A canonical schema for a generalised data model with local interfaces

S. M. Deen

Department of Computing Science, University of Aberdeen, Aberdeen, Scotland

Based on the concept of a canonical data model, a global schema is proposed for a generalised data base system capable of supporting interfaces to other systems—notably Relational, Codasyl and Adabas—through appropriate local schemas and data manipulation languages. The proposed global schema consists of entries in the form of normalised relations describing data and data relationships. The investigation of the Relational, Codasyl and Adabas interfaces demonstrates the capacity of the model to support all of them effectively. However, the model is found to impose on the local models certain constraints which help remove inconsistencies and enhance user facilities. The findings of this study are being implemented in a prototype called PRECI.

(Received February 1979)

1. Introduction

There appears to be a general concensus of opinion that future data base systems (DBSs) should provide both the Codasyl and Relational facilities under the same umbrella. It is also believed by many that a canonical data model that can interface not only with the Codasyl and Relational models, but also with others—notably Adabas, IMS, Total—would be a very useful facility, particulary in a distributed environment where all users could participate through their local interfaces. These considerations prompted us to investigate the feasibility of such a canonical data model with a global schema in a canonical form to be described by normalised relations.

We chose this relational framework chiefly because of its elegance, conciseness and convenience in data analysis and data description (Maddison, 1978; DDSWP, 1978). It is assumed that a canonical data model, such as the one we are considering here, will be based on a three level architecture of a global schema, a storage schema and local schemas or sub-schemas (supporting the local interfaces) as shown in **Fig. 1.** In this report we shall describe the content and the structure of the global schema—also referred to as the canonical schema—and its inherent capability to interface with the Relational, Codasyl, and Adabas models. The plan of the paper is as follows: in Section 2, we define a canonical data model, followed in Section 3 by the description of our proposed global schema. The three local interfaces are examined in Section 4, with a conclusion in Section 5.

2. Canonical data model

We define a canonical data model as 'a model of data which represents the inherent structure and the usage constraints of the data in a standardised form, independent of its local models'.

The local models are the interfaced models as seen by the users through the local schemas. Any existing data model should in principle be able to act as a local model. The global schema of such a canonical model should be composed of the following entries:

(*a*) entity record descriptions

(*b*) entity relationship descriptions

(*c*) access constraints

(*d*) integrity constraints.

The first two entries describe the data and their inherent characteristics and the last two the usage constraints that need to be imposed for the protection of data from unauthorised usage and from damage and corruption as discussed below. 'Access constraints' implies the constraints or controls that must be exercised to prevent unauthorised users from accessing the data base. These constraints are usually imposed by establishing locks or passwords on selected items or groups of items (Deen, 1977a), and can be specified conditionally if desired through predicates; their specification in a relational global schema is fairly straightforward (Michaels *et al*, 1976; Date, 1977). There are however other forms of access controls particularly those relating to data ownerships and transfer of authorisation, which seem to generate some open-ended problems without any comprehensive theoretical foundation (Griffith and Wade, 1976; Chamberlin *et al*, 1975; Fagin, 1978).

A data base must always be maintained in an error free and a self-consistent state (Deen, 1977a; Hawley *et al*, 1975; BCS DBAWG, 1975): It is preferable to apply all the necessary controls at the global level so that the system can automatically enforce them, rather than leaving them to the local level relying

