DATA STRUCTURE DIAGRAMS

By Charles W. Bachman

Successful communication of ideas has been and will continue to be a limiting factor in man's endeavors to survive and to better his life. The invention of algebra, essentially a graphic technique for communicating truths with respect to classes of arithmetic statements, broke the bond that slowed the development of mathematics.

Whereas "12+13=25" and "3+7=10" and "14+(-2)=12" are arithmetic statements, "a+b=c" is an algebraic statement. In particular, it is an algebraic statement controlling an entire class of arithmetic statements such as those listed.

Data Structure Diagrams

The Data Structure Diagram is also a graphic technique. It is based on a type of notation dealing with classes—specifically, with classes of entities and the classes of sets that relate them. For example, individual people and automobiles are entities. When they are taken collectively, they make two quite different classes of entities. On the other hand, all the automobiles belonging to a particular person constitute a set of entities that are subordinate to the owner entity.

The Data Structure Diagram has been used fruitfully over a period of five years by a limited but rapidly growing audience. This audience (where the technique originated) consists of the users of General Electric's Integrated Data Store (I-D-S) data management system. I-D-S employs language statements that directly support the relationships implied by the Data Structure Diagrams. The technique is now being used to study, document, and communicate information structures, even in those cases where no mechanized implementation is intended. The purpose of this paper is to document the technique of data structure diagramming so that it may be studied, evaluated, and put to work where it appears useful.

Definitions

Four terms: *entity, entity class, entity set,* and *set class* are central to the understanding of Data Structure Diagrams. This text will use the term *entity* to mean a particular object being considered; the term *entity class* will mean an entire group of entities which are sufficiently similar, in terms of the attributes that describe them, to be considered collectively. Many different entity classes may exist. The text will use the term *entity set* to mean a different kind of entity grouping—one that associates a group of entities of one entity class with one entity of a different entity class in a subordinate relationship. The concepts of entity class and entity set are independent of each other and can be thought of as being at right angles or orthogonal. Figure 1 illustrates this point.

CHARLES W. BACHMAN is Manager, Applications Technology, for General Electric, Phoenix, Arizona. He is the creator of G.E.'s Integrated Data Store (I-D-S), a generalized data storage and retrieval system. He is also a member of the COBOL Data Base Task Force.

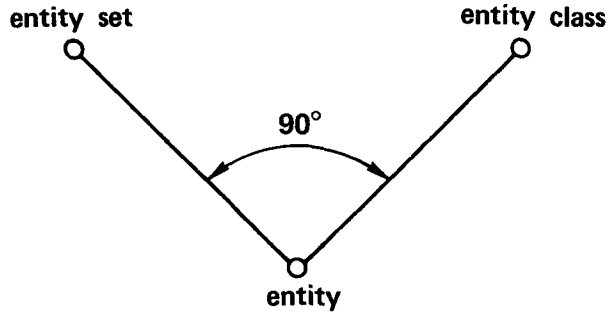


Figure 1

The term *set class* will be used in the text to mean an entire group of entity sets which are sufficiently similar, in terms of the attributes that describe them, to be considered collectively. Specifically, it is limited to those groups of sets in which the same entity-to-entity subordinate relationship exists. Figure 2 expands on Figure 1 to put all four of these terms into a spatial relationship.

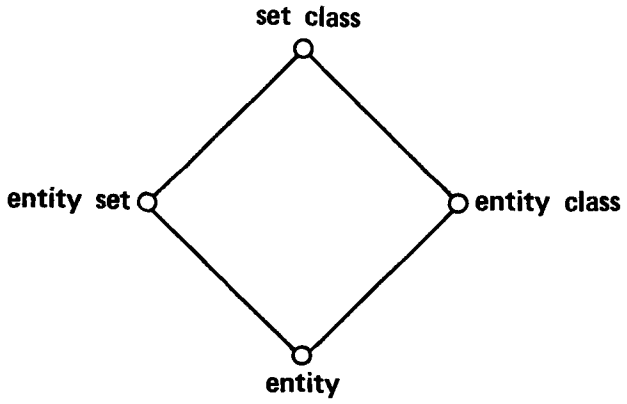


Figure 2

Many different *set classes* may exist. For example, the entities that we might consider in a management information system are the employees and the departments. All the employees in the company, when considered together, would make one entity class, while all the departments would make another entity class. Although the departments and employees may be considered independently of each other for some purposes, the relationship between the group of employees who work for the same department and that department may also be very important. Insofar as a department has a set of employees currently assigned to it, these employees can legitimately be considered as subordinate entities or sub-entities of that department. Each department is considered to be the *owner* of the set in which its employees are the *members.* When all of these

sets of employees are considered collectively they constitute a set class.

