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 occur frequently in CAD and engineering applications. At higher levels of abstraction they are treated 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, which 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.

1. Introduction

The relationship of abstract data types to databases is becoming progressively more important. Some time ago, abstract data types were identified as an important connection between programming languages and databases ([Sch77], [Web78], [Row79], [Was79]). More recently, they have been instrumental in supporting CAD, engineering, and statistical database applications ([Sto83], [Lor83a-b], [Joh83], [Su83], [Bro83a], [Has82]).

Presently there are two distinct approaches to the integration of abstract data types to logical data models. One approach introduces sophisticated data types to augment the standard data types (e.g., integer, string, real, etc.) that underly relations. The works of Stonebraker et al. and Su et al. are instances of this approach.

Stonebraker et al. ([Sto83]) have advanced the idea that user-defined types are fundamental to the support of non-traditional database applications. These types and their associated operators are called 'abstract data types'. Examples include linear and multidimensional arrays, polygons, and complex numbers. ADT-INGRES is an implementation of this proposal ([Ong84]).

Su et al. ([Su83], [Bro83a]) have proposed that a set of system-defined types (e.g., set, vector, matrix, time series) are required for DBMS support of statistical database applications. In addition, they proposed ways 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 column of a relation could be a system-defined type or a relation. The resulting data types and their attendant operators are called 'complex data types'.

The second distinct approach does not deal with columns of a relation, but rather treats collections of heterogeneous tuples as objects. These objects, which we will call *molecular objects*, have the property that they are given different representations at different levels of abstraction. At higher levels, a molecular object is represented by a single tuple. At lower levels, it is represented by a set of interrelated tuples from different relations. Molecular objects are instances of object types, which are defined in terms of more primitive types and their interrelationships. Thus, molecular objects are analogous to abstract data types. Lorie et al. ([Lor83a-b], [Has82]) and Johnson et al. ([Joh83]) have independently proposed restricted notions of molecular objects. Lorie's objects are called 'complex objects'; Johnson's are called 'structures'.

As an example of molecular objects, consider a catalog of data structure diagrams. On each page of the catalog is a data structure diagram of some database that is being designed. 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.

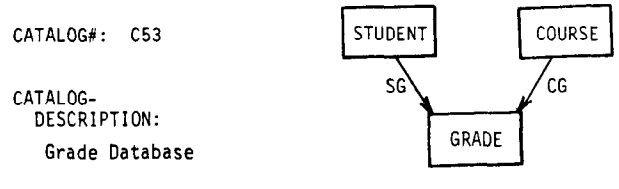


Figure 1. An Entry in a Catalog of Data Structure Diagrams

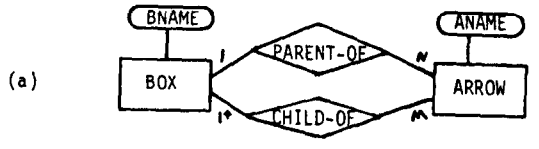
This work was supported by the National Science Foundation under Grant MCS-8317353.

Suppose the contents of this database are described using the notation of the E-R model ([Che76]). An E-R diagram of a data structure diagram is shown in Figure 2a. A data structure diagram is composed of two types of constructs: boxes and arrows. Each box and arrow has a name. An arrow connects one box (the owner record type) to one or more different boxes (the member record types). [1] 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. [2] It is this set of seven tuples that represents the molecular data structure diagram object/entity 'C53'. Because the 'C53' entity and its relationship to these seven tuples are not represented, the E-R diagram of Figure 2 is incomplete.

Using standard modeling techniques, two possible ways of completing this diagram are shown in Figure 3. Both utilize the Smith and Smith notion of aggregation ([Smi77a-b]), which we will call *atomic aggregation*. Atomic aggregation is denoted in E-R diagrams by drawing a box around the relationship to be aggregated. Both PARENT-OF and CHILD-OF relationships are aggregated in Figures 3a-b.

Figures 3a-b show the addition of data structure diagram (DSD) entities whose attributes are catalog number (CAT#) and catalog description (CAT-DES). Figure 3a relates DSD entities to their underlying BOX, ARROW, PARENT-OF, and CHILD-OF entities by separate relationships. Note that BOX, ARROW, PARENT-OF, and CHILD-OF entities are shown as *weak entities* as they are existent dependent on DSD entities. That is, if a DSD entity is deleted, so are its dependent entities. Figure 3b



(b)

BOX(BNAME)	ARROW(ANAME, PARENT-BNAME)	CHILD-OF(ANAME, BNAME)
(STUDENT)	(SG, STUDENT)	(SG, GRADE)
(GRADE)	(CG, COURSE)	(CG, GRADE)
(COURSE)		