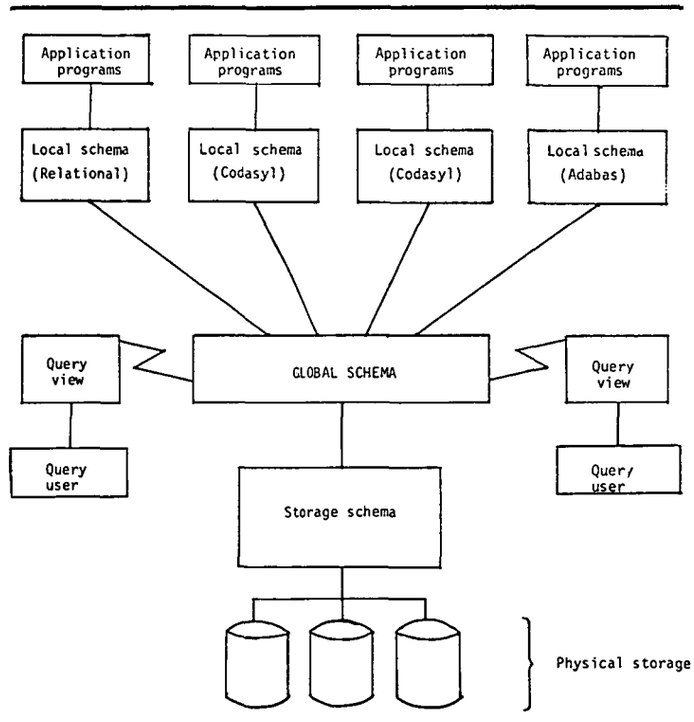


Fig. 1 Architecture of PRECI. The same global schema may support more than one storage schema each with its own data base. There can be any number of local schemas for every local model, each local schema catering for an arbitrary number of application programs

on the frailties of individual programmers. Some of these constraints such as the validation check on individual items for the permissible characters and range of values can be specified and enforced without much problem as has been done in the Codasyl model, but others are not so easy to implement. Although it is relatively trivial to state the validation and consistency requirements for a given record occurrence, any attempt to generalise them into an abstract form suitable for specification in the global schema is likely to be difficult. There is also a question of cost effectiveness, since it is often substantially cheaper if the enforcement of integrity controls is left to the application programmers.

We use the term access path to imply facilities to access entity records in specified orders, or by specified keys. Access paths determine access efficiency, and therefore the data base administrator (DBA) should be free to alter them periodically to improve the overall performance of the data base (Deen, 1977a; BCS DBAWG, 1975), and as such they should be invisible at the local level, that is the change of access paths should not require changes in the local schema and application programs. This means that the global schema is not the rightful place for access path specification.

3. Global schema

In this report we shall consider only a basic global schema consisting of entity records and their relationships, but not usage constraint which will be dealt with in a separate document later.

3.1 *Representation of entity records*

Entity records in our global schema are represented as relations in 4NF (Sharman, 1975; Fagin, 1977; Zaniolo, 1976), which may be defined as

> 'A normalised relation is in fourth normal form, if every multideterminant is a candidate key.'

If domain A multidetermines domain B, that is if B is multivalued dependent on A, then A is multideterminant. Both A and B can be composite. Since the presence of multivalued dependency and its subset (full) functional dependency cannot be automatically deduced from the data by the system (Chamberlin, 1976), it will be up to the DBA to ensure, by using aids if available, the correct normalisation of the data. In the remainder of this paper, we shall assume, unless otherwise qualified, that all relations are correctly normalised, and hence the terms (entity) record and tuple, and (entity) record type and relation type are synonymous.

The occurrences of each entity record type in our model are identified by the values of an Entity Identifier Domain (EID) which is required to be visible at the local level so that the application programmers can use its values when necessary to identify entity records, and therefore EIDs must be specified in the global schema. If an EID is composite, we shall assume the order of the appearance of its components in the global schema as the order of their concatenation (see **Fig. 4**).

Although it is increasingly recognised that the normalised relation is the most convenient tool for representing data (Maddison, 1978; DDSWP, 1978), there is a small number of cases where this might be debatable. Consider, for instance, the problem of representing data on bus routes, each route being defined by a route number and a set of bus stop names. Assuming that each route has exactly the same n number of stops, then we can represent this data by a relation of $n + 1$ meaningfully named domains, including one domain for the route number. However, if the number of stops vary from route to route, then we must break this repeating group by allocating to each bus stop a unique stop number, from 1 to n, in the correct sequence for n number of stops. We can then represent the data by a relation containing route number, stop number and stop name. Additional domains containing other information on each bus stop, such as the frequency of arrivals, can also be included there.

