Why restrict the modelling capability of codasyl data structure sets?

by CHARLES W. BACHMAN
Honeywell Information Systems
Billerica, Massachusetts

ABSTRACT

Several issues have been raised concerning changes to the capabilities of the CODASYL Data Description Language specifications for data structure sets. The paper argues against new restrictions suggested and for removal of existing restrictions. The issues are:

- allow recursive set declaration
- keep multiple member declaration
- allow alternate owner declarations

The concept of "record-roles" is introduced to justify the need for these capabilities. The expanded capabilities described offer an alternate means of achieving the same end result without the need to introduce the "record-role" into the CODASYL Data Description Language.

INTRODUCTION

The concept of data structure sets has been well established through the publicity and use of I-D-S, the Honeywell Integrated Data Store system.¹⁻³ In recent years this concept has been adopted by the various CODASYL committees and imbedded in the CODASYL Data Description Language¹ and the COBOL Data Manipulation Language.⁵ A number of hardware and software suppliers have implemented the data structure sets of these languages (DDL/DML) as part of their systems. They include:

- IDMS (Cullinane for IBM 360/370)
- I-D-S (Honeywell GE200, GE400, GE600, H6000)
- I-D-S II (Honeywell for H66, H64)
- EDMS (Xero Sigma 6/78)
- DMS 1100 (Univac for Univac 1100 Series)
- PHOLAS (Philips for Unidata 7000, P1000)
- PHOLAS (Siemens for S4004)

Other implementations have been reported for CDC and DEC.

The capabilities of the data structure set, as developed in I-D-S and now defined in both the CODASYL DDL and the COBOL DML, provide for set-type declarations which:

(1) restrict the record type declared as owner to be different from any of the record types declared as member.
(2) permit declaration of one or more record types to serve as member records of an occurrence of a set type, and
(3) restrict to one the number of record types which can be declared to serve as owner records of occurrences of a set type.

THE PROPOSALS

This paper is a refinement of a working paper written in response to an assignment accepted at the IFIP-TC2 meeting on Data Description Languages held in Namur, Belgium in January 1975. There were three closely related proposals for changes to the CODASYL DDL discussed at that meeting. These proposals relate directly to three points enumerated in the prior paragraph. Assignments were given to defend a number of such proposals. The proposals interrelated and my working paper treats them as a single concept.

The first proposal, which was unanimously supported at the meeting, was to remove the restriction that the record-type declared as owner could not also be one of those declared as member. I strongly concurred with this proposal as a removal of an unnecessary restriction.

The second proposal was to add a restriction that only one record-type could be declared as member of a particular set-type. This proposal received mixed support. I strongly disagreed with this proposal, for essentially the same reasons that I support the first and third proposals. It adds an unnecessary restriction.

The third proposal was to remove the restriction that only one record-type can serve as owner of a particular set-type. This proposal received scant attention. The meeting did not express an opinion on the subject. I strongly recommended it, as I have done to the DBTG at least seven years ago. It is the removal of a restriction.
ARGUMENTS

There are several arguments for permitting a set-type to permit the record-type declared as member to be the same as the record-type declared as owner. There is a specific argument which will be treated first and then a general argument which relates to all three proposals mentioned earlier.

The specific argument treats the need for tree-like data structures, catalogues, organization structures and parsing trees. For these structures, it is necessary to support recursive sets, which provides the capability to build trees, with branches which have branches, which have branches, etc. An example of this is illustrated in Figure 1.

If all the straight lines in Figure 1 are considered to be "branches," then this structure can be built with a single record type and a single set-type. However, the membership of the "branch" record-type in the "fork" set-type must not be mandatory. A record declared to be as the lowest level branch is never a member of a "fork" set-occurrence. This special branch is characterized as the trunk. Figure 2 is a data structure diagram which illustrates the branch/fork structure.

In this data structure diagram, the "fork" set-type is illustrated with a broken line, meaning that the "branch" record-occurrences are sometimes members of it. That is, they are members if they are not the "trunk" branch of the tree. Note that all branch record-occurrences, whether or not they are the trunk, own "fork" set-occurrence with zero, one, or more subordinate "branches" records.

The third Namur proposal, which supports alternative owners in data structure sets, would permit the trunk branch to be treated as a distinctly different type of entity. Figure 3 is a redrawing of Figure 1.

It illustrates a tree with one trunk and many forks and branches. When this is drawn as a data structure diagram, the illustration of Figure 4 is developed. This structure has some advantages.