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

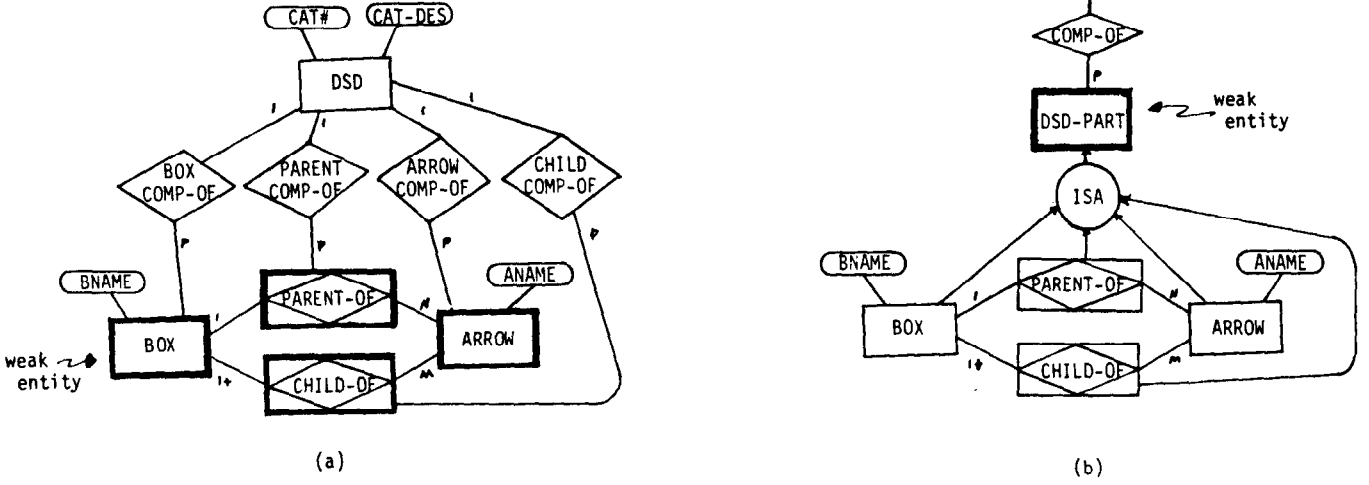


Figure 3. Approximate Models of a Molecular Object

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

[2] 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(BNAME, ANAME) table has been merged with the ARROW(ANAME) 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.

uses the Smith and Smith notion of generalization ([Smi77a]) to define an entity set DSD-PART which is the union of the BOX, ARROW, PARENT-OF, and CHILD-OF entity sets. DSD-PARTs are related to DSD entities by the COMP-OF relationship.

Both solutions are unsatisfactory for two reasons. First, existing modeling techniques and their notations fail to clearly indicate the molecular nature of DSD entities and the level of abstraction that separates DSD entities from their components. Database users are aware of this distinction, but it is not evident from the data model itself. This problem is not isolated to just the E-R model. Even models that are thought of as being semantically rich do not provide the necessary concepts for modeling molecular objects satisfactorily ([Cod79], [Shi80], [Che76]). In Section 3, we will return to this example and show how it can be modeled correctly.

Second, existing techniques fail to capture the operational semantics that are normally associated with molecular objects. The retrieval of a molecular object, for example, results in the output of a database of tuples (i.e., multiple relations and their occurrences) that comprise the object (see [Lor83b]). Again, database users are aware of such semantics, but there is little or no DML support for such operations.

The interest in molecular objects stems from their semantics and utility. There is a growing need to provide users with data modeling capabilities and run-time support for describing, manipulating, and retrieving molecular objects as primitives. It is certainly the case that molecular objects can be stored in existing DBMSs. However, the semantics and operations associated with these objects are completely specified by users and are buried in their application programs. We believe that support for molecular objects should be an integral part of future DBMSs.

As a first step toward this goal, we present in this paper data modeling techniques for molecular objects. A type of aggregation, called *molecular aggregation,* is introduced. Instrumental to the development of our model is the *programming language/data structure paradigm.* This paradigm asserts that data structures (trees, lists, etc.) can be viewed as molecular objects, and the power of a data model can be tested in terms of its ability to accurately represent the relationships among the nodes of a data structure. The paradigm enables us to identify four distinct types of molecular objects: disjoint/nondisjoint and recursive/nonrecursive. No existing DBMS that we are aware presently supports all of these types.

We believe our work is a prerequisite to the study of operational semantics and DBMS run-time support for molecular objects. Although we do not examine run-time support in this paper, important contributions to this subject in the context of disjoint molecules have already been made ([Has82], [Lor83b]).

We begin our discussions with an explanation of the programming language/data structure paradigm and its relationship to abstract data types and molecular objects.

2. The Programming Language/Data Structure Paradigm

Programming languages and databases are coming progressively closer ([Bro83b]). Many of the data abstraction capabilities that are present in today's advanced programming languages are only now appearing in logical data models. Although a unification of databases and programming languages is a long way off, clues to a unification can be found in common examples of abstract data types, namely data structures. Data structures are main-memory databases where nodes of a data structure can be viewed as tuples in one or more (main-memory) relations. Thus, data structures can be modeled as molecular objects. This connection proves to be quite useful in two ways. One is identifying different types of molecular objects. Another is testing the power of logical data models. We will briefly consider each in turn.