In the above example, we had to create a bus stop number which did not exist. If the problem was represented by a record type with repeating group, then such creation would not have been necessary; however we believe that the introduction of an extra domain is a small price to pay for the convenience of the relational representation. Besides it can be argued that since sequencing is an inherent characteristic of that data, the information on sequencing should ideally be represented as part of the data, in the form of an extra domain containing the relative position. The representation technique used in the earlier example of a fixed number of bus stops should be considered as an expedient, rather than rigorous, procedure. That expedient works only if a fixed number of atomic domains are involved. (Note that this problem can also be solved without using bus stop numbers, if we represent a pair of adjacent bus stops, rather than a single stop. Apart from the redundancy introduced, it will not permit other information on individual bus stops to be maintained; but it can store information on the pair, for instance the distance between them.)

The problem of the type discussed above will arise whenever information on sequence is an inherent characteristic of data, as found in the data for routes through a set of points, or for shapes of irregular polygons derived from the vertex coordinates (Brown, 1978).

3.2 *Representation of entity relationship*

Relationships among entities are categorised as one to many ($1:N$), and many to many ($M:N$); these relationships are logical in the sense that the same entity can participate in many such relationships. Since an $M:N$ relationship can be resolved conveniently into two $1:N$ relationships, a generalised DBS should at least facilitate an effective representation of $1:N$ relationships. We shall consider below a representation technique for $1:N$ relationships, followed by that of $M:N$ relationships.

$1:N$ *relationship*

In a one to many relationship one record (owner) is logically linked to N records (members). In general these related records can belong to one or more record types, and in that respect they may be subdivided as follows:

(*a*) a single record type contributing to both the owner and the members

(*b*) two distinct record types, one for the owners, and the other for the members

(*c*) several record types for owners, and several other record types for members

(*d*) the same as (*c*), but owners and members being able to share some or all of the record types.

Relationships of the type (*a*) and (*b*) are adequately represented by what is sometimes referred to as the 'single member' set type of the Codasyl model; its characteristics are:

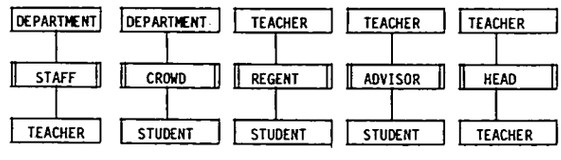


Fig. 2 Some codasyl set types

a set name
one owner record type
one member record type
(the same record type may be declared as both owner and member).

A set (occurrence) has one owner record, and zero, one or more member records.

As an example let us consider the relationship among three record types, DEPARTMENT, TEACHER and STUDENT with the following five set types, each representing a collection of $1:N$ relationships (**Fig. 2**).

In a university, each student is usually allocated a regent and an advisor; the regent acts as the academic guardian, but leaves it to the advisor to suggest suitable courses. It is assumed that a student belongs to only one department (the department of his/her Honours subject) which is not necessarily the department of his/her regent or advisor. The Codasyl set type also permits more than one record type as members (but only one record type as owner). We wish to extend the scope of these 'multi-member' set types to fulfil the requirements in (*c*) and (*d*) by introducing the concept of role sets as proposed by Bachman (1977). Consider for instance two set types, TRANSPORT and WORK-FORCE as shown in **Fig. 3.**

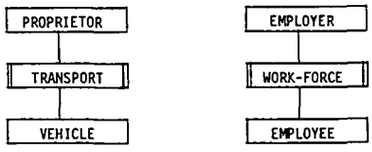


Fig. 3 Role sets

A proprietor can be a person, firm, or a government department, whereas a vehicle can be a car, truck, or a van, each from a different record type. Therefore PROPRIETOR and VEHICLE represent not record types, but roles played by a collection of record types. Similarly an employer can be a person, a firm or a government department, and an employee a person. There are a number of problems in such representations which we intend to deal with, along with some other associated issues, in a later document; in this paper we shall restrict ourselves to the relationships of types (*a*) and (*b*) only.

