CO-EXISTING DATA BASE MANAGEMENT SYSTEMS

(CODASYL DBTG and System R Approaches)

Hamid W. Mirza

30 November 1979

OREGON STATE UNIVERSITY

Department of Computer Science

ABSTRACT

The paper addresses the problem of communication between co-existing DBMSs. Specifically, two data models--that is, the Network and Relationals models--have been analyzed using their respective implementations CODASYL DBTG and System R.

Feasibility of communication is discussed and mapping mechanisms are suggested between the two DBMSs at the following levels:

- Data Model
- Data Definition Language
- Data Manipulation Language
- Access method

Restrictions arising from the individual data models in the mapping process have been outlined, in particular the levels at which communication takes place between the two systems: from Network to Relational at the query language level and the converse at the access method level.

TABLE OF CONTENTS

I. INTRODUCTION

The Multi-model Environment

Rapid advances in the information-processing technology, both in terms of cost and speed, have made accessible to organizations vast quantities of data to be used as information. This development has its negative aspects, however, since the users are not insulated from the vagaries of a changing technology. Providing a logical data-view independent of much of its implementational details has been a solution to the enormous expenditure involved in modifications to existent software.

Data Base Management has come to be recognized as a comprehensive discipline for providing the medium by which information processing may develop with minimal cost and disruption inherent in such a process.

In the recent past data models or the logical views of data have been the subject of considerable research effort. Many data models have been proposed and implemented, each having its own concepts and terminology. Even though the underlying objective has been the same, each data model approach differs substantially from the others in:

their definition of the logical data base structure,

what constitutes a data-independent query language,

access paths provided,

encoding the information,

storing the information physically.

As long as different application needs exist and different commercially

available DBMSs are to be found, the proliferation of Data Base Systems shall continue. In fact, it has been suggested as a healthy trend in developing an optimal canonical/meta-model towards which these varying approaches may eventually converge.

In this paper the problem of cooperation between two heterogenous DBMSs is addressed--specifically two of the more accepted approaches, that is, the Relational and Network views of modelling data. Typically, such an exchange would take place with two local implementations communicating via a computer network or users of one of the data models interfacing to an implementation of the other model. In any case, such intercommunication requires an analysis of the ramifications of translation and mapping of the model and its associated data sublanguage.

In general, communication between DBMS's may be established at a number of levels:

Interactive query level.

Host language level.

Access method level.

Microscopic (data storage level).

Communication at one level does not exclude communication at another level, e.g., since queries are compiled into access requests, communicating at the access method level may be an indirect way of query-level communication.

The last choice, of course, would burden the user with a mass of details relating to the actual data storage and bypassing the system security controls as well as the facility of any optimized access paths. The

cost effectiveness in terms of both system resources and user time would dictate eventually the level of communication chosen without affecting the semantics of the DBMS model addressed.

In the modelling approach of two systems, such as the Relational and Network, the underlying philosophy and implementational aspects (e.g., on the network model and its implementation DBTG the DDL and DML are designed for a logical structure that closely corresponds directly to the storage structure and its related access paths) are diametrically opposed to each other. Which brings us to finding a more general method of characterizing a DBMS such that it lends itself to coexistence in a multi-model environment.

The ANSI/X3/SPARC/SG DBMS Architecture

The American National Standards Committee on Computers and Information Processing (ANSI/X3) established a Study Group of its Standards Planning and Requirements Committee (SPARC) charged with investigating the subject of Data Base Management Systems with the objective of determining which, if any, aspects of such systems are suitable candidates for the development of American National Standards. To provide a context for the investigation, a working definition of the discipline of Data Base Management was viewed as "records, fields, files, sets, and the descriptions of all these, and all the indices, mapping techniques, access methods, file organizations and end user languages."

The architecture of DBMS's recommended is partly based on the concept of 'nested machines.' The outermost machine is most closely related to the

real world view of and most closely aligned to the functionality and support of the data base. As we descend inwards through successive machines, we pass the various logical views of the data base on to its physical attributes until ultimately we arrive at the actual secondary storage devices. The need for data manipulation interfaces for multiple classes of users and multiple data declaration interfaces for achieving data independence has been recognized and defined in the proposed architecture.

A three-schema approach has been suggested:

Conceptual Schema: embodies the 'real world' view of the enterprise being modelled in the data base. The enterprise is described in terms of the entities with which the enterprise is concerned, the attributes by which they are described and the relationships existing between them. It also provides the basis for integrity and security declarations imposed by the enterprise on the various users and a data description basis for restructuring.

The objects utilized are:

attributes (conceptual fields)

conceptual groups

conceptual records

conceptual plexes

conceptual record set

conceptual data base

External Schema: Various users of the data-base operate on subsets of the total enterprise model relevant to their particular needs. These subsets contain the names and characteristics of the objects

visible to a family of applications as well as the associations and structures in which they are related and the operations permitted on them.

The objects defined are:

external fields

external groups

external record

external plex

external record-set

Internal Schema: describes the 'machine view' of the data specifying the stored representation of the enterprises information.

The objects defined are:

internal model space

internal field data (element)

internal field aggregates

internal record (stored record)

internal record aggregate

space extent

form extent

internal record-set (data set)

internal data base

data bank

These objects provide a precise definition of how the internal data base is represented, organized, stored and accessed.

The external and internal schemas must be consistent with and derivable

from the conceptual schema. This is accomplished by specifying mappings between the conceptual and the external, internal schemas. Each schema is specified by its related administrator and the mapping functions maintained and executed by the related schema processor. Figure 1 illustrates this architecture.

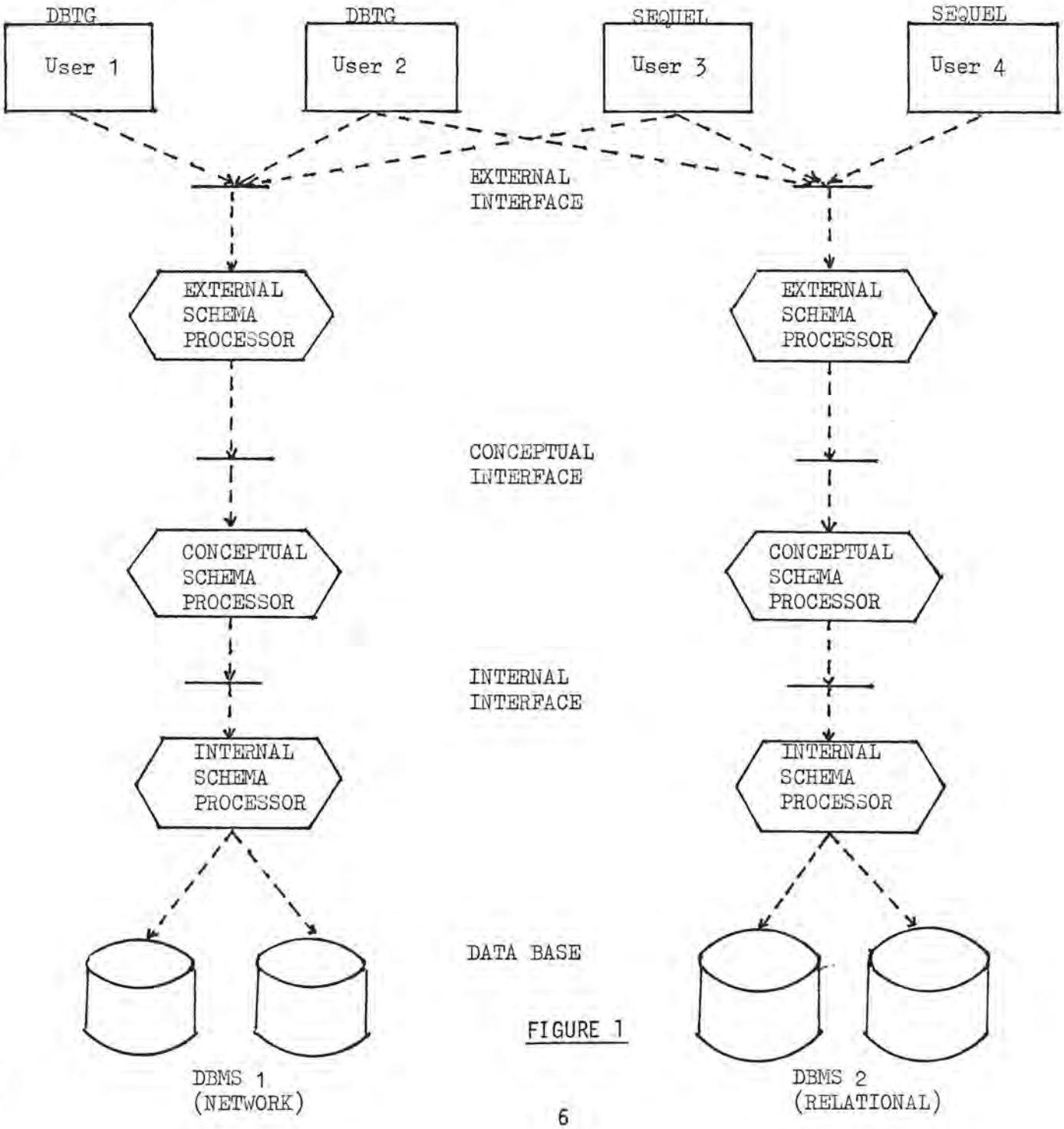


FIGURE 1

Some Basic Concepts Underlying Data Models

In viewing data, people logically organize their view of the real world at various levels. At one level they perceive the real world; at another they interpret (or give meaning to) the real world. Finally, they record and describe their ideas of the real world on some physical medium, such as data in computers.

When giving meaning or in order to understand independent objects of the real world, we interpret them as sets of entities or entity sets. Entity sets correspond to objects that have an independent existence and can be meaningfully considered by themselves.

An entity set can be meaningfully described in terms of its attributes. Although there may be a real distinction between an object and its characteristics in the real world, this distinction becomes a bit vague when representing ideas about the real world by attributes and entity sets.

For each entity set, its attributes can assume certain values. Data values by themselves say nothing meaningful; it is only when a relationship is established between members of value sets that some information is derivable, e.g., sets of street name values and city name values do not really communicate anything until one can relate the street name to a city name. A relationship then is a correspondence or mapping between members of two sets. Relationships may be 1:1, 1:N or N:M correspondence between the members of the two sets. It is useful to distinguish between two types of relationships:

In one case when we are considering characteristics of an object, these

characteristics remain of interest as long as the object exists. The relationship that exists between these characteristics is called an 'attribute relationship,' that is, a relationship between attributes of an entity set.

In the other case, a relationship may be considered between two objects, each of whom has an independent existence without regard to the other. In such an instance, the relationship defined between the two entity sets is termed as 'association.' Such relationships may be 1:1, 1:N or N:M.

When storing data in computers, we usually organize the data according to some pattern that represents entity sets and the relationships between them. This we term as the 'data model.' Data models used by different DBMS's can be distinguished mainly by how they represent data relationships.

The Network and Relational Data Models

The Network model: is a formal model for representing attribute relationships of an entity set and the associations between the entity sets. The data model consists of record types and connections among them which are represented as links. 'Record types' are used to represent the relationships among the attributes of an entity set and 'links' to specify the associations between entity sets.

A record type is defined as a collection of data items, the data item being the smallest unit of logical data and the record type being the generic description representing a set of record occurrences consisting of the defined data item values.

Associations are usually effected by explicit mappings between different record types, a link being defined as the representation of an association. Thus, while an association is an abstract perception of the real world, a link is a concrete object representing the association in a network data model.

Links, since they represent relationships, may be 1:1, 1:N, N:M. Two kinds of links may be distinguished 'information carrying' and 'non-information carrying':

> An information-carrying link represents an association that cannot be expressed as a closed form property between two sets, i.e., non-presence of any items in the data sets that could express a relationship between objects of the sets.
>
> Non-information carrying links do not carry any extra information regarding a relationship that could not be constructed by the data-items available, they are for convenience.

Information carrying links are usually constructed manually by selecting records and explicitly connecting them. Non-information-carrying links can be constructed algorithmically once the property of the data items defining the link is specified. Such links can be constructed and maintained automatically.

In a general network data model, there are no restrictions on the relationships represented by the links. They can be 1:1, 1:N or N:M; a link can also connect a record type with itself sometimes called 'recursive links.' Specific implementations may, however, place some limitations

on the use of links, such as the DBTG proposal, which we shall discuss in some depth later.

In a data base organized according to the network data model, consisting of record types and links, a user traverses the data base according to the connections defined by the links between the records. A record is selected using some qualifications, then another occurrence following a connection according to a link, and so on. The system keeps track of the record occurrences the user visits by maintaining pointers called 'currency indicators' to the record occurrences. The user can also manipulate these currency indicators.

Most languages for network systems implement some sort of navigation with either explicit or implicit currency indicators.

The Relational Model: is a formal model for representing relationships among attributes of an entity set and the associations between entity sets. Given a set of domains S_1, S_2, ..., Sn, R is a relation defined over these domains, such that it is a set of tuples, each of which has its first element from S_1, its second element from S_2, and so on. In other words, R is a subset of the Cartesian product S_1 X S_2 ... X Sn. The set S_J is referred to as the J^{th} domain of R. As defined, R is said to have degree n.

A relation represents an entity set both in terms of its intentions, that is, the entity set name, its attributes and their properties, and in its valid extension, that is, the possible values the attributes may have.