In a like manner, if employees, as an entity class, were considered in conjunction with their spouses and children, which comprise yet another entity class, then a set class could be established on the basis of the sets with employee entities as owners and their spouse and children as members. The concept of owner and member, the one owner to many member ratio, and the fact that these may be treated on a class basis, are central to the purpose and graphics of the Data Structure Diagram.

Graphic Symbols

The Data Structure Diagram technique uses two basic graphic symbols: the block, to represent an entity class; and the arrow, to represent a set class and to designate the roles of owner/member established by that set class. The arrow points from the entity class that owns the sets to the entity class that makes up the membership of the sets.

The diagram in Figure 3 states that an entity class exists and that an entity class name is to be assigned.

No information is implied as to how many entities make up the entity class. The only implication is that the entity class has been declared and is subject to such operations as may be defined.

CLASS OF ENTITIES

Figure 3

The diagram in Figure 4 states that two entity classes have been defined and that their entity class names are: "department" and "employee." If a particular company were being studied, there would be as many department entities and employee entities as that company had departments and people.

TWO CLASSES OF ENTITIES

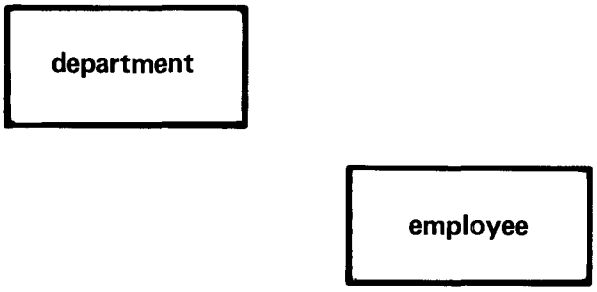


Figure 4

The diagram in Figure 5 states not only that two entity classes exist, but also that they are related by a set class named "assignment." The direction of the arrow is read to mean that each employee is a member of a set of employees that belong to a particular department, and further, that each department has such a set of employees.

SET ASSOCIATION OF ENTITIES

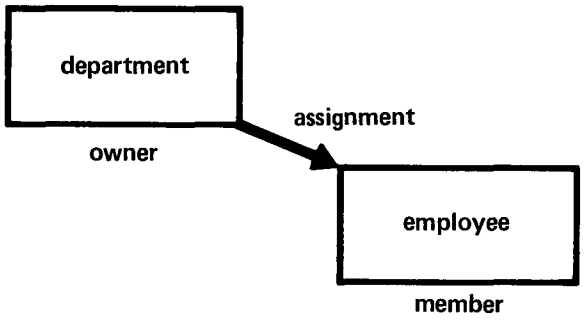


Figure 5

The Data Structure Diagram is topological in nature. Only the blocks, arrows, and names have meaning. The size, position, and proportion are selected for readability. Figure 6 is exactly equivalent to Figure 5, even if somewhat contorted.

TOPOLOGICAL GRAPHICS

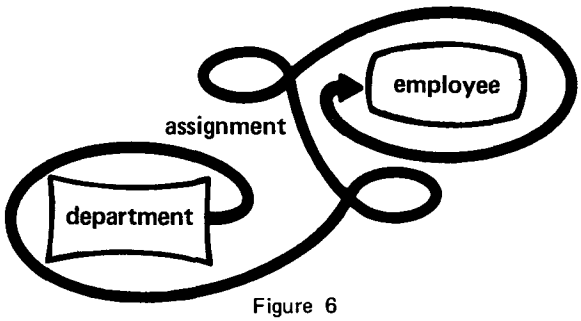


Figure 6

A Data Structure Diagram may contain as many blocks and arrows as necessary to establish the particular information structures under study. Any two entity classes may be associated as entity class/sub-entity class by zero, one, two, or more different set classes with the same or opposite ownership.