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. From the programming language/data structure paradigm, there are implementations of abstract data types (i.e., molecules) that are *not* disjoint. Figure 4 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. Johnson et al. ([Joh83]) present other examples of non-disjoint molecules. [3]

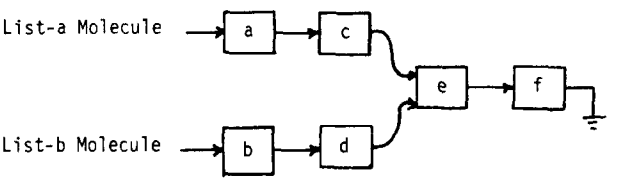


Figure 4. Two Non-disjoint Molecules

Molecules are *recursive* if they are composed of other molecules of the same type. A linked list, for example, can be defined recursively as a node followed by another (possibly empty) linked list. As another example, a circuit is a graph which can be defined recursively as interconnections between more primitive circuits (graphs). Most molecules, however, are not recursive. The DSD molecule of Figures 2 and 3 is an example.

It is our belief that disjoint/nondisjoint and recursive/nonrecursive molecules occur naturally and are quite common. We conjecture that as more CAD, statistical, and other special-purpose applications are supported by database systems, restricting the type of molecules that are supported (as is done in System R) will not be sufficient to handle the needs of many applications. A general facility

[3] Our notion of disjointness deals with molecules of a single type. The atoms of molecules of *different* types may overlap, thus giving rise to another notion of disjointness. We examine this topic in more detail in Section 3.2

for handling molecular objects is needed. In the following section, we will discuss at length the modeling of these different types.

A second benefit of the programming language/data structure paradigm is a way of testing the power (or demonstrating the limitations) of a data model. This can be done by modeling fundamental data structures as found in standard texts ([Knu73], [Aho74], [Hor78]). As we did in Figure 4, 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 this technique several times in this paper to illustrate and develop the modeling concepts that are proposed later.

In the next section, we present a general framework for studying molecular objects. In Section 4 we give additional examples of molecular objects that are taken from non-traditional database applications.

3. 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 ([Smi77a-b], [Cod79], [Dat82], [Has82], [Lor83a-b], [Sto83]).

Atomic aggregation ([Smi77a-b]) is an abstraction of a *single* relationship into a higher level entity. *Molecular aggregation* is an extension of this concept; it is an abstraction of a *set* of entities and their relationships into a higher level entity.

We believe that the concept of molecular aggregation is independent of the model or notation that is used to express it. However, for the purposes of exposition, 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. [4]

In the following sections, models of disjoint/nondisjoint and recursive/non-recursive molecules are presented. These models are progressively developed and are illustrated by a set of examples.

3.1 Modeling Non-Recursive, Disjoint Molecular Objects

Levels of abstraction allow molecular objects to have different representations. At higher levels, a molecular object is an atomic entity; at lower levels, the atoms that compose the molecule are seen. At any particular level, existing modeling techniques 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 5a.

At a lower level, the implementation details of data structure diagrams are captured by the E-R diagram of Figure 5b (Figure 2a). 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. [5] 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.

A general scenario for non-recursive, disjoint molecules is shown in Figure 6a. 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 $A_{i,1} \cdots$. 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. 6b). [6]

In the case of Figures 5a-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 5c.

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 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

[4] There is no guarentee that the tables produced will be in a certain 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.

[5] Note the notational difference between atomic aggregation and molecular aggregation. An atomic aggregation is shown as a box drawn around a single E-R *relationship*. A molecular aggregation is shown as a box drawn around an E-R *diagram*.

[6] 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.

