MOLECULAR OBJECTS, ABSTRACT DATA TYPES, AND DATA MODELS: A FRAMEWORK

D. S. Batory and A. P. Buchmann

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-84-07 March 1984

Molecular Objects, Abstract Data Types, and Data Models: A Framework *

D.S. Batory
Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712

Alejandro P. Buchmann
IIMAS
National University of Mexico

ABSTRACT

Molecular objects are treated at higher levels of abstraction as atomic units of data; at lower levels they are defined in terms of a set of tuples possibly from different relations. System R's complex objects are examples of molecular objects.

In this paper, we present a framework for studying a generalized concept of molecular objects. We show that abstract data types unify this framework, and the framework itself encompasses some recent data modeling contributions by researchers at IBM San Jose, Berkeley, Boeing, and Florida. A programming language/data structure paradigm is seen as a way of developing and testing the power of logical data models. A primary consequence of this paradigm is that future DBMSs must handle at least four distinct types of molecular objects: disjoint/non-disjoint and recursive/non-recursive. No existing DBMS presently supports all these types.

* This work was supported by the National Science Foundation under Grant MCS-8317353.

1. Introduction

The role of abstract data types and data abstraction mechanisms in logical data modeling is becoming progressively more important. A classical relationship between abstract data types and data models was established some time ago by Schmidt ([Sch78]), Weber ([Web78]), Rowe ([Row79]), Wasserman ([Was79]), and others who identified relations (of Relational databases) with abstract data types; a fixed set of operations could be performed on relations while the implementations of relations themselves were hidden.

More recently, new and important relationships have been independently recognized in the context of database support for CAD, engineering, and statistical database applications ([Sto83], [Lor83a-b], [Joh83], [Su83], [Bro83], [Has82]). These relationships can be understood in the following way: part of the schema definition of a relation is equivalent to a PASCAL, ADA, C, etc. record definition where the underlying data types are strings, integers, and reals. In the context of CAD databases, Stonebraker et al. ([Sto83]) proposed that the underlying set of data types be enlarged by the addition of user defined types. They present practical examples of linear and multidimensional arrays that have special properties and operators. The name given to this proposal is 'abstract data types'.

In the context of statistical and scientific databases, Su et al. ([Su83], [Bro83]) started with a similar idea, but developed it differently. A set of system-defined data types (e.g., set, vector, matrix, time series) was proposed in addition to the standard ones, and ways were provided in which these types could be used to define new data types. A relation, for example, could be defined as a data type, and a component of a relation could be another relation. 'Complex data types' was the name given to this idea.

The underlying theme and direction of both works was to introduce sophisticated data types - i.e., abstract data types that are nested or unnested, user or system defined - to relations. Moreover, both works are characterized by the notion of what we will call *atomic aggregation*. That is, a single tuple (of whatever composition) becomes an atomic unit of data. Atomic aggregation is synonymous with the aggregation concept of Smith and Smith [Smi77].

In the context of CAD databases, Lorie et al. ([Lor83], [Has82]) and Johnson et al. ([Joh83]) independently proposed restricted but important notions of what we will call *molecular aggregation*. That is, a set of related tuples (possibly from different relations) becomes an atomic unit of data. Lorie's molecular aggregations are called 'complex objects'; Johnson's are called 'structures'.

The interest in molecular objects stems from their semantics and utility. There is a growing need to provide users with the capability of manipulating and retrieving molecular objects as primitives, rather than placing the burden on users to hunt for an object's underlying tuples (see [Has82], [Lor83a-b]). Support for molecular objects is especially important in CAD and other non-traditional environments where molecules arise naturally and consist of tens or even thousands of tuples each. Here's an example.

Consider a catalog of data structure diagrams. On each page of the catalog is a data structure diagram of some corporate, university, or governmental database. Each diagram is identified by a catalog number and is explained by a catalog description. The contents of this catalog are to be stored in a computerized database. Figure 1 shows a page from this catalog. [1]

Figure 1. An Entry in a Catalog of Data Structure Diagrams

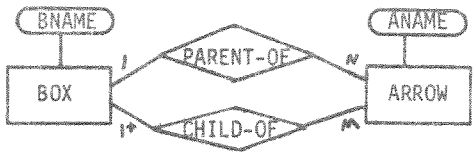
Suppose we describe the contents of this database using the notation of the E-R model ([Che76]). (We will present a justification of this choice in Section 2). An E-R diagram of a data structure diagram is shown in Figure 2a. A data structure diagram consists of box and arrow entities. Each box and arrow has a name. An arrow entity connects one box (the owner record

[1] We use the data structure digram example in this paper to avoid the confusion that arises when making reference to applications that are unfamiliar to most readers.

type) to one or more different boxes (the member record types).[2] These connections are represented by the PARENT-OF and CHILD-OF relationships.

The underlying tables of Figure 2a and the seven tuples that define the data structure diagram of Figure 1 are shown in Figure 2b.[3] It is this set of seven tuples that represents the molecular catalog entity 'C53'. Note however that the E-R diagram of Figure 2 is not complete because the catalog number 'C53' and catalog description 'Grade Database' have not been associated with these tuples. Nor is there a 'catalog entity set' which relates this collection of tuples to a particular catalog entity. In Section 2, we will return to this example and show how it can be modeled correctly.

(a)



(b)

BOX(<u>BNAME</u>)

(STUDENT)
(GRADE)
(COURSE)

ARROW(<u>ANAME</u>, PARENT-BNAME)

(SG,	STUDENT)
(CG,	COURSE)

CHILD-OF(<u>ANAME, BNAME</u>)

(SG,	GRADE)
(CG,	GRADE)

Figure 2. An E-R Diagram of a Data Structure Diagram and its Underlying Tables

Like all other objects, *molecular objects* or *molecular entities* can have only certain operations performed on them. For example, one cannot add boxes and arrows to a data structure diagram at random; notions of connectivity must be preserved. For this reason, molecular objects can be understood in terms of abstract data types.

Molecules are *disjoint* if the underlying sets of tuples that define them are disjoint. The molecular objects that are supported by System R (i.e., complex objects) are disjoint molecules.

---

[2] The restriction that owner record types cannot also be member record types of the same set would be expressed by an integrity constraint.

[3] Usually there is a distinct table that underlies each entity set and relationship set of an E-R diagram. In Figure 2, the PARENT-OF(<u>BNAME, ANAME</u>) table has been merged with the ARROW(<u>ANAME</u>) table. This is possible because each arrow starts at precisely one box, so the entries of the PARENT-OF table are in 1:1 correspondence with entries in the ARROW table.

From common knowledge of programming languages and data structures, there are implementations of abstract data types (i.e., molecules) that are *not* disjoint. Figure 3 shows an example of two nondisjoint molecules. It shows two lists that have two nodes in common. Each list is a molecule; each node can be represented as a single tuple in some relation.

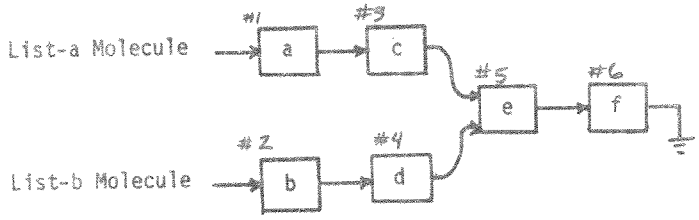


Figure 3. Two Non-disjoint Molecules

It is our belief that tuple/atom sharing in molecules occurs naturally and is quite common. Johnson et al. ([Joh83]) present an example of non-disjoint molecules. Additional examples are given in Sections 2 and 3. We conjecture that as more CAD, statistical, and other special-purpose applications are supported by database systems, restricting molecules to be disjoint (as is done in System R) will not be sufficient to handle the needs of many applications. Nondisjoint molecules must also be supported.

From the above discussions, it is evident that recent efforts to extend the data modeling capabilities of DBMSs to handle non-traditional applications are unified by the concept of abstract data types. Extrapolating the research direction indicated by these works leads to the conclusion that *logical data models need the data abstraction capabilities that are already present in today's advanced programming languages (e.g., CLU [Lis77], ALPHARD [Wul76], ADA [Geh83])*. That is, the types of 'entities' and 'relationships' that can be expressed in programming languages today cannot be handled by existing data models. Future data models should be more powerful in this respect. [4]

[4] It is worth noting that the same conclusion could be drawn from a different line of reasoning. In the 1950's, special purpose programs were written to handle the processing of some fairly simple-formatted data. As the volume of data grew, file management and database management systems were developed. Today, special purpose programs are processing very complex-formatted data. We are now seeing efforts to develop the next generation of file and database management systems, but this time the data records that are being processed must be expressed in terms of abstract data types. Thus the next generation of database management systems must incorporate abstract data types in their data models.

Two important implications follow. First, we are not aware of any application which cannot be expressed in a programming language in terms of abstract data types. If DBMS data models have the data abstraction capabilities of programming languages, then they should be general enough to handle any application (CAD, statistical, A.I., etc.). [5]

How record types and 'relationships' in programming languages can be embedded or identified with the record types and relationships supported by a DBMS is not yet clear. It may be the case that record type declarations of programming languages will someday be supplanted by some (future) data model whose design techniques (e.g., E-R diagrams, normalization, etc.) can be used to develop the schemas of both internal databases (e.g., node formats of data structures such as lists and trees) and external databases. (We will present direct evidence that this is possible later in the paper). Moreover, a single set of operators would be used to access and manipulate both internally and externally stored records. This is one approach that would achieve a unification of programming language and logical data modeling constructs. Such a unification, unfortunately, is a long way off. However, it is worth noting that research is already underway to tie the abstract data types (and subtypes) of experimental programming languages to database systems ([Smi83], [Laf83], [Ger84]). We believe more of this type of research is likely to become popular in the future.

Second, an easy way to test the power (or demonstrate the limitations) of a data model is to model fundamental data structures as found in standard texts ([Knu73], [Aho74], [Hor78]). As we did in Figure 3, it is easy to devise a data structure with certain properties that will reveal the limitations of existing data models or data modeling concepts. The advantage of this approach is that it provides application independent tests; one does not have to be intimately familiar with a peculiar database application in order to comprehend the example. We will use the *programming language/data structure paradigm* many times in this paper to illustrate and develop the modeling

---

[5] Clearly performance would be a major concern for such a DBMS. Recent advances in physical database modeling have shown that it is possible to develop internal database software automatically from a small set of specifications ([Bat83], [Bat84]). It seems likely, therefore, that special storage architectures to support customized DBMSs will be common in the future, rather than being exceptions.

concepts that are proposed later.

In the following section, we present a general framework for studying molecular aggregations. In Section 3 we will further illustrate molecular aggregation concepts with examples taken from non-traditional database applications.

2. A Framework for Studying Molecular Aggregations

Most of the work relating abstract data types and data models has involved the Relational model. It is well-known that the Relational model has semantic limitations. Therefore, attempts have been made to expand it to include different types of aggregation ([Smi77], [Cod79], [Dat82], [Has82], [Lor83a-b], [Sto83]).

We believe that the concepts of molecular aggregation are independent of the model or notation that is used to express them. However, for the purposes of developing and explaining these concepts, we have found it convenient to use the diagrammatic notations of the E-R model ([Che76]). An advantage of using E-R diagrams is that they can be reduced to tables which, in turn, can be identified with relations. [6] As we will see later, the E-R model, like the Relational model, has its limitations.

In our framework, we identify recursive and non-recursive molecules and disjoint and non-disjoint molecules. The ideas required to describe such molecules are progressively developed and are illustrated by a carefully chosen set of examples. The development is motivated by basic observations that follow from the programming language/data structure paradigm.

2.1 Modeling Non-Recursive, Disjoint Molecular Entities

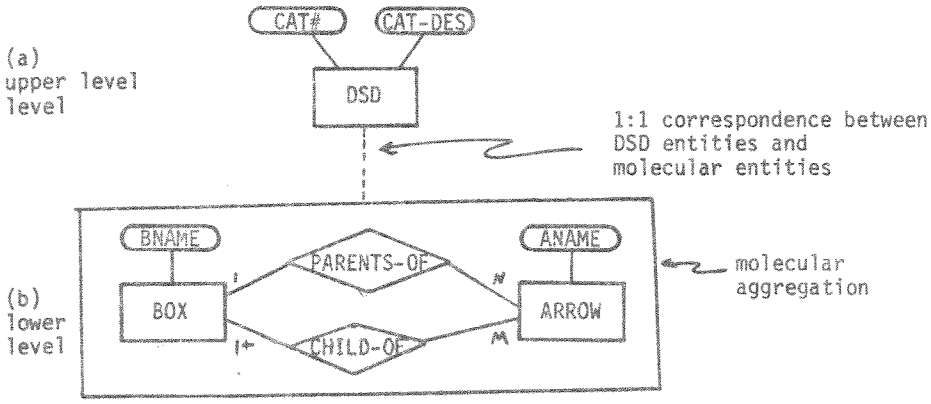
Abstract data types allow data to be modeled at different levels of abstraction. That is, one can treat a molecular entity at one level of abstraction as an atomic entity; at a lower level, the atoms that compose the molecule are seen. At any particular level, existing modeling techniques

---

[6] There is no guarentee that the tables produced will be in X-normal form. Techniques of normalization may need to be applied to reduce these tables to the desired state (e.g., BCNF). We are not proposing that the E-R model is a substitute for the relational model; we are simply using its diagrammatic conventions.

should be adequate to express the entities and relationships that may exist. It is the *mapping* or *correspondence* of entities, attributes, and relationships at one level of abstraction to those of another that needs to be introduced. This gives rise to the modeling construct *correspondence*. Here is an example.

Recall the data structure diagram catalog. At a higher level of abstraction, we are dealing with 'catalog' entities or data structure diagram (DSD) entities. DSD entities are identified by their catalog number (CAT#) and are explained by their description (CAT-DES). The E-R diagram which represents the database at this level of abstraction is shown in Figure 4a.



(c)
tables

DSD(CAT#, CAT-DES)

(C53, Grade Database)

BOX(CAT#, BNAME)

(C53, STUDENT)
(C53, GRADE)
(C53, COURSE)

ARROW(CAT#, ANAME, PARENT-BNAME)

(C53, SG, STUDENT)
(C53, CG, COURSE)

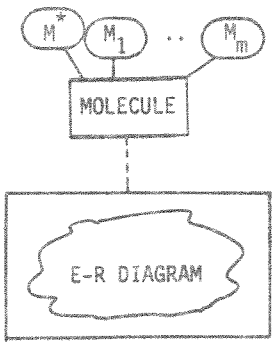
CHILD-OF(CAT#, ANAME, BNAME)

(C53, SG, GRADE)
(C53, CG, GRADE)

Figure 4. A Non-recursive, Disjoint Entity

At a lower level, the implementation details of data structure diagrams are captured by the E-R diagram of Figure 4b (Figure 2b). The tuples (atoms) that define an occurrence of this diagram are combined by disjoint molecular aggregation to form a DSD entity. Disjoint molecular aggregation is shown by a box drawn around the E-R diagram which defines the relationships among atoms of the molecule. The dashed line connecting the DSD entity box and the molecular box denotes that each molecular entity corresponds to precisely one DSD entity, and vice versa.

The general scenario for non-recursive, disjoint molecules is shown in Figure 5a. The tables that underly this diagram are formed by separately reducing the E-R diagrams at both levels to tables. This can be done using standard techniques ([Che76]). At the upper level, there will be a MOLECULE table with primary key M^* and descriptive attributes $M_1 \cdots M_m$. At the lower level, there are tables $T_1 \cdots T_n$. Table T_i has primary key K^i and descriptive attributes $A_{i1} \cdots$. The correspondence between levels is made by specifying to which molecular entity each tuple of tables $T_1 \cdots T_n$ belongs. This is accomplished by augmenting the primary key of a molecule to its underlying tuples. Thus, the primary key of table T_i becomes (M^*, K^i). All non-recursive, disjoint molecular aggregations can be reduced to tables in this way (see Fig. 5b). [7]



MOLECULE($\underline{M^*}$, M_1, ..., M_m)

$T_1(\underline{M^*, K^1}, A_{11}, \ldots)$
$T_2(\underline{M^*, K^2}, A_{21}, \ldots)$
...
$T_n(\underline{M^*, K^n}, A_{n1}, \ldots)$

(a) (b)

Figure 5. A Model of Non-recursive, Disjoint Molecules

In the case of Figures 4a-b, this procedure yields table DSD(CAT#, CAT-DES) at the upper level and BOX(CAT#, BNAME), ARROW(CAT#, ANAME, PARENT-BNAME), and CHILD-OF(CAT#, ANAME, BNAME) at the lower level. (Primary keys are underlined). These tables and their tuples are shown in Figure 4c.

Note that there may be several data structure diagrams that have a box labeled 'student'; the names given to box and arrow entities are distinct within the confines of a particular data

[7] The inheritance of M^* in underlying tables is similar in function to System R's COMPONENT-OF attribute of non-root tuples of complex objects.

structure diagram, but need not be distinct throughout the catalog. This is the reason why the primary key of the upper-level entity is inherited by each table at the lower level. [8] In contrast, System R generates primary keys for 'entities' that are unique to the entire database. These keys, called *surrogates* ([Cod79]), can be treated in the E-R model like any other attribute, but the underlying tables that are formed will have redundancies. For example, a table that represents an E-R relationship will contain the primary keys of the entities that are related, plus the primary key of the molecule. The presence of the molecule key is redundant - i.e., transitively dependent - because its association with each of the entity keys is already recorded in the tables that describe these entities. Such redundancies can only be eliminated by applying normalization techniques.

2.3 Modeling Non-Recursive, Non-Disjoint Molecular Entities

Suppose that data structure diagrams of subschemas for each catalog entry are also to be present in the database (see Fig. 6). This can be modeled at a higher level of abstraction by data structure diagram (DSD) entities and their dependent subschema data structure diagram (SDSD) entities. The HAS relationship relates both entity sets (see Fig. 7a). Note that SDSD entities are *weak* entities because they are existant dependent on DSD entities. That is, if a DSD entity is deleted, so are its dependent SDSD entities.

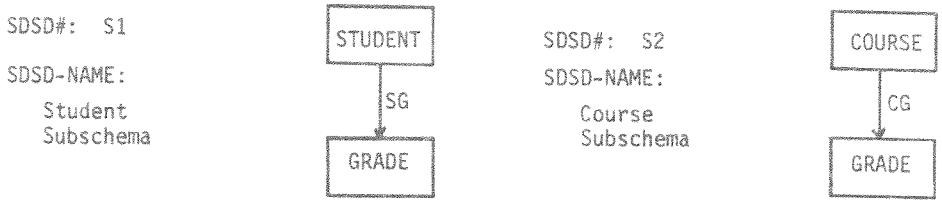
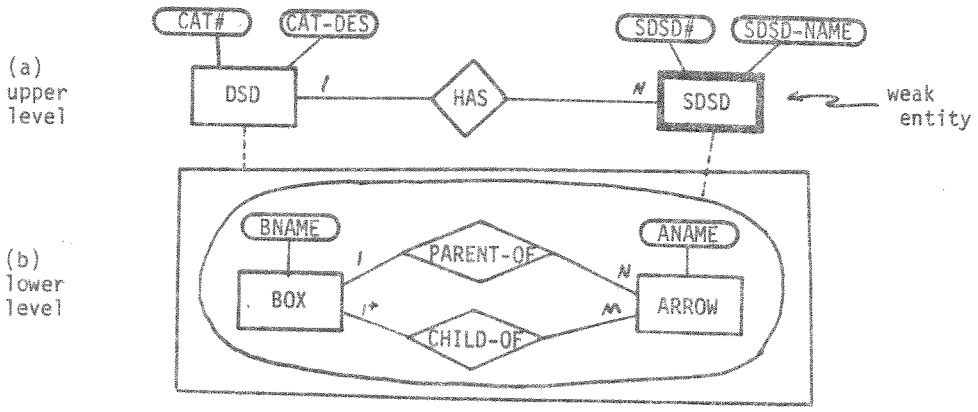


Figure 6. Subschemas of Figure 1

Because different subschemas may share record and set types, the sets of tuples that underly two SDSD molecules need not be disjoint. We denote non-disjoint molecular aggregation by

[8] Readers might recognize a similarity of this molecular aggregation with the E-R notion of *weak* entities ([Che76]). That is, the existence of the entities and relationships that underly a molecular entity are dependent on the existance of that molecular entity.



(c) tables

BOX-CT(CAT#, SDSD#, BNAME)

(C53, S1, STUDENT)
(C53, S1, GRADE)
(C53, S2, COURSE)
(C53, S2, GRADE)

ARROW-CT(CAT#, SDSD#, ANAME)

(C53, S1, SG)
(C53, S2, CG)

CHILD-OF-CT(CAT#, SDSD#, ANAME, BNAME)

(C53, S1, SG, GRADE)
(C53, S2, CG, GRADE)

SDSD(CAT#, SDSD#, SDSD-NAME)

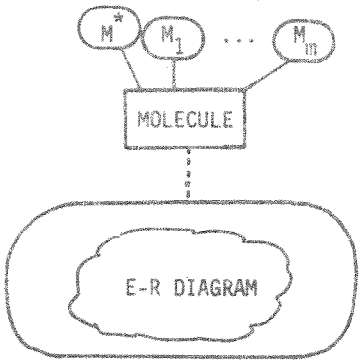
(C53, S1, Student Subschema)
(C53, S2, Course Subschema)

plus tables of Figure 4c.

Figure 7. A Non-Recursive, Non-Disjoint Entity

circling the E-R diagram whose instances define a molecule. Figure 7a-b shows both disjoint and nondisjoint aggregations and their correspondences.

The general scenario of non-recursive, non-disjoint molecules is shown in Figure 8a. The tables that underly this diagram are formed by separately reducing the E-R diagrams to tables at both levels. As before, there will be a MOLECULE table and the underlying tables $T_1 \cdots T_n$. The correspondence between levels is again made by specifying to which molecular entity each tuple of tables $T_1 \cdots T_n$ belongs. This is accomplished by creating for each table T_i a *component table* or *correspondence table* CT_i which pairs the primary keys of component tuples (K^i) with the primary keys of their molecular entities (M^e). (M^e, K^i) is the primary key of CT_i tuples. All non-recursive, non-disjoint molecular aggregations can be reduced to tables in this way (see Fig. 8b).



(a)

MOLECULE($\underline{M^*}$, M_1, ..., M_m)

$T_1(\underline{K^1}, A_{11}, \ldots)$ $\quad CT_1(\underline{M^*, K^1})$

$T_2(\underline{K^2}, A_{21}, \ldots)$ $\quad CT_2(\underline{M^*, K^2})$

....

$T_n(\underline{K^n}, A_{n1}, \ldots)$ $\quad CT_n(\underline{M^*, K^n})$

(b)

Figure 8. Modeling Non-Recursive, Non-Disjoint Molecules

In the case of Figures 7a-b, this procedure yields tables DSD(CAT#, CAT-DES) and SDSD(CAT#, SDSD#, SDSD-NAME) at the upper level and BOX(CAT#, BNAME), ARROW(CAT#, ANAME, PARENT-NAME), CHILD-OF(CAT#, ANAME, BNAME), BOX-CT(CAT#, SDSD#, BNAME), ARROW-CT(CAT#, SDSD#, ANAME), and CHILD-OF-CT(CAT#, SDSD#, ANAME, BNAME) at the lower level. These tables and their tuples are shown in Figure 7c. [9]

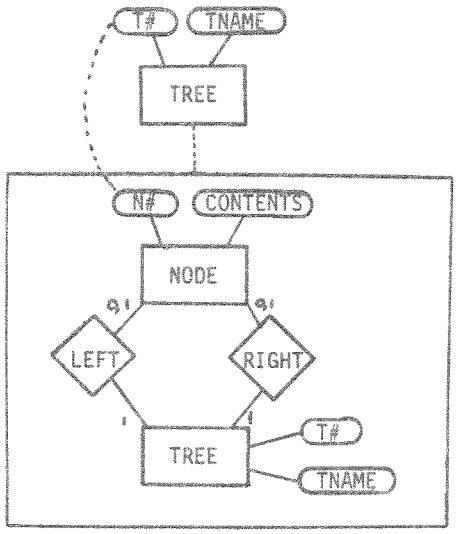
It is worth noting that the notion of disjointness to which our discussions have centered deals with molecules that are of a single type. That is, all DSD molecules are disjoint; not all SDSD molecules are disjoint. A concept of disjointness can also be defined for molecules of different types. Figure 7 provides an example. A DSD molecule can share atoms with an SDSD molecule. The disjointness or non-disjointness of molecules of different types are possibilities that follow naturally from our model and do not require special attention. We will give additional examples in Section 3.

[9] Note that the BOX and CHILD-OF tables contain only key attributes. Other attributes could be present such as LOCATION-MODE for BOX entities and INSERTION-CLASS and RETENTION-CLASS for CHILD-OF.

2.3 Modeling Recursive, Disjoint Molecular Entities

A binary tree can be defined as being either empty, or a root node with left and right binary (sub)trees. A recursive disjoint molecular diagram defining binary trees is shown in Figure 9a. At a higher level there are only tree entities which have tree number identifiers (T#) and tree names (TNAME). At the lower level, there are node entities, which have node number identifiers (N#) and contents (CONTENTS). There are also (sub)tree entities, described as in the higher level. The left and right subtree relationships are modeled by LEFT and RIGHT.

(a)



(b) NODE-TREE(T# or N#, CONTENTS, LEFT-T#, RIGHT-T#, TNAME)

Figure 9. A Model of Binary Trees

Once again we see that for each molecular entity (a NODE tuple and its associated left and right subTREE tuples) there is a corresponding TREE tuple at the upper level. The E-R diagram also shows that there is a correspondence between TREE identifiers and NODE identifiers. This has a simple interpretation. The primary key of a NODE or TREE can be understood as a pointer: a pointer is used to identify a single NODE and it also used to identify the TREE rooted at that NODE. A distinction is made according to the level of abstraction at which the pointer reference is made. As this example shows, when objects exist at two or more levels of abstraction it is possible for there to be a shift in their semantics going from one level of abstraction to the

next. We will see that such shifts are common.

The general scenario for recursive, disjoint molecules is shown in Figure 10a. The MOLECULE and $T_1 \cdots T_n$ tables that underly this diagram are formed exactly as if the molecules were non-recursive with one exception. The upper-level MOLECULE table is unnecessary since a copy of it also exists at the lower level. To account for the correspondence, the primary key of the upper-level molecule (now denoted M'') is inherited as an ordinary attribute of each table at the lower level and is not subsumed as a key prefix. Thus, table T_i has primary key K^i and has descriptive attributes M'' and $A_{i1} \cdots$. The MOLECULE table has primary key M' and has descriptive attributes M'' and $M_1 \cdots M_m$ (see Fig. 10b).

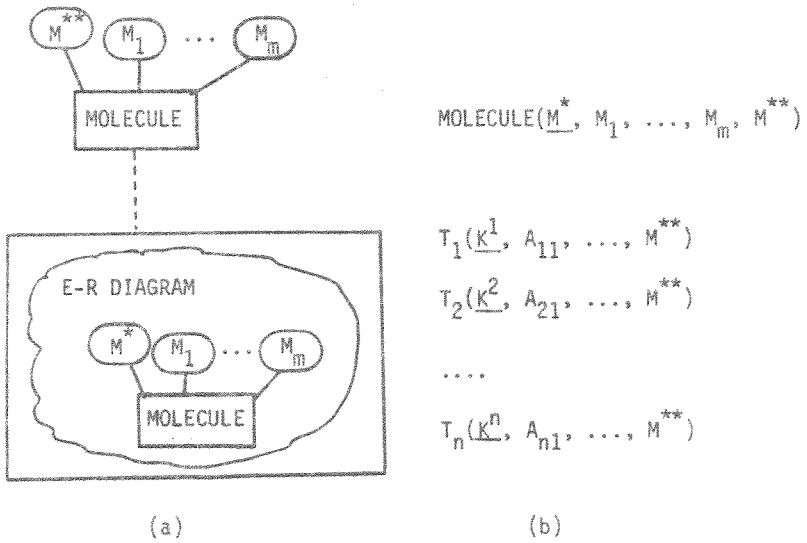


Figure 10. Modeling Recursive, Disjoint Molecules

Two tables result in applying this procedure to Figure 9a: TREE(T#, TNAME, PARENT-T#) and NODE(N#, CONTENTS, LEFT-T#, RIGHT-T#, PARENT-T#). [10] Because of the 1:1 correspondence between node numbers with tree numbers, data values that are assigned to the PARENT-T# and N# attributes of the NODE table are identical, and hence can be combined into a single attribute. That is, the PARENT-T# of a NODE is the T# of the TREE which has that NODE as its root. Moreover, since NODE entities and TREE entities are in 1:1 correspondence, the NODE and TREE tables can be combined into a single table NODE-TREE(T# or N#, CONTENTS, LEFT-T#, RIGHT-T#, TNAME, PARENT-T#). The two names given to the

table's primary key reflect the possible interpretations of its values.

From our experience it appears to be common for recursive E-R diagrams to have more than a single upper-level entity - molecular aggregation correspondence. In the binary tree example, the additional correspondence is that between node numbers and tree numbers. In every example that we are aware, additional correspondences introduce relationships which cause the molecular primary key M'' to be redundant, and hence optional.

The presence of the PARENT-T# attribute in table NODE-TREE is redundant. Given any subtree, we know that its parent tree is associated with one NODE-TREE tuple. In the parent tuple, the given subtree is referenced as either the LEFT-T# or RIGHT-T#. Thus, if PARENT-T# were stripped from NODE-TREE, its value could be inferred.

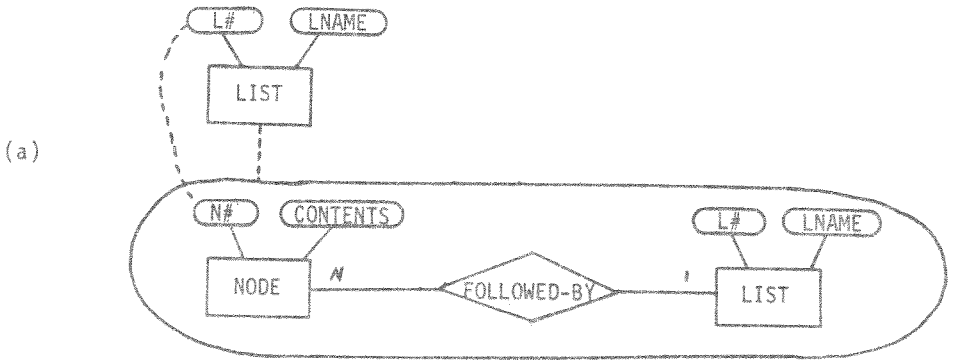
It is worth noting that when PARENT-T# is removed, the table that results (Fig. 9b) is a record layout that is commonly used in programs to manipulate binary trees. This is evidence that it is possible to derive the intuitive record layouts of data structures (i.e., schemas of internal databases) using molecular aggregation modeling techniques. We will see another example of this in the next section.

