ABSTRACT

The Data Base Management System is now a well established part of information systems technology, but the many architectures and their plethora of data models are confusing to both the practitioner and researcher. In the past, attempts have been made to compare and contrast some of these systems, but the greatest difficulty arises in seeking a common basis. This paper attempts to show how a generalized data system (GDS), represented by two different models, could form such a basis; it then proposes that data policy definitions can restrict the GDS to a specialized model, such as a relational or DBTG-like model. Finally, it proposes that this concept forms a better basis for data structure design of specific system applications.

INTRODUCTION

The seventies has seen the acceptance of the database management system (DBMS). Commercial systems and research efforts have proliferated, and the subject has become a major conference topic. However, the potential user is still left with most of the questions that first appeared: Which is the best system? Am I locking myself into one technique or implementation method? There have been attempts at explaining similarities and differences in the basic classes of systems, debate on the effectiveness of different data models, and description of the selection and acquisition process, but confusion remains.

Possibly the reason for difficulty is:

1. The topic is complex. DBMS exist, but they are so different that they defy simple comparison. They also run the gamut of size and sophistication.
2. They differ in methodology of data modelling, retaining, and querying, as well as their internal storage.

Testing is expensive: some representative system must be implemented on several DBMS for comparison, or difficult simulations are needed.

Further problems arise in large scale database research and there is need for a common basis as a formalism for describing such systems. Methods of defining the functionality of DBMS include set theory and graph theory constructs. This paper attempts to define such a common basis, and show how it can be used to compare models.

DEVELOPMENT OF A FRAMEWORK

There are at least three distinct levels in an information system: the information and its structure, the data model, and the storage structuring. Obviously, no short paper can cover all three, and we will concentrate on the data model. However, consideration must be given to the information system/data model interface to set the stage for ways to define a good data model with its need to reflect the way data are interrelated, manipulated, and protected.

A data model is a system in which a schema may be defined; the DDLC’s definition language is principally a mechanism for defining the names and attributes of data elements, groupings, and relationships, while the definition of policy (integrity, security, efficiency, etc.) is almost an afterthought.

The information system—Data model interface

There is an important interface between the organization view of information and the data model constructed to represent it. This interface is being investigated by researchers who are attempting to define a process for producing a good data base design given a set of user needs or aspirations. Reference 9 is a survey of current techniques. Complete knowledge of the information system and its data usage characterizes the company. Operating policy, however, summarizes the internal constraints of the organization, and the way that the functional subsystems interact; one author refers to these as operative and directive.

* The authors wish to express their gratitude to the U.S. Army Computer Systems Command who, through grant number DAAG29-76-G-0300 (Title: Problems in the Translation and Standardization of Relational and Network Type Data Base), provided partial support and technical advice.
information structures. Several researchers have recently advocated the collection of transactions as a basis for designing logical data structures.

Our goal, however, is to develop a framework in which to study data models, and to incorporate important parameters of the interface into the data model: Data Utilization and Operating Policy. By incorporating these parameters into the data model framework we expect to examine some classes and:

- Explain subtle differences,
- Explore declarative versus procedural aspects, and
- Characterize their "semantics."

Data architecture—A level concept

A recent paper proposes that there are four abstraction levels for data machines and models. These, in increasing abstraction, are:

1. The Defined and Populated Database: a fully operational data system, with database defined via some definition language.
2. The Database System: it involves no data, but represents a specific system, with its description.
3. The Data Model: the system prior to its application. The class(es) of data structures that may be supported by the system have been fixed, but not used.
4. Data Model Theory: a conceptual or generalized data base management system generator, assumed to be able to support all classes of data models.

These levels form a progression: From one to four is abstraction—from four to one is utilization; each level naturally subsumes or subsets the previous one.

As an example, Level 3 may be a Relational Model, i.e., it can support a relational data system, but no other. Then Level 2 might be the implementation of a payroll data definition; i.e., a definition of those items (and their attributes) with procedures making up a relational payroll system. At Level 1, we see sets of payroll tuples; i.e., entries for specific people.

We shall use this concept as a basis for the paper. However, there are special operations performed in going from one level to another, defined as follows:

- Level 4 to 3: Data Policy Definition.
  In this step, the management and information system designers state the major constraints on the operational system. The resulting data model (or database machine) at Level 3 is restricted: only some classes of data structures are now allowed; some types of operation are restricted, allowing privacy or security (policy) decisions to be stated; some actions are performed automatically, allowing validation and integrity (policy) to be stated.
- Level 3 to 2: Data Operation Definition.

Here, the administration is working with a restricted system in which data structure, some efficiency, and specific policy considerations may be stated: e.g., Data Policy Definition specified a relational system with validation at input, now Data Operation Definition defines a database of 2-tuples involving social security number and name, where the former is a nine digit element. This also involves definition of the procedures for building, maintaining, manipulating and retrieving data.

- Level 2 to 1: Data Population and Utilization.
  Finally the data must be loaded and used.

The data model generator—generalized data system

Level 4 data architecture or system can be viewed as either a theory or a machine: i.e., as either an abstract description of a method, or as a machine implementation of that method. If the concepts of Level 4 can be expressed in set theory, then the "machine" could be either a theoretical or working set processor. In this paper, we discuss two candidates for Level 4 machines, and then show how each may be restricted to Level 3 machines. It is, however, of tantamount importance that these machines be truly general, and this exercise is an attempt to show the need and generality, as well as to illustrate the parts and use of such machines. Obviously, if the Level 4 machine is sufficiently general, it will cover all possible data machines at Level 3 (at least relational, hierarchic, and network models). It may be considered a meta-data model or generalized data system (GDS). Furthermore, the use of such a system provides a framework for data model comparison.

The two models discussed here are:

i. The Functional Model and
ii. The Set Theoretic and Extended Set Processor modified to allow data policy and data operation definition.

THE FUNCTIONAL MODEL OF DATA

Here we consider the Functional Model of Data as a GDS. First, Level 4 structure is presented, then data policy constraints are shown to add form and structure to the model. The constraints are semantic, and allow the Functional Model to be viewed in restricted cases as either relational or DBTG (network) data models.

Level 4 structure

Level 4 is a meta-data level where the Functional Model of data is viewed as a directed graph; its nodes represent sets and its arcs represent total functions. Nodes are either entity sets or value sets. Entity Sets may have any number of incoming or outgoing arcs; Value Sets may have only incoming arcs, because "values" are the ultimate logical
representation of information, so no arcs leave value set nodes. A typical Level 4 Functional Model graph is shown in Figure 1.

The definitional facilities for the Level 4 Functional Model consists of three creation and naming operations for value sets, entity sets, and functional specification of an entity set (i.e., specification of functions whose domain is the entity set).

There are also operations for deleting value sets, entity sets, and functions. The deletion operations have the following side-effects:

- Deletion of an entity set also implies deletion of those functions incident on it (both incoming and outgoing);
- Deletion of a value set also implies deletion of those functions incident on it;
- Deletion of a function does not affect its domain and range sets, but some may become isolated nodes and may no longer be relevant.

Data policy definition

Data policy decisions are of the following types:

- The methodology to be used to obtain the data structures.
- The representation of elements in the nodes (i.e., sets) of the Functional Model graph.
- The logical access mechanisms to be supported at Level 3.

While both information and management policy ramifications may be stated in a declarative fashion (CODASYL DDL), management policies are often enforced by transaction-driven "triggers." To refine these ideas we first address the ramifications of data policy. The most important decision is the choice of data model methodology; this will determine the richness and complexity of the allowable data structures. The methodology adopted in the Functional Model is semantic predication analysis, a process developed to analyze the semantic structure of sentences.

A predication represents a whole sentence: e.g., an assertion, a command, or a question; it may be decomposed into zero, one, or two arguments and a predicate. Arguments may themselves be predications. "Downgraded predications" may qualify arguments (the semantic equivalent of adjectival clauses) or may modify predicates (the semantic equivalent of adverbial clauses). The lowest semantic level consists of semantic features which serve as atomic semantic description units. Downgraded predications play the role of semantic features of the arguments or predicates that they qualify or modify.

Here we assume that a specification of data interrelationships is available. The information analyst's role is to obtain the corresponding predication structures and map these to the Functional Model (an abstraction process).

Consider the statement:

"Companies supply parts to departments in some volume"

The predication structure is shown in Figure 2, where the main predication structure "companies supply parts" is represented by the predication $PN_1$ with arguments $A_1$ (COMPANIES), $A_2$ (PARTS) and predicate $P_1$ (SUPPLY). The arrow under SUPPLY denotes the direction of the relationship represented by $PN_1$; it corresponds to the active voice of $P_1$. $PN_2$ and $PN_3$ are downgraded modifying predications representing the indirect object and adverbial
information, respectively. The X’s refer to PI’, which their corresponding predications modify. Semantic features are not present in this example, but correspond to descriptions of the arguments (COMPANIES, PARTS, DEPARTMENTS and VOLUME). For example, DEPARTMENTS might be characterized by NAME, ADDRESS, and NUMBER.

The choice of the abstraction used to map predication structures to Functional Model data structures is part of data policy. As an example, the model might be restricted as follows:

- Semantic features map to functions whose range sets are value sets.
- Arguments corresponding to “real-world” entities map to named argument sets.
- Predications map to named predication sets, and the arcs pointing to arguments become named functions. Also the predicate and its arrow are attached to the predication set.
- Downgraded predications are represented by functions whose domain is the main predication set and range is either an argument set or a value set.

Figure 3 depicts the Functional Model data structure based on the predication structure of Figure 2 and the above abstraction rules.

Thus the choice of the methodology used to model the organization is one aspect of data policy which induces structure in passing from Level 4 to Level 3. At Level 3 the Functional Model has entity sets classified as either predication sets or argument sets (denoted P and A, respectively) and value sets remain unchanged. Functions perform two roles: a function may provide information about a predication structure or it may represent a semantic feature. The data structures supported by the predication analysis and abstraction process are depicted in Figure 4. All these data structures are possible for the Functional Model, but most applications use cases b, c, and d. If we restrict further:

- Functions emanating from a predication set cannot have predication sets as ranges, then only cases c and d are admissible. This is precisely the case in the Entity-Relationship model, where predication and argument sets are called relationship sets and entity sets, respectively. In the Functional Model arcs all represent total functions, whereas in the Entity-Relationship model arcs are either 1:N mappings (for entity sets to relationship sets) or functions (for entity sets to value sets).

So far, we have only considered the methodology for obtaining data structures. Although the set types play an important role, the function types are important in expressing logical access mechanisms. Consider a binary relation α from set A to set B, and its representation, Rα, as the set of ordered pairs

\[ R_\alpha = \{(a, b) \mid a \in A \land b \in B \land a \rho b\} \]

Obviously, there exists an “inclusion” function, i, which assigns an element (a, b) of Rα to its corresponding element (a, b) of AxB, the Cartesian product of A and B. In addition, there exist functions f: Rα→A and g: Rα→B such that the diagram of Figure 5 “commutes.” In terms of the Functional Model predication structures, the binary relation α corresponds to a predication structure consisting of a predication node (labelled “α”), argument nodes (A and B) and arcs (f and g) (see Figure 5). The actual representation of the elements in the predication set is by means of the set Rα.

The functionality types of f and g are important in modeling the semantics of α: i.e., whether α is a relation, a partial function, or a total function. There are sixteen possible configurations for the predication structure, because f and g may be one-to-one, onto, both of these, or none of these. The most important configurations are
summarized in the following:

**Fact:**
Let \( \alpha \) be a binary relation from A to B with functional specification \( f \) and \( g \).
Then:

1. If neither \( f \) nor \( g \) are one-to-one (1:1), then \( \alpha \) is a relation;
2. If \( f \) is 1:1, then \( \alpha \) is a partial function;
3. If \( f \) is 1:1 and onto, then \( \alpha \) is a total function;
4. If \( f \) is 1:1 and onto, while \( g \) is 1:1, then \( \alpha \) is a 1:1 function;
5. If \( f \) is 1:1 and onto, while \( g \) is onto, then \( \alpha \) is an onto function;
6. If \( f \) and \( g \) are both 1:1 and onto, then \( \alpha \) is also 1:1 and onto.

The functionality type of \( f \) determines whether \( \alpha \) is a relation, a partial function, or a total function. If \( \alpha \) is a function, then the functionality of \( g \) determines whether \( \alpha \) is one-to-one, onto, a one-to-one correspondence, or none of these.

In terms of predication structures, a predication set may thus represent a relation or a function, which implies that, in the latter case, \( \alpha \) may be represented by an arc. This is indeed true, but it must be considered a Data Policy decision.

One-to-one functions play an important role in accessing a particular element of a set. If the function \( f \) in Figure 5 is one-to-one, a particular \( a \in A \) will participate in (at most) one ordered pair \((a,b) \in R_a\), so that the second element of the ordered pair may be obtained by evaluating the function \( g \). Thus, any one-to-one function outgoing from a set is a candidate (key) for accessing the elements of the set.

The notion of logical access can be extended to predication sets involving more than two argument sets and various value sets (as in Figures 3 and 4). In this case, the predication node represents an n-arity relationship, where each element may be viewed as an n-tuple of entities and values. For n-ary predication sets, a composite key is precisely the concept in Reference 22.

Finally we illustrate a management policy constraint whose predication structure has another predication node as an argument (e.g., Figure 4(f)). The operational constraint "A company must supply at least three parts to some department during a quarter to remain a valid supplier." could be modelled by a predication structure consisting of a "must" predication set with argument sets COMPANY and SUPPLY (a Figure 3 predication set), and semantic features (functions to value sets) TOTAL-PARTS and QUARTER. Elements of the "must" set might be updated by program on the occurrence of a SUPPLY transaction, but the validation would probably take place at the end of each operating quarter.

### The relational model of data

The data policy constraints on the Functional Model predication structures which transform it into the Relational Model are:

- Value sets are *domains*;
- Argument sets and predication sets are *relations* whose elements are represented by *tuples* of values from domains;
- Functions whose range sets are value sets become the *attribute names* of their respective relations;
- Functions from predication sets to argument sets are replaced by the attribute name corresponding to the key (perhaps the composite key) of the argument set.

The consequence of these restrictions is to allow only nodes which represent Level 4 entity sets as relations: represented by tuples. The arcs can now only point away from nodes, and their functions are the names of the value sets of each element of the tuple: see Figure 6.

### The DBTG data model

The Functional Model can also be transformed into DBTG type Data Structures by data policy definitions. First, the record will be simplified by ignoring repeating groups—in this case a record is a tuple, and may be represented in the same way as the relational data structure. The DBTG-set, in its simplest form, is a representation of a functional predication (i.e., a predication which is a function). The arrow of the function name is in the opposite direction to the arrow in the equivalent "Bachman" Diagram. For a more complicated DBTG-set structure, there is a predication (a record) between two other relations (also records). This represents an intermediate or link record between two others; these structures have been discussed as linking records between two relations in Reference 25 and termed "associations" in Reference 26.

Whether the model transforms a function name into a DBTG-set or follows the Predication (Record) name model is a data policy decision: the same results may be obtained provided that one record is functionally dependent on the other—but not if the relation is N to M.

The insertion property of a DBTG-set is either AUTOMATIC or MANUAL. Which set has which property is defined at Level 3, but the ability to define these properties of a function is a Data Policy decision, which delimits the Level 4 to 3 structure and defines the DBTG Model. Similarly, the deletion properties (MANDATORY and OP-
a) Relational Data Structures

b) DBTG-like Data Structures

Fig. 6—Level 3 data structures in the functional model

TIONS) are Level 4 to 3 data policy decisions which define the operation of a DBTG model. These two properties are therefore declaratives associated with the arcs, similar to the functionality types in a functional model, but obviously having stronger effect. The enforcement of this policy is, of course, procedural.

Comparisons

It is useful to compare and contrast relational and DBTG data policy operations. Relational systems like System R and INGRES use the theory of normal forms to provide good data structures and then utilize these simple structures.
through data-name linkages and data operations like JOIN and PROJECT; they apply other data policy such as consistency, integrity, and validation through system modification of the user statement (INGRES), which is immediately initiated, or through system triggers (SYSTEM R) which are transaction driven, but may be delayed. These two implementations differ in the fact that INGRES specifies policy in declaratives with procedural enforcement, while SYSTEM R is procedural both in declaration and enforcement.

The DBTG data policies are both declarative and procedural: some, such as AUTOMATIC, are declarative and apply a primitive function on operation implying a "semantics" at storage of a record; others, the Data Base Procedures, are procedural and are invoked by some trigger: e.g., on update (validation), privacy and security checking.

THE SET THEORETIC DATA MODEL

The set theoretic processor owes its existence to the concept of an extended set. