Extending the Relational Algebra to Capture Complex Objects

Bernhard Mitschang
University of Kaiserslautern, 6750 Kaiserslautern, FRG

Abstract

An important direction in database research for non-standard applications (e.g. engineering or design applications) deals with adequate support for complex objects. Without doubt, the provision of network structures and shared subobjects as well as support for dynamic object definition and appropriate manipulation facilities is urgently needed for natural and accurate modeling as well as for efficient processing of the applications' objects. These concepts are the major concern of the molecule-atom data model (MAD model) and its molecule algebra which is introduced in this paper. They make the model stand out compared to the relational model and even to models limited to hierarchical and statically defined complex objects. By means of the molecule algebra a precise and complete specification of one conceivable kind of complex object processing and its inherent semantics is provided. Furthermore, this algebra is used as a sound basis to express the semantics of the high level query language MQL (molecule query language) that is able to deal with complex objects in a descriptive manner.

Keywords:

complex objects, structural object orientation, relational algebra extensions, formal specification

1. Motivation

Over the last few years 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 design, knowledge-based systems, and office applications has emerged as an important direction in database system research.
One uniting characteristic over all these advanced applications is adequate support for complex objects, which is quite different to conventional data processing in business applications. Thus, most current research topics refer to some kind of object-orientation and extensibility reflected in the data model and in its implementation. There are different approaches which are distinguishable starting from only a few selected extensions of the relational model [RS87,LK84] and leading up to the integration and superposition of structures on relations [Da86, PSS-WD87,CDV88].
Though many of these proposals are interesting, they are mostly limited to hierarchical complex objects, which are in addition in most cases statically defined (i.e. fixed in the database schema). As stated in [BB84] *), the

- provision of shared subobjects, and
- support for dynamic object definition (of course in combination with powerful manipulation facilities)

is urgently needed for natural and accurate modeling and efficient processing of the applications' objects.
In this paper we attack the above mentioned problems by means of the molecule-atom data model (MAD model [Mi88b]), which is an advancement to the relational model and is based on a generalized notion of complex objects. These objects are called molecules and are dynamically constructed from atoms, which are used as basic building blocks. Since molecules can overlap having non-disjoint atom sets, the model allows in a natural way for the sharing of subobjects. With the relational model in mind, chapter two offers an easy to understand introduction to the relevant concepts underlying the MAD model. When dealing with these network-like structures it is necessary to rely on a sound and precise definition of the model's data structures and its operations. Since a formal specification of the data model seems to be indispensable, we introduce in chapter three the so-called *molecule algebra* capable of handling complex objects exhibiting network structures and shared subobjects. With that, a formal description of complex object processing (here, simply called molecule processing) and its inherent semantics is provided. Thus, the molecule algebra appears to be an extension to the relational algebra [UI80] and also to the non-first-normal-form algebra [SS86] that supports only hierarchical complex objects without shared subobjects. In the fourth chapter, we show how such a formal specification (here, it is the molecule algebra) is used as a sound basis to define the semantics of a higher level query and manipulation language (here, it is the molecule query language MQL). Finally, we conclude with a short comparison to other models and an outlook to upcoming future research topics. As far as is known to the author, no satisfying complex-object algebra (capable of handling network and recursive structures as well as dynamic object definition) has been proposed until now.

*) [BB84] '... support for molecular objects should be an integral part of future DBMSs ...', where 'molecular' objects are classified according to their structure, leading to disjoint/non-disjoint and recursive/non-recursive complex objects.

Proceedings of the Fifteenth International Conference on Very Large Data Bases

Amsterdam, 1989

2. An Informal Introduction to the MAD Model

Throughout the paper we use a simple example from a geographic application area for further explanatory purposes. As shown in the upper part of fig.1, we concentrate on a cartographic view to our universe of discourse (here, Brasil, its states, cities, rivers etc.), which is also modeled by means of the well-known ER model. The corresponding ER diagram depicts a geographical model as part of the schema that is used to share all geographic structures (i.e. points to build up edges, which are in turn used to construct areas and nets) among all application objects (e.g. cities, states, rivers etc.) thereby avoiding any data redundancies: point-like objects (e.g. city), network-like objects (e.g. river, street, flight), and area-like objects (e.g. state) are modeled by means of this common geographical model. Thus, different complex objects are contained in one schema sharing common subobjects. For example, the river Parana shares with the states Minas Gerais, Sao Paulo, and Parana some edge and point tuples representing in one case the course of the river and in another case the border of the states. This sharing of components is expressed by the n:m relationship types. It is easy to imagine that a transformation to the relational model becomes quite cumbersome, since all n:m relationship types have to be modeled by some auxiliary relations. With this, the queries and their processing obviously become more complicated and perhaps less efficient. To overcome these problems and to provide an effective complex object concept, we have developed the MAD model as an extension of the relational model.