2.4 Modeling Recursive, Non-disjoint Molecular Entities

Earlier we gave the example of two list molecules sharing the same nodes. Figure 11a shows how this can be modeled. It is based on a recursive definition of a list: a list is either empty or it is a node followed by another (sub)list. There can be any number of nodes that immediately preceed a (sub)list. Once again, NODE entities (and their primary keys) are in 1:1 correspondence with LIST entities (and their primary keys).

The general scenario for recursive, non-disjoint molecules is shown in Figure 12. The procedure for reducing such diagrams to tables is identical to that of reducing non-recursive, non-

[10] A table underlies each entity set and relationship set. In Figure 9a there would be tables for the NODE and TREE entity sets and tables for the LEFT and RIGHT relationship sets. Each entry in the LEFT (and RIGHT) table is in 1:1 correspondence with NODE table entries (because each NODE entry has precisely one LEFT and one RIGHT subtree). For this reason, the LEFT, RIGHT, and NODE tables can be combined - as done above - into a single 'NODE' table.



(b) NODE-LIST(L# or N#, CONTENTS, FOLLOWING-L#, LNAME)

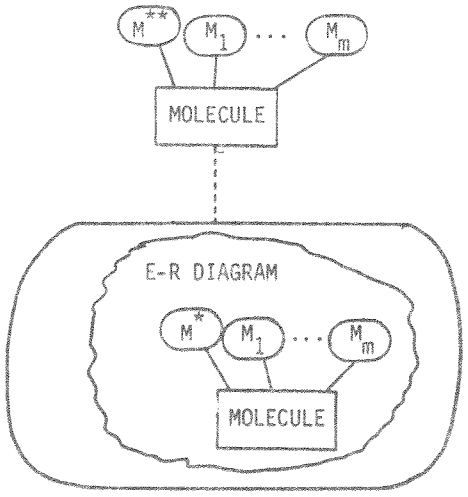
(#1,	a,	#3,	list-a)
(#2,	b,	#4,	list-b)
(#3,	c,	#5,	-)
(#4,	d,	#5,	-)
(#5,	e,	#6,	-)
(#6,	f,	null,	-)

LIST-CT(PARENT-L#, L#)

(null,	#1)
(null,	#2)
(#1,	#3)
(#2,	#4)
(#3,	#5)
(#4,	#5)
(#5,	#6)

Figure 11. Non-Disjoint Lists

disjoint molecules with the exception that the upper-level MOLECULE table is not represented twice. The primary key of the upper-level molecule is denoted by M^{**}. Note that the MOLECULE-CT table pairs the primary key M^{**} of a parent molecule with the primary key M^{*} of each of its submolecules.



(a)

MOLECULE($\underline{M^{*}}$, M_1, ..., M_m) MOLECULE-CT($\underline{M^{**}, M^{*}}$)

$T_1(\underline{K^1}, A_{11}, \ldots)$ $CT_1(\underline{M^{**}, K^1})$

$T_2(\underline{K^2}, A_{21}, \ldots)$ $CT_2(\underline{M^{**}, K^2})$

...

$T_n(\underline{K^n}, A_{n1}, \ldots)$ $CT_n(\underline{M^{**}, K^n})$

(b)

Figure 12. Modeling Recursive, Non-Disjoint Molecules

Applying this procedure, four tables exist at the lower level of the list molecule example:

LIST(L#, LNAME), NODE(N#, CONTENTS, FOLLOWING-L#), NODE-CT(PARENT-L#, N#), and LIST-CT(PARENT-L#, L#). Because of the 1:1 correspondence between node numbers (and node entities) and list numbers (and list entities), the NODE and LIST tables can be combined into a single table NODE-LIST. Also, since the attributes N# and L# always assume the same data value in each NODE-LIST tuple, i.e., a pointer to a NODE is identical to the pointer to the LIST headed by that NODE, N# and L# can be combined into a single attribute. For this same reason, identical data values are assigned to PARENT-L# and N# attributes for each entry of a NODE-CT table. That is, NODE-CT tuples are ordered pairs where both elements of a pair are equal. Therefore, the NODE-CT table is redundant and can be eliminated (see Fig. 11b).

The LIST-CT table is redundant and also could be eliminated. Entries in it are pairs of (parent-list, sub-list) identifiers. Just as in the binary tree example, if the LIST-CT table were eliminated, its contents could be inferred. That is, given the identifier of the parent-list, there is a NODE-LIST tuple which corresponds to this list. The FOLLOWING-L# data value of this tuple is the sub-list identifier. Note that the result of eliminating these redundancies is NODE-LIST: a record format that is commonly used in programs to manipulate linked lists. Here again we see molecular aggregation modeling techniques can be used to derive record formats of common data structures.

2.5 Modeling External Features of Molecular Entities

The previous sections have described a general methodology for modeling molecular objects. In this section we will show that the programming language/data structure paradigm leads us to a better understanding of how molecular objects are addressed at higher levels of abstraction. In particular, we examine the idea of molecules having external features. At the same time, we will also find limitations in the E-R model.

The approach taken in CLU, ADA, and other programming languages is that instances of abstract data types are featureless entities. For example, M could be an instance of a matrix. Very often it is useful to give some of the internal features of an abstract data type external

projections so that they may be referenced easily. Viewing the element M[2,3] is such an example. For this reason, projection operators (i.e., [i,j]) are defined for that type. A similar situation applies to molecular entities: molecules have internal features that need external projections.

Consider the circuit of Figure 13 which defines a 4-input AND gate in terms of 2-input AND gates, wires, and terminals. At the higher level there is a GATE entity with its dependent PIN entities. PIN entities are *external features* of GATEs (see Fig. 14a). At the lower level there are TERMINAL entities (i.e., pins that are not associated with lower-level gates), GATE entities and their dependent PIN entity features, and WIRE entities which connect pins and terminals to other pins and terminals. Note that upper-level PINS correspond to lower-level TERMINALS: PINS are weak entities; TEMINALS are strong entities. Here again is a shift in semantics between levels of abstraction, a shift that is similar to that of node and tree identifiers.

The type of correspondence that is shown in Figure 14a is slightly different than what we have encountered before. It is called *subset correspondence*. It means that every molecular entity is a GATE entity, but not all GATE entities are molecular entities. In the example, the 4-input AND gate is certainly a molecular entity, but its 2-input AND gates are primitive and are not defined in terms of lower level gates.