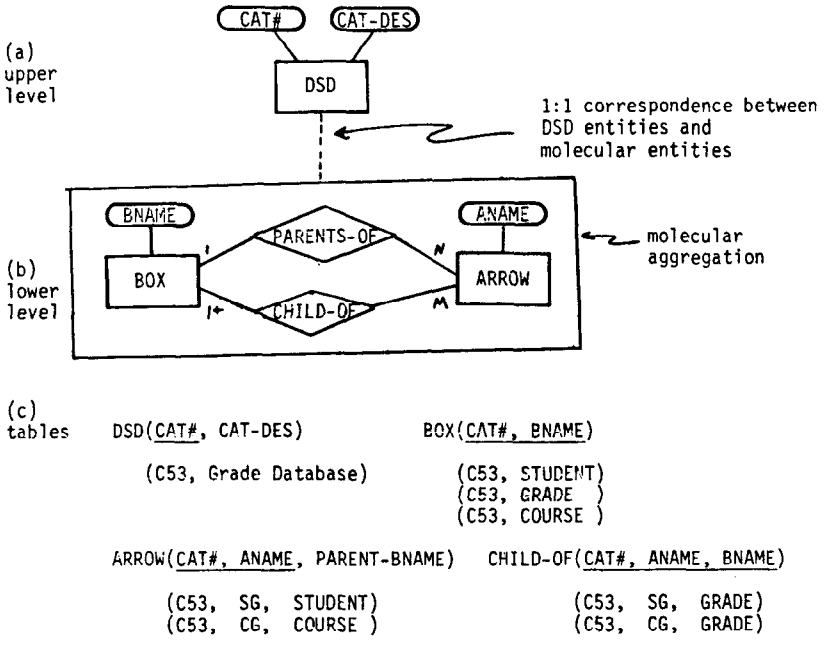


Figure 5. A Non-recursive, Disjoint Entity

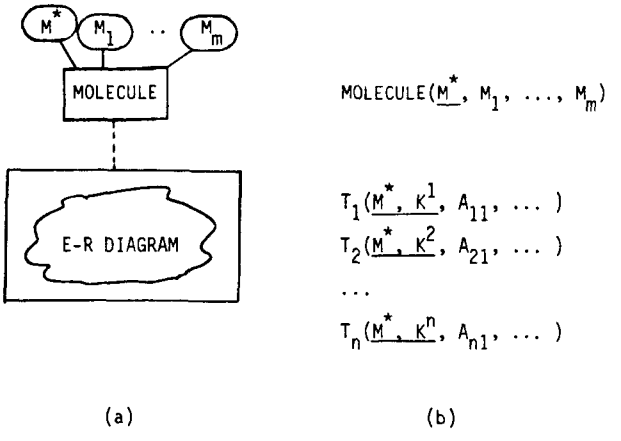


Figure 6. A Model of Non-recursive, Disjoint Molecules

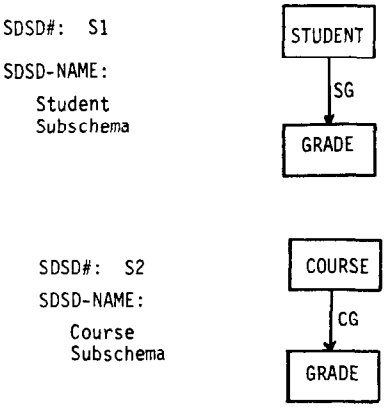


Figure 7. Subschemas of Figure 1

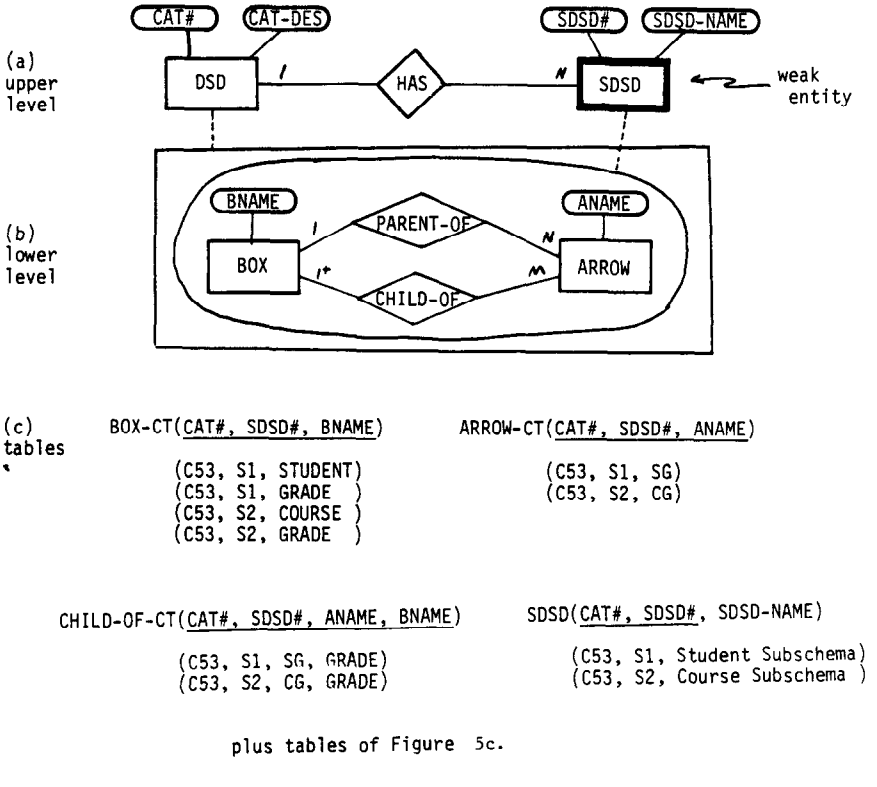


Figure 8. A Non-Recursive, Non-Disjoint Entity

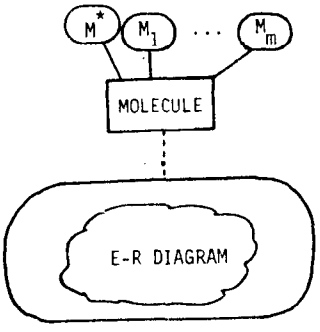
table at the lower level. In contrast, System R generates *surrogates* for 'entities' that are unique across the entire database ([Cod79]). This approach avoids the problem of key inheritance but requires an internal-level solution to the problem of relating a molecule tuple to its underlying atom tuples ([Lor83b]).