Hierarchies

The term hierarchy has been used rather ambiguously in the field of information technology. Data Structure Diagrams provide one possibility for non-ambiguous definition, i.e., an information hierarchy can be said to exist wherever there is a set-class relationship. Therefore, an information hierarchy exists whenever there are two or more levels of associated entity classes. Figure 7 integrates the department/employee association of Figure 5 with the employee/dependent association mentioned earlier to provide an example of a three-level hierarchy.

HIERARCHY

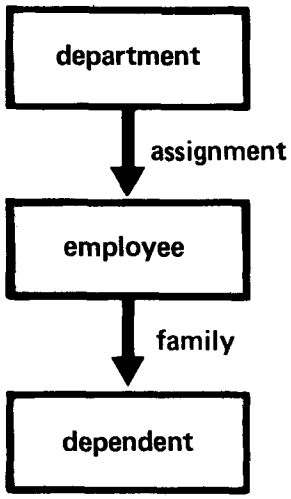
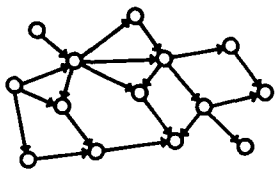


Figure 7

Many actual structures can be modeled either as a hierarchy, network or tree. When the elements in a real world hierarchy are like entities (i.e.,all people, all organizations) and their reporting level is subject to change, then a network or tree structure may prove to be more satisfactory in modeling the situation than a hierarchical structure.

Networks

Many information models involve networks of information. PERT or CPM diagrams are examples of such networks. Another example is the "T" account double-entry accounting system in which every entry affects the debit side of one account and the credit side of another account. Figure 8 is a generalized picture of a network with nodes connected to each other in a directed sense, or as a directed graph.



NETWORK

Figure 8

The Data Structure Diagram that defines a network is seen in Figure 9.

SIMPLE NETWORK STRUCTURE

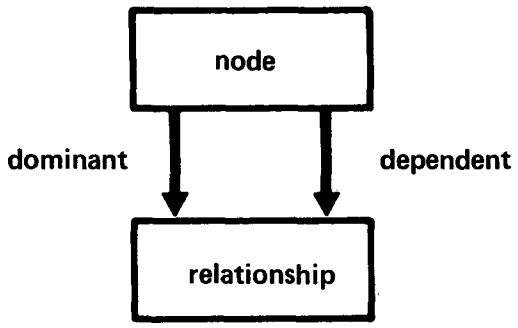


Figure 9

The two entity classes labeled "node" and "relationship" relate to the nodes and lines between nodes in Figure 8. By setting up a table of equivalences (Table 1), several different models which are network-oriented can be quickly defined. Please do not confuse the arrow direction in the network (Figure 8)—meaning the dominance of one node over another with the arrow direction in the network Data Structure Diagram (Figure 9) meaning an owner/member role.

APPLICATION	ENTITY CLASS NAME		SET CLASS NAME	
	NODE	RELATIONSHIP	DOMINANT	DEPENDENT
PERT/CPM	Event	Activity	Prior	Succeeding
GENERAL ACCOUNTING	Account	Transaction	Debit	Credit
PARTS LISTS	Material Item	Component	Call-Out	Where Used
GENEALOGICAL CHARTS	Subject	Relationship	Parent	Child
SUBROUTINE STRUCTURE	Subroutine	Call	Enter	Return
ORGANIZATION CHARTS	Organization	Component	Sub-Unit	Report-To

Table 1

The similarity of the PERT/CPM diagrams to a network is obvious. That of the general accounting model may be less obvious. But what are the transactions, except directed quantitative relationships between accounts (nodes); the trail balance should always be zero. Manufacturing parts lists consists of material items, which are made out of material items. Genealogical charts are similar to manufacturing parts lists except that each item is made from only two other things, its parents.

The interrelationship of a set of subroutines that call on each other is also a network because each subroutine may call many subroutines or be called by many subroutines.

Tree Structures

Organization charts usually are special cases of a network, the tree structure, in which each node has one

dependent relationship and many dominant relationships. I say usually, because military organizations are rife with situations where units are assigned one place for command purposes and another for rations and quarters. Figure 10 illustrates a tree structure.