The underlying tables of Figures 14a-b are given in Figure 14c. They follow directly from the rules for reducing recursive, disjoint molecules in the previous section: GATE(G#, TYPE, PARENT-G#), POINTS(G#, PIN#, I/O, PARENT-G#), and WIRE(W#, START-G#, START-PIN#, END-G#, END-PIN#, PARENT-G#). The PARENT-G# attribute of all three tables is redundant and optional for the following reason. It is possible to determine the underlying tuples of a GATE molecule simply by starting at the GATE's terminals and by following wires. This is identical to the procedure for determining the nodes of a list: start at the head of the list and follow pointers. The inclusion of PARENT-G# in these tables may be necessary for performance reasons: it is much faster to consult a single attribute to find component gates than by following wires. [11]

[11] It is worth noting that System R handles the problem of finding underlying tuples of complex objects

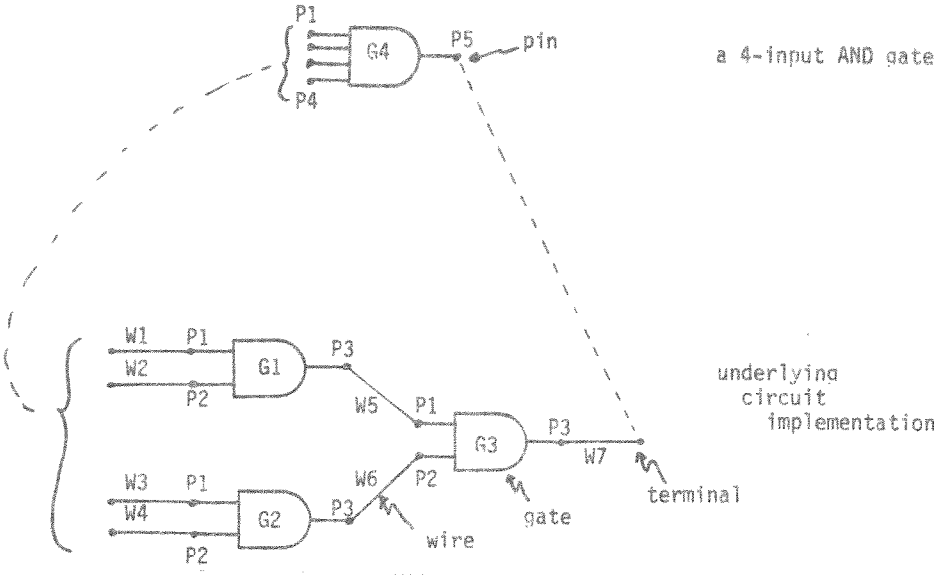
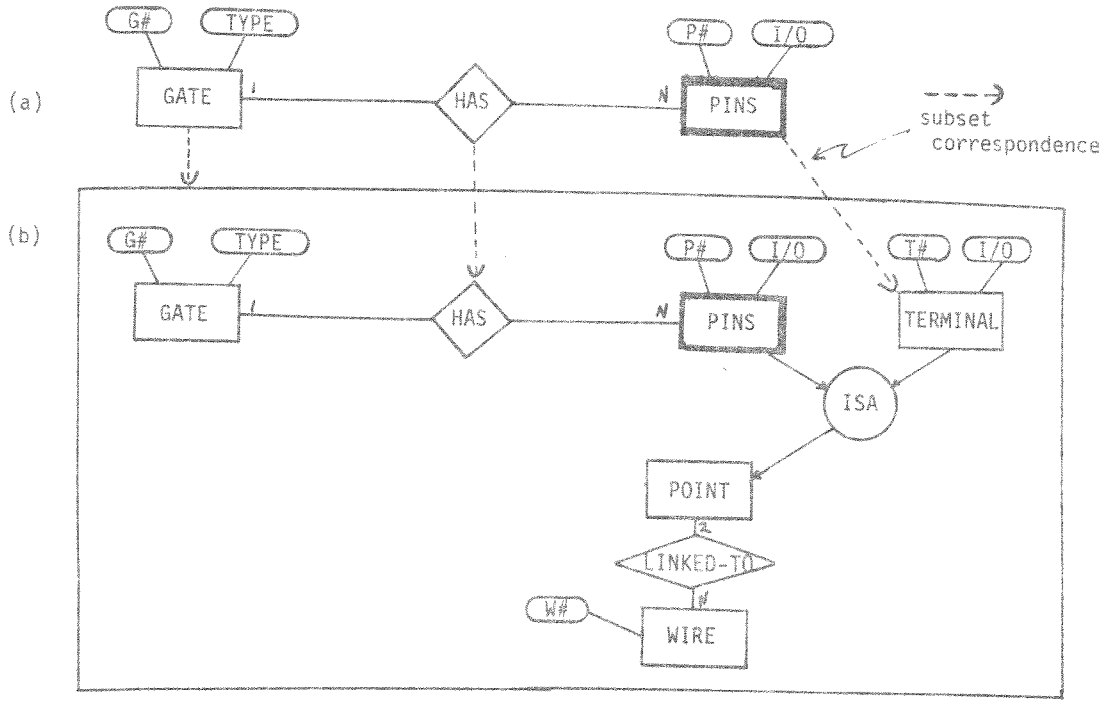


Figure 13. A Circuit Diagram



(c)

GATE(G#, TYPE, PARENT-G#)

(G1, 2-AND, G4)
(G2, 2-AND, G4)
(G3, 2-AND, G4)
(G4, 4-AND, -)

POINTS(G#, P#, I/O, PARENT-G#)

(G1, P1, I, G4)
(G1, P2, I, G4)
(G1, P3, O, G4)
(G2, P1, I, G4)
(G2, P2, I, G4)
(G2, P3, O, G4)
(G3, P1, I, G4)
(G3, P2, I, G4)
(G3, P3, O, G4)
(G4, P1, I, -)
(G4, P2, I, -)
(G4, P3, I, -)
(G4, P4, I, -)
(G4, P5, O, -)

WIRE(W#, START-G#, START-P#, END-G#, END-P#, PARENT-G#)

(W1, G4, P1, G1, P1, G4)
(W2, G4, P2, G1, P2, G4)
(W3, G4, P3, G2, P1, G4)
(W4, G4, P4, G2, P2, G4)
(W5, G1, P3, G3, P1, G4)
(W6, G2, P3, G3, P2, G4)
(W7, G3, P3, G4, P5, G4)

Figure 14. A Model of a Circuit Database

The model of our circuit has some deficiencies. Consider the POINTS table. It is evident that external feature tuples of 2-input AND gates are repeated. Clearly what is needed is the notion of templating: a *template* is a definition; instances of a template simply refer to this common definition. If the definition is changed, so are all of its instances. The source of the problem in our circuit example is not due to molecular aggregations, but rather it is with the E-R model itself. Templating appears to be a difficult concept to express in the E-R model; it requires special notations and rules for reducing template diagrams to tables. Expressing such tables in the Relational and network models is trivial. (System R, in fact, supports templating of complex objects).

A much better design would normalize the POINTS table by replacing it with a POINT-TEMPLATE(TYPE, P#, I/O) table that lists the external features common to all GATE instances. The relationship between the POINTS table and the GATE and POINT-TEMPLATE table is expressible in relational algebra as a join of GATE and POINT-TEMPLATE:

$$POINTS = \Pi_{(G\#,\ P\#,\ I/O,\ PARENT\text{-}G\#)}\ GATE[\ TYPE = TYPE]\ POINT\text{-}TEMPLATE$$

The contents of the POINT-TEMPLATE in our example would be:

POINT-TEMPLATE

(TYPE,	P#,	I/O)
(2-AND,	P1,	I)
(2-AND,	P2,	I)
(2-AND,	P3,	O)
(4-AND,	P1,	I)
(4-AND,	P2,	I)
(4-AND,	P3,	I)
(4-AND,	P4,	I)
(4-AND,	P5,	O)

It is clear from the above example that the E-R model, like the Relational and other models, has its limitations. In this and preceeding sections, we have explained and developed some basic concepts of molecular aggregation; concepts that can be applied to any model. It is beyond the

efficiently through the use of special storage structures. This essentially makes logical connections, such as PARENT-G#, unnecessary from a practical viewpoint, although from a purely logical modeling viewpoint - one that is implementation independent - it is not altogether satisfactory.

scope of this paper to correct problems that are peculiar to a particular notation, in this case templating and the E-R model. We will address the templating problem in future research.

In the following section, examples are presented of nondisjoint molecular objects that occur in non-traditional database applications.

