On Coupling of Database and Expert Systems (in German), in: State of the Art, Vol. 1, No. 3, 1986, pp. 23-34.
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 Sysem IMS/VS, in: IBM Systems Journal, Vol. 16, No. 2, 1977, pp. 84-168.
Mi75	Minsky, M.: A Framework for Representing Knowledge, in: The Psychology of Computer Vision (editor: Winston, P.), McGraw-Hill Book Company, 1975.
Mi86	Mitschang, B.: MAD - A Data Model Managing Complex Objects (in German), SFB124 Research Report No. 20/85, Univ. Kaiserslautern (revised Summer 1986).
ML83	Meier, A., Lorie, R.: A Surrogate Concept for Engineering Databases, in: Proc. 9th VLDB Conf., Florenz, 1983, pp. 30-32.
ML85	Mylopoulos, J., Levesque, H.J.: An Overview of Knowledge Representation, in: On Conceptual Modelling (eds. Brodie, M.L., Mylopoulos, J., Schmidt, J.W.), Topics in Information Systems, Springer-Verlag, 1985, pp. 3-17.
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, in: ACM SIGMOD Conf., San Francisco, 1987, pp. 196-207.
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.
RS87	Rowe, L.A., Stonebraker, M.R.: The POSTGRES Data Model, in: Proc. 13th VLDB Conf., Brighton, 1987, pp. 83-96.
Sh81	Shipman, D.: The Functional Model and the Data Language Daplex, in: ACM Trans. on Database Systems, Vol. 6, No. 1, 1981, pp. 140-173.
SR86	Stonebraker, M., Rowe, L.A.: The Design of POSTGRES, in: Proc. ACM SIGMOD Conf., Washington, D.C., 1986, pp. 340-355.

SS77 Smith, J., Smith, D.: Database Abstractions: Aggregation and Generalization, in: ACM Trans. on Database Systems, Vol. 2, No. 2, 1977, pp. 105-133.

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