3.2 Modeling Non-Recursive, Non-Disjoint Molecular Objects

Suppose that data structure diagrams of subschemas for each catalog entry are also to be present in the database (see Fig. 7). 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. 8a). Note that SDSD entities are weak entities because they are existence dependent on DSD entities.

Because different subschemas may share record and set types, the sets of tuples that underly two SDSD molecules need not be disjoint. In Figure 7, the SDSD molecules for subschemas S1 and S2 will share the tuple that describes the GRADE record type. We denote non-disjoint molecular aggregation by circling the E-R diagram whose instances define a molecule. Figure 8a-b shows both disjoint and nondisjoint aggregations and their correspondences.

A general scenario of non-recursive, non-disjoint molecules is shown in Figure 9a. 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') with the primary keys of their molecular entities (M^*). (M^*, K') 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. 9b).



(a)

MOLECULE($\underline{M^*}$, M_1, ..., 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 9. Modeling Non-Recursive, Non-Disjoint Molecules

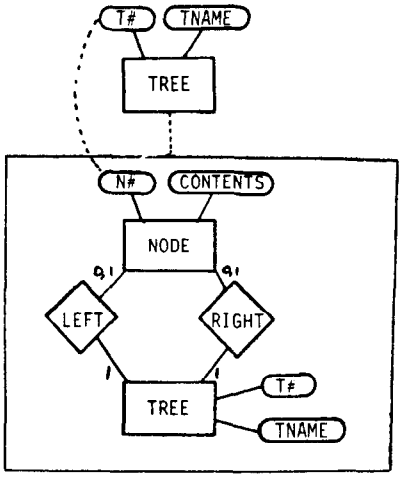
In the case of Figures 8a-b, this procedure yields tables DSD($\underline{CAT\#}$, CAT-DES) and SDSD($\underline{CAT\#, SDSD\#}$, SDSD-NAME) at the upper level and BOX($\underline{CAT\#, BNAME}$), ARROW($\underline{CAT\#, ANAME}$, PARENT-NAME), CHILD-OF($\underline{CAT\#, ANAME, BNAME}$), BOX-CT($\underline{CAT\#, SDSD\#, BNAME}$), ARROW-CT($\underline{CAT\#, SDSD\#, ANAME}$), and CHILD-OF-CT($\underline{CAT\#, SDSD\#, ANAME, BNAME}$) at the lower level. These tables and their tuples are shown in Figure 8c.

It is worth noting that the notion of disjointness on which our discussions have centered deals with molecules that are of a single type. Disjointness can also be defined for molecules of different types. Figure 8 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 4.

3.3 Modeling Recursive, Disjoint Molecular Objects

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 10a. 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($\underline{T\# \text{ or } N\#}$, CONTENTS, LEFT-T#, RIGHT-T#, TNAME)

Figure 10. 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 is 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 a shift in their semantics is possible in going from one level of abstraction to the next. We will see that such shifts are common.

A general scenario for recursive, disjoint molecules is shown in Figure 11a. 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 $A_{i,1} \cdots$. The MOLECULE table has primary key M^* and has descriptive attributes M^{**} and $M_1 \cdots M_m$ (see Fig. 11b).

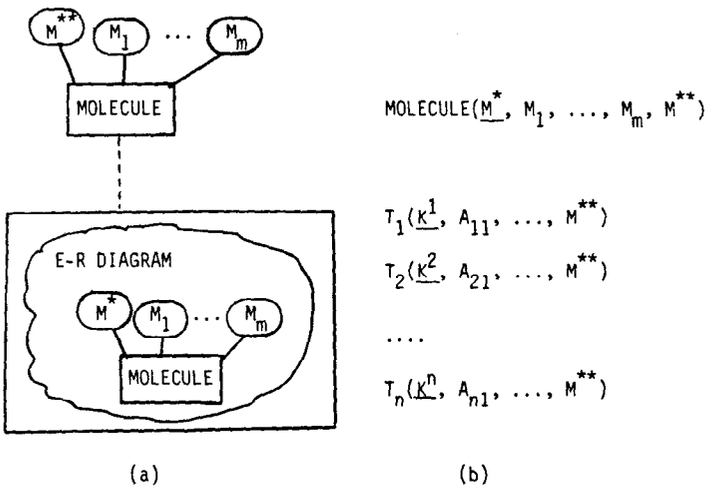


Figure 11. Modeling Recursive, Disjoint Molecules

Two tables result in applying this procedure to Figure 10a: TREE(T#, TNAME, PARENT-T#) and NODE(N#, CONTENTS, LEFT-T#, RIGHT-T#, PARENT-T#).[7] 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 that recursive E-R diagrams commonly 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, 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. 10b) 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.