In the MAD model *atoms* are used as a kind of basic element to represent the real world entities. They play a similar role to tuples in the relational model. Each atom consists of attributes of various data types, is uniquely identifiable, and belongs to its corresponding *atom type*. Contrary to the foreign/primary key connections of the relational model, all relevant relationships between the entities are expressed by so-called *links* that are defined

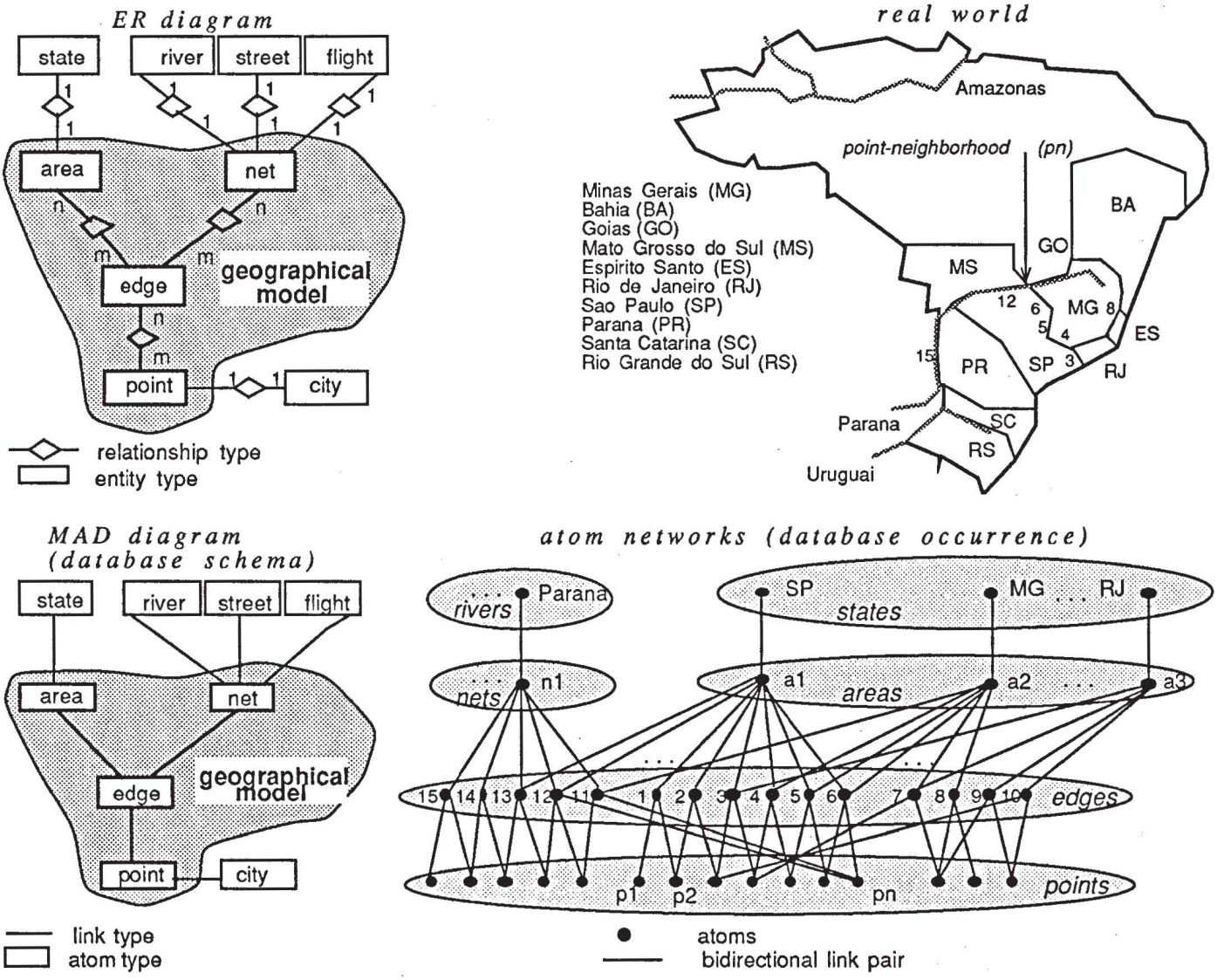

Figure 1: Sample geographic application

as *link types* between the corresponding atom types. Link types are used to accurately map all types of relationships (1:1, 1:n, and n:m). (Note, the link concept is a logical concept and shall not be confounded with physical storage structures). The direct representation and the consideration of bidirectional, i.e. symmetric links establish the basis of the model's flexibility. Hence, in the database all atoms connected by links form meshed structures, called *atom networks*. The corresponding *database schema* exhibits a set of atom types, whereof some are connected via link types establishing nondirectional graphs. The lower part of fig.1 shows these concepts applied to our cartographic application.

A closer look at the ER diagram and the corresponding MAD diagram in fig.1 reveals that there is a one-to-one mapping from the ER model to the MAD model associating each entity type with an atom type and each relationship type with a link type. Compared to the relational model, here we don't have to use any auxiliary structures.

Based on the atom networks the model's complex objects are dynamically definable as higher level objects seen as a structured (i.e. coherent) set of possibly heterogeneous atoms. In analogy to chemistry, we call these complex objects *molecules* to express more vividly that several atoms could 'combine' in different ways to build up the desired molecules. The 'formula' or the procedure to determine what kinds of molecules one is interested in is specified by means of the *molecule structure*. This description is a subgraph of the database schema, which establishes a coherent, directed and acyclic type graph, whose nodes are the atom types and whose edges are the link types. Each graph has a unique starting point called the root atom type. To each molecule structure, there exists a corresponding *molecule set*, which groups all molecules adhering to the specified structure. At least from the concept level point of view the *derivation of molecules* proceeds in a straight-forward way using the molecule structure as a kind of template, which is laid over the atom networks. Thus, for each atom of the root atom type one molecule is derived following all links determined by the link types of the molecule structure to the children, grandchildren atoms etc. till the leaves are reached. Derivation of the children atoms means performing the hierarchical join [LK84] along the specified branches. Both, the molecule structure together with its derived molecule set are denoted *molecule type*.