TREE

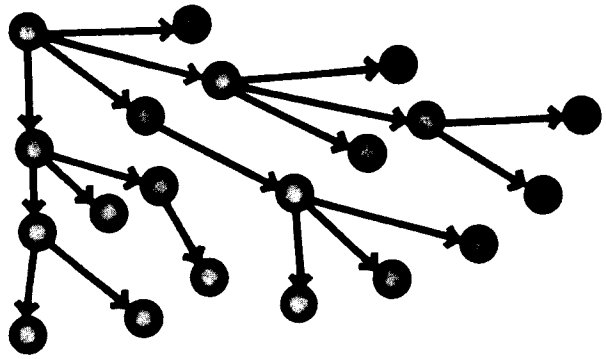


Figure 10

The Data Structure Diagram in Figure 9 is equally good for modeling a network or a tree. The Data Structure Diagram in Figure 11 is more specialized in that it supports a tree model but does not support a network.

TREE STRUCTURE

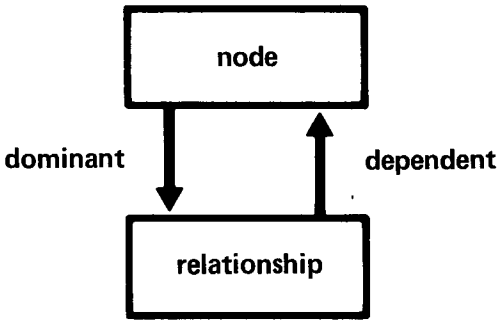


Figure 11

In the Tree Structure Diagram the owner/member relationship (arrow) of the "dependent" set class has been reversed. The freedom to make the reversal is based upon the fact that the tree allows only one or no relationships on the dependent side of the node. Modelling the tree with a Data Structure Diagram permits two options: (1) one relationship entity owning a one-member set of node entities (Figure 11), or (2) one node entity owning a one-member set of relationship entities (Figure 9). Therefore, the direction of the entity/sub-entity relationship is somewhat arbitrary. From the Data Structure Diagram in Figure 11, it is a short step in structure simplification to reach the diagram in Figure 12, which still represents the tree.

TREE STRUCTURE DIAGRAM

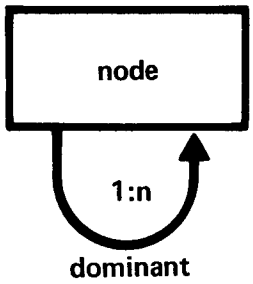


Figure 12

The "dependent" set had been limited by definition of a tree to a 1:1 relationship between the "node" and "relationship" entities. This was the fact which permitted reversing the dependent entity/sub-entity association in Figure 11 and still further supported merging the "node" entity class with the "relationship" entity class. The diagram simply illustrates that each "node" has a "dominant" relationship with other nodes that, in turn, are limited to one relationship on what is considered their "dependent" side.

The Data Structure Diagrams in Figures 11 and 12 create a chicken and egg situation. A member entity cannot exist until there is a set in which to insert it. In Figures 11 and 12 the "node" entity is both owner and member. Therefore, one such entity cannot exist alone unless it is its own owner. Two different structure solutions are available for this dilemma. These are the sometime member entity classes, described in the next section, and the alternate owner set classes, which will be discussed later in the text.

Sometime Member Entity Classes

When it is necessary to document a set class in which the member relationship may or may not exist, a dashed arrow is used. Figure 13 illustrates the graphic convention. Entities of the owner entity class always own a set of the set class specified. Entities of the class at the head of the arrow in Figure 13 individually may (or may not) be members.

SOMETIME MEMBERSHIP

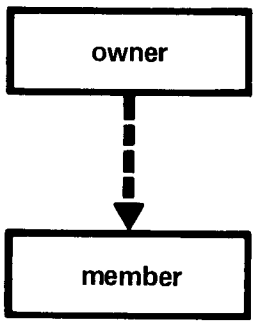


Figure 13

An example of a sometime member can be drawn from the banking industry, where demand deposit accounts entities have a sometime relationship with an overdrawn account entity. Whenever an account's balance is less than zero, the relationship is invoked until paid up. Otherwise, the relationship doesn't exist.

Compound Networks

We have just used some examples of simple networks to illustrate the usage of Data Structure Diagrams. By simple network, I mean networks of like entities, i.e., all events, or all "T" accounts. Compound networks are the result of the association of unlike entities within a network. One example of a compound network is developed by the author association between books and the people who wrote them. If you were to examine the books in any library, you would find that some books had one author while other books, especially in the technical and educational fields, had two and maybe three or four authors. If you considered the people who were authors of these books, you would find that some were authors of more than one book. Certain prolific authors would have many books to their credit. Therefore, the network created by the book/people/author relationship would consist of nodes for books (book entities), nodes for people (people entities), and a relationship between a book entity and a person entity that records authorship. Figure 14 is the Data Structure Diagram which represents this compound network.