The "branch" record-occurrences can now be treated as mandatory members of the "fork" set-type, i.e., no branches are floating in the air. This can be very important from a naming point of view, as each branch needs a field for its "branch name" while the trunk does not need such a name. Branches are frequently named with an articulated grammar. All the branch names at a single fork of the tree must have unique local names. The higher level branches (i.e., the branches which are farther from the trunk), are uniquely named by concatenating their unique local names to the tree unique name of the branch immediately below. This is expressed below in BNF (Backus/Nauer/Form):

\[
\text{branch-name} ::= (\text{character-string}) | \\
(\text{branch-name})(\text{articulation-character}) \\
(\text{character-string}) \\
(\text{character-string}) ::= (\text{character}) | (\text{character-string}) \\
(\text{character}) ::= \text{a} | \text{b} | \text{c} | \ldots | \text{z} | 0 | 1 | \ldots | 9 \\
(\text{articulation-character}) ::= \text{any symbol which is not a blank or a character}
\]

In our information systems today, there are many examples of tree structures. In catalog (file) systems, we find...
directories of directories of directories of... of named files. In corporate organizations we find companies which have departments, which have departments, etc. Figure 5 is a data structure diagram which illustrates this.

This example illustrates a specific need for this type of structure and supports the proposal. Some will argue that this is an incorrect approach to the fundamental corporate organization structure and that they are really hierarchies of different types of organizational-units. At one time, the General Electric Company had a well defined hierarchical organization structure which is illustrated in Figure 6.

If you were a "section" manager, you knew exactly where you were in the management hierarchy. This is an easier structure to handle manually than by computer, as people did not get quite as upset when someone thought it appropriate for a particular "unit" manager to report directly to a "section" or "department" manager. If this organization structure were declared to an I-D-S database system with the structure illustrated in Figure 6, then no "unit" could directly report to a "section" or "department", it would have to be assigned to a "subsection." The proposal to support alternative owner record-types for a single set-type should be accepted because it is useful. It does not require that the database administrator use either the structure of Figure 5 (alternative owners) or Figure 6 (unique owners). It should be supported because it allows each administrator the choice.

The more general argument, for the support of alternative owner record-types for a set-type, also supports the need to retain the capability for multiple member record-types for a set-type. In an information system, real world entities are represented by records. These entities are classified by entity-type in order to facilitate the processing of data concerning them. Further, each entity-type may portray several concurrent roles or behavior patterns and sometimes these roles are shared by other distinctively different entity-types. For example, a person, or a company, or a governmental unit may serve in the role of "employer" of people, and as an "owner" of property. Within such a designated role, the record-types representing the entity-types should be capable of being the owner or member of a set-type which is role related, and be the holder of a field which is role related. Figure 7 is a data structure diagram which illustrates the employer role played by persons, companies and government units.

To model this structure, it is necessary to declare the "person" record-type, "company" record-type and "gov-
governmental unit” record-type such that all are able to assume the role of owner of the “employer → employee” set. It is necessary to declare the “person” record-type as a sometime set member, “sometime,” since all persons are not necessarily employers.

In the case of a person who is self-employed, the same “person” record would be the owner and member of the same occurrence of the “employer → employer” set. Both the alternative owner (prop. 3) and recursive set (prep. 1) proposals would need to be accepted to support this structure.

The reader should glance back to Figure 6, one of the possible means of implementing the organizational unit aspects of a corporate structure. Given this structure, now imagine how the organization-to-employee relationship would have to be handled. Each organizational record, from the “corporation” record through to the “unit” record, must be able to handle the role of employer. All seven of the organizational records need to be declared as alternative owner types to the “organization → employee” set. Figure 7 illustrates this extension to Figure 6.

If one assumes that each of the units needs to have a manager, who is a person, then each of these seven organizational unit role record-types must also be declared as a member in the “managed” set. The support of the “managed” set gives an example of the usefulness of the ability to declare multiple record-types as members of the same set-type, (Proposal 2). Figure 8 illustrates the further extension of Figure 6 to include the “manager → organization” set.

RECORD-ROLE CONCEPT

For the theoreticians (and it is they who have largely argued for reducing the number of member record-types declarable for a set-type to one, and keeping the owner types declarable to one) the introduction of the “record-role” concept may be of great importance. This is because there will be no argument from the practitioners over having only one role declared as the owner of a set-type and only one role declared as the member of set-type if roles become declarable entities. Furthermore, the owner and the member declarations could be restricted to be different “roles.”

In the work at Honeywell Information Systems on this subject, the word “record-role” has been used to characterize the role concept introduced above. The following definitions apply:

- A “record-occurrence” is the database representation of a real world entity.
- A “record-role” is a declaration of the a collection of the properties (fields and sets) which a record-occurrence may represent on behalf of one role of a real world entity.
- A “record-type” is a declaration of a collection of one or more record-roles which a record occurrence may represent, while roles are all played concurrently by a real world entity.
- A “record-class” is the collection of all record-occurrences of a particular record-type. A record-occurrence is always in only one record-class, defined by a record-type.
- A “role-class” is the collection of all record-occurrences of a particular record-role. A record-occurrence

![Figure 7](data structure diagram)

![Figure 8](data structure diagram)
Multiple Member "Department" Set
(data structure diagram)

- A "role-occurrence" is a subdivision of a record-occurrence which is the representation of that record-occurrence playing one role.

This distinction between record-role and record-type has not been made in existing database systems. While a record-type may have represented several record-roles, there was no mechanism of sharing the record-role declarations between two or more record-types. Thus the same fields and sets, relative to the role, had to be multiply declared, once for each record-type which played the role. This led to the requirement for multiple member declarations and alternate owner declarations for sets.