The flexibility of the MAD model stems from the fact that the same database (i.e. atom networks) can be used to derive totally different molecule types, just by specifying and deriving different molecule structures. This is shown in fig.2. The reason that it works well, firstly lies in the direct and bidirectional link concept allowing for a symmetric use of the database: e.g. looking at the 'point neighborhood', i.e. going from point to edge, and from edge to both area and state as well as to net and river, as exemplified in fig.2. Secondly, the database schema is primitive in the sense that it is not superposed by some static structures used, for complex object definition, as it is the case for example in [LK84,Da86]. Our complex object definition is defined on demand in the queries and not fixed in the schema. Therefore, we term our approach *dynamic object definition*. Since it is possible to combine molecules having non-disjoint atom sets, (which show a general graph structure, and not only a strict hierarchical one) the model allows in a natural way for the sharing of subobjects. This fact is especially depicted in fig.2. Of course, disjoint objects showing only hierarchical (graph) structures are just special cases thereof.

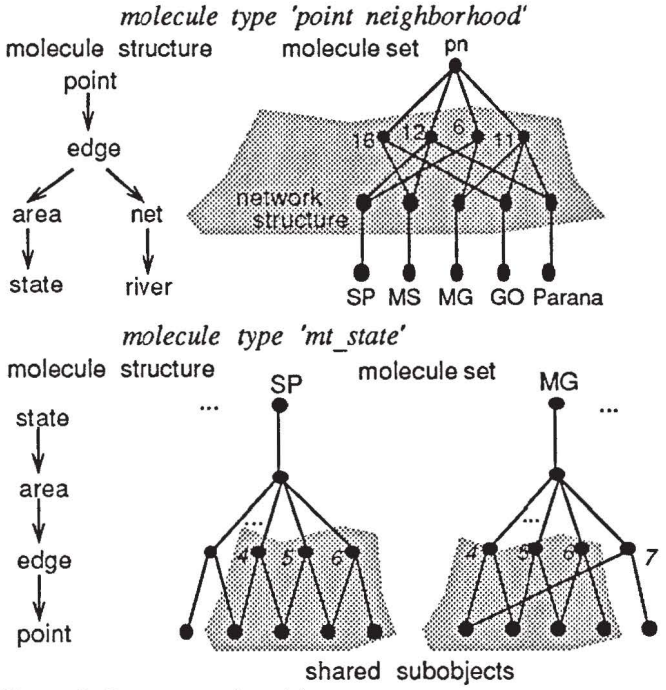

Figure 2: Some complex objects

3. The Molecule Algebra

In this chapter we provide a precise definition of the MAD model's data structures and of its algebra operations. These formal specifications will contain the relational model as well as the basic non-first-normal-form model (with some minor cuts) as degeneration. At a first glance, the reader might be astonished at the formalism introduced for more or less common facts. But in case of the MAD model and its inherent support for network structures and dynamic object definition, we cannot dispense with formal definition of basic concepts (e.g. type, structure or description, and occurrence information), because otherwise we could not define the effect of the algebra operators formally and we could not prove the closure of the molecule algebra. In the following we presume that the reader is familiar with the relational model and its formalization (e.g. [Ul80]) as well as with basic mathematical notations. Furthermore, we assume a given set N the elements of which will be used for naming purposes and the symbol ∇ indicating the end of definitions, theorems, and proofs.

3.1 Basic Concepts

Here we define the basic building blocks of the MAD model that is

- on the data structure side: atom and atom type, link and link type as well as atom network and database
- on the operational side: operations for projection, restriction, cartesian product, union, and difference, all working on atom types (and showing some effects on link types).

The data structures and the corresponding operations introduced here are very close to those known from the relational model. Therefore, we omit a detailed formal definition of well-known and common facts and concentrate on the aspects intrinsic to the MAD model.

Definition 1 *Atom Type*

The triple at=<aname,ad,av> is an *atom type* iff aname$\in$N, ad is a valid atom-type description, and av is a valid atom-type occurrence. A valid atom-type description consists of a set of attribute descriptions, and a valid atom-type occurrence is a subset of the description's domain, which is the cartesian product of the attribute domains used. An element of the atom type occurrence is denoted *atom* ∇

Obviously, there is a direct association to the relational model. Fig. 3 compares the MAD concepts against the well known relational concepts.

relational concepts	MAD concepts
attribute	attribute
attribute domain	attribute domain
relation schema	atom-type description
tuple set	atom-type occurrence
tuple	atom
relation	atom type
database	database
-	link
-	link-type description
-	link-type occurence
-	link type
referential integrity (?)	referential integrity(!)
'relation domain'	database domain

Figure 3: Comparison of corresponding concepts

Each atom type is uniquely identified by its atom-type name. AT* denotes the set of all correctly defined atom types. To deal with atom types easily, we define for each at$\in$AT* the following auxiliary functions:

- nam(at) = aname provides the atom-type name,
- des(at) = ad provides the atom-type description,
- ext(at) = av provides the atom-type occurrence, and
- atyp(aname) = at is inverse to the function nam with atyp(nam(at))=at.

If C is a set of atom-type names, the function atyp is extended in the following way:

atyp(C) = {at | at=atyp(aname) and aname$\in$C}.