A data base relation is also time-varying since the entity set which the

relation represents changes over time as entities are inserted, deleted, and modified; this is one important aspect in which they differ from mathematical relations.

An n-ary relation can be represented as a table, each column of the table called an attribute corresponds to a domain of the relation and each row to an n-tuple. The ordering of rows is immaterial and all rows are distinct, i.e., an entity's representation as a tuple cannot appear more than once in the table. If each column is labelled with the name of its corresponding domain, then the ordering of the columns is also insignificant.

To ensure unique identification of an attribute (column), attribute names within relations must be unique. When more than one attribute of a relation take their values from the same underlying domain, the distinct roles played by each attribute are distinguished by prefixing each appearance of the attribute name by a role-name.

Normally subsets of the values of some attributes of a relation uniquely identify each tuple of the relation. This 'key' of the relation has the following time-independent characteristics:

Unique identification: in each tuple of the relation the key uniquely identifies the tuple.

Non-redundancy: no attribute in the key can be discarded without destroying its property of unique identification.

All attribute relationships and all associations are viewed as relations at the data model level. However, additional semantic information may

be introduced to distinguish the semantic properties of different relations.

Relations may already exist in the data base or may be generated from existing relations using relational operators. Both the operand(s) and the result of a relational operator are a relation.

Relational operators may be described using relational calculus, relational algebra or set-oriented expressions. Any of these approaches are valid in that they are 'complete' in their ability to provide data manipulation facilities on base relations.

II. MODEL DEFINITION AND TRANSLATION

In order to facilitate our analysis of the Network and Relational data models and establish a framework in which the model definition and translation capabilities can be assessed, we shall consider the following implementational specifications for these data models:

CODASYL DBTG (Network).

System R (Relational).

DBMS Specifications

CODASYL DBTG: The Data Base Management System developed by CODASYL (*CO*nference on *DA*ta *SY*stems *L*anguages) DBTG (Data Base Task Group) is a host language-oriented system whose data base capabilities are provided by enhancing the data description and data manipulation facilities of a programming language. The outline of the DBMS is versatile and based on modular constructs.

The data base has associated with it:

A schema: This represents the community view of the data base and is variously called data schema, conceptual schema.

A storage schema: to describe the physical aspects of the data base or internal schema.

Sub-schemas: representing the user view which is a subset of the corresponding schema.

The overall architecture of the CODASYL DBMS (Fig. 2) has four major

sets of modular constructs.

User working area (UWA): being the loading and unloading zone between the DBMS and an application.

System Buffers: being the loading and unloading zone between the DBMS and the operating system.

Data base: itself.

DBMS, Object Schema and Sub-Schemas: containing the descriptions of the data base (storage and data structures) and the user's view of the schema.

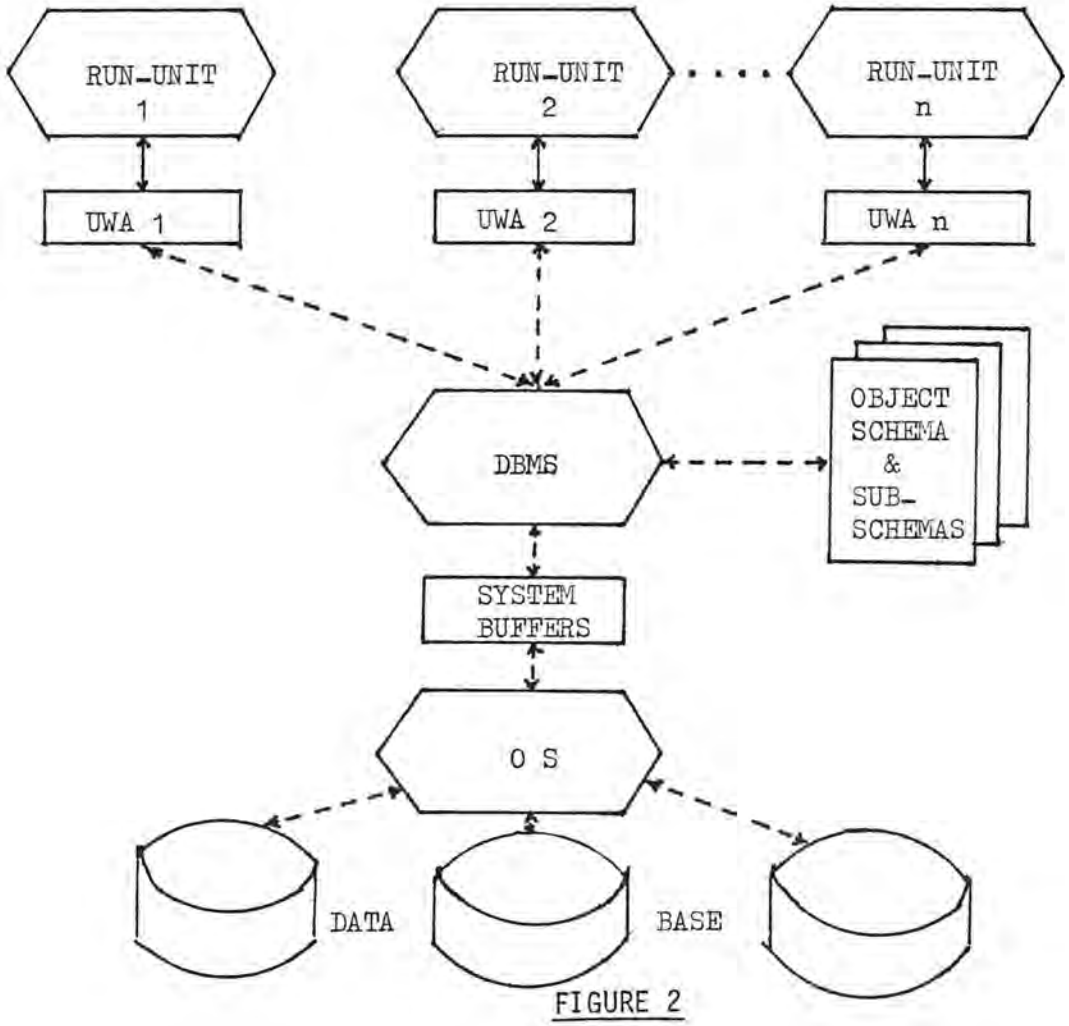


FIGURE 2

Since the general functions and objectives of the User Working Area and the System Buffers are self-evident, we shall concentrate on the salient features of the DBMS, Object Schemas and Sub-schemas.

The characteristics of the DBMS are declared in the schema specifications using the DDL. This set of declarations encompasses with a few limitations the object schema and the sub-schemas and includes all logical units of data.

The schema declarations closely follow COBOL syntax and being a host-language-oriented system, we shall try to describe the DBMS using the DDL constructs as a guideline.

Data item: is the smallest named logical unit of data.

Record type: is a named collection of data items. Data items within a record may be represented as:

data aggregate: being a named collection of data items, which may again take the form of:

vector: a collection of data items with the same characteristics.

repeating group: a collection of data items, vectors or repeating groups occurring an arbitrary number of times.

Set: represents a named 1:N link among record types. It consists of one owner record and one or more member record types; the owner must be of a different type than its members. Here it would be appropriate to point out the major departure between the DBTG implementation and the standard network model definition.

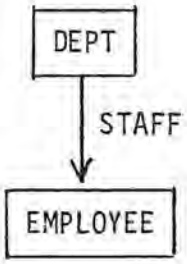
All links are 1:N, that is, no record occurrence can participate in more than one set occurrence of the same type.

No recursive links are allowed, that is, an owner and member records of a set have to be of different types.

The set concept is the single most important concept of the DBTG model "the (CODASYL) set mechanism is a basic building block which can be used to construct complicated data structures."

One variation in the set construct is a 'singular set' for which the owner declared in the schema is the system. Any number of singular sets may be declared, each has precisely one occurrence.

An example of two record types DEPT and EMPLOYEE linked through a set STAFF which connects employees working for a specific department would be a simple illustration:



Data base key: this is a unique identifier of a record within the data base distinguishing it from all other record occurrences during the time the record exists within the data base. This does not imply that the data-base key is a physical address; it is generally a symbolic value from which the physical address can be derived.

Area (Realm): is a named logical subdivision of the addressable storage space in the data base. For each type of record the schema specifies the area or areas into which occurrences of that record are to be placed when they are entered into the data base. An area-id is a data base parameter containing the area name in which the record is to be located. The Area concept is used for:

On/off line problem in large data bases.

Exclusive access mechanism in concurrency.

Aid in recovery.

As a protection mechanism for security.

Currency indicators: are conceptual entities that are maintained by the DBMS to keep track of record retrieval and storage. The content of a currency consists of a data-base key value for the record occurrence most recently accessed for the following:

Each area.

Each type of set.

Each type of record.

Run-unit, i.e., any type of record for the current execution of the program.

We will end this general overview of the CODASYL DBMS with a note on the Sub-schemas and Object or Storage Schemas:

Sub-schemas: are a definition (or redefinition) of a portion of the logical data (schema) in the data base of interest to an application program in a form suitable for its intended use. Any number of sub-schemas may be defined on a given schema and any number of programs

may share a given schema.

Although a sub-schema must be self-consistent within its operating environment, it may differ from the schema in:

declaration of one or more areas.

declaration of one or more sets.

declaration of one or more records.

declaration of one or more data-items.

These are some other points where it may differ which we shall exclude for lack of relevance.

Storage (Internal) schemas: provide the physical mapping model between the schema and the physical storage as well as the run-time facilities for navigating the data base (access paths).

Most of the physical mappings are declared statically as part of the schema and only limited modifications are permitted in the sub-schema.

System R: is a relational Data Base Management System providing a high-level data interface using the set-oriented sublanguage (SEQUEL) approach.

The system permits a variety of relational views on common underlying data, at the same time providing data independence by isolating the end user as much as possible from underlying storage structures and data access paths.

The overall architecture of the system will be discussed in this section leaving the detailed analysis of individual components and their functions to later review.

The system provides multi-user support for a variety of host and query languages which may themselves be employing differing data model views (Fig. 4).

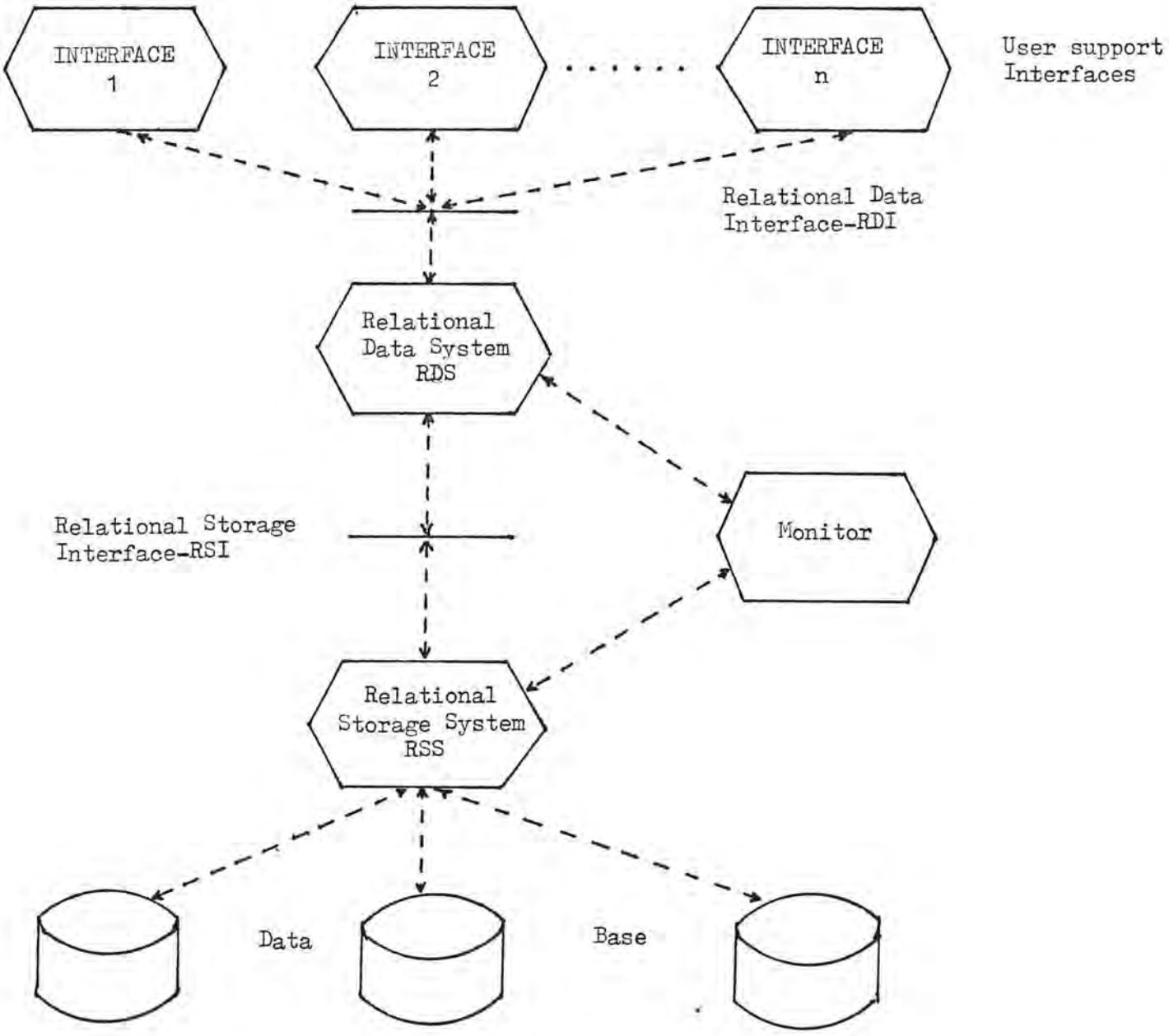

INTERFACE 1
INTERFACE 2
INTERFACE n
User support Interfaces
Relational Data Interface-RDI
Relational Data System RDS
Relational Storage Interface-RSI
Monitor
Relational Storage System RSS
Data
Base

Figure 4.