3.4 Modeling Recursive, Non-disjoint Molecular Objects

Earlier we gave the example of two list molecules sharing the same nodes. Figure 12a 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).

A general scenario for recursive, non-disjoint molecules is shown in Figure 13. The procedure for reducing such diagrams to tables is identical to that of reducing non-recursive, non-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.

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

[7] A table underlies each entity set and relationship set. In Figure 10a 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.

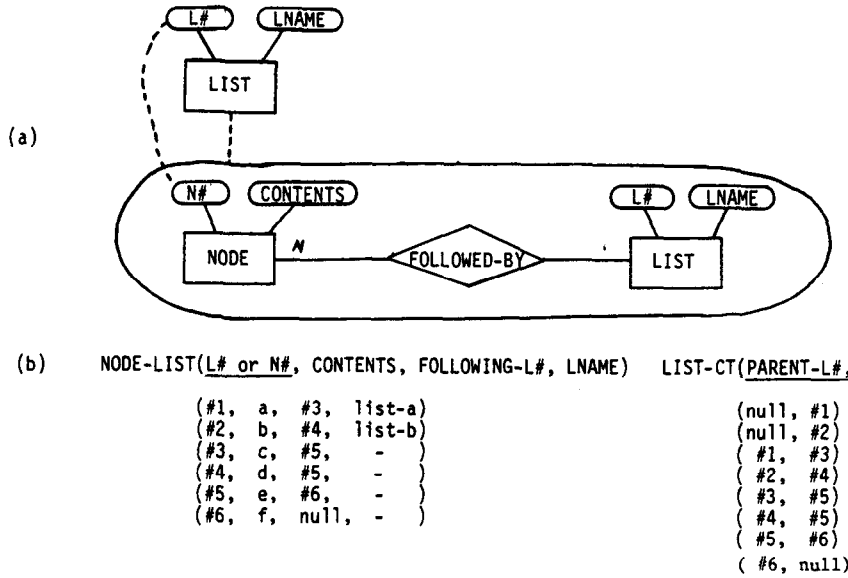


Figure 12. Non-Disjoint Lists

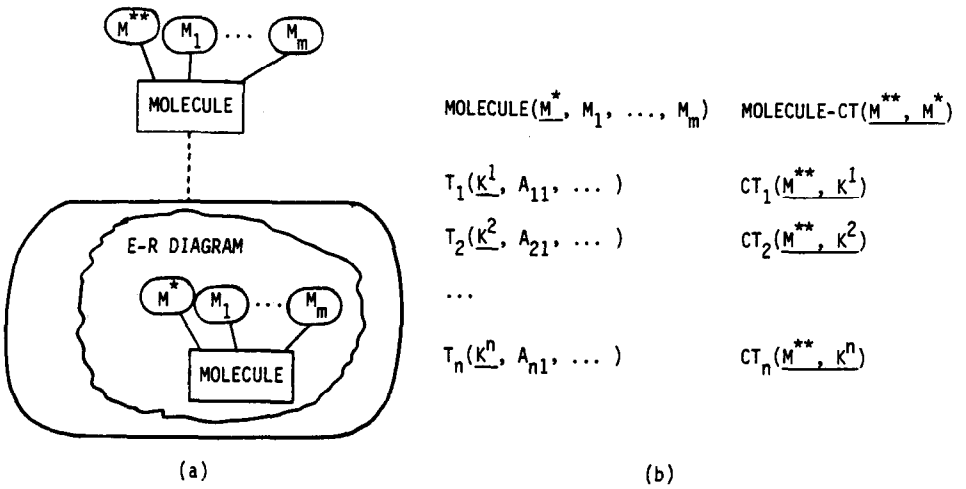


Figure 13. Modeling Recursive, Non-Disjoint Molecules

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. 12b).

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.

3.5 Modeling External Features of Molecular Objects

The previous sections have described general procedures 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 ([Lis77]), ADA ([Geh83]), and other programming languages is that instances of abstract data types are featureless entities. 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 [2,3] of matrix M is such an example. For this reason, projection operators (i.e., [i,j])