COMPOUND NETWORK STRUCTURE

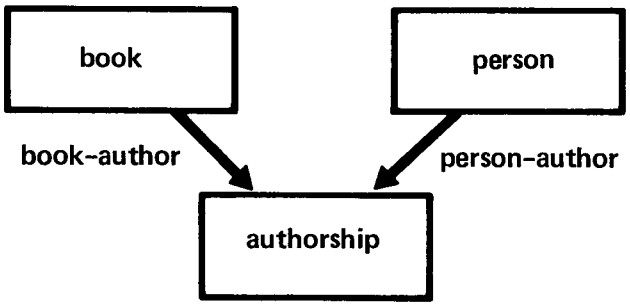


Figure 14

Information models of any complexity usually are compound networks in one or more ways. If our original department/employee model had represented a university, then the "employee" entity might have been the same entity class name as the "person" entity with a new association. Figure 15 combines the Data Structure Diagrams of Figure 14 and one similar to Figure 7 to create a more comprehensive information model.

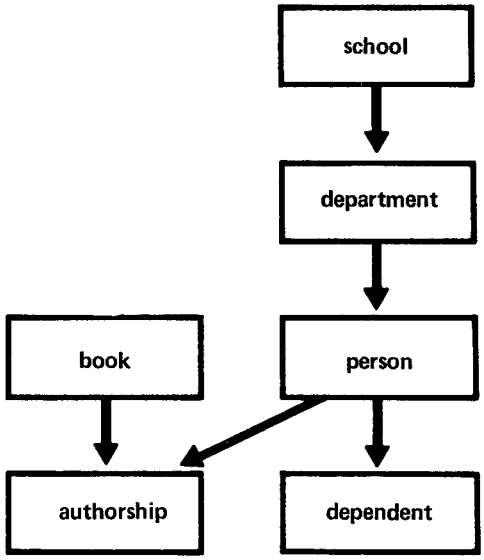


Figure 15

This model includes a school/department/person organization-oriented hierarchy, the person/dependent personnel-oriented hierarchy and the person/authorship/book compound (publish-or-perish) network. Given the necessary computer hardware and software, and access to any one entity in a specific data base created according to this Data Structure Diagram, then all other associated entities could be determined by moving through the sets. Both General Electric's Integrated Data Store (I-D-S) and General Motors' Associative Programming Language (APL) are software systems specifically designed and implemented to encourage and support the construction, maintenance, and use of data bases organized around such complex structures.

Multimember Set Classes

When it is necessary to document a set class with more than one class of entities in the role of member, a multi-headed arrow is used. Figure 16 illustrates the graphic convention.

MULTIMEMBER SET CLASSES

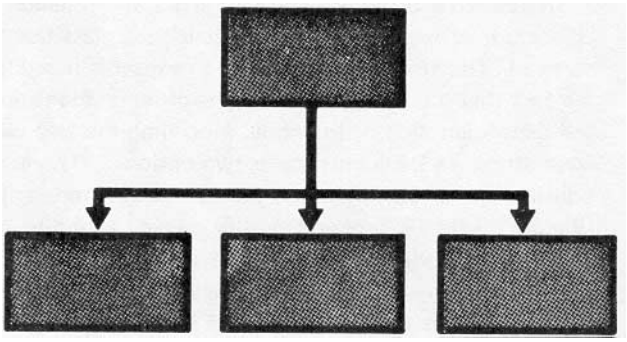


Figure 16

An example of multimember set classes can be drawn from the banking industry, where each customer may have several different types of business with the bank. Different entity classes are established for demand deposit accounts, savings accounts, loan accounts, and trust accounts. Figure 17 illustrates this data structure. With this structure, each customer can have multiple numbers of accounts of the same or different types.