System Architecture

Relational Data Interface (RDI): is the external interface called directly from a programming language, or used to support various emulators and other interfaces. The high-level SEQUEL language is embedded within the RDI and is used as the basis for all data definition and manipulation.

The Relational Data System (RDS): is the sub-system that implements the RDI. It provides authorization, integrity enforcement and support for alternative views of data. The RDS maintains the catalog of external names and contains an optimizer which plans the execution of each RDI command choosing a low-cost access path to data from among those provided by the Relational Storage System.

Relational Storage Interface (RSI): is an internal interface which handles access to single tuples of base relations. Calls to the RSI require explicit references to the location of data (segments) and the access paths (images, links, identifiers).

The Relational Storage System (RSS): is a complete storage subsystem in that it manages devices, space allocation, storage buffers, transaction consistency and locking, deadlock detection, backout, transaction recovery and system recovery. It also maintains indexes on selected fields of base relation and pointer chains across relations.

Finally, the RSS provides support for the RSI.

The Monitor: contains the system administrator facilities which controls logon authorization and initializing the data base for each user. It also schedules periodic checkpoints and maintains usage and performance statistics for reorganization and accounting purposes.

System Constructs

Relations: the main data object of the RSS is the n-ary relation consisting of a time varying number of tuples each containing n-fields. A new relation can be defined at any time; an existing relation with its associated access paths dropped and extensions to tuples of a relation can be made without a data-base reload. Two field types are supported: fixed and variable length. Using a special RSI protocol fields can be generated with an undefined value. Tuples are stored as a contiguous sequence of fields.

Tuple Identifiers (TID): associated with every tuple of a relation is a tuple-identifier. This is generated by the RSS and is available to the RDS as a concise and efficient means of addressing tuples. TIDs are also used by the RSS to refer to tuples from index structures and pointer chains. However, they are not intended for the user above the RDS level. They are essentially pointers to the physical location of the tuple in internal storage.

Views: are dynamic windows on the data base providing a view on a relation derived from one or more relations as a result of a query

operation. This view may then be used as a base relation for further update operations so long as the tuples of the view are associated one-to-one with the underlying base relation. Views may be made permanent or dropped, in which case all further views defined in terms of it are also dropped.

Images: are a logical reordering of an n-ary relation with respect to values in one or more fields. They provide associative access capabilities and low-level support of simple views.

A new image may be defined or an existing one dropped at any time. In order to conduct a scan on a given image the parameters passed should include the sort field values and the TID of the given tuple. Once defined, an Image is automatically maintained by RSS during all insert, delete and update operations on it.

Links: are an access path in the RSS which are used to connect tuples in one or two relations. The RDS determines which tuples will be on a link and their relative position through explicit connect, disconnect operations. Links may be of two types:

Unary links: involve a single relation and provide a partially defined ordering of tuples. They can be used to maintain ordering specifications not value ordered and an efficient access path.

Binary links: provide a path from single tuples (parents) in one relation to sequences of tuples (children) in another relation. A given tuple can appear only once within a given link. They are used to connect child tuples to parents based on value matches

thus providing past associative access to a relation without the use of an extra index.

Links are maintained by storing TIDs in prefix of linked tuples.

Segments: are a collection of logical address spaces employed to control physical clustering. They are used to store data, access path structures, internal catalog information and intermediate results generated by the RDS. All tuples of any relation must reside within a single segment, however a given segment may contain several relations.

Several types of segments are supported with its own function and overheads. A user-specified maximum length is associated with each setment. Segments consist of a sequence of equal-sized pages with the RSS maintaining a page map for mapping to physical storage.

Cursors: are a means by which the RDI interfaces SEQUEL to a host programming language. A cursor is a name used at the RDS to identify a set of tuples called its 'active set,' and furthermore to maintain a position on one tuple of the set. The cursor is activated by means of the RDI operator SEQUEL; the call has the effect of associating the cursor with the set ot tuples that satisfy the query and positioning it before the first such tuple, e.g., CALL SEQUEL (C1,'SELECT NAME, SAL FROM EMP WHERE JOB = 'PROGRAMMER'');

Optimizer: the objective of the optimizer is to find a low-cost means of executing a SEQUEL statement given the data structures

and access paths available. The cost measure of the optimizer is based on storage page accesses. The physical clustering option of tuples in the data base is a major consideration. The optimizer begins by classifying the statements into groups according to the presence of various language features. Next it examines the system catalogs to find the set of images and links pertinent to the given statements and applies a decision procedure to find a reasonable method of execution.

After analysis of the SEQUEL statements the optimizer produces an Optimized Package (OP) containing the parse tree and execution plan. If the statements require fetching tuples, the OP is used to materialize tuples. If the statement is a view definition the OP is stored in the form of a Pre-Optimized Package which can be utilized whenever an access is made via the specified view. If any change is made to the structure of a base relation or to the access paths maintained on it the POP's of all views defined on it are invalidated.

Clustering: is the property of physically storing tuples near each other according to some criterion. The system accepts clustering specifications for:

Relations: in the form of its TID's associated with a storage page.

Images: at most one image per relation may be clustered.

Links: may be declared to have the clustering property.

Implementational Aspects of Major Constructs

Before we can proceed with establishing an equivalence between the model

constructs that have been discussed so far, we shall try to analyze the suggested implementational techniques in order to show the viability of mapping the constructs between models.

CODASYL DBTG:

The User Work Area (UWA): In principle two approaches may be established to define this area : The static approach where the layout of this area is completely determined at sub-schema compile time and the dynamic approach where the layout is determined at run-time.

In the static approach it is assumed that the displacements of items within record type are fixed and that the position of record-areas within the UWA are also fixed. The current COBOL DML and sub-schema DDL offer no facilities to distinguish between different record-areas for the same record type or to specify any explicit overlay requirement for record areas of different record type. The advantage of the static approach is that at run-time no allocation and interpretation is required.

In the dynamic approach the position of the record-area within the UWA is part of the interface, so the address where a record, or part of it, has to be delivered or obtained by the DBMS is one of the parameters passed to the DBMS.

Areas (Realms): should be logical as well as physical non-overlapping storage units. Since, in general, an area will be too large to read in or to store as whole, it has to be divided into physical

blocks (or pages). A simple and efficient manner of organizing these pages would be by using the direct organization (relative recording mode) with fixed length pages. The techniques of mapping the data-base information into these areas we will discuss in the Record and Set sections.

Records: The physical description of the record is clear in the DDL specifications. There are, however, a few issues associated with the maintenance of the record in the data base.

Data base keys. Since the key is a unique identifier of the record within the data base which will remain unchanged during the record's existence, it is not practical to utilize a physical address; a symbolic value must be used from which the physical address can be derived.

This symbolic address must consist of two parts: one being the identification of the record type of the record, and the other a sequentially assigned number by the DBMS per record type. The DBMS maintains a table per record type, which contains an entry for each of the record type occurrences.

Physical placement. This takes two forms: one being within a specified area and the other at a specific place within an area (clustering). The first situation requires a standard page access optimization algorithm.

The second form can again be split into two cases: one where the specific place is determined by hashing a user-key (CALC-KEY) and the other where the place is determined by the placement of associated records. The first option requires a solution to the situation of duplicate hashed-keys, therefore an adequate page overflow mechanism. The second option we shall discuss under set-traversal facilities.

Record retrieval is accomplished either sequentially using the set concept or directly using primary and secondary keys. In all these cases use is made of the internal referencing scheme in which the data base key is used for identification purposes through indirect 'transformation tables.'

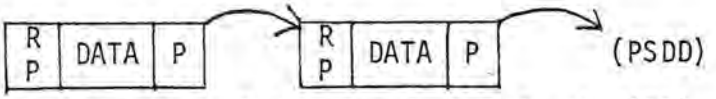
Sets: This concept is one of the cornerstones of the CODASYL DBTG proposal. In its current context a set is defined as 'the definition of an access path through a number of related records.'

Basically sets can be implemented in two ways:

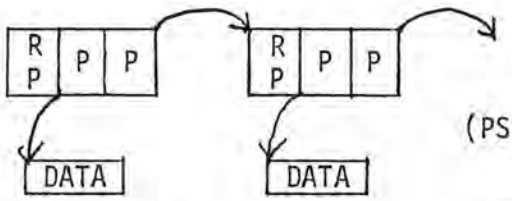
pointer sequential (chaining).

physical sequential (lists).

These two categories vary in their level of indirection. Each of the methods can be further subdivided into a further level of indirection:

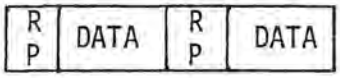


Pointer Seq. Data stored at Ref. pt. (RP).



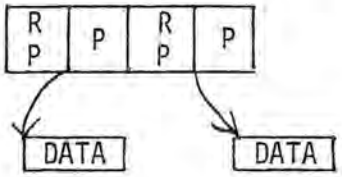
(PSDI)

Pointer Seq. Data not stored at Ref. pt.



(FSDD)

Physical Sequential Data at Ref. pt.



(FSDI)

Physical Seq. Data not at Ref. pt.

These four basic methods may be refined further through

bidirectional chains,

linked to owner options,

attached to owner for lists and pointer arrays.

What becomes apparent is that for maintenance of sets, besides records, tables would also have to be stored in the data base, for example:

pointer arrays,

index-tables for search keys,

record lists.

One advantage to the implementator is that since the type of the tables is identical a uniform table-handling concept is applicable.

System R:

Relations: consist of a time varying number of tuples each containing n fields. These fields may be fixed or varying length and may contain a value or be undefined. Associated with every tuple of a relation is a tuple identifier or TID. Each TID is generated by the RSS and is available to the RDS as a means of addressing tuples. The RSS may also use these TID's to refer to tuples from index structures and to maintain pointer chains.

Tuples are stored as a contiguous sequence of fields within a single page of a segment. Field lengths are also provided for variable length fields. Tuples are only stored on 'data' pages. Tuples from different relations may be stored on a single page according to clustering specifications.

A prefix is stored with each tuple for use by the RSS containing:

- the relation identifier
- pointer fields (TID's) for link structures
- number of stored data fields
- number of pointer fields.

Each tuple identifier consists of:

- a page number within the segment
- a byte offset from the bottom of the page.

This indirection allocates a slot to the tuple in a page with the facility of moving the tuple by merely changing the pointer value in the page TID. If a tuple overflows the page it is moved to a nearby page and its overflow position is kept in a tagged over-

flow record to which the TID points. In subsequent overflow the tagged overflow record pointer is again modified.

Images: are logical reordering facility of a relation with respect to one or more value fields. The RSS maintains each image through the use of a multi-page index structure. An internal interface is needed for associative or sequential access along an image and also for insertion, deletion or updating of index entries. The parameters passed across this interface include the sort field values along with the TID.

Each index is composed of one or more pages within the relation segment. The pages of an index are organized into balanced hierarchic trees (B-trees). Each page is a node within the hierarchy and contains an ordered sequence of entries. Non-leaf nodes contain entries consisting of a <sort value, pointer> pair, the pointer addresses another page in the structure and the value indicates sort value entries less than or equal to the given one. Leaf nodes contain entries having the sort value and an ascending list of TID's for tuples having those values. Leaf pages are chained in a doubly linked list for sequential access support.

Links: are access paths used to connect tuples in one or more relations. The RDS determines which tuples will be linked and their position through explicit operations. The RSS maintains internal pointers for insertion and deletion in linked structures. The RSS maintains links by storing TIDs in the prefix of tuples. When a new link is defined a portion of the prefix is assigned to hold

the entry if necessary by enlarging the prefix. No tuples are accessed until the tuples are connected to the link when the assigned space is formatted.

Views: are dynamic windows on the underlying data base tables. Their impact on the implementation is as a mechanism for authorization to read or access base tuples.

Data Base Transformation versus Simulation

Before we proceed to establishing an equivalence between the two models' constructs that we have reviewed so far, the topic of 'application migration' should be introduced.

The first aspect of this problem is the total conversion of one DBMS to the data representation of the other. This one-time conversion is known as data-base transformation. In a sense it accomplishes the purpose of preserving the original information such that the entire data base is made available to the new model both in its

physical data structure, and

its logical data structure.

However, this is an enormous task for a data base of any reasonable size, and is not feasible as a solution for coexisting data-base systems.

What becomes a viable proposition is the task of query translation in a suitable fashion such that the sub-schema of either model is maintained and the sub-schema definitions and operations are translated in order to simulate the model that is being addressed. This simulation process, however, must make use of concepts that can be equated, both logically

and in terms of physical implementation in order to avoid extreme inefficiency. In the remainder of this chapter this is the subject that we shall address.

Model Equivalencing

In this section we shall try to compare the main data structuring elements of the Relational and DBTG models, and then in the context of their implementation establish an equation between these structural constructs.