ID	*NAME*	*TYPE*	*SIZE*	*OWNER*
REL	DEPARTMENT			
EID	DNO	CHAR	5	
DOM	DNAME	CHAR	20	
REL	TEACHER			
EID	STAFF	SET		DEPARTMENT
EID	TNO	INTE	5	
DOM	TNAME	CHAR	30	
DOM	HEAD	SET		TEACHER
REL	STUDENT			
EID	SNO	INTE	4	
DOM	SNAME	CHAR	20	
DOM	CROWD	SET		DEPARTMENT
DOM	REGENT	SET		TEACHER
DOM	ADVISOR	SET		TEACHER
DOM	YEAR	INTE	4	

Fig. 4 A canonical schema. Please note that SIZE here denotes the logical, but not the stored, size of attribute values

An example of the proposed canonical schema with five set types, STAFF, HEAD, CROWD, REGENT, and ADVISOR is shown in Fig. 4. In our representation, set names are replaced by domain names which contain the entity identifiers of the owners. Assuming that TNOs are unique only within department, the EID of relation TEACHER is ⟨STAFF⟩⟨TNO⟩, that is ⟨DNO⟩⟨TNO⟩.

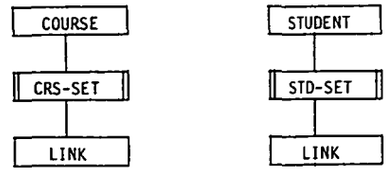
Since the teacher's entity identifiers include department's entity identifiers, the teachers—in the Codasyl term—are the Automatic Fixed members of set type STAFF, which Kay (1975) calls 'dependent' set type. In relation TEACHER, the identity of two separate domains STAFF and TNO making the EID is maintained, but this is not so in set domains REGENT and ADVISOR (of relation STUDENT), each of which contains the entity identifiers of teachers as a single attribute concatenated from DNO and TNO. This protects the students from automatically becoming members of his/her advisor's and regent's departments, in addition to his/her own (through set domain CROWD), and thus prevents confusion which will otherwise ensue. If students are required to be members of their regents' department, then a separate set domain must be provided for the purpose.

If in implementation every entity record in the data base is given an unique internal identifier independent of storage, to be referred to in this paper as surrogate (Hall *et al*, 1976), then in storage the owner identifiers can be replaced by the appropriate surrogates. Note also that the storage characteristics of the attribute values are meant to be specified in the storage schema, the global schema gives their types and logical sizes only.

Each $1:N$ relationship discussed above has an owner record; in general, however, there can be logical collections of records without having any owners. Consider for instance employee records which can logically be grouped as employees on overtime, employees on monthly salary, employees on commission etc. These groups do not have any owners and are not necessarily disjoint, and therefore suggest a need to extend the capability of our set types to include ownerless sets, one set per set type, which can be generated by storing special indicators in the set domain, say 1 for the members of the set and 0 (or space) otherwise.

$M:N$ relationships

The entries discussed above show our technique of specifying $1:N$ relationships. To represent $M:N$ relationships, we follow the link record approach (Deen, 1977a) which involves splitting a network into 2 one to many relationships. For instance if we have COURSE and STUDENT records, where each course is taken by a number of students and each student takes a number of courses, then we create a third record type, often referred to as LINK records, one link record for each pair of related entities, each record basically containing two entity identifiers, say course number (CNO) and student number (SNO). Additional information, such as a student's performance (EXAM) in a course, can also be stored there. Using these link records the network can be represented by two set types, say CRS-SET and STD-SET, one owned by COURSE and the other by STUDENT, with LINK records as members in both, as shown below:



In the global schema we add the following entries:

ID	*NAME*	*TYPE*	*SIZE*	*OWNER*
REL	COURSE			
EID	CNO	CHAR	6	
DOM	CNAME	CHAR	20	
REL	LINK			
EID	CRS-SET	SET		COURSE
EID	STD-SET	SET		STUDENT
DOM	EXAM	INTE	3	

The relationship between the student and the course is of interest to us and hence can be assumed as an entity (Senko *et al*, 1973), its entity identifier being made up of CNO and SNO. It is up to the user to specify the order of concatenation, but here we have arbitrarily assumed CNO as the major key. The advantage of using link record approach is that it not only simplifies network representation but also provides a proper construct for storing information on the relationship itself.

4. Interfaces

The global schema specified above is capable of supporting local interfaces to other existing models through appropriate local schemas—to be constructed in accordance with the specification of the local model, with minimal variation. The local user should be able to access the data base for both the retrieval and update purposes using the DML commands of the local model. However there will be some inevitable variations in the specification of those local schemas and DML commands; the possible sources of these variations are:

1. *Entity records*

Local records may be constructed from some or all of the domains of the global records, using surrogates to identify each global record, particularly if EID's are absent in the local records. If local records must contain repeating groups, they can only be provided by reconstruction, presumably only from the Automatic Fixed class of set types.

2. *Entity relationships*

Most data base systems provide some form of $1:N$ relationships, usually more restrictive than that for which we have allowed. Our model is therefore upward compatible, except in the case of Codasyl 'multi-member' set types and direct representation of $M:N$ relationships. It is however possible to support both indirectly.

3. *Access paths*

The removal of access paths to the storage schema will affect some of the current practices, and will require changes in the affected areas of the local model. These changes may in fact provide more flexibility in data manipulation.

4. *Usage constraints*

The need to impose access and integrity constraints on a piece of data depends on the usage of that data, independent of the model employed to process that data. If the model cannot support all the constraints that need to be imposed, then some of them will be left unattended, thus risking breach of access and loss of data base integrity. A canonical schema of a generalised data base system should therefore support as many controls as feasible, irrespective of their impact on the local models. Obviously the security of the data base must be regarded as paramount, and cannot be jeopardised by relaxing the constraints for a local model, just because it does not happen to support them; on the contrary, the local models must be forced to conform to all the constraints that can be specified in the canonical schema. This will imply some inevitable changes in the local schemas and DML commands. For instance in Adabas, one can delete the owner of a non-empty set, which we might not allow. If a record type is declared as Automatic Fixed member (see later) of a set type, no local user should be able to violate it either. However since we have not yet specified the access and integrity protection facilities in our canonical schema, it is too early to consider this problem in this paper.

We shall examine in this section three potential local models, namely Relational, Codasyl and Adabas as regards item (*a*), (*b*) and (*c*) listed above. Of these, only the Codasyl interface will be discussed in some depth.

4.1 *Relational interface*

Since the global schema consists of normalised n-ary relations, a relational subschema will be the easiest to interface. Each local relation may consist of only a subset of the domain in the global relation interpreting set domains as foreign keys and EIDs as primary keys. For instance one might specify relation STUDENT as

RELATION	STUDENT	
PKEY	SNO	INTE 6
DOM	REGENT	CHAR 10
DOM	CROWD	CHAR 5

Renaming of global datanames should be permitted in the local schema. The system however may not allow the deletion of an owner tuple, unless all its member tuples (that is where this tuple appears as the foreign key) are deleted first. Relational algebra or a calculus based sublanguage can be used as either a query language or a data manipulation language (with an underlying host language) or both.

4.2 *Codasyl interface*

In considering a Codasyl interface, we shall confine ourselves to only the major features of the Codasyl model (1975; 1978), namely:

1. Records and sets.
2. Set membership class.
3. Set selection criteria.
4. Record keys and Search keys.
5. Set order criteria and set order keys.
6. Subschema entries.