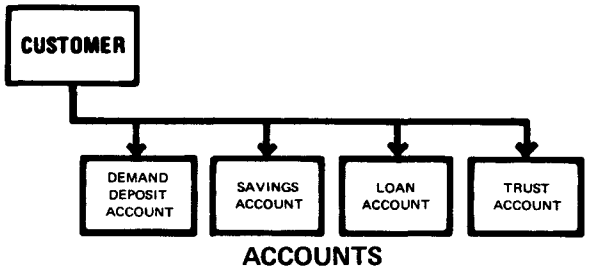


Figure 17

The need to provide for multicustomer associations with specific accounts, i.e., joint accounts, usually leads banks to work with a structure that includes joint account entities. Figure 18 shows this extension. With this data structure, each customer can have multiple numbers of accounts, and each account has one prime customer. In addition, each account may have any number of additional joint account customers. Thus, again a compound network is described.

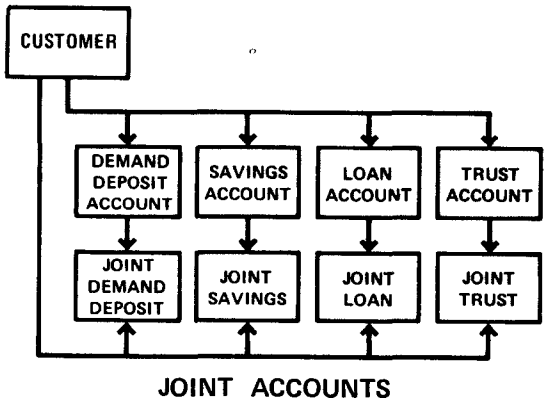


Figure 18

Alternate Owner Set Classes

When it is necessary to document a set class with potentially more than one class of entities in the role of owner, a multitail arrow is used. Figure 19 illustrates the graphic convention.

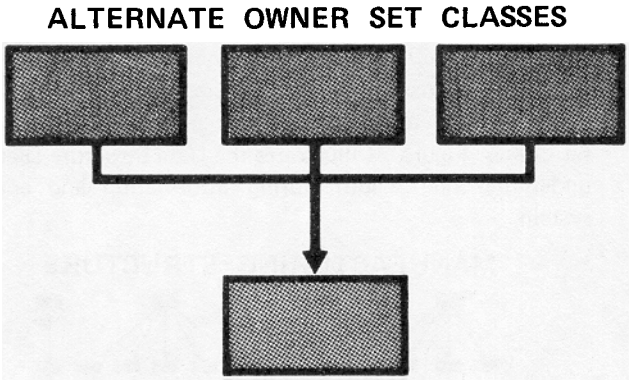


Figure 19

A particular set has only one owner. Each owner entity has its own set, but the set class name and the member entity classes of its set are the same regardless of which entity is the owner.

An example of alternate owner set classes can be drawn from the manufacturing industry, where a manufacturing order may be placed on the shop by an internal organization, a distributing organization, or a customer. For accounting and reporting purposes, different entity classes are established for each type of organization because different information must be maintained or because they may be involved in other and different set classes. However, from the shop's point of view, they are the "customer" for an order regardless of their other nature.

Another example relates back to the Tree Structure Diagram in which initiation of the tree posed a problem. Figure 20 is an alternate solution. If the "dominant" set class had alternate owners (either a primary node entity that is owner but not member or a secondary node entry that is both owner and member), then the problem is trunk of the tree. The corollary of this structure, if imposed on a data management system, is as follows, "If the primary node were removed, then all of the secondary nodes and thus the entire tree goes with it."

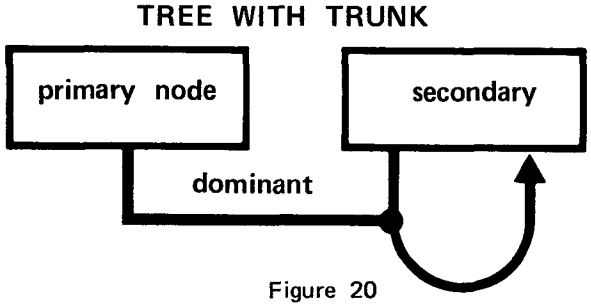


Figure 20

Complex Structures

Very large Data Structures Diagrams have been designed and used in the last five years in the design and implementation of various information systems. "Large" in this case is measured in terms of the number of entity classes and set classes. Figure 21 illustrates the Data Structure Diagram underlying one manufacturing information and control system.