The term 'equivalent' is used in the sense that whenever one collection of data is mapped into a subset of the other collection there will be some facts omitted or constraints placed on the expressive powers of the source data after the mapping. In general, however, we can state that the collections are equivalent in the sense that every fact that can be deduced from the sub-collection can also be deduced from the main collection.

In the process of establishing equivalent mappings, the subject of affected data structures becomes integral to the discussion. A 'data structure class' is the class of data structures which can be constructed from a given set of primitives using a given set of operators in a system.

Model views:

From a general model standpoint we shall discuss four elements of the data structure aspect;

entity

attribute

function

access path.

Defined again are the terms used above (ref. ch. I) to emphasize their role in providing an effective comparison:

attribute: is a single valued characteristic of someone or something.

entity: is a collection of one or more attribute values such that at least one subset of the collection called identifier, identifies the entry within the set, while all other attribute values in the collection are functionally dependent on the identifier. An entity set is a collection of entities with the same attributes.

function: is a rule which assigns to each entity of a given entity set exactly one entity of some other entity set.

access path: for one or more attributes of an entity set given the constituent attribute values an access path provides a mechanism for accessing the entity other than scanning all the entities in the set.

Comparison Table (Model View)

MODEL	DATA STRUCTURE ELEMENT			
	attribute	*entity*	*function*	*access path*
Network	yes	yes	yes	yes
Relational	yes	yes	yes/no	no

The attribute concept is known in the DDL as a data item; its equivalent concept in the Relational model is the domain-role.

The entity concept in the DDL is a record occurrence and a tuple

in the Relational model.

The function concept in the DDL is a record-owned set-type. The set-type, however, entails two aspects:

- establishment of navigational routes:

  from owner record to associated member record occurrences.

  from member records to other member record occurrences belonging to the same owner record.

  from member record occurrences to associated owner record occurrences.

- As a semantic rule indicating that an element in the domain (member record) cannot exist without the corresponding element in the range (owner record).

In the Relational model an equivalent of the semantic rule is an integrity constaint. The concept of predefined navigational routes are not integral to the model so that one may either say that there exist no predefined navigational routes or that all possible routes are dynamically materialized.

The access path concept corresponds to the ownerless or system-owned 'singular set' in CODASYL. The Relational model definitionally does not support the concept of an access path.

Implementation View:

From the above discussion, we see there is a close correspondence in the data structuring views of the Network and Relational models. The points of divergence are at the conceptual level, however both

models in their implementation utilize approaches that allow us to bridge this gap without too much difficulty. The language features which we shall discuss in Chapter 3 because of their fundamental differences in approach may further emphasize these conceptual divergences but adopting a simulative approach again allows us to resolve the difference and achieve model communication.

To establish this correspondence we shall set up a table of equivalent constructs which would enable us to further analyze the different viewpoints.

Comparison Table (Implementation View)

SYSTEM R	CODASYL
field (domain)	data item
tuple	record occurrence
relation	record type
tuple identifier	data base key
integrity assertions/ binary links	sets
images/unary links	singular sets
cursors	currency
view	sub-schema
segments	areas (realms)
clustering	area-id
transaction	run-unit

Field/data item: the concept of the attribute in the respective models is field and data item. The data types supported

in both are varied enough to support any requirements. The major difference arises in the CODASYL facility of the presence of repeating groups of data items within records. Since the relational model expects the relations to be normalized, that is, separation of any subset of the domain elements that in themselves constitute a relation. We will treat for purposes of comparison the existence of repeating groups as a relation with the special facility of clustering with the main record.

Tuple/record occurrence: the main entity of data objects is the tuple or record occurrence. All accesses to the data base are made in terms of this entity. Both constructs can contain any number of occurrences of fields or data items respectively. The storage characteristics of both follow more or less the same pattern, consisting preferably of a contiguous sequence of fields/data items, the exception being the repeating group mentioned above.

Relation/Record type: The entity set is manifested through these two constructs. In the relational system operations are defined in terms of one relation or more, however in the CODASYL system a record type is addressed in terms of its set with the exception of the singular set. The record type serves therefore as a generic identifier activated in terms of the set. The objective served is the same.

Tuple identifier/Data-base key: Every entity of the respective entity set's Relation or Record type is uniquely identified

through its identifier. So long as the record/tuple remains in existence within the data base the associated identifier remains with it. In the CODASYL implementation the data-base key is required to be additionally abstract, that is, it should be (pointer indexes) disassociated from the physical address as far as possible to allow mobility in physical record placement. System R tends to achieve a practical mobility through pointer TID's, if necessary maintaining pointer chains.

(Binary links/integrity assertions)/Sets: The function concept of establishing navigational routes between two entity sets and maintaining a semantic rule on this relationship is the basic role of the Set concept in CODASYL DBTG. Building on this basic concept sets develop as a powerful data structuring facility. Sets fall into two categories:

Information bearing: where a set carries its own relationship information, a distinction is made between the existence of a relationship and values used to determine the relationship. There is the presence of path dependency; a record's uniqueness is contextually derived from a series of owners together providing uniqueness of keys.

Non-information bearing: where matching values in owners and members of sets carry the relationship information. No distinction is made between the existence of a relationship and the values used to determine it. There is no path dependency since the owner record contains a primary

key that uniquely identifies it, all members of that owner occurrence being related through value matches.

Although a Relational model does not support predefined navigational paths, binary links in System R provide an access path from single tuples of one relation (parent) to sequences of tuples (children) in another relation. The connection of these links is, however, value-dependent, in order to translate information bearing sets--unique sets of keys must be generated between each level of owner and member records.

The supporting 'optimizer' provides an OP (optimized package) access path to the defined tuple of the owner base relation and subsequently follows the established access path to the physically linked members. The semantic rules are applied through integrity assertion allowing the system to automatically maintain the relationship.

(Images/Unary links)/Singular Sets: Access paths in a single entity set are accomplished in CODASYL through ownerless or system-owned sets. Since a Relation in general has no ordering sequence and, hence, no access path, an Image which is a logical reordering of a relation on one or more fields, or a Unary link which encompasses only one relation-linking single tuples into chains provides an equivalent mechanism.

Cursors/currencies: For purposes of accomplishing navigation the system must indicate the currently accessed or accessible

entity in the entity set being operated upon. The CODASYL system provides a set of currency indicators for each of the following:

area

record type

set

current run unit.

System R provides an equivalent concept to manipulate a tuple or set of tuples at a time through the cursor concept. Any number of cursors may be active for a given transaction which can conveniently be translated and maintained for a (segment, relation or a transaction).

Views/Sub-schemas: A sub-schema represents a subset of the schema. It must be consistent with the schema declaration but may omit or modify many of the schema specifications. In a sense it represents the user's view of the schema.

In System R the view mechanism becomes the users prime access ability. The query power of SEQUEL allows a view to be defined as a relation derived from one or more relations. This derived relation may then be used as a base relation for operation or definition of subsequent views.

Segments/(Areas/Realms): In order to make the data base more manageable both for purposes of operational efficiency and to assist the task of the data base administration, the total address space is divided into logical sections called Segments

in System R and Areas or Realms in CODASYL. In physical terms these sections would be environment dependent. Segments and Areas are subdivided into pages for physical access convenience.

In System R, since all data, including the system data such as indexes, etc., are also stored in segments, various categories of segments are specified. Although CODASYL specifications do not comment on this aspect, it would seem a rational implementational criterion.

Clustering/Area-ids: Physical placement of entities or entity sets present divergent philosophies in principle. The Relational model minimizes physical storage considerations from the user. CODASYL, on the other hand, expects explicit specification of its entities and entity sets.

In the CODASYL system, the user specifies the area or areas into which occurrences of a given record are to be placed. Before an access can be made to an area it must be opened and before a record can be stored its associated area must be specified.

System R does not allow such explicit storage manipulation strategies, however the system accepts clustering specifications for images and links. A relation is placed within a given segment thereby allowing a tentative, if not explicit, control over the physical storage of data.

Transaction/Run-unit: Activation of system resources for a user entails considerable system activity. In the CODASYL system the run-unit provides a system basis for maintaining integrity and recovery, it is also necessary for updating currency indicators a necessary mechanism for DBTG DML.

In System R a transaction constitutes a series of RSI calls on behalf of one user involving making available requisite system resources and providing a unit for recovery and consistency support.

Model Translation

This section will investigate the strategy and mechanics of accessing:

CODASYL DBTG data base described by a DDL schema, by a sub-schema presenting the user with a relational (System R) view of the data base.

Relational data base having the architecture described by the System R DBMS presenting the user with a CODASYL DBTG Network view of the data base.

The suggested strategy assumes the presence of an interface program which allows the mapping to be accomplished between the external schema view and the underlying conceptual schema. The mapping is essentially simulative in nature and equivalent model constructs are utilized to provide the facilities expected of the model being simulated.

We have shown that although the views of data held by the Network and Relation are quite different, the implementation techniques utilize mechanisms that are similar. It is, therefore, possible to establish a mapping

from one to the other without loss of information.

Relational (System R) data base with a Network (CODASYL) sub-schema:

The general strategy adopted would be to represent the network constructs by their closest constructs in the Relational model. The interface program would map the DBTG DDL declarations on to these constructs and maintain a 'data-dictionary' of the correspondence between the two DDL's. At query execution time the translated DML query would be passed to the System R RDI in its equivalent SEQUEL terms allowing accesses to individual tuples and a simulated navigation of the underlying data base.

Record representation: A CODASYL record type is transformed into a TABLE declaration, the details of each record being converted into a field definition list, that is, tuple definition according to the following schema. Consider a record type with the following structural definition:

```
RECORD NAME IS R

    . . .
-02  A
 -03  B
  -04  C    PIC 9(2)
  -04  D    PIC 9(2)
 -03  E     PIC x(5)
-02  C      PIC x(5)
```

For each record occurrence of this type the corresponding System R tuple includes the following fields:

```
<B.C. , INTEGER>
<D    , INTEGER>
```

<E , CHAR (5) >

<R.C , CHAR (5) >

each elementary data item being selected as a field, the names of the data items being qualified by names at higher levels to create uniqueness.

Where a record type has present within it a repeating group (generally a form of DDL to be discouraged) the repeating group would constitute a separate relation containing the key fields of the record type itself and the fields constituting the repeating group. In order to maintain the clustering property between two such relations and a rapid access path, a 'binary link' would be declared on the record key value having a clustering specification.

Set representation: A CODASYL set allows a functional mapping between an owner record type occurrence and occurrences of its members. Schema declarations for each set whether information bearing or non-information bearing would generate the creation of a binary link between the corresponding record types. In the case of information bearing sets an additional indicator would be set up for the two concerned relations requiring generation of additional fields in the member records corresponding to the key field of the owner record, thus, removing any path dependencies.

System-owned sets involving only one record type would simply generate an image specification on the target relation.

Any orderings specified on the record types would be included in

the links or images created.

Area specifications: Allows users to have explicit control over data base storage areas as to record placement strategies and processing control through opening and closing them. System R manages such tasks itself and maintains a subsystem, the RSS, to accomplish this. All DBTG commands relating to specific areas would therefore be redundant and consequently ignored in the mapping process.

Access paths: DBTG allows two levels of access to its data base, which usually operate in conjunction:

LOCATION MODE: specifies the access strategy to an occurrence of the record type.

SET OCCURRENCE SELECTION: specifies the access stragegy to an occurrence of the specified set.

Since the occurrence of a set requires first locating the corresponding owner record, the LOCATION MODE of the owner record is the most usual manner of specifying SET OCCURRENCE SELECTION.

System R allows an analogous prespecification of an access strategy to a given relation's tuple occurrence. This is accomplished by entering a view definition into the system, which represents the 'currently visible tuples' of each relation whenever the view is opened. At definition time the requisite access path in the view is generated by the 'optimizer' utilizing any links or images available in the RSS system catalog in the form of a POP (Pre-

Optimized Package). Thereafter, whenever a view is utilized the POP provides an optimal access path to the selected tuple(s).

Currency indicators: Navigation through the data base in the DBTG systems requires a heavy dependence on currency indicators which tell the user the current position in the data base for:

Record type.

Set.

Area.

Run unit.

System R utilizes an equivalent concept called 'cursor' which is used to identify the position on the current set of tuples (active set). Since multiple cursors may be declared during the course of a single transaction, the translator program could maintain and update a set of cursors for each of the following:

Relation.

View.

Transaction.

Since a mapping to System R would not give a user explicit control over area (segment) an area cursor becomes redundant.

UWA (User Work Area): The user work area where the DBTG user has available to him, data to or from the data base is created using a set of BIND commands for each field of each record type declared in the sub-schema. These variables are where data are delivered as the result of a query or in which data are stored for utilization as arguments of a query.

Further detailed treatment of variant constructs for each major construct discussed here is given in language translation and mapping.

Network (CODASYL DDL) data base
with a Relational sub-schema:

The CODASYL DBTG report defined a schema Data Description Language (DDL) which was intended to be independent of but common to many other high-level programming languages. In this section we consider the case of interfacing it to SEQUEL as implemented in System R.

The power of SEQUEL lies in expressing its definitions and operations in a set-oriented form. Since the user is not supposed to procedurally navigate through the data base system, support facilities must be provided which evaluate the SEQUEL DML and DDL statements and convert them into appropriate low-level access primitives (RSI operators). The sub-schema control environment is established through additional operators (RDI) of which SEQUEL itself is but one.