Records and sets

The Codasyl model permits records with repeating groups which, as indicated earlier, can be generated for our interface without heavy overheads from the Automatic Fixed class of set types. There should not be any problem in defining local records from only a subset of the domains in the global records with optional renaming and regrouping. Since our set types are identical with Codasyl 'single member' set types, there is no interfacing problem, except for the 'multi-member' set types which at present can be supported only indirectly.

Set membership class

In the Codasyl models, records can be made members of a set either automatically (the AUTOMATIC option) by the data base control system (DBCS), or by the application programmer (the MANUAL option) using the CONNECT commands. The automatic option should not be viewed as a mere labour saving device, because it contains an integrity declaration for the DBCS which must ensure that all the entries of the relevant set domain contain only valid owner identifiers, permitting null values solely in the case of the Manual option.

Given a member record, its removal from the membership is controlled by three Codasyl options: FIXED, MANDATORY and OPTIONAL. If Fixed, the record cannot change its set

(i.e. the owner within the set type) until the record is deleted; if Mandatory, it can be switched from one set to another within the same set type, but must remain a member of the declared set type. If Optional, member records can be removed without any restriction. Therefore they also represent integrity constraints. There is no doubt that membership class should be supported in canonical global schema as part of integrity constraints.

Set selection criteria

The set selection clause in the Codasyl model specifies criteria for 'selecting the correct set occurrence of a set type by the DBCS, for the following cases:

(*a*) to place an Automatic member in the correct set after its insertion into the data base by the STORE command

(*b*) to change the set membership of a record by the MODIFY command

(*c*) to select a record from a set type by the record selection format 7 of the FIND command.

These are access facilities which depend not only on the local model but also on the application, and hence should be specified only on the local schema.

Record keys and search keys

The difference between a record key and a search key is that while the former is used to access an occurrence of a record type irrespective of its set types, the latter operates only within a named set type; in other words, if a set domain is used as the major key in a record key, then the result is a search key; and yet in the Codasyl model, the record keys must be specified in the Codasyl schema and search keys in the application program, access to the data base being faster if the search key used by an application program (in record selection format 7 of the FIND command) happens to be supported in the storage schema. Since both are access facilities and functionally equivalent, we propose not to make any artificial distinction between them—both must be declared in the application programs irrespective of whether or not they are specified in the storage schema.

Set order criteria and set order keys

In the Codasyl model, one can access the first, last, or n^{th} member record, or the next or prior member record with respect to a given member record, of a set by using the record selection format 4 of the FIND command. This selection of the member record is made in accordance with the set order criteria specified in the Codasyl schema. The options provided for this criteria are: FIRST, LAST, NEXT, PRIOR, SYSTEM-DEFAULT, WITHIN RECORD-TYPE, and DEFINED KEYS. It seems that only the SYSTEM-DEFAULT and DEFINED KEY options are really important. Insertion of records into a set as FIRST or LAST in the set, or NEXT or PRIOR to another member of the set, loses significance due to subsequent updates as they are not sorted, and thus become equivalent to the SYSTEM-DEFAULT option which is unordered. The WITHIN RECORD-TYPE and DEFINED KEY options are specified for sorting, a separate KEY clause (referred to in this paper as set order key) being defined under each member subentry. In the WITHIN RECORD-TYPE option the members are sorted according to the KEY clause within each record type, although the record name may not necessarily be used as the major key; in the DEFINED KEYS option, on the other hand, record name can be specified as the major key, and if it is not specified then all member records are sorted irrespective of record types. Therefore the WITHIN RECORD-TYPE does not have much to offer and hence can be dropped. We are thus left with only two

```
MAPPING DIVISION.
ALIAS SECTION.
AD CROWD BECOMES POP.
SN CROWD BECOMES MOB.
AD STAFF BECOMES DEPT.
STRUCTURE DIVISION.
SET SECTION.
SD STAFF
    SET SELECTION THRU DATA-BASE-KEY EQUAL
    TO DEPT.
SD REGENT
    SET SELECTION THRU CURRENT OF SET.
SD MOB
    SET SELECTION THRU STRUCTURAL
    CONSTRAINTS POP EQUAL TO DNO.
RECORD SECTION.
01 STUDENT.
   02 SNO          PIC 9(4) COMP.
   02 POP          PIC X(5).
   02 SNAME        PIC X(20).
01 TEACHER.
   02 DEPT         PIC X(5).
   02 TNO          PIC 9(5).
01 DEPARTMENT.
   02 DNO          PIC X(5).
   02 DNAME        PIC X(20).
```