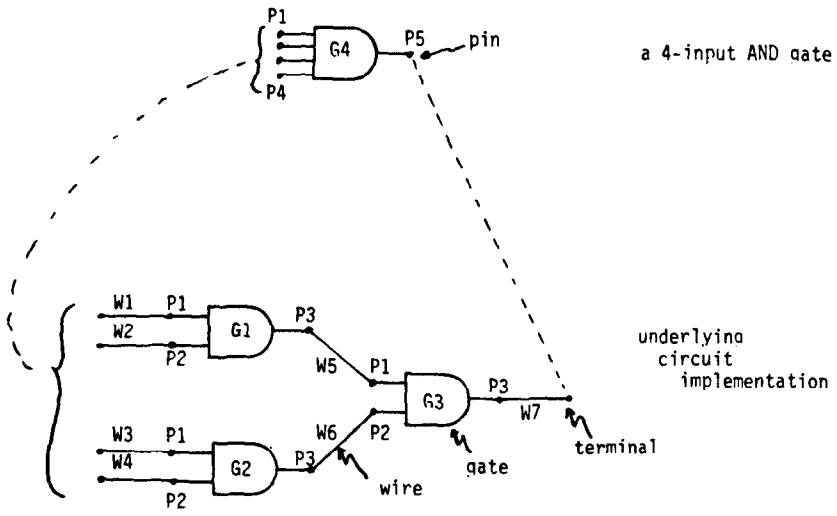
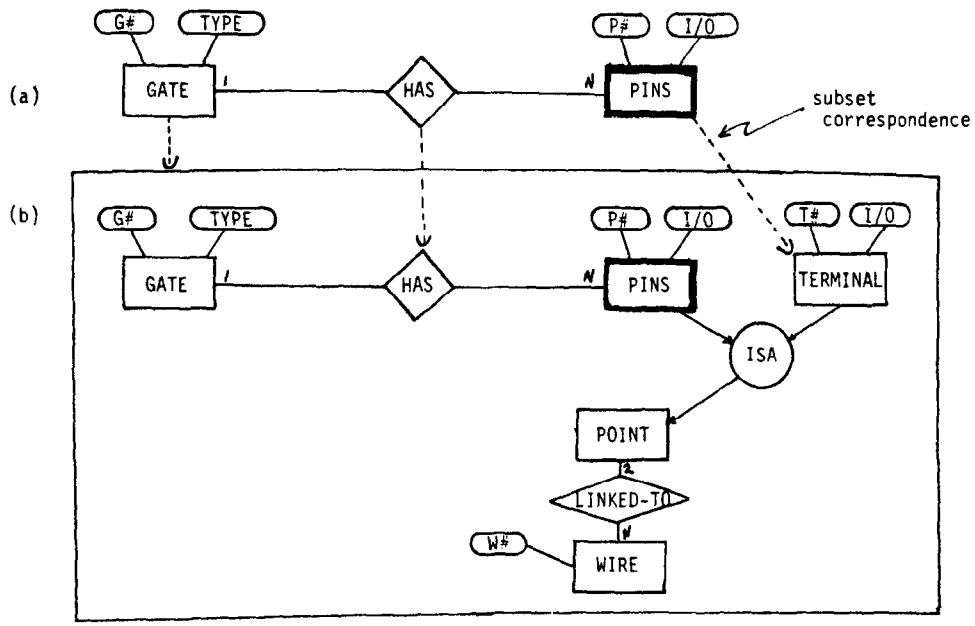


Figure 14. A Circuit Diagram



(c)

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

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

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)

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, -)

Figure 15. A Model of a Circuit Diagram

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 14 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. 15a). 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 15a 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 15a-b are given in Figure 15c. 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. [8]

The model of our circuit has some deficiencies. Consider the POINTS table. It is evident that external features of 2-input AND gates are repeated. That is, each two-input AND gate (e.g., G1 - G3) has two input pins (P1 and P2) and an output pin (P3). Pin names and their I/O functions are external gate features; data on these features are duplicated in each gate instance. 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 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 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.

4. 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 metadata. 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

[8] It is worth noting that System R handles the problem of finding underlying tuples of complex objects 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 may not be altogether satisfactory.

a scheme which relied on different databases that interact ([Buc84]). 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 3) 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 different levels of abstraction. To illustrate some of the modeling and data management problems that were present, Figures 16, 17, and 18 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 8 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 4. 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.

5. Conclusions

A general framework for modeling molecular objects has been proposed. 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.

A unification of the data abstraction mechanisms of programming languages and data models is inevitable. It is our conjecture that the programming language/data structure paradigm will be an important factor to the unification, just as it was instrumental to the development of our model of molecular objects. The paradigm states that data structures are main-memory 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. We have shown that some fundamental data structures can be modeled accurately using molecular aggregation and other data modeling constructs.

Four types of molecular objects were identified in our model/framework: 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. Since research on molecular objects is in its early stages, it is likely that additional types of molecular objects will be discovered. Further research should concentrate on run-time support for molecular objects.

Acknowledgements. We gratefully acknowledge the help of Sham Navathe for his constructive suggestions on improving an earlier version of this paper.