In our mapping strategy we would therefore essentially take into consideration the RDI and RSI constructs rather than the SEQUEL query which would essentially be converted or presented as one of them to the translator program prior to being interfaced to the DBTG system.

Program variables: Each variable that is declared through the BIND statement constitutes essentially the user work area and represents the sub-schema declaration to be derived from the declared schema.

Cursors: Each cursor to be utilized by the user is declared together with its associated relation or view name. The translator

would keep a record of such declarations for later correspondence between the 'current of' and the specific cursor names.

Relations: Each declared table utilized by the user would be mapped as a specified record type and the field definition list be converted into definitions of the corresponding data item.

Since relations do not contain any prespecified ordering and access to tuples is obtained through the TID, the LOCATION MODE specified would be using DATA BASE-KEY item into which the system would place the TID for access.

Access paths: Availability of system utilized physical access paths is stored in the RSS in the form of images and links. The user would declare a binary link for each set declaration available in the DBTG schema which would be utilized by him.

SET OCCURRENCE SELECTIONS would be through LOCATION MODE OF OWNER since System R's optimizer would most closely follow such a path. Maintenance of the set, in this instance a binary link, is done by System R through explicit CONNECT, DISCONNECT operators. Therefore, membership would either have to be MANUAL for such sets or the RSI operators CONNECT+DISCONNECT would be ignored for AUTOMATIC sets by the translator program.

Query (SEQUEL): A SEQUEL query is parsed and the execution strategy for physical data-base access derived by the optimizer. In this process the optimizer would make use of physical access paths and ordering available in the data base. Since the corresponding

structures stored in the RSS for a DBTG data base would consist of mainly binary links and member record orderings, the optimizer would generate an execution strategy simulative of a network traversal.

To additionally aid the optimizer in deriving an execution strategy the user would declare VIEWs which correspond to the set declarations available in the DBTG schema and utilize the POP's in an inverse fashion to the mappings suggested in the last section.

Most of the SEQUEL operators are available under the DBTG schema. The one major exception is the dynamic generation of new relations using the assignment operation (<-). Since the DBTG sub-schema assumes a static schema new relations may not be added without prior modifications to the schema.

Segment: System R maintains explicit control over its segments through specifications in its RSI operators. The translator program would maintain references to such segments and pass them on to the DBTG system as references to AREAS.

Details of both DDL and DML operators in System R at RDI, SEQUEL and RSI level will be further discussed in the next chapter on language translation and mapping.

III. LANGUAGE DEFINITION AND TRANSLATION

Until now the discussion has addressed the possibilities and facilities available in the Relational and Network systems for mapping the model constructs onto each other. In this process, the discussion has not limited itself to the classical features of the two models, but has taken into account implementational facilities of System R and DBTG in order to make the mapping feasible wherever a one-to-one correspondence between the models was not available.

In order to fully evaluate the feasibility of such mappings being implemented in the framework of two heterogenous DBMS's coexisting and having access to a common underlying data base organized according to one of the data models, we must consider the problem of language definition and translation. The scope of the discussion shall be limited, however, to the data definition and manipulation languages in the System R and DBTG systems, particularly in the context of their providing the facilities required to accomplish the mappings already discussed.

Given the characteristics of the Network and Relational models as to the degree of system support required for implementing their DDL and DML, the point at which the two systems interface with each other shall vary. In other words, whereas the DBTG language can be directly translated into equivalent System R commands, the System R sub-language SEQUEL must be reprocessed into actual access requests prior to being interfaced with the DBTG system.

With this consideration in mind certain extensions of the basic language features will be taken into account in order to accomplish an effective translation scheme.

Definition Languages

CODASYL DBTG: The logical view of the data base is established using the Data Description Language (DDL). This is implemented at two stages:

Schema level: is a logical description of all the data stored in the data base. The schema DDL is considered to be a language independent of any host language and at a higher level than the Device and Media Control Languages (DMCL) and the Storage Structure Definition Languages.

Sub-schema level: to provide the applicaton programmer view normally using COBOL as the host language a sub-schema DDL is used.

The user of the schema DDL can:

name a SCHEMA.

describe the records and their constituent items or aggregates in the data base.

describe the AREA's of the data base.

describe the SET's of the data base.

The types of declarations allowed are:

Naming a SCHEMA:

```
SCHEMA NAME
PRIVACY LOCK
```

The SCHEMA NAME assigns a name to the schema.

The PRIVACY LOCK's declared control access to the schema, that is, to the data base's description.

Declaring an AREA:

AREA NAME

TEMPORARY

PRIVACY LOCK

ON

The concept of an AREA is singular--thus only one occurrence of a named AREA exists in the data base. It provides the DBA control over physical storage maintenance.

A TEMPORARY AREA is a 'scratch-file' within the data base and exists only during the life of a run-unit.

PRIVACY LOCKS declared control users from retrieving or updating data in an exclusive, protected or unrestricted mode.

Procedures may be invoked, via ON when an area is opened or closed. These procedures may be used to gather statistics about the use of the areas.

Declaring Records:

When writing a schema a DBA completes a record entry for each type of record to be stored in the data base--thus a record-entry describes a 'type' of record and many occurrences (that is records) of each record type may occur in the data base:

RECORD NAME

LOCATION MODE

WITHIN (area)

PRIVACY LOCK

ON

at the item or group level:

LEVEL NUMBER (as punctuation only)

ITEM NAME (Data-base-data-name)

PICTURE

TYPE

OCCURS

ACTUAL/VIRTUAL RESULT

ACTUAL/VIRTUAL SOURCE

ON

PRIVACY LOCK

CHECK

ENCODE/DECODE

The LOCATION MODE defines precisely how the programmer is to go about storing new occurrences of the record and affects their placement when they are so stored; there are 3 modes:

DIRECT data-item: the data item is initialized with a data-base-key value in order to place it at the data-base-key position or near it. The item may not be part of a record but must be explicitly declared as type DATABASE-KEY.

CALC procedure

USING data item,...

DUPLICATES ARE [NOT] ALLOWED: the data items comprising the CALC-key must be initialized, the new occurrence being placed at the data-base-key position specified as a result of the procedure.

VIA set SET: the data-items required for the SET OCCURRENCE SELECTION clause are initialized, a data-base key is assigned to the record which is 'as close as possible to the actual or probable logical insert point in the selected set occurrence.'

The WITHIN tells the DBMS where to store the record.

The PRIVACY LOCKS at record level allow control of the storing, deletion, updating, retrieval of records; at the item level they only control fetching, modifying and storing; the ability to modify the sets in which the record participates is also controlled by PRIVACY LOCKS. The ON function allows procedures to be invoked at these control points.

The ACTUAL/VIRTUAL RESULT allows an item to be specified as a result of a procedure or actually stored.

The ACTUAL/VIRTUAL SOURCE allows a record to actually or apparently contain a data item stored elsewhere.

The CHECK clause permits data validation procedures to be declared.

The ENCODE/DECODE clause allows the specification of additional data conversion routines.

Declaring SETS:

When writing a schema a DBA completes a set entry for each set type to be maintained in the data base--many occurrences (that is sets) of each set type may occur in the data base. A DBTG set is a particular kind of relationship among records characterized by one or no (system-owned) owner record and several member record(s)

or none occurrences and/or types.

SET NAME

MODE

ORDER

PRIVACY LOCK

ON

OWNER

MEMBER

ASCENDING/DESCENDING KEY

SEARCH

SET OCCURRENCE SELECTION

The MODE clause allows the DBA to choose between certain specified mechanisms for maintaining the set-relationship between member records.

The ORDER clause specifies the ordering of the member records declared in ASCENDING/DESCENDING KEYS.

PRIVACY LOCK allows control of the reordering insertion and removal of record instances in a set occurrence and also of its use for retrieval. The ON clause allows procedures to be invoked at the same control points except for when the SET occurrence is used for data retrieval.

All sets must have an OWNER record except system-owned sets.

A set may contain several MEMBER record types. The membership of the type(s) are a combination of the following two categories:

MANDATORY/OPTIONAL: A MANDATORY membership specifies that the

record type once entered in the set can never have an existence in the data base not as member of that set. The only way to destroy the membership is by deleting the occurrence from the data base. An OPTIONAL membership allows removal of a record type occurrence from a given set without affecting its presence in other portions of the data base.

AUTOMATIC/MANUAL: An AUTOMATIC membership specifies that whenever an occurrence of a record type is created and placed in the data base (using STORE) the DBMS will automatically insert it into the appropriate set occurrence. A MANUAL membership requires the program to explicitly store the record type occurrence (using INSERT) into the appropriate set occurrence.

The specified membership class is concerned with set maintenance.

The sort keys for ordering the set occurrence are specified by using the ASCENDING/DESCENDING clause provided, of course, that the set's order has been declared to be sorted.

Keys and indexes to be used to search the set are specified in the SEARCH declaration.

The SET OCCURRENCE SELECTION allows the specification of an access strategy which the DBMS will use to isolate the unique set occurrence which a user wishes to process. This clause may not be specified for a system-owned set. The processing involved does not affect currency indicators of the run-unit. There are 3 forms of SELECTION:

THRU CURRENT OF SET: the current occurrence of the set is to be

used, which has presumably been procedurally selected by the program.

THRU LOCATION MODE OF OWNER USING data-item: the implication of this form is that the owner must have a location mode declared as VIA set SET of which it is a member. The initialized data-item is used to search for the owner in its location mode specified set. This set may in turn have the same SET OCCURRENCE SELECTION to an arbitrary number of levels.

THRU LOCATION MODE OF OWNER: the set occurrence is selected by locating the corresponding occurrence of its owner recording using its declared LOCATION MODE, which must be of the types CALC or DIRECT.

The user of the Sub-schema DDL can:

name a SUBSCHEMA and its 'parent' schema.

declare alternative names for data-entities named in parent schema.

selectively copy with a few minor changes the descriptions of areas, records, sets included in the parent schema.

describe the data-items in a form compatible with COBOL.

The declared sub-schema describes only a subset of the data in the corresponding data base. The rules governing derivation of this subset are:

PRIVACY LOCKS: in all sections except the Renaming Section new privacy locks may be declared which override any privacy lock

declared in the schema for that entity. The user-program must satisfy the privacy locks declared for the schema in order to gain access to it.

No equivalent to the ON clause is available in the sub-schema DDL.

All names must be unique within the sub-schema. The Renames Sections allows for declaration of synonyms for names specified in the schema, the uniqueness rules still hold.

The AREAs as described in the schema may be selected; the only attribute which may be redefined are their PRIVACY LOCKS.

The SETS declared in the schema may be selected apart from the PRIVACY LOCKS; the only set attributes which may be redefined are the SET OCCURRENCE SELECTION criteria. This may be necessary when the criteria depend upon schema sets which have been excluded.

RECORD's declared in the schema may be selected and redefined in a COBOL compatible fashion. Apart from redeclaration of PRIVACY LOCKS the only attributes of records which may be altered is the WITHIN clause. This may be necessary because the sub-schema may exclude certain AREAS of the data base. The rules governing redefinition of records are:

Although SOURCE and RESULT data-items cannot be redefined they may be referred to in the sub-schema.

The correspondence between the items/data aggregates declared in the parent schema and the sub-schema is established

in the Renaming Section. The user selects which items are to be copied by naming them and excluding the rest by non-declaration.

The DBMS includes appropriate data conversion routines by comparison of format description in the schema and sub-schema.

System R (SEQUEL)

SEQUEL (Structured English Query Language) has been used by System R as the language for expressing the Data Definition facilities which are supported by the Relational Data Interface (RDI). However, the RDI can support other non-relational interface programs written on top of it, which would simulate the other language/model requirements.

Regardless of the model or language that is being interfaced to the RDI, the following Data Definition facilities expressed in SEQUEL syntax will be supported. The sub-schema or user view of the data together with any associated redefinition facilities are explicitly made available through the SEQUEL statement GRANT, which in turn can be applied recursively to other users.

The following types of ddl-statements are supported:

create a TABLE.

expand a TABLE.

keep a TABLE.

create an IMAGE.

create a LINK.

define a VIEW.

DROP a system-entity.

Creating a TABLE:

CREATE

[PERMANENT | TEMPORARY]

[SHARED | PRIVATE]

TABLE <table-name>:

<field-name > (type[,NONULL])

creates a new base (physically stored) relation:

a PERMANENT table is retained by the system beyond the current transaction sequence until explicitly destroyed. A TEMPORARY table is automatically destroyed when the user who created it logs off.

a SHARED table is publicly accessible to other users. A PRIVATE table is only accessible to users who have been granted specific access authority.

a table-name is associated with each created table, the system uses this as the reference for all further commands made against the table.

each field/domain of the table is given a field-name and a data type is specified for it. The data types INTEGER, SMALL INTEGER, DECIMAL, FLOAT and CHARACTER (both fixed and varying lengths) are supported. The fields may be specified to not utilize the undefined field feature of the RSI.

Expanding a TABLE:

the user may add new fields to an existing TABLE, specifying the name and data types of these fields in much the same fashion as a new table creation. All views, images and links defined on the original table are retained. All existing tuples are interpreted as having null values in the expanded fields until they are explicitly updated.

Keeping a TABLE:

KEEP

TABLE < table-name >

causes a TEMPORARY table to become PERMANENT.