3. Examples of Molecular Aggregation Taken From Non-Traditional Applications

A preliminary data analysis for a fully integrated chemical process plant resulted in the identification of approximately 8000 items. The analysis spanned from the process design, and instrumentation and piping to the complete specification of the plant itself. Because of the nature of the design process, different types of data were identified: namely, approved data for project-wide use, preliminary data used by designers working on a small subportion of the project (which is integrated with the project-wide data after approval), active and passive catalogs, and meta-data. The nature of this data and problems such as versioning, handling of design alternatives, handling long interactive transactions, and the control of update propagations resulted in a scheme which relied on different databases that interact ([Buc83]). A project-wide database was defined to contain approved data, and workspaces were defined for exploring preliminary solutions and design alternatives. Setting up workspaces often required massive extraction of data. Although molecular objects (as defined in Section 2) were not supported by the CODASYL-based DBMS that was used to manage these databases, expressing data aggregates as molecular objects would have considerably simplified the extraction specification.

Some of the 'molecular objects' of this database were pieces of equipment, isometric views of different pipelines, and the whole plant modeled at a certain levels of abstraction. To illustrate some of the modeling and data management problems that were present, Figures 15, 16, and 17 are documents about a heat-exchanger: they respectively show a small portion of a process flow sheet (PFS), a piping and instrumentation diagram (P&ID), and a specification sheet. Although the specific details of a heat-exchanger are unimportant to this paper, what is important is that each document can be viewed as a molecular object. Although every document appears to be self-contained (i.e., molecularly disjoint), in fact they are not; the PFS and P&ID molecules share

some of the tuples that make up the specification sheet molecule. This is not unlike the schema and subschema molecules of Figure 7 where molecules of different types have underlying atoms in common. Clearly, data management problems arise in such cases: automatically deleting all the tuples that belong to a particular molecule can have unforseeable effects in other portions of the design. Currently available DBMSs cannot prevent inconsistencies when deleting non-disjoint molecules.

While the previous example shows non-disjointness among molecules of different types, there are also examples of non-disjoint molecules that are of the same type. A pipeline consists of straight runs of pipes and fittings, such as 'L' and 'T' fittings, flanges, valves, etc. It is common to represent the connectivity of a pipeline as a list of these elements, quite similar to the example of non-disjoint lists in Figure 3. Each pipeline is a molecular object, and since pipelines normally merge or branch into other pipelines, there is a sharing of elements (i.e., atoms). Once again, data management problems arise: when a branch is eliminated from a pipeline its elements are dropped, and a 'T' fitting and connecting runs of pipe are replaced by a longer run of pipe. Because of the sharing, updates may cause inconsistencies.

Examples of recursive molecules can be found in other non-traditional applications. Maps in geographic databases are examples ([Bar82], [Bar84]). Each level of recursion is given a different name: blocks (the smallest surveyed region) are aggregated into counties, counties aggregate to states, states to countries, and so on. Although the regions represent disjoint areas, they share common boundaries. If the boundaries of a region are to be represented non-redundantly, then region-boundary molecules will need to be non-disjoint.

The examples presented above are only a small sampling of cases where molecular aggregation (disjoint vs. non-disjoint, recursive vs. non-recursive) is commonly encountered. They show that a complete framework for handling molecular aggregates is required if a DBMS is to respond adequately to the demands of non-conventional database applications.

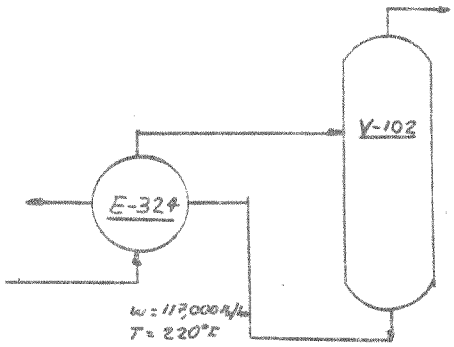


Figure 15. A Process Flow Sheet for a Heat Exchanger

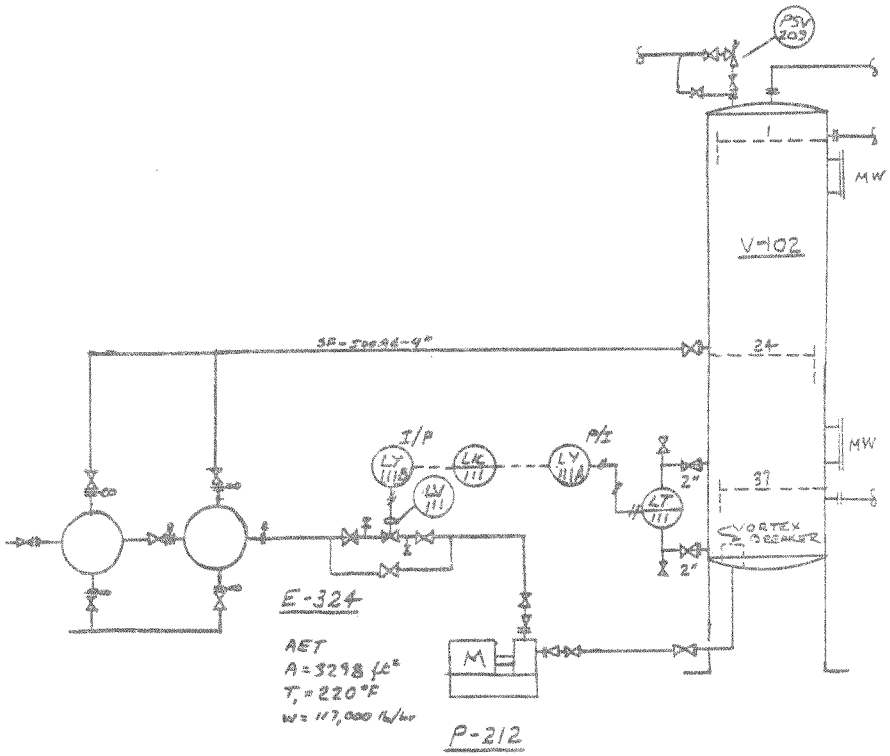


Figure 16. A Piping and Instrumentation Diagram for A Heat Exchanger

CLIENT XYZ Company
LOCATION AUSTIN Tx
E-324
CLIENT JOB NO.
JOB NO. 8301
DATE 9/12/83
BY RGM

EXCHANGER SPECIFICATION SHEET

STRIPPER FEED HEATER

SIZE 24" x 240 AET CONNECTED IN PARALLEL
SURFACE PER UNIT 3298 ft² SHELLS PER UNIT 2 SURFACE PER SHELL 1649 ft²

PERFORMANCE OF ONE UNIT

		SHELL SIDE IN	SHELL SIDE OUT	TUBE SIDE IN	TUBE SIDE OUT
FLUID CIRCULATED		STRIPPER BOTTOMS		STRIPPER FEED	
DESCRIPTION		STRIPPED WATER		SOUR WATER	
TOTAL FLUID ENTERING	#/HR	92,000		117,000	
VAPOR	#/HR	-0-	-0-	-0-	-0-
LIQUID	#/HR	102,000	102,000	117,000	117,000
NON-CONDENSIBLES	#/HR	-0-	-0-	-0-	-0-
SPECIFIC GRAVITY	@T	0.95	0.99		
MOLECULAR WT	VAPOR	—	—	—	—
MOLECULAR WT	NON-COND	—	—	—	—
VISCOSITY	CENTIPOISES	0.24	0.30	0.42	0.27
SPECIFIC HEAT	BTU/#	1.0	1.0	1.0	1.0
LATENT HEAT	BTU/#	—	—	—	—
THERM COND	BTU/HR FT² °F/FT	0.396	0.394	0.362	0.392
TEMPERATURE	°F	242	163	100	200
OPERATING PRESSURE	PSIG	120		100	
VELOCITY	FPS	—		20	
NUMBER OF PASSES		ONE		FOUR	
PRESSURE DROP	PSI	ALLOW 10	CALC 4	ALLOW 10	CALC 6