Fig. 5 A possible subschema (Title Division and Realm Section are not shown)

options:

(*a*) an unsorted option which can be provided as an efficient system default option

(*b*) a sorted option in accordance with a defined set order key, (which might optionally include record name as the major key if 'multi-member' set types are allowed).

The set order criterion is used to provide access paths and hence should be specified in the storage schema. Access should be faster if the key declared in the DML command ORDER coincides with the storage schema specification. The scope of the record selection format 4 could also be extended, in line with the facilities of format 7, by including in it an optional user defined key which may contain surrogate. As with search keys, access should be faster if this key is present in the storage schema; if format 4 is used without a key, then the DBCS should automatically invoke the storage schema specification.

Subschema entries

Most subschema entries will remain unaffected in our model except for the ALIAS SECTION and set selection criteria. To avoid confusion between a set name and the possible use of the same set domain as a data item in the record, we need to provide additional renaming facilities. In the attached sample subschema (**Fig. 5**) we have used SN as a new exclusive entry for renaming set names, but alternative approaches are also possible. Note that the system should be able to distinguish between a set name and a domain name without requiring such renaming. The set selection clauses used are drawn from the provisions of the Codasyl schema (but with somewhat reduced verbosity), except the data base key (that is surrogate) option which we have retained as a useful facility although no longer supported in the Codasyl schema.

4.3 *Adabas interface*

Like the Codasyl model, Adabas (Adabas, 1971; Olle, 1978) also permits records with repeating groups and multiple record keys (descriptors in the Adabas term) per record type. A

descriptor itself can have repeating groups thus becoming variable length. There does not appear to be any great advantage in supporting these repeating groups in our interface, particularly when the global records are normalised; but of course one can provide them, if need be, through a special mapping.

Like the relational model, $1:N$ relationships in Adabas are represented by including owner's key in the member records, except that the uniqueness of the owner is not guaranteed. In our Adabas interface it will be only sensible to restrict owner key to EID (or surrogates), thereby guaranteeing uniqueness of the owner. Adabas permits the direct representation of $M:N$ relationships, by including a repeating group of keys of the first record type into the second, the occurrences of the first record type then being accessible from the second, by a variable length descriptor. In view of the more flexible facilities of our model, we are dubious of the need to support such $M:N$ representations and variable length descriptors in the Adabas interface, which can be provided only through a special mapping.

The ISNs of Adabas are equivalent to our surrogates. The incremental restructuring permitted in the Adabas can also be facilitated, if our model is implemented properly.

5. Conclusion

We have developed a canonical global schema based on the concept of a canonical data model which can support other systems through local interfaces. Our global schema is incomplete in the sense that the access and integrity control facilities have yet to be incorporated; its facility for set representation can also be enhanced to include role sets. Within these limits however, we have shown that this global schema is effective and can potentially support local interfaces. We have not of course examined the details of these interfaces—this can only be done by implementing the ideas expressed here. We have indeed undertaken a project called PRECI (*P*rototype of a *Re*lational *C*anonical data model with local *I*nterfaces) (Deen, 1977b; Deen *et al*, 1979) to construct such a system with Relational and Codasyl interfaces and to study its effectiveness. PRECI is expected to be used principally as a test vehicle for research into the data base area (including distributed data bases).

In designing this canonical data model, one question which we faced repeatedly is 'to what extent should such a model be made to support some of the redundant and somewhat dubious facilities of the local models, such as repeating groups and $M:N$ relationships (of the Adabas model)'. When the global schema provides normalised nonduplicate tuples and better structure for $M:N$ relationships, is there really a good reason for simulating the old facilities? We feel strongly that these redundant facilities should not be supported when better alternatives exist, unless one needs to provide a continuity to the old user programs.

References and comments

ADABAS PUBLICATIONS (1971). Software ag, Darmstadt, W. Germany.

BACHMAN, C. W. (1977). Why restrict the modelling capability of Codasyl data structure sets?, Proc. National Computer Conference 1977, p. 69.

BCS DATA BASE ADMINISTRATION WORKING GROUP (1975). *BCS/Codasyl DDLC DBAWG Report*, June 1975, BCS.

BROWN, A. G. P. (1978). The irregular polygon problem first presented in a Codasyl DDLC working paper (ICL WP 7801) as an example of limitations of the relational approach; he favoured a repeating group solution.

CHAMBERLIN, D. D. (1976). *Computing Surveys*, Vol. 8 No. 1, p. 43.

CHAMBERLIN, D. D. *et al.* (1975). *Proc AFIPS 1975 NCC*, Vol. 44, p. 425.

CODASYL. *Cobol Journal of Development*, May 1975.

CODASYL. *DDLC Journal of Development 1978.*

DATE, C. J. (1977). *Introduction to Database Systems*, Addison-Wesley, second edition 1977, chapters 23 & 24.

DEEN, S. M. (1977a). *Fundamentals of Data Base Systems*, Macmillan, chapters 3 & 4.

DEEN, S. M. (1977b). PRECI, Report Number AUCS/7701.

DEEN, S. M., NIKODEM, D. and VASHISHTA, A. (1979). Design of a Canonical data model with local interfaces: (PRECI), Report Number AUCS/TR-7902 (to be published).

FAGIN, R. (1977). *ACM TODS*, Vol. 2 No. 3, p. 262.

FAGIN, R. (1978). *ACM TODS*, Vol. 3 No. 3, p. 310.

DDSWP (1978) Minutes of the BCS Data Dictionary Systems Working Party meeting, September 20-22, 1978. (Discussions at the 4th International Conference on very large data bases, held in Berlin, 1978 also seem to indicate a preference for normalised relations.)

GRIFFITH, P. P. and WADE, B. W. (1976). *ACM TODS*, Vol. 1 No. 3, p. 242.

HALL, P. *et al.* (1976). *Relations and Entities—Modelling in DBMS*, edited by Nijssen North-Holland.

HAWLEY, D. *et al.* (1975). *The Computer Journal*, Vol. 18 No. 3, p. 201.

KAY, M. H. (1975). Data Base Description, edited by B. Douque and G. Nijssen (North-Holland 1975), p. 199.

MADDISON, R. N. (ed.). (1978). Discussions in the BCS symposium on data analysis for information system design. University of Loughborough, June 1978. Our own experience coincides with this.

MICHAELS, A. S. *et al.* (1976). *ACM Surveys*, Vol. 8 No. 1, p. 125.

OLLE, T. W. (1978). *The Codasyl Approach to DBMS*, Wiley, chapter 23.

SENKO, M. E. *et al.* (1973). *IBM Systems Journal*, Vol. 12, p. 30.

SHARMAN, G. C. (1975). Technical Report TR12.136, IBM Laboratory, Hursley, England.

ZANIOLO, C. (1976). Analysis and design of a relational schemata for data base systems, Ph.D. Diss., Tech Dep UCLA-ENG-7669, University of California, Los Angeles, Calif, July 1976.