Definition 2 *Link Type*

The triple lt=<lname,ld,lv> is a *link type* iff lname$\in$N, ld={aname$_1$, aname$_2$} with atyp(aname$_1$)=at$_1$, atyp(aname$_2$)=at$_2$, and at$_1$, at$_2\in$AT* as well as lv$\subseteq${l | l=<a$_1$,a$_2$> with a$_1\in$ext(at$_1$), a$_2\in$ext(at$_2$) and l is an unsorted pair}.

Here lname is called the link-type name, ld denotes the link-type description, and lv marks the link-type occurrence consisting of a set of elements called *links* ∇

Each link type is uniquely identified by its link-type name. LT* specifies the set of all correctly defined link types. The use of nondirectional link types (and links) integrates the symmetry of the relationships of the MAD model into our formalization. It is allowed to define several link types using the same two atom types as well as using only one atom type (reflexive link type). For example, when modeling the bill-of-material application with its super-component and sub-component view, we just have to define one reflexive link type called 'composition' on the atom type 'parts'. Exploiting the link type's symmetry it is now easy to evaluate either the super-component view or only the sub-component view. Analogously to above, we define for each link type lt$\in$LT* the following auxiliary functions:

- nam(lt) = lname provides the link-type name,
- des(lt) = ld provides the link-type description, and
- ext(lt) = lv provides the link-type occurrence.

Using definition 1 and 2, we are now able to define the notion of a database:

Definition 3 *Database*

Let AT$\subset$AT* be a set of different atom types and let LT$\subset$LT* be a set of corresponding link types. The pair DB=<AT,LT> specifies a *database* and DB*=<AT*,LT*> denotes the so-called database domain comprising all valid databases ∇

The definition of DB* is necessary, because all subsequently defined operations are closed under this database domain (see theorem 1). Each operation uses one or two atom types of a specified database and produces a new atom type with new link types that are all contained in a correspondingly enlarged database being part of the database domain. For the relational model its closure is defined in the same way: the result of each relational operation is a new relation that is contained in the 'relation domain' (see fig.3) comprising all valid relations.

Definition 4 *Atom-Type Operations*

Let DB=<AT,LT> be a database and let at=<aname,ad,av>, at_1=<$aname_1$,ad_1,av_1>, and at_2=<$aname_2$,ad_2,av_2> be atom types within DB, and ad, ad_1 and ad_2 are in pairs disjoint. Now the following operations are defined:

- atom-type-projection
  $\pi[proj(ad)](at)=<aname_\pi,ad_\pi,av_\pi>=at_\pi$
  with $proj(ad)\subseteq ad$, $aname_\pi \in N$, $ad_\pi=proj(ad)$, and
  $av_\pi=\{a_\pi \mid a_\pi=a(proj(ad))$ and $a\in av\}$
- atom-type-restriction
  $\sigma[restr(ad)](at)=<aname_\sigma,ad_\sigma,av_\sigma>=at_\sigma$
  with $restr(ad)\in$ qual-formulas(ad), $aname_\sigma \in N$, $ad_\sigma=ad$, and $av_\sigma=\{a \mid qual(restr(ad),a))$ and $a\in av\}$; qual is a predicate that decides whether the atom at hand fulfills the qualification condition restr(ad).
- cartesian product
  $\chi(at_1,at_2)=<aname\chi,ad\chi,av\chi>=at\chi$
  with $aname\chi \in N$, $ad\chi=ad_1\cup ad_2$,
  $av\chi=\{a\chi \mid a\chi=a_1\&a_2$ and $a_1\in ext(at_1)$, $a_2\in ext(at_2)\}$
  and '&' denotes the concatenation of the two given tuples.
- atom-type-union
  $\omega(at_1,at_2)=<aname_\omega,ad_\omega,av_\omega>=at_\omega$
  with $name_\omega \in N$, $ad_\omega=ad_1=ad_2$, $av_\omega=av_1\cup av_2$
  and '$\cup$' denotes the union of two sets.
- atom-type-difference
  $\delta(at_1,at_2)=<aname_\delta,ad_\delta,av_\delta>=at_\delta$
  with $name_\delta \in N$, $ad_\delta=ad_1=ad_2$, $av_\delta=av_1-av_2$
  and '-' denotes the difference of two sets ∇

The detailed definition of these atom-type operations should be clear, so we decided to leave it out. All atom-type operations have indirectly some effects on the link types: the link types of the operand atom types are 'inherited' to the resulting atom type. Thus, the result atom type could be reused in subsequent operations. In particular this is necessary for the molecule operations (see next section), since the dynamic molecule derivation relies on the existence of link types. In order to save space, we omit the formal definition of the inheritance of link types. The interested reader is referred to [Mi88a].
Based on the above introduced constructs, we are now able to set up and prove the following theorem:

Theorem 1 *Atom-Type Algebra*

The set of the atom-type operations π, σ, χ, ω, δ forms an algebra on the database domain DB*, i.e. the result of each atom-type operation is representable in DB* ∇

Proof:
(1) To prove that each result atom type is valid:
This is already proved by the result atom type's construction (cf. def. 4)
(2) To prove that all inherited link types are well-defined:
This follows directly from their careful construction (cf. [Mi88a]) ∇

In fig.4 we show the formal specification of our geographic application of fig.1. Both the database schema and the atom networks (i.e. database occurrence) are defined within the database definition. Only the relevant data are shown.
Based on this database definition we are now able to raise some atom-type operations, whilst showing the corresponding relational algebra operations:

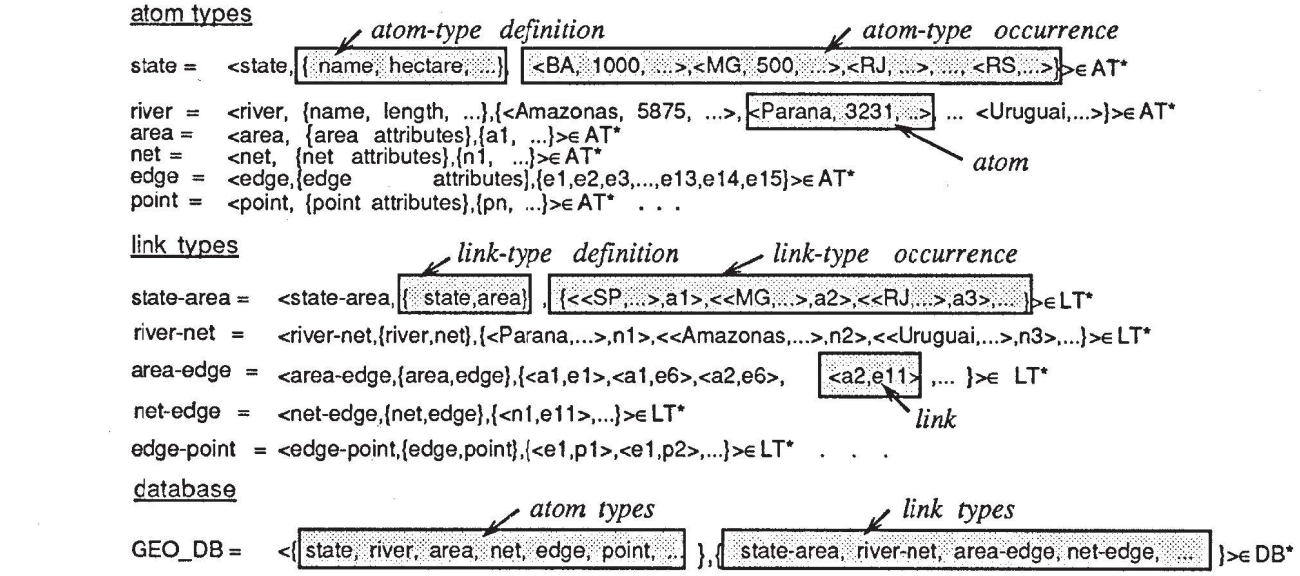

Figure 4: Formal specification of the geographic database

• Cartesian product χ(area,edge) = border
with border =<border,{attributes of both area andedge},
{a1&e1,a1&e2,...}>

All link types having as component area or edge are inherited to the result atom type border. Analogously, the links are inherited to the result atoms.
The relational 'equivalent' looks like: border = area x edge with 'x' being the relational cartesian product.
In both cases the result border gets all attributes from its operands. On the relational side, the foreign keys indicating relationships (i.e. links) are implicitly inherited, too.

• Atom-type restriction σ[restr(hectare>1000)](border)
corresponds to the relational expression:
σ[hectare>1000](border).

Here it becomes obvious that the atom-type algebra exhibits the power of the relational algebra. Moreover, it avoids the problem of enforcing referential integrity, since the relevant relationships (i.e. the foreign/primary key concept of the relational model) are explicitly represented and maintained by means of the link concept. (There are no dangling references (i.e. links) and it is even possible to control cardinality restrictions specified in an extended link-type definition). In the next section we show how this explicit representation of the relationships serves to dynamically specify and derive the desired molecules.

3.2 Higher Level Structures and Operations

Analogously to the previous section, we introduce first the data structures and afterwards the operations for the higher level objects (i.e. complex objects, here called molecules). The central concept of the molecule algebra is the notion of the molecule type. According to the atom type, the definition of molecule type is based on both a molecule-type description and its corresponding molecule-type occurrence.

Definition 5 *Molecule-Type Description*

The pair md=<C,G> is a molecule-type description iff
- $C = \{aname_1,...,aname_n\}$ and $atyp(C) \subset AT^*$
- $G = \{dl_i \mid dl_i = <lname_i, aname_{i1}, aname_{i2}>$ with $<lname_i,\{aname_{i1},aname_{i2}\},lv> \in LT^*$ and $i=1,...,n\}$
- md_graph(md) ∇

Def. 5 formalizes the notion of 'molecule structure' used in the previous chapter introducing a (type) graph whose nodes are atom types and whose edges are directed link types (cf. fig.2). Each triple dl has as first component its name, as second one the start node and as last component its end node. By means of the predicate md_graph we guarantee that this graph has the following properties: directed, acyclic, coherent, and having only one root node. At the same time each molecule-type description determines the domain of its corresponding molecule-type occurrence. This is expressed by the function m_dom:

Definition 6 *Function m_dom*

m_dom(md) = {m | m=<c,g> with
$c \subseteq \{a \mid \exists at: a \in ext(at)$ and $nam(at) \in C\}$ and
$g \subseteq \{l \mid \exists lt: l \in ext(lt)$ with $ltyp(dl)=lt$ and $dl \in G\}$
and mv_graph(m,md)}