HEAT EXCHANGED-BTU/HR 11,700,000 CORRECTED LMTD LMTD 103
TRANSFER RATE-SERVICE 120.1 CLEAN 200.8 FOULING RESISTANCE: SHELL 0.001 TUBES 0.001

CONSTRUCTION

PRESSURE	PSIG	MAX OP 135	DES 175	MAX OP 115	DES 175
TEST PRESSURE	PSIG	263		263	
TEMPERATURE	°F	MAX OP 240	DES 300	MAX OP 220	DES 300
TUBES NO. 420	OD 3/4"	BWG 16	LENGTH 20 ft	PITCH 15/16" TRI.
TRANSVERSE BAFFLE-TYPE	SEGMENTAL	SPACING 14.2 IN	% CUT 25	DIRECTION HORIZ
LONG BAFFLE-TYPE	—	IMPINGEMENT BAFFLE	YES		

MATERIALS

ITEM	MATERIAL	SPECIFICATION	DESIGN TEMP.	THICKNESS	SR	IR
TUBES	CS	SA-214	300	0.065 IN	YES	NO
TUBESHEETS	CS	SA-515-70	300	1"	NO	NO
SHELL & COVER	CS	SA206	300	1/2"	NO	NO
CHANNEL & COVER	CS	SA206	300	1/2"	NO	NO
FLOAT HEAD COVER	CS	SA206	300	1/2"	NO	NO
BAFFLES & SUPPORTS	CS	SA206	300	1/4"	NO	NO

BOLTS-SHELL COVER — GASKETS-SHELL 1/16" ASBESTOS
CHAN-SHELL — CHANNEL
FLTG HEAD — FLTG HD COV
CHAN-COVER — TIERODS-SPACERS
CONNECTIONS: SHELL INLET 4" 150# RF — TUBE INLET 4" 150# RF
OUTLET 4" 150# RF — OUTLET 4" 150# RF
CLEANING — CLEANING 3/4" RF
DRAINS, VENTS, Etc. (1) 2" 150# RF RF — DRAINS, VENTS, Etc. —
CORROSION ALLOWANCE-SHELL 1/8" TUBE — INSULATION THICKNESS 2" CAL. SIL.
TEMA CLASS R — CODES TEMA, ASME — STAMP YES
WEIGHTS-SHELL 12,000 LB BUNDLE 5000 LB FULL OF WATER 22,000 LB
REMARKS

Figure 17. A Specification Sheet for a Heat Exchanger

4. Conclusions

A general framework for modeling molecular objects has been presented. The framework is unified by the concept of abstract data types and was shown to relate recent contributions on DBMS support for CAD, engineering, and statistical database applications. Although the framework was explained in terms of the E-R model, the ideas should be portable to all models.

Four types of molecular objects were identified: disjoint vs. non-disjoint and recursive vs. non-recursive. We have shown that all arise in practice yet no DBMS that we are aware supports all types. System R, for example, only handles disjoint molecules.

It is our belief that someday there will be a unification of the data abstraction mechanisms of programming languages and data models. Central to this belief is the programming language/data structure paradigm which was instrumental to the development of our model of molecular objects. The paradigm states that data structures are databases and logical data models should be able to describe them. Although unification is a long way off, we feel the ideas of molecular aggregation presented in this paper are a step forward to this goal. Further research should tie operations to molecular objects.

References

[Aho74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Mass., 1974.

[Bar82] R. Barrera and A.P. Buchmann, 'Schema Definition and Query Language for a Geographical Database System', *CAPAIDM workshop,* Hot Springs, Virginia, Nov. 1982.

[Bar84] R. Barerra, 'GEOBASE - A Reconfigurable Geographic Database System', IIMAS Tech. Rep., Feb. 1984.

[Bat83] D.S. Batory, 'Modeling the Physical Structures of Commercial Database Systems', Tech Rep. TR-83-21, Dept. of Computer Sciences, University of Texas at Austin, 1983.

[Bat84] D.S. Batory, 'Conceptual-to-Internal Mappings in Commercial Database Systems', *ACM 1983 PODS.*

[Bro83] V.A. Brown, S.B. Navathe, and S.Y.W. Su, 'Complex Data Types and Data Manipulation Language for Scientific and Statistical Databases', *Proc. 1983 International Workshop on Statistical Database Management,* Los Altos, California.

[Buc79] A.P. Buchmann and A.G. Dale, 'Evaluation Criteria for Logical Database Design', *CAD* 11, May 1979.

[Che76] P.P.S. Chen, 'The Entity-Relationship Model - Toward a Unified View of Data', *ACM Trans. Database Syst.* 1 #1 (March 1976), 9-36.

[Cod79] E.F. Codd, 'Extending the Database Relational Model to Capture More Meaning', *ACM Trans. Database Syst.* 4 #4 (Dec. 1979), 397-434.

[Geh83] N. Gehani, *ADA: An Advanced Introduction,* Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

[Ger84] M. Gerzo, 'TM - An Object Oriented Language', IIMAS Tech. Rep. 84-01, Jan. 1984.

[Has82] R. Haskin and R. Lorie, 'On Extending the Functions of a Relational Database System', *ACM SIGMOD 1982,* 207-212.

[Hor78] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms,* Computer Science Press, Rockville, Maryland, 1978.

[Joh83] H.R. Johnson, J.E. Schweitzer, and E.R. Warkentine, 'A DBMS Facility for Handling Structured Engineering Entities', *Proc. 1983 ACM Engineering Design Applications,* 3-12.

[Kat82] R. Katz, 'DAVID: Design Aids to VLSI Using Integrated Databases', *IEEE Database Engineering* 5 #2 (June 1982), 29-32.

[Knu73] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching,* Addison-Wesley, Reading, Mass., 1973.

[Laf83] G.M.E. Lafue, 'Basic Decisions About Linking an Expert System with a DBMS: A Case Study', *IEEE Database Engineering* 6 #4 (Dec. 1983), 56-64.

[Lis77] B.H. Liskov, et al., 'Abstraction Mechanisms in CLU', *Comm. ACM* 20 #8 (Aug. 1977),

564-576.

[Lor83a] R. Lorie and W. Plouffe, 'Complex Objects and Their Use in Design Transactions', *Proc. 1983 ACM Engineering Design Applications,* 115-121.

[Lor83b] R. Lorie, et al., 'User Interface and Access Techniques for Engineering Databases', to appear in *Query Processing in Database Systems,* W. Kim, D.S. Batory, and D. Reiner, ed., Springer-Verlag 1983.

[Row79] L.A. Rowe and K.A. Shoens, 'Data Abstractions, Views and Updates in RIGEL', *ACM SIGMOD 1979,* 71-81.

[Sch77] J.W. Schmidt, 'Some High Level Language Constructs for Data of Type Relation', *ACM Trans. Database Syst.* 2 #3 (Sept. 1977), 247-261.

[Smi77] J.M. Smith and D.C.P. Smith, 'Database Abstractions: Aggregation and Generalization', *ACM Trans. Database Syst.* 2 #2 (June 1977), 105-133.

[Smi83] R.G. Smith, 'STROBE: Support for Structured Object Knowledge Representation', *Proc. 1983 IJCAI.*

[Sto83] M. Stonebraker, B. Rubenstein, and A. Guttman, 'Application of Abstract Data Types and Abstract Indices to CAD Databases', *Proc. 1983 ACM Engineering Design Applications,* 107-114.

[Was79] A.I. Wasserman, 'The Data Management Facilities of PLAIN', *ACM SIGMOD 1979,* 60-70.

[Web78] H. Weber, 'A Software Engineering View of Data Base Systems', *VLDB 1978,* 36-51.

[Wul76] W.A. Wulf, R.L. London, and M. Shaw, 'An Introductin to the Construction and Verification of ALPHARD Programs', *IEEE Trans. Soft. Engr.* (Dec. 1976), 253-265.