MANUFACTURING STRUCTURE

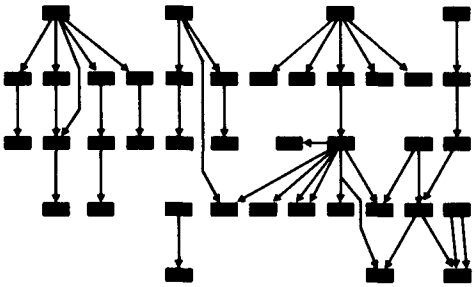


Figure 21

It has 39 entity classes and 38 set classes. Can you, in examining the diagram, find a five-level hierarchy? Two simple networks? Five compound networks? A simple network with an extra hierarchical level in one leg? Note that there are no tree structures. Without any rigorous definition, a complex structure is one composed of many entity classes and many set classes. Within a complex structure, we typically find multilevel hierarchies, simple networks, compound networks and trees.

Large Data Structure Diagrams, in terms of the number of elements, should be clearly distinguished from large data bases with many entities (records), which have been built in response to a Data Structure Diagram. Although each entity in a data base needs an entity class to define and control it, that one entity class may represent zero, one, ten thousand, or a million records in storage. The largest system in operation today contains 60 entity classes controlling over half a million data records. Larger systems are being installed.

Summary of Structure Types

Simple and compound networks are differentiated by the fact that one, the simple network, is constructed of node entities of a single entity class, while the other is constructed of node entities of several different entity classes. Trees and networks are differentiated by the fact that one, the tree, is constructed under the rule that each node has only one "dominant" node while the other is constructed with unlimited association between nodes. The two concepts: tree vs. networks, and simple vs. compound, are independent of each other and can be thought of as being at right angles or orthogonal. Figure 22 illustrates this point and brings the hierarchial structure into perspective. It is a compound tree, i.e. with more than one entity class as node and each node with only one dependent association.

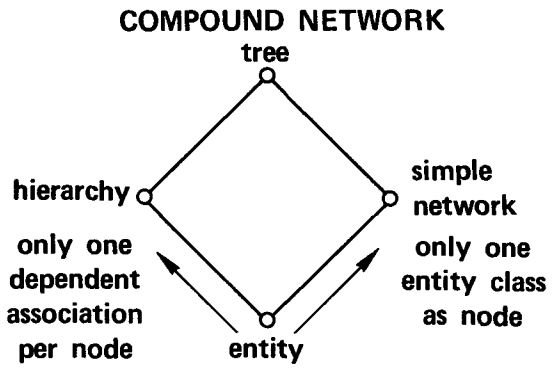


Figure 22

Summary

The Data Structure Diagrams, consisting of two kinds of elements (blocks representing entity classes and arrows representing set classes), have been defined. Examples have illustrated their application. Their practical usage in the design and study of mechanized information systems has been described. Two data management languages, I-D-S and APL, provide for the description and manipulation of data bases with the characteristics that are describable through the Data Structure Diagrams. In addition, there is a new Data Description Language and Data Manipulation Language currently being specified by the Data Base Task Group of the COBOL Programming Languages Committee. This offers promise for an industry standard data management language that would operate in conjunction with FORTRAN, ALGOL, PL/I, as well as COBOL allowing a DATA BASE to be built in one language while being accessed in yet another.

Bibliography

(1) Bachman, C. W., Williams, S. B., "A General Purpose Programming System For Random Access Memories," *Proceedings of the Fall Joint Computer Conference,* San Francisco, California. October 1964.

(2) Bachman, C. W., "Software For Random Access Processing" *Datamation,* April 1965.

(3) "Integrated Data Store, A New Concept in Data Management," AS-CPB-483A General Electric Information Systems Group, Phoenix, Arizona.

(4) Bachman, C. W., "Integrated Data Store Data Base Study" *Second Symposium on Computer-Centered Data Base Systems,* also available as CPB-481A General Electric Information System Group, Phoenix, Arizona.

(5) Dodd, G. G., "APL—A Language for Associative Data Handling in PL/I," Fall Joint Computer Conference, 1966.

(6) "Report to the CODASYL COBOL Committee, January 1968. COBOL Extensions to Handle Data Base" prepared by Data Base Task Group.

(7) "Data Description Language and Data Manipulation Language Report," April 1969. Prepared as a report to the CODASYL COBOL Programming Language Committee by the Data Base Task Group.