The function ltyp is used to determine for each directed link type its associated nondirectional link type being an element of LT*. The elements of m_dom(md) are denoted *molecules* and the predicate mv_graph guarantees the correctness of the molecules with respect to the given molecule-type description md:

mv_graph(m,md) <=> md_graph(m) ∧ total(m,md)

The already known predicate md_graph tests the above mentioned graph properties of the (molecule) graph whose nodes are atoms and whose edges are links. Of course we have to demand the same properties for both graphs (i.e. that of the molecule-type description and that of each molecule) in order to guarantee the correspondence of type and occurrences. The predicate total assures the molecule graph is maximally:

total(m,md) <=> ∀a∈c: contained(a,m,md) and
∀a∉c, but a∈ ext(atyp(aname)) with
aname∈C: ¬contained(a,m,md)

contained(a,m,md)<=> (a∈ ext(root(md))) ∨
(∀ <lname,aname$_i$,aname>∈G,
∃a$_i$∈c with a$_i$∈ ext(atyp(aname$_i$)):
contained (a$_i$,m,md) ∧ <a$_i$,a>∈g)

with a∈ ext(atyp(aname)) with aname∈C,
and <a$_i$,a>∈ ext(ltyp(lname)).

The predicate root determines the root node of a given graph and the predicate contained is recursively defined on the molecule structure graph. It is used to derive all the atoms and links that are necessary to build up the desired molecules ∇

In some sense the above formalization defines the 'synthesis of molecules', that is a procedure explaining how to derive all molecules of a molecule-type occurrence according to a given molecule-type description. It is exactly the same procedure as already informally introduced in chapter 2: the derivation of the children atoms, i.e. hierarchical join along the specified branches is incorporated in the predicate contained. Based on def. 5 and 6 we are now able to make concrete our notion of molecule type:

Definition 7 *Molecule Type*

Given a database DB=<AT,LT>. The triple mt=<mname,md,mv> is a *molecule type* (defined over the database DB) iff the name of the molecule type mname∈N and md is a molecule-type description with md=<C,G> and atyp(C)⊆AT and ltyp(G)⊆LT as well as mv comprises the molecule-type occurrence with mv=m_dom(md) ∇

Definition 8 *Molecule-Type-Definition*

Given a database DB=<AT,LT>, C={aname$_1$,...,aname$_n$} with atyp(C)⊆AT, and G={dl$_1$,...,dl$_m$} with ltyp(G)⊆LT. Furthermore let be mname∈N and md_graph(<C,G>). Then the operator *molecule-type-definition* is defined as follows:

α[mname,G](C)=mt$_\alpha$=<mname,md,mv>

with md=<C,G> and mv=m_dom(md) ∇

Obviously, the molecule type mt$_\alpha$ is valid according to def.7.

To prove the closure of the following molecule-type operations we have to introduce the function prop that materializes or 'propagates' the result of some molecule-type operations into a given database. Thereby, the database is enlarged by new atom types and link types in such a way that the previously propagated result set is now derivable as a corresponding molecule type over this extended database.

Definition 9 *Propagation of Result Sets*

Given a result set rst=<mname,rsd,rsv> with mname∈N, rsd=<C,G> with atyp(C)⊆AT*, ltyp(G)⊆LT*, and md_graph(rsd) as well as rsv={m|mv_graph(m,rsd)}. The function prop is defined as follows:

prop(rest,DB)=<mt,DB'> with

mt=<mname,md,mv> and DB'=<AT∪atyp(C'),LT∪ltyp(G')>

Furthermore, md=<C',G'> and mv=m_dom(md) with C' is the set of renamed atom types used in rsd that exhibit the same atom-type description but only a restricted atom-type occurrence: the corresponding atoms are selected only from the elements within rsv. The same happens to G' being the set of inherited link types that are used in rsd. Of course, md is built using only propagated atom types or inherited link types within C' or G' but it still shows the same graph structure as rsd ∇

Even without a formal proof (cf.[Mi88a]) it is obvious that mt is a valid molecule type over DB' such that mt=α[mname,ltyp(G')](atyp(C')). This is easily conceivable because md satisfies md_graph and all molecules that are constructible with respect to the molecule-type description md also belong to the molecule-type occurrence mv. Furthermore, mv is restricted in an appropriate way so that for each element within rsv there is exactly one equivalent molecule within mv and vice versa.

Figure 5 shows the approach used to define all molecule-type operations: the effect of each molecule-type operation is expressed in several portions starting with some operation-specific actions followed by the propagation of the specified result set or the execution of some atom-type operations. The last part to accomplish the molecule-type operation op$_{mt}$ is to perform the corresponding molecule-type definition α.

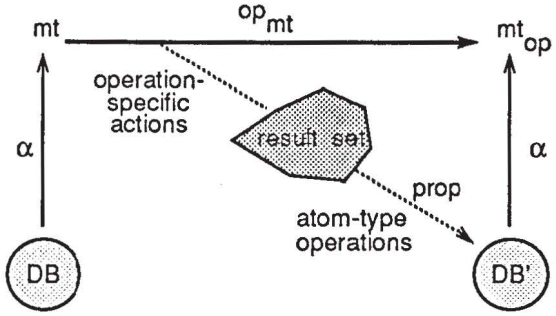

Figure 5: Diagrammatic view to the definition of molecule-type operations

Definition 10 *Molecule-Type-Restriction*

Given a database DB=<AT,LT> and the a molecule type mt=<mname,md,mv>. Furthermore let restr(md) be a qualification formula over md with qual(m,restr(md)) being a predicate that decides whether a molecule m∈mv fulfills the qualification conditions raised in restr(md). Then the operator *molecule-type-restriction* is defined as follows:

Σ[restr(md)](mt) = mt$_\Sigma$

with <mt$_\Sigma$,DB'> = prop(rst,DB)

and rst = <mname$_\Sigma$,rsd,rsv>

with mname$_\Sigma$∈N, rsd=md, and

rsv = {m | m∈mv and qual(m,restr(md))} ∇

Theorem 2

The molecule type mt$_\Sigma$ is a valid molecule type over DB' ∇

Proof:

(1)To prove that rst is a valid result set, we only have to look at rsv, since mname$_\Sigma$∈N and rsd=md. By definition it is true that rsv⊆mv. Thus all elements m∈rsv fulfill mv_graph(m,rsd).

(2) The proof that mt$_\Sigma$ is a valid molecule type over DB' follows straight-forwardly from the correctness of the propagate function prop (cf. def. 9) ∇

Due to space limitation we omit the definition of the following molecule type operations (again, the interested reader is referred to [Mi88a]):

- *molecule-type-projection* (Π),
- *molecule-type-cartesian-product* (X),
- *molecule-type-union* (Ω), and
- *molecule-type-difference* (Δ).

They are mostly defined using the molecule-type propagation and the atom-type operations of the previous section. Of course we can prove that the molecule types resulting from these 4 operations are valid over their correspondingly enlarged databases. Basically, these proofs are similar to that shown above for the operation molecule-type-restriction. Thus, the following theorem holds:

Theorem 3 *Molecule Algebra*

The set of molecule-type operations α, Σ, Π, X, Ω, Δ form an algebra on the set of molecule types. Thus, the result of each molecule-type operation is again describable as a molecule type ∇

The most important outcome of theorem 3 is that the described data model is closed under its operations, i.e. the MAD model is closed under its molecule-type operations. Thus, we are able to concatenate several molecule-type operations to build higher level and more complex operations and objects. For example, we can express the operation molecule-type-intersection Ψ by using the operation molecule-type-difference twice:

$$\Psi(mt_1,mt_2) = \Delta(mt_1, \Delta(mt_1,mt_2))$$

with mt_1, mt_2 being molecule types.

4. From an Algebraic Approach to a High Level Query Language

In this chapter we describe and exemplify a way how to exploit a model's algebra to define a high level as well as user-friendly query language for that model. Firstly, we show some examples of our molecule algebra using the geographic application described in fig.1, fig.2, and fig.4. Translating these algebra expressions into statements of our molecule query language (for short MQL) introduces the constructs and the semantics of this high level query language by means of the sound algebra definition provided in the previous chapter. As syntactical basis of MQL we have chosen an SQL-like formalism because of its widespread use, its syntactical simplicity, and its user-friendliness.

The first example shows the molecule-type-definition of the molecule type 'mt_state' (cf. fig.2):

```
α[mt_state,{<state-area,state,area>,<area-edge,area,edge>,
      <edge-point,edge,point>}](state,area,edge,point)=mt_state
 with
mt_state=<mt_state, <{state,area,edge,point},
                    {state-area,area-edge,edge-point}>, [mv]>
```

molecule-type definition *molecule-type occurrence*

```
 and
mv = {m1=<{<SP,...>,a1,e1,e2,e3,...,p1,p2,...},
          {<<SP,...>,a1>,<a1,e1>,<a1,e2>,...} >,
     m2=<{<MG,..>, a2 ,e3,...},{<<MG,..>,a2>, <a2,e3>..},...}
```

molecule *component atoms* *component links*

Some elements of this molecule-type occurrence are already shown in the lower part of fig.2 in a graphical representation.

The equivalent MQL statement looks as follows:

```
SELECT ALL
FROM mt_state(state-area-edge-point);
```

Obviously, the whole molecule-type definition is expressed in the FROM clause. (If there is only one link type defined between two atom types we can simplify the syntax of the molecule-type definition by using the symbol '-' instead of the link-type's name.)

The following MQL statement exploits the symmetry within the model's data structures:

```
SELECT ALL
FROM point-edge-(area-state,net-river)
WHERE point.name='pn';
```

The result of this query is shown in the upper part of fig.2. To express this in our molecule algebra, we firstly have to build the molecule-type 'point-neighborhood' using the operation molecule-type-definition:

```
point-neighborhood =
    α[point-neighborhood,
      {<point-edge,point,edge>,<edge-area,edge,area>,
      <area-state,area,state>,<edge-net,edge,net>,
      <net-river,net,river>}]
      (point,edge,area,state,net,river)
```

In the second step we have to concatenate this molecule-type-definition with a molecule-type-restriction:

$$\Sigma[restr(point.name='pn')](point\text{-}neighborhood)$$

This example stresses the flexible and symmetric use of a link type to build up quite different molecule types. Molecule restriction in MQL is expressed within the WHERE clause, and molecule projection is accomplished within the SELECT clause.