Creating an IMAGE:

CREATE

[UNIQUE and/or CLUSTERING]

IMAGE <image-name >

ON <table-name >

(ord-spec-list)

an image which is a logical reordering of a base relation may be declared with respect to values in one or more sort fields. Once defined an image is maintained automatically by the RSS during all INSERT, DELETE and UPDATE operations. It may also be dropped at any time.

an image may be declared to be UNIQUE which forces each combination of sort field values to be unique in the relation. At most, one image per relation may have CLUSTERING which causes tuples whose sort field values are close to be physically stored

near each other.

specifies the base relation ON which the image has been declared.

any combination of fields can be specified as the image key; furthermore, each of the fields may be specified as ascending or descending.

Creating a LINK:

CREATE

[CLUSTERING]

LINK <link-name>

FROM <table-name> (field-name-list)

TO <table-name> (field-name-list)

[ORDER BY (ord-spec-list)]

a link is an access path which is used to connect tuples in one or two relations. The RDS determines which tuples will be on a link and determines their position through explicit operations.

a link may be declared to have the CLUSTERING property which causes each tuple to be physically stored near its neighbour in the link.

the link has a declared name associated with it.

the user specifies the field names upon whom the links would be established on matching values between the fields in a value-dependent way.

the tuples in the link may be ORDERED on some fields in a value-dependent manner.

Defining a VIEW:

DEFINE

[PERMANENT | TEMPORARY]

VIEW <table-name>

[(field-name-list)]

AS <query>

a view may be defined as a relation derived from one or more other relations, and then may be used as a base relation. When an update or modification to a view is issued, updates to the underlying base relations will be effected so long as the tuples of the view are associated one-to-one with tuples of the underlying relation.

a view may be PERMANENT or TEMPORARY to the transaction sequence.

each view has a name associated with it.

the field/domain names of the derived relation which the view defines.

the derivation of the view is done based on a SEQUEL DML query expression which is prespecified.

DROPping a system-entity:

DROP <system-entity>

an authorized user may issue a DROP command against the following system entities: TABLE, IMAGE, LINK, VIEW, ASSERTION,

TRIGGER, causing them and any dependent system entities to disappear from the system. The last two entities are system control facilities which we have not discussed.

Manipulation Languages

CODASYL DBTG: The DBTG Data sublanguage is a COBOL host language-oriented Data Manipulation Lanugage (DML). The operators are activated using COBOL-like statements and excepting for the STORE command are implicitly dependent on the values of 'currency-indicators' maintained by the system.

The reason for this dependence is the procedural navigation expected of the programmer in manipulating the data base. Considerable assistance is provided by the system through the predefined access paths available through the LOCATION MODE and SET OCCURRENCE SELECTION clauses. However, apart from these, the programmer must procedurally assess the position at which he is in the data base and the logic to be determined for his next action.

The following major DML statements are provided:

FIND locates an existing record occurrence establishing it as current of run-unit and updating other appropriate currencies.

GET retrieves current of run-unit.

MODIFY updates the current of run-unit.

INSERT inserts the current of run-unit into one or more set occurrences.

REMOVE removes the current of run-unit from one or more set occurrences.

DELETE deletes the current of run-unit.

STORE creates a new record occurrence and establishes it as the current of run-unit also updating other appropriate currencies.

FIND:

The function of the FIND statement is to locate a record occurrence in the data base and make it current of the -run-unit, area, record type and of all appropriate sets. There are seven formats associated with the FIND statement:

format 1:

FIND < record-name >

USING < data base-key >

the initialized data-base key which must be local to the run-unit.

format 2:

FIND

[OWNER IN < set-name > OF]

CURRENT OF

(< area > AREA | < record > RECORD | < set > SET | RUN-UNIT)

utilizes the current of AREA, RECORD, SET or RUN-UNIT to locate the appropriate owner occurrence in set-name, or merely updates appropriate currencies from specified currency.

format 3:

FIND

(NEXT | PRIOR | FIRST | LAST | integer-value)

<record-name > RECORD

OF <set-name > SET

depending on the position specified the record name is located in the named set.

format 4:

FIND

OWNER RECORD

OF <set-name > SET

locates the owner record of the current member record in a specified set.

format 5:

FIND

NEXT DUPLICATE

WITHIN <record-name > RECORD

utilized when LOCATION MODE is CALC and DUPLICATES ARE ALLOWED to find all record occurrences having the same hash value derived by the CALC procedure.

format 6:

FIND

<record-name >

VIA [CURRENT OF] <set-name >

USING <data-item > ,...IN <record-name >

If CURRENT OF is omitted, SET OCCURRENCE SELECTION is

used for the value initialized in the data items in the UWA.

format 7:

FIND

< record-name >

VIA [CURRENT OF] <set-name >

USING < data-item > ,...IN <record-name >

essentially the same as format 6, except the match is not with values in the UWA but a sequential scan to match values with those of the current set occurrence.

GET:

GET

[< record-name >]

[;<data-item >,<data-item>,...]

brings the current of run-unit into the appropriate location in UWA. If record-name is specified the system will verify that the current of run-unit is the correct record type. If the data-items are specified values for only those data-items are accessed.

MODIFY:

MODIFY

<record-name >

[;<data-item >,<data-item>,...]

[USING <data-item >]

replace (portions of) the current of run-unit with values taken from UWA. If the list of data-items is included only those are replaced otherwise the entire record. The using clause causes automatic SET OCCURRENCE SELECTION for the modified record.

INSERT:

INSERT

< record-name >

INTO <set-name >,<set-name>,...

inserts the current of run-unit into the current occurrence(s) of the specified set(s).

REMOVE:

REMOVE

<record-name >

FROM <set-name >,...

removes the current of run-unit from the occurrence(s) of the specified set(s) containing it only.

DELETE:

DELETE

<record-name >

[ONLY | SELECTIVE | ALL]

NO qualification: the current of the run-unit is deleted if it is not the owner of a non-empty set occurrence.

ONLY: the current of the run-unit is deleted, together with MANDATORY member occurrences of which it is the owner (OPTIONAL member occurrences being merely removed).

SELECTIVE: same as ONLY excepting that OPTIONAL member occurrences not participating as members in any other set are deleted.

ALL: current of run-unit is deleted together with all members of any set occurrences of which it is the owner.

STORE:

STORE

<record-name>

The record is stored with its values available in the UWA. Data item for its LOCATIONS MODE and the SET OCCURRENCE SELECTIONS of sets in which it participates must be initialized if membership has been declared as AUTOMATIC.

MISCELLANEOUS:

SUPPRESS

(<record-name> | <set-name> | <area-name>)

CURRENCY UPDATES

currency updates for specified system entities are suppressed.

OPEN

AREA <area-name> ,...

USAGE MODE IS

(UPDATE | RETRIEVAL)

[PROTECTED | EXCLUSIVE]

makes available for access the specified areas and their mode of access.

CLOSE

AREA <area-name> ,...

user must close areas on end of session.

MOVE

CURRENCY STATUS

FOR< currency-name>

TO <data-item >

allows user to access and save currency status, that is, the data-base key currently associated with that currency.

System R

The Relational Data Interface (RDI) which is the principal external interface of System R consists of a set of operators which may be called from an interface program. The operator SEQUEL makes available to the external program the facilities of the data sublanguage SEQUEL which is the Data Manipulation Language used by the system. The RDI interfaces SEQUEL to a host programming language through a 'cursor' which is a name used to identify a set of tuples called the 'active set.' The tuples may then be retrieved one at a time.

Actual materialization of tuples is done one at a time, using the FETCH operator and specifying the cursor.

The following RDI operators are provided:

OPEN

(< cursor-name > , < name-of relation or view >)

associates the cursor-name with the specified relation or defined view.

CLOSE

<cursor-name>

deactivates the cursor during transaction execution.

KEEP

(< cursor-name> ,< relation-name >,< list-of-field-names>)

causes the tuples identified by the cursor to be copied to form a new relation in the data base, having some specified relation name and field names.

DESCRIBE

(< cursor-name >,< degree >,< pointers-to-IO-locations>)

when a program does not know in advance the degree and data-types of tuples associated with a cursor, the DESCRIBE operator returns the degree and the data-types of the fields in the locations pointed by the set of pointers.

BIND

<prog-var-name >

<prog-var-address >

defines to the system the program variables and their addresses into which the query will return the results.

FETCH

<cursor-name >

[<pointers-to-IO-locations>]

each call to this operator delivers the next tuple of the active set; if the pointers are specified individual components of the tuple are delivered to the locations specified.

FETCH-HOLD

<cursor-name>

[<pointers-to-IO-locations>]

works in the same fashion as FETCH except that an exclusive lock is placed on the tuple delivered until an explicit RELEASE is given.

SEQUEL

[<cursor-name>]

<sequel-statement>

the following SEQUEL DML statements are supported:

Assignment:

relation-name

[(field-name-list)]

←<query-expr>

creates a new table with the specified relation name as a result of the query expression. If the field name list is provided the new relation will contain only those fields.

Insertion:

INSERT INTO

relation-name

[(field-name-list)]

(<query-expr> | <literal> |<constant>)

inserts a new tuple into the specified relation. If the field name list is provided values for only the field names will be active all other fields being set to null. The values of the new tuple may be derived as the result of a query expression, literal values or the values in the program variable names.

Deletion:

DELETE

<relation-name >

[WHERE boolean |

WHERE CURRENT

[TUPLE] OF

[CURSOR] <cursor-name >]

deletes tuples either in a set-oriented manner based on evaluation of the boolean expression or an individual tuple depending on the specified cursor position.

Update:

UPDATE

<relation-name >

(SET <field-name > = <expr >|

SET <field-name > = <query-expr >)

[WHERE boolean |

WHERE CURRENT

[TUPLE] OF

[CURSOR] cursor-name]

updates relations by either value of an expression or results of a query expression and updating the field names specified in the SET list. Tuples can be selectively updated through appending the WHERE clause.

Query:

SELECT [UNIQUE]

((expr [: <host-location >]|

<var-name> * |

<relation-name> *) |

*)

FROM <from-list>

[WHERE boolean]

[GROUP BY (field-spec-list)

[HAVING boolean]]

[ORDER BY (ord-spec-list)]

a query expression in SEQUEL provides a query facility in a simple block-structured English keyword syntax with simple operations on tables. The relevant data to be accessed is described by set operations, although the RDI operators allow tuple-wise access. For purposes of this discussion some of the more detailed syntactical constructs of a SEQUEL query have been omitted, suffice it to say that they provide substantial flexibility and power of expression.

The Relational Storage Interface provides simple tuple at a time operators on base relations. Since evaluation of a feasible interface with the DBTG system must take into account actual storage manipulation and access requests, the major RSI operators into which all RDI's operators are eventually converted will be considered.

Calls to the RSI require explicit use of data area (segments) along with access paths (images and links) through RSS generated numeric identifiers for data segments,

relations, access paths and tuples. The selection of efficient access paths is handled by the RDS in its optimizer function.

Most of the operators have self-evident meanings corresponding to the initial RDI commands the only difference being that the RSI operators are directed towards the formatted (storage) control blocks.

Operators on segments:

OPEN-SEGMENT (<segid>)

CLOSE-SEGMENT (<segid>)

SAVE-SEGMENT (<segid>)

RESTORE-SEGMENT (<segid>)

All segments must be explicitly opened prior to data access and closed at the end of a transaction session. The SAVE and RESTORE operators allow for convenient backup and recovery.

Query Equivalence

Before proceeding to the general strategy for mapping queries from one system to the other in this section, a brief review is made of the ability of both languages to view a data base in an equivalent logical fasion. The equivalence is in terms of both data definition as well as manipulation statements.

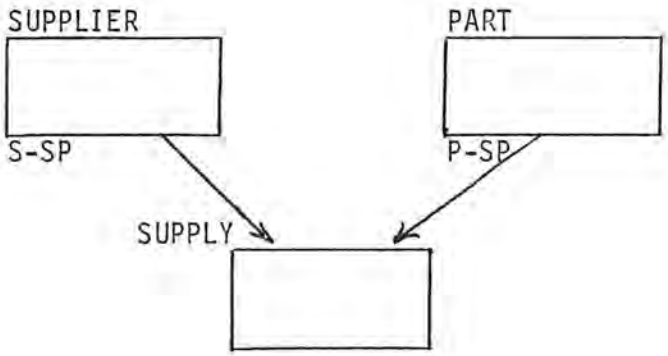
The suppliers-and-parts data model which Codd and Date have used for expounding their views on the Relational model will be used for purposes of illustration.

Data Model:

Relational:

Relation	data fields
SUPPLIER	SNO, SNAME, STATUS, CITY
PART	PNO, PNAME, COLOR, WEIGHT
SUPPLY	SNO, PNO, QTY

Network:



System R (DDL):

Table definition:

```
CREATE
PERMANENT SHARED
TABLE SUPPLIER
SNO     CHAR(5)
SNAME   CHAR(20)
STATUS  DECIMAL(3,3)
CITY    CHAR(15)

CREATE
PERMANENT SHARED
TABLE PART
```

```
    PNO    CHAR(6)
    PNAME  CHAR(20)
    COLOR  CHAR(6)
    WEIGHT DECIMAL(4,4)

    CREATE
    PERMANENT SHARED
    TABLE SUPPLY
    SNO  CHAR(5)
    PNO  CHAR(6)
    QTY  DECIMAL(5,5)
Access paths:
    CREATE
    CLUSTERING LINK
    FROM SUPPLIER (SNO)
    TO SUPPLY (SNO)
    ORDER BY  SUPPLY.PNO  ASC

    CREATE
    CLUSTERING LINK
    FROM PART (PNO)
    TO SUPPLY (PNO)
    ORDER BY SUPPLY.PNO  ASC

    CREATE
    UNIQUE CLUSTERING
    IMAGE  S
    ON SUPPLIER  SUPPLIER.SNO  ASC
```

```
    CREATE
    UNIQUE CLUSTERING
    IMAGE  P
    ON PART  PART.PNO  ASC
```

CODASYL DBTG

Schema Definition:

```
SCHEMA NAME IS SUPPLIERS-AND-PARTS
AREA NAME IS DATA-AREA
RECORD NAME IS SUPPLIER;
    LOCATION MODE IS CALC HASH-SNO
    USING SNO IN SUPPLIER
    DUPLICATES ARE NOT ALLOWED,
WITHIN DATA-AREA.
02 SNO    ;  TYPE IS CHARACTER 5.
02 SNAME  ;  TYPE IS CHARACTER 20.
02 STATUS ;  TYPE IS FIXED DECIMAL 3.
02 CITY   ;  TYPE IS CHARACTER 15.

RECORD NAME IS PART;
    LOCATION MODE IS CALC HASH-PNO
    USING PNO IN PART
    DUPLICATES ARE NOT ALLOWED;
WITHIN DATA-AREA.
02 PNO    ;  TYPE IS CHARACTER 6.
02 PNAME  ;  TYPE IS CHARACTER 20.
02 COLOR  ;  TYPE IS CHARACTER 6.
02 WEIGHT ;  TYPE IS FIXED DECIMAL 4.
```

```
RECORD NAME IS SUPPLY;
WITHIN DATA-AREA.
02 SNO   ;  TYPE IS CHARACTER 5.
02 PNO   ;  TYPE IS CHARACTER 6.
02 QTY   ;  TYPE IS FIXED DECIMAL 5.
```

Access paths:

```
SET NAME IS S-SP.
    MODE IS CHAIN;
    ORDER IS SORTED;
    OWNER IS SUPPLIER;
    MEMBER IS SUPPLY;
        MANDATORY AUTOMATIC;
        ASCENDING KEY IS PNO IN SUPPLY
            DUPLICATES ARE NOT ALLOWED;
        SET OCCURRENCE SELECTION IS THRU
            LOCATION MODE OF OWNER.

SET NAME IS P-SP;
    MODE IS CHAIN;
    ORDER IS SORTED;
    OWNER IS PART;
    MEMBER IS SUPPLY;
        MANDATORY AUTOMATIC;
        ASCENDING KEY IS SNO IN SUPPLY
            DUPLICATES ARE NOT ALLOWED;
        SET OCCURRENCE SELECTION IS THRU
            LOCATION MODE OF OWNER;
```

The data definitions of the two systems provide us access facilities to the data model example which are equivalent in their scope if not identical in the implementational details.

System R:

Three relations have been created as permanent tables.

Facilities for traversal between the relations (SUPPLIER and SUPPLY) and (PARTS and SUPPLY) have been set up through binary links.

Functional mapping for the SUPPLY relation on its connection with the SUPPLIER and PART relation is established in the binary link.

CODASYL DBTG:

The three record types are defined together with keys for the access paths utilized by SUPPLIER and PART record types.

Two set types S-SP and P-SP have been created. Both having SUPPLY as the member record and SUPPLIER and PART respectively as the owner.

Ordering on the member records has been established, with membership being declared as MANDATORY AUTOMATIC and SET OCCURRENCE SELECTION through the location mode of the owner.

The scope of these data definitional facilities are apparent in the examples given below which express equivalent queries on the two data bases.

DML:

#1. Get supplier name and city for supplier 'S4'.

```
DBTG:
    MOVE 'S4' TO SNO IN SUPPLIER
    FIND SUPPLIER RECORD
    GET SUPPLIER; SNAME, SCITY.

System R:
    CALL BIND ('X', ADDR(X));
    CALL BIND ('Y', ADDR(Y));
    CALL SEQUEL (C1, 'SELECT SNAME:X, SCITY:Y FROM SUPPLY WHERE
            SNO  = "S4"');
```

#2. Add 10 to the status value for supplier S4.

```
DBTG:
    MOVE 'S4' TO SNO  IN SUPPLIER
    FIND SUPPLIER RECORD
    GET SUPPLIER; STATUS
    ADD 10 TO STATUS IN SUPPLIER
    MODIFY SUPPLIER; STATUS

System R:
    CALL SEQUEL ('UPDATE SUPPLIER
                  SET STATUS = STATUS + 10
                  WHERE SNO =
                      SELECT SNO
                      FROM SUPPLIER
                      WHERE SNO = "S4"')
```

#3. Create a new supply occurrence for 'S5/P6/7'.

```
DBTG:
    MOVE 'S5' TO SNO IN SP
    MOVE 'P6' TO PNO IN SP
    MOVE 7 to QTY IN SP
    MOVE 'S5' TO SNO IN SUPPLIER
    MOVE 'P6' TO PNO IN PART
    STORE SP

System R:
    CALL SEQUEL ('INSERT INTO SUPPLY:
                <S5,P6,7> ');
```

#4. Find the corresponding owner in another set of current run-unit member.

```
DBTG:
    FIND OWNER IN P-SP OF CURRENT OF S-SP SET.

System R:
    CALL SEQUEL ('SELECT *
                  FROM PART
                  WHERE PNO = PNO OF CURSOR C1 ON SUPPLY');
```

#5. Find the qty. of part P5 supplied by supplier S4.

```
DBTG:
    MOVE 'S4' TO SNO IN SUPPLIER
    FIND SUPPLIER RECORD
    MOVE 'P5' TO PNO IN SUPPLIER
    FIND SUPPLIER VIA CURRENT OF S-SP USING PNO IN SUPPLIER
    IF ERROR-STATUS = 0326 GO TO NON-SUPPLIED
    GET SUPPLIER
```

System R:

```
CALL BIND ('X', ADDR(X))
CALL SEQUEL (C1, 'SELECT QTY:X
                  FROM SUPPLY
                  WHERE SNO = 'S4'
                  AND PNO = 'P5'');
```

Translation Strategy

The general translation strategy will consist of:

1) making available to each of the systems which is effecting a data access to an object system, equivalent system facilities of the object system to provide a simulated data-base transformation. This should be done with the goal of preserving original information such that all operations previously possible may still be performed.

2) providing run time query translation which converts the data management calls into the object systems operators.

CODASYL DBTG to System R:

Schema Declaration: Since most of the schema declaration constructs have been previously analyzed and their equivalent facilities in System R have been identified, the translation mechanism suggested here shall concentrate on the SET construct and its suggested mappings only.

Representation of set types: A set type as has been observed consists of one owner record type and one or more members record types.

Set membership: Since access paths in System R are made available to the RDI as static references, these paths can be declared by setting up binary links between the relations. However, since a binary link can only be established between two relations, multiple binary links would have to be declared if there were more than one member record types in a declared set. In general the facility of multi-membership in sets is not recommended [Nijssen].

DBTG specifies membership in sets through the membership class. Since relations by definition and System R (SEQUEL) operate on set-oriented definitions, the MANUAL class of membership may not be supported in the DDL. Record-type membership in sets is thus restricted to < MANDATORY | OPTIONAL > < AUTOMATIC > eliminating arbitrary user placement strategy.

Ordering: In the CODASYL system the logical position of a member record within a set occurrence may also be considered to be of significance, the ordering being established in two ways.

On the order of insertion of a new entry into the set type (such as when order is declared to be NEXT, PRIOR, FIRST, LAST). Since relation and System R in particular does not contain any access paths containing any logical information other than that derivable from the data values themselves, such an order cannot be supported (see Set Membership MANUAL class).

MANUAL insertion may be simulated, however, by placing the partial burden of set maintenance on the interface program. Records which participate as members of information bearing sets would include the keys of their owner record as one of their data items. The order of insertion for the record in the set occurrence would be maintained by generating sequence numbers as one of the data items in the record.

This technique would maintain the value dependent link requirements of System R and accomplish the DBTG facility for:

manual INSERTS.

allow for Information-bearing sets.

Ordering based on certain sort keys of the member records are declared as the order specifications in the binary links.

Set Occurrence Selection: Access strategies provided by DBTG for selecting a given set occurrence are utilized by the system under the following circumstances.

when a member record of a given set has been declared as an AUTOMATIC member of a specified set, the system uses the selection strategy for its proper placement in the data base. Such a situation is easily handled in System R since all links are automatically maintained by the system and placement of the new tuple is effected in its prespecified ordering.

when a modification to the record key accomplishes an automatic deletion and insertion into the respectively new set occurrence. This is a mere programming trick and is

not supported by System R since it violates the value dependent ordering rule of links. What should be done is to delete the record and insert it afresh after key modification to maintain system consistency.

The last application of SET OCCURRENCE SELECTION is as an access strategy in a FIND statement which we describe as follows:

For each DBTG set declared in the schema, a VIEW definition is entered in System R, the linkage between the relations being established through the use of cursors.

Using the SUPPLIER and PART example SET S-SP would be declared as:

```
DEFINE VIEW S AS
SELECT *
FROM SUPPLIER

DEFINE VIEW S-SP AS
SELECT *
FROM SUPPLY
WHERE SNO = SNO OF CURSOR C1 ON SUPPLIER
ORDER BY SUPPLY.PNO ASC
```

The definitions of S and S-SP call for tuples of SUPPLIER and SUPPLY which have the same SNO as cursor C1; furthermore, they specify that when these views are used, the cursor C1 will be active on SUPPLIER.

These VIEW definitions are parsed and optimized and retained by the system in the form of POP's. During the optimization presence of the physical support (defined binary link) will be utilized for fast access. Thus, providing a direct analogy for a SET traversal mechanism.

Data Manipulation

Retrieving Set Occurrences: If an entire set occurrence is needed, the query is translated into a two-stage retrieval.

Accessing the owner type: The owner view is opened and the cursor positioned on the specific owner record/tuple. Locating the owner record may be:

CALC or DIRECT: in which case the normal System R accessing is done.

VIA set SET: in which case the translator must declare additional views of the records on which the current view would be dependent and the translated query allow for mapping the cursor position to the current view.

The key of the link between the records is expressed as a query.

Traversing Set Types: A set type is traversed by executing a sequence of DML statements of the form:

FIND NEXT (or PRIOR, or LAST, etc.) RECORD OF SET set-name.

The first implication of such traversal statements is that the interface program has generated requisite owner and sequence number keys.

FIND [OWNER | MEMBER] OF CURRENT SET set-name.

Standard traversal involving the current of owner and member records.

The following procedure accomplishes a simulation of such traversals.

the cursor is positioned on the owner record VIEW.

the subordinate VIEW is then opened and either the member record is selected using a key value match or the translator provides the specific sequence number of the record using the current value to accomplish (NEXT, PRIOR, etc.).

GET, MODIFY, REMOVE, INSERT, DELETE, STORE.

The above commands are directly translatable into SEQUEL equivalents and as can be observed in their details, have approximately the same implications in System R as in DBTG. The differences can be easily compensated by a translator program.

System R to CODASYL DBTG

As has been mentioned earlier any strategy for translating queries from System R to CODASYL DBTG must take into account the underlying system support facilities which convert high-level set-oriented SEQUEL statements into lower-level access primitives.

The translation mechanism which interfaces System R to a DBTG system shall make use of these low-level access primitives rather than translating the SEQUEL query itself. The reasons for this become self-evident when we review the query execution process under System R.

System R components: The system comprises two interfaces and their supporting sub-systems respectively:

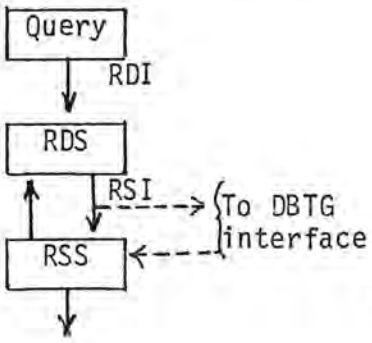
Relational Data Interface (RDI) is the external interface, providing high-level data independent facilities for data retrieval, manipulation, definition and control. It consists of a set of operators including SEQUEL which allows all facilities of the SEQUEL sub-language.

Relational Data System (RDS) is the sub-system which supports the RDI. It contains an 'optimizer' which plans the execution of each RDI command, this is done by choosing a low-cost access path to date from among those provided by the RSS.

Relational Storage Interface (RSI) provides simple tuple at a time operators on base relations. Calls made to the RSI require explicit use of segments (data areas) and access paths called images and links identified appropriately for the RSS.

Relational Storage System (RSS) has functions which can be found in many other systems (relational and non-relational) such as the use of indexes and pointer chain structures appropriately identified and maintained for use by the RDS in choosing an access path and generating its RSI calls.

The flow of a System R query is thus:



dependent on the interrelated system components before being effectively translated by an interface program.

Data Definition facilities.

Most of the system facilities for defining data have been discussed previously and their counterparts in the DBTG system have also been identified. Here the specific clauses will be analyzed together with any limitations arising therefrom.

Representation of Relations: Each relation utilized by the system is specified by the 'create-table' ddl command; this is converted and maintained by the RSS. These relational declarations will correspond to the record types declared in the DBTG schema.

In the DBTG schema the table will generate the following additional features:

LOCATION mode is DIRECT: Relational systems depend on the tuple-id rather than a user defined physical location (CALC) or arbitrary placement (CURRENT OF SET).