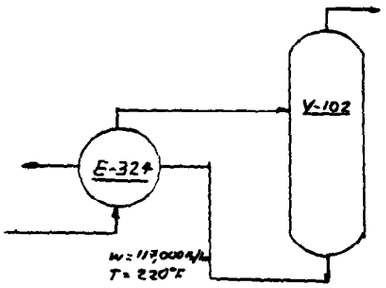


Figure 16. A Process Flow Sheet for a Heat Exchanger

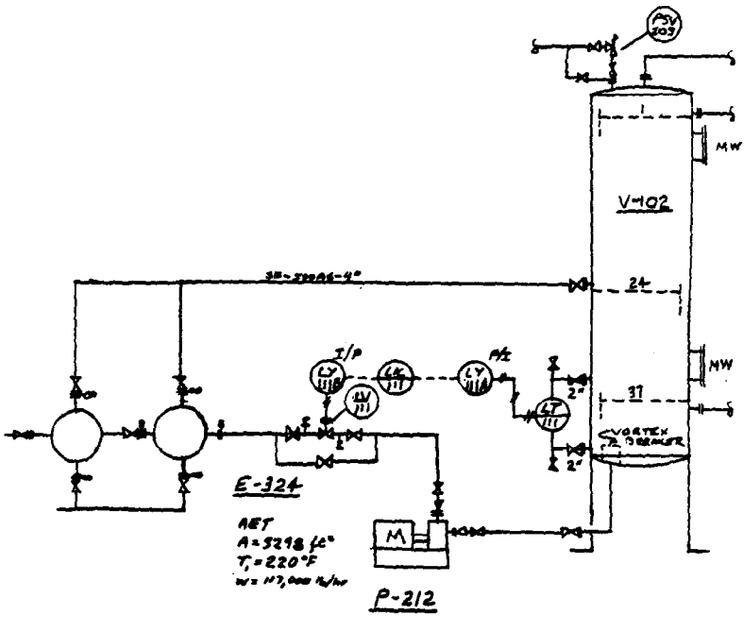


Figure 17. A Piping and Instrumentation Diagram for A Heat Exchanger

CLIENT XYZ Company
LOCATION AUSTIN, TX
E-324
CLIENT JOB NO.
JOB NO. 3301
DATE 9/12/83
BY RGM

EXCHANGER SPECIFICATION SHEET

STRIPPER FEED HEATER

SIZE 24" x 240 AET CONNECTED IN PARALLEL
SURFACE PER UNIT 3218 ft² SHELLS PER UNIT 2 SURFACE PER SHELL 1643 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	1/1	0.98	0.99	-0-	-0-
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	0.384	0.362	0.382
TEMPERATURE	°F	242	142	100	200
OPERATING PRESSURE	PSIG	120		100	
VELOCITY	FPS	—		7.0	
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 107
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	265				265			
TEMPERATURE	°F	MAX OP 250	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	SR
TUBES	CS	SA-214	300	0.065 IN	YES	NO
TUBESHEETS	CS	SA-516-70	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	1/2"	NO	NO
BOLTS-SHELL COVER	CS	SA206	300	1/2"	NO	NO

CHAN-SHELL		GASKETS-SHELL	1/16" ASBESTOS
FLTG HEAD		CHANNEL	
CHAN-COVER		FLTG HD COV	
CONNECTIONS: SHELL INLET	4" 150 # RF	TIERODS/SPACERS	
OUTLET	4" 150 # RF	TUBE INLET	4" 150 # RF
CLEANING	—	OUTLET	4" 150 # RF
DRAINS, VENTS, Etc.	(1) 2" 150# RF, RF	CLEANING	6" 150 # RF
CORROSION ALLOWANCE-SHELL	1/8" TUBE	DRAINS, VENTS, Etc.	—
TEMA CLASS R	CODES TEMA, ASME	INSULATION THICKNESS	2" CAL. SIL.
WEIGHTS-SHELL 12,000 LB	BUNDLE 5,000 LB	STAMP	YES
REMARKS		FULL OF WATER	22,000 LB

Figure 18. A Specification Sheet for a Heat Exchanger

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.

[Bro83a] 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, 188-195.

[Bro83b] M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, eds., *On Conceptual Modelling,* Springer-Verlag, 1984.

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

[Buc84] A.P. Buchmann, 'Current Trends in CAD Databases', *CAD,* to appear.

[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.

[Dat82] C.J. Date, *An Introduction to Database Systems, 3rd. Ed.* Addison-Wesley, 1982.

[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.

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

[Ong84] J. Ong, D. Fogg, and M. Stonebraker, 'Implementatin of Data Abstraction in the Relational Database System INGRES', ACM SIGMOD Record 14 #1 (March 84), 1-14.

[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.

[Shi81] D. Shipman, 'The Functional Data Model and the Data Language DAPLEX', *ACM Trans. Database Syst.* 6 #1 (March 1981), 140-173.

[Smi77a] J.M. Smith and D.C.P. Smith, 'Database Abstractions: Aggregation and Generalization', *ACM Trans. Database Syst.* 2 #2 (June 1977), 105-133.

[Smi77b] J.M. Smith and D.C.P. Smith, 'Database Abstractions: Aggregation' *Comm. ACM* 20 #6 (June 1977), 405-413.

[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.

[Su83] S.Y.W. Su, '*SAM**: A Semantic Association Model for Corporate and Scientific-Statistical Databases', *Infor. Sci.* 29 (1983), 151-199.

[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.