5. Comparison and Outlook

After having sketched the MAD model and its molecule algebra as well as the high level language MQL, it is worthwhile to draw a comparison to other models and their algebras. A rough but expressive comparison can be done just by comparing the different complex object concepts supported: It is obvious that the MAD model with its provision for network structures and shared subobjects comprises all models that are based on flat or only hierarchically structured objects. Thus, the relational model, the extended relational model [LK84], and even the non-first-normal-form models [AB84,Da86] are just special cases thereof. A more detailed comparison is possible using the models' formalizations: comparing the molecule algebra with the NF^2 relational algebra [SS86] or other non-first-normal-form approaches [OY85,RKS85,BK86] results in the same outcome. Meanwhile, there are attempts to extend the basic NF^2 idea with sharing of subobjects, recursiveness, and dynamics in object definition (cf. [Oz88]).

As already mentioned above, the MAD model allows for reflexive link types and for other cycles in the database schema; e.g. for modeling a bill-of-material application. These cycles are normally queried in a recursive manner, for example asking for the parts explosion (i.e. sub-component view) of a given part. Therefore, we have started to extend the algebra as well as MQL to provide for recursive query facilities [Schö89] introducing recursive molecules types as data model objects.

Moreover, it is worth-while to recall the MAD model's capability for dynamic object definition: the complex

objects, i.e. the molecule types, to work with are defined in algebra expressions or high level language expressions and are not statically fixed in the database schema as it is the case in most of the above mentioned approaches. Our molecule algebra incorporates these facilities and, hence, offers a great flexibility in complex object management.

The comparison between the MAD model and the well-known (binary) ER model (without relationship attributes) is quite obvious. On one hand, the modeling power of the MAD model comprises the modeling capabilities of this type of ER model as already pointed out in chapter 2. On the other hand, it could also serve as a descriptive high-level 'ER language' with the molecule algebra serving as a sound 'ER algebra' (cf. [PS85,HG88,PRYS89]).

Meanwhile a first prototype implementation of MAD/MQL called PRIMA [HMMS87] has been finished. Its internal architecture shows two main components influenced by the construction of the molecule algebra: the basic component provides an atom-oriented interface (similar to the functionality of atom-type algebra) for the second component that performs molecule processing and implements an MQL interface (similar to the functionality of molecule algebra) to the application programs. Thus, this implementation endeavor exploits quite directly the theoretical experiences gained from our algebra definitions. Furthermore, we are confident that we can conveniently exploit the algebra to considerably simplify and enhance query transformation and query optimization as well as using it as a focal point for detailed investigations in query parallelism.

6. Acknowledgment

I'm deeply indebted to K. Meyer-Wegener who has considerably contributed to clarify and improve the presentation of important issues. T. Härder and H. Schöning have read and improved an earlier version, and M. Gaß, I. Littler, and H. Neu have helped in preparing the manuscript.

7. References

[AB84] Abiteboul, S., Bidoit, N.: Non First Normal Form Relations to Represent Hierarchically Organized Data, in: Proc. PODS Conference, 1984, pp. 191-200.

[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.

[BK86] Bancilhon, F., Koshafian, S.: A Calculus for Complex Objects, in: Proc. PODS Conference, 1986, pp. 53-59.

[CDV88] Carey, M., DeWitt, D.J., Vandenberg, S.: A Data Model and Query Language for EXODUS, in: Proc. ACM SIGMOD Conf., Chicago, 1988, pp. 413-423.

[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.

[HG88] Hohenstein, U., Gogolla, M.: A Calculus for an Extended Entity Relationship Model Incorporating Arbitrary Data Operations and Aggregate Functions, in: Proc. 7th Int. Conf. on Entity-Relationship Approach, Rome, 1988, pp. 1-20.

[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.

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

[Mi88a] Mitschang, B.: A molecule-atom data model for non-standard applications - requirements engineering, data-model design, and implementation concepts (in German), PhD Thesis, University Kaiserslautern, 1988, published by Springer Publ. Co. in the series IFB 185.

[Mi88b] Mitschang, B.: Towards a unified view to design data and knowledge representation, in: Proc. 2nd Int. Conf. on Expert Database Systems, Tysons Corner VA, 1988, pp. 33-49 (further publication by Benjamin/Cummings Publ. Co.).

[OY85] Ozsoyoglu, Z.M., Yuan, L.-Y.: A Normal Form for Nested Relations, in: Proc. PODS Conference, 1985, pp. 251-260.

[Oz88] Ozsoyoglu, Z.M. (ed.): Special Issue On Nested Relations, in: Data Engineering, IEEE, Vol. 11, No. 3, Sept. 1988.

[PRYS89] Parent, C., Rolin, H., Yetongnon, K., Spaccapietra, S.: An ER calculus for the entity-relationship complete model, Research Report 890401, Dep. d'Informatique, Ecole Polytechnique Federal de Lausanne, 1989.

[PS85] Parent, C., Spaccapietra, S.: An Algebra for a General Entity Relationship Model, in: IEEE Transactions on SE, Vol. 11, No. 7, July 1985, pp. 634-643.

[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.

[RKS85] Roth, M.A., Korth, H.F., Silberschatz, A.: Extended Algebra and Calculus for ¬1NF Relational Databases, Technical Report TR-84-36 of Univ. of Texas at Austin, 1985.

[RS87] Rowe, L.A., Stonebraker, M.R.: The POSTGRES Data Model, in: Proc. 13th VLDB Conf., Brighton, 1987, pp. 83-96.

[Schö89] Schöning, H.: Recursion in the MAD Model - Recursive Molecules as Objects of the Data Model (in German), in: IFB 204, Springer Publ. Co, 1989, pp 389-407.

[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.

[Ul80] Ullman, J.D.: Principles of Database Systems, Pitman Publishing Limited, 1980.