WITHIN area: the segment-id associated and maintained by the RSS would be converted to the more descriptive AREA names used by the DBTG system through a conversion scheme.

PRIVACY LOCK: any ASSERT statements declared on the relation would be converted to PRIVACY LOCK statements with ON conditions.

Representation of Access paths: Since the major access path utilized in the DBTG system is the SET declaration. Any translation scheme must be heavily dependent on utilizing this particular system

facility.

Binary links: are used to connect tuples in two relations. Such links can be directly expressed in terms of sets under the DBTG system with the following qualifications.

MODE IS CHAIN: which is the normal implementation mechanism under System R and is efficient in access terms.

ORDER: would be declared if the ORDER BY ord-spec-list has been specified in the LINK.

OWNER/MEMBER: are correspondent to the FROM table-name TO table-name clauses in System R.

SET OCCURRENCE SELECTION: in keeping with the relational viewpoint would be the LOCATION MODE OF OWNER.

MEMBERSHIP: as has been discussed earlier would be OPTIONAL/MANDATORY AUTOMATIC which is the system requirements for links.

Images: are logical reordering of relations providing associative paths in the relation itself. Since the only system path supported under DBTG involving a single record type is a singular or system-owned set, an image would be translated to a system-owned set together with the ordering specifications declared.

The sequential nature of such sets together with their ordering would provide the associative access facility of an image.

The RSS would store these access paths and allow the optimizer to select its access path generating tuple-wise RSI

calls identifying the particular access path being utilized which would correspond to the path available under the DBTG system.

Before the conversion of a SEQUEL query into RSI calls is made. The optimizer function should be briefly reviewed.

Optimizer: attempts to minimize the expected number of pages to be fetched from secondary storage into buffers during execution of a statement. It accomplishes this by classifying the SEQUEL statement into one of several statement types according to the presence of various language features such as JOIN and GROUP BY. Next the optimizer examines the system catalogs to find the set of images and links which are pertinent to a given statement. A rough decision procedure is then executed to derive an expected cost formula for each 'reasonable' access method available and an optimal path selected.

In the translation strategy suggested it is the aim to bias the optimizer towards using access paths which are more naturally available under the DBTG system. This is readily accomplished by heavy utilization of binary links, which would orient the access path decision toward a simulation of network type traversal and retain system efficiency after the access calls are converted into DBTG commands.

Query Translation process: In order to evaluate the feasibility of effectively translating a query we will follow the various constructs utilized by the System R in DBTG terms.

BIND: establishes the variable names into which tuple values are delivered. This would correspond to the UWA of DBTG which is utilized by the sub-schema user.

SEQUEL: expresses the query content itself involving one or more relations. As discussed earlier a SEQUEL query is parsed, analyzed and an optimized access path for it generated by the 'optimizer'. The RDS would utilize this access strategy which is a combination of the DBTG system supported access paths of LOCATION MODE and SET OCCURRENCE SELECTION clauses to position the defined cursors which are essentially the RECORD, set and run-unit currency indicators at the tuple or record-type occurrence.

FETCH: materializes the desired tuple into the variable names specified in the BIND statement which is the UWA, GET.

Any of the SEQUEL dml statements with the exception of those dynamically generating new relations (assignment) have appropriate DBTG commands into which they can be adequately translated (INSERT, DELETE, UPDATE-System R) into (INSERT, STORE, REMOVE, DELETE, MODIFY-DBTG).

Since the interface program which would handle actual translation of System R access primitives expects tuple commands together with support information to navigate in a DBTG environment, the level of interface is logically the RSI operators.

The following RSI operators together with the supporting environment information are available for querying:

FETCH (< segid >, < relid >, <identifier: tid or link id or image id, key values), < field list> ,< pointers to IO locations > [,HOLD].

fetches the specified tuple based on identifier values into the variables (work area) and places a lock if option specified; equivalent to a GET.

OPEN-SCAN (< segid >, < path:relid or image id or link id>, < start-point = key values for image, or tid for link, or scanid for link>) RETURNS (< scanid >)

utilizes information of access path and tuple identifier to retrieve the position of required tuple or set of tuples; DBTG FIND.

NEXT (< segid >, < scanid >, < field list >, < pointers to IO locations > [,< search argument > [,HOLD])

sequential scan of the link members and retrieval into work area, together with optional search arguments and locks--analogous to sequential navigation facilities of DBTG FIND.

INSERT

DELETE

UPDATE

together with requisite segment (AREA) and path (SET) information allow effective data manipulation facilities on tuples which can be translated into DBTG terms without much difficulty. Set maintenance by the system would be managed by explicit operators for links.

CONNECT

DISCONNECT

which though value-dependent would serve adequately in AUTOMATIC category of sets.

From the foregoing it is apparent that translation of SEQUEL (System R) queries into analogous DBTG commands is feasible and can be accomplished with minimal loss of system facilities.

Limitations

In deriving any translation strategy some restrictions must be placed on individual system facilities to allow for an effective mapping of the language constructs to be accomplished. So long as these restrictions do not cause any excessive loss of information, they are acceptable.

Given below are the major limitations which would have to be taken into account in the translation process:

DBTG to System R:

Repeating groups: within records to be discouraged, since it conflicts with basic normalization of data (a conceptual viewpoint of organizing data). Although a translator could split these repeating groups into separate relations, generating owner tuple keys and linking them with binary links to the owner record with clustering specification. Thus accomplishing ready access.

VIRTUAL SOURCE/RESULT: this clause allows data items to be virtually existent, that is, either actually available in some other record type

or the result of a procedure. The underlying concept being saving of storage space, such a mechanism has many system maintenance implications and is not supported under System R.

MANUAL membership: which depends on the physical insertion of records into sets allowing users arbitrary control over placement stretegy is not permitted in a data value dependent relational system.

LOCATION MODE IS CALC: is another instance of utilizing user defined storage address computational procedures which is not permitted under System R. All tuple placement is controlled through the TID which is under system control, any other strategy conflicting with this would create considerable complications in system efficiency.

System R to DBTG:

The major restraint on any translation process in this direction is the lack of support for dynamic generation of new relations. In other words, the system allows sub-schema users to operate on a static data base, any major modifications to the schema being under the control of the DBA.

IV. CONCLUSION

In this paper a mechanism has been presented for interfacing the Relational model to the Network model and vice versa. Throughout the discussion an effort has been made to allow a feasible interface preserving as far as possible the intactness of the model implementations, that is, System R and CODASYL DBTG, respectively.

Since the context of this discussion is within the framework of the above-mentioned implementations certain limitations come to light:

1. Whereas in the DBTG DML direct use of the schema facilities is made in specifying the access paths the user utilizes to navigate within the data base, no such facility is made available to the higher level SEQUEL user. In order to accomplish effective interfacing the level at which the two implementations communicate are therefore disparate:

   DBTG to System R at the host language level (DML).

   System R to DBTG at the access method level (RSI operators).

   This point has been discussed by (Nijssen) when he remarks that the relational language although simpler and more powerful than the Network DDL and DML does not provide any facility for defining an access path. Thus, whereas in the 'relational + access path' a distinct separation between the logical aspects of information and access function is implied, in 'DDL + discipline ' there is no possibility of separating the two aspects.

2. The dynamic nature of the relational model which allows users under System R to generate new relations as the result of a query gets restricted under the DBTG model. Users under DBTG cannot make major changes to the schema during a dynamic interchange with the data base, that is, in a query session.

3. The control over the physical storage which DBTG allows its users falls into the following three statement clauses:

   AREA and WITHIN: which specifies setting up storage areas and allocation record types to be stored in them. Subsequent DML statements require explicit opening of the required areas and associating any data base activity with an area.

   LOCATION MODE: user can have his own predefined hashing procedure which maps a record to its physical storage location.

   MANUAL SETS: which allow user discretion/strategy in including a record type within an entity set relationship (set).

   Under System R users have no control over the data base physical storage. The system decides over the size of segments and the placement strategy of tuples and relations.

   The logical and physical addresses of tuples are controlled by the system through the generation of TID's. Any optimization to be accomplished is done through user-defined clustering specifications. Apart from this the user has no control over the physical address of the tuples.

   Any relationships between entity sets, such as binary links, is value-dependent and, thereafter, automatically maintained by the

system. A user may not arbitrarily include or exclude individual tuples from a physical link once defined and created.

4. The duplication of logical facilities caused by the provision of both sets and repeating groups under DBTG causes unnecessary complications when interfacing with System R. As mentioned earlier, although repeating groups may be simulated under System R by locating them as separate relations they burden the translator with unnecessary maintenance work and may conveniently be excluded.

5. The VIRTUAL/SOURCE clauses available under the DBTG schema perform two functions:

   firstly they provide the data base administrator control over the storage of different data items in different records.

   secondly they provide a certain amount of data independence in the COBOL sub-schema.

   However, the first function should be handled much more comprehensively in the suggested DMCL and the second under the COBOL suc-schema DDL.

   Under a relational system such facilities become redundant because of the dynamic nature of the DDL, although a distant analogy may be suggested between the view facility of System R and the VIRTUAL/SOURCE clauses.

Although the contents of the foregoing paper place certain limitations on the process of communication between the subject DBMS's, it demonstrates the feasibility of both logical and physical mapping on both sides. Until such time as a more standard Data Base Model is accepted as the medium of

implementing large data stores, this non-optimal but practical approach to communicating between different data models remains a necessary if cumbersome reality.

REFERENCES

Astrahan, M.M., et al. (1976) "System R: Relational Approach to Database Management," ACM TODS June 1976, 97-137.

Adiba, M., and C. Delobel. (1977) "The Problem of Cooperation between different D.B.M.S." Proc. IFIP TC2 Jan. 1977, 165-186.

Banerjee, J., and D.K. Hsiao. (1978) "The Use of a Database Machine for supporting Relational Databases," Proc. 5th Wkshp. on Computer Architecture, Aug. 1978.

Banerjee, J., and D.K. Hsiao. (1978) "A methodology for Supporting Existing CODASYL Databases with New Database Machines," Proc. ACM '78 Conf., Dec. 1978.

Bracchi, G., P. Paolini, and G. Pelagatti. (1975) "Data independent descriptions and the DDL specifications," Proc. IFIP TC2 Jan. 1975, 259-267.

Codd, E.F. (1970) "A Relational Model of Data for Large Shared Data Banks," CACM June 1970, 377-387.

Codd, E.F. (1974) "Normalised Data Base Structure: A Brief Tutorial," Proc. ACM SIGFIDET Wkshp. on Data Description Access & Control, 1-17.

Codd, E.F. (1974) "A Data Base Sublanguage Founded on the Relational Calculus," Proc. ACM SIGFIDET Wkshp. on Data Description Access & Control, 35-68.

Codd, E.F., and C.J. Date. (1974). "The Relational and Network Approaches: Comparison of the Application Programming Interfaces." Proc. ACM SIGMOD (R. Rustin, ed.), 83-113.

Codd, E.F., and C.J. Date. (1974) "Interactive Support for Non-Programmers: The Relational and Network Approaches," Proc. ACM SIGMOD (R. Rustin, ed.), 11-41.

Chamberlin, D.D., and R.F. Boyce. (1974) "SEQUEL: A Structured English Query Language," Proc. ACM SIGMOD Wkshp. on Data Description, Access & Control, 249-264.

Date, C.J. (1975) An Introduction to Database Systems, Addison Wesley, Reading, Massachusetts.

Engles, R.W. (1971) "An Analysis of the April 1971 Data Base Task Group Report," Proc. ACM SIGFIDET Wkshp. on Data Description, Access & Control, 69-91.

Nijssen, G.M. (1975) "Set and CODASYL Set or Coset," Proc. IFIP TC2 Jan. 1975, 1070.

Nijssen, G.M. (1974) "Data Structuring in the DDI and Relational Model," Prox. IFIP Working Conf. on Data Base Mgt., Apr. 1974, 363-379.

Kershberg, L., A. Klug, and D.C. Tsichritzis. (1976) "A Taxonomy of Data Models," Systems for Large Data Bases, 43-64.

Kay, M.H. (1975) "As Assessment of the CODASYL DDL for use with a Relational Subschema," Proc. IFIP TC2 Jan. 1975, 199-212.

Martin, J.T. (1975) Computer Data-Base Organization, Prentice Hall, Englewood Cliffs, New Jersey.

Metaxides, A. (1975) "'Information bearing' and 'non-information bearing' sets." Proc. IFIP TC2 Jan. 1975, 363-368.

Schenk, H. (1974) "Implementational Aspects of the CODASYL DBTG Proposal," Data Base Management (Klimbie & Koffeman, eds.), 399-412.

Sibley, E.H. (1974) "On the Equivalence of Data-Based Systems," Proc. ACM SIGMOD (R. Rustin, ed.), 43-76.

Su, S.Y.W. (1976) "Application Program Conversion due to Data Base Changes," Systems for Large Data Bases, 143-157.

Tsichritzis, D.C. (1975) "A Network Framework for Relational Implementation," Proc. IFIP TC2 Jan. 1975, 269-282.

Tsichritzis, D.C., and F. Lochovsky. (1976) "Views on Data," SHARE XLIV, Chicago, Illinois, 51-65.

Whitney, V.K. (1974) "Relational Data Management Implementation Techniques," Proc. ACM SIGFIDET May 1974, 321-345.

Yormark, B. (1976). "The ANSI/X3/SPARC/SGDBMS Architecture," SHARE XLIV, Chicago, Illinois, 1-21.