With the record-role concept, the declaration of fields, groups and sets are all associated with the record-role declaration. Field may be accessed using field-names which are qualified by record-role-name rather than record-type names. Sets are ordered by role declared fields. Set owner selection is based upon role declared fields. Record occurrences of different record-types coexist within the same set type as owners or members when the record-types share the same record-role with the declared function.

In the paper "The Evolution of Data Structures," there was a sequence of data structure diagrams which were used to illustrate the progressive introduction of new meta objects and new inter-object relationships into data structuring capabilities. The first two of the following three data structure diagrams are reprinted from that article. Figure 10 illustrates the meta entities: record, field, group, owner, member and data-structure-set, where there may be an unlimited number of member record-types and owner record-types declarable for a set-type. The diagram of Figure 10 is the meta object structure which I recommend to DBTS and have recommended and used for a number of years.

Figure 11 is a simplification of Figure 10 where the restrictions of a one owner entity-role and a one member entity-role have been placed on the set-type. The set-owner and set-member meta entities have been merged with the set-type meta entities, as they exist on a 1:1:1 basis. This yields the restricted structural capability which was recommended by some of the attendees at the Namur Conference.

The data structure diagram of Figure 12 introduces the meta entity "record-role". In this structure, the "record-role" meta entity has displaced the "record-type" meta entity in its direct relationship to the set, group and field. The record-type concept is now at the side, associated by a declared relationship with one or more "record-roles".

"Record-role" declaration may be associated with one or more record-types. From the viewpoint of the set-type, there is only one "owner" record-role and only one "member" record-role. This fits more easily into the viewpoint of both the relational model and the Data Independent Access...
Model. However, the requirements of the real world which we wish to model can be satisfied as each real world entity can be recognized acting in one or more roles and its record-occurrences are combined with the declared role occurrences.

At this time I have no interest in trying to introduce the 'record-role' concept into existing data description languages and data manipulation languages. Rather, I wish to provide the rationale, within these languages, for:

1. recursive set-types,
2. multiple member set-types, and
3. multiple owner set-types

which provide an alternate means for achieving the objectives achieved by the record-role concept while remaining within the limitations of the presently available meta entities. However, the eventual introduction of the 'record-role' may be the unifying factor that we seek.

The data structure diagramming technique has been extended to support the concept of record-role. Figure 13 is a redrawing of Figure 7 with focus on the record-roles of employer and employee.

They are illustrated by the two hexagons so designated. The record-types with which the roles are associated are designated by the background boxes. The employer role is played by the company, person and governmental-unit record-types. If classical data structure diagrams were thought to represent record-types and the relationships between them, then the new diagrams illustrate record-roles, their associations with record-types and their relationships with other record-roles. Record-roles are illustrated by hexagons and the background boxes name the record-types which play the role. Each record-type is considered to have one or more roles. In this example "person," "company" and "government-unit" are the record-types. "Employer" and "employee" are record-roles. A complete data structure diagram would show each record-type once as a box on top of a stack of hexagons. Each hexagon representing a record-role played by the record-type. It would also show each record-role, once as a hexagon at the top of a stack of boxes. Each record-type would appear once more as a background box behind each record-role it plays. Figure 13 is thus an incomplete data structure diagram as it shows the record-types playing each record role but does not graphically illustrate each record-

From the collection of the Computer History Museum (www.computerhistory.org)
Person, Company and Governmental-Unit Record-Type

Figure 14

type with its record-roles. Figure 14 illustrates each record-type with the record-role that it plays. Thus most old data structure diagrams can be considered as being examples where the record had only one record-role. Thus no role factoring is necessary. If alternative owners or multiple members exist in those diagrams, then the record-roles for them has not yet been factored. The importance of the record-role concept to data structure diagrams may not be immediately obvious at the first comparison of Figures 7 and 13. However, consider the following analogy. If identical programming code appears in several parts of a computer program, it is common to factor this code out as subroutines or at least as macro procedures so that the documentation is more easily understood. The record-role concept is the data structure diagram equivalent of a subroutine call. The diagram illustrates the shared aspects of the record-role and also all the places it has been invoked. For data structure diagrams, representing complex organizations with many record types, record roles and their relationships, the record-role has proven to be extremely useful in simplifying the diagrams. They are much more readable. Of necessity, these diagrams are only effective after the new concept has been understood, used awhile and accepted.

SUMMARY

The data structure set is almost the only structural tool currently available to the database administrator to represent the relationships between entities in his enterprise. At this time when all of its usages are unknown, it seems desirable not to place any restrictions upon its application. Proposal 1, to permit recursive sets (where owner and member are of the same record class), is a proposal to remove a restriction. Proposal 2, to prohibit multiple member record-types, is a proposal to add a restriction. Proposal 3, to permit alternative owner declarations, is a proposal to remove a restriction. These facilities, within today's record-network model would provide a workable implementation of the role concept, an expression of the evident and important multiple behavior patterns which are characteristic of real world entities.

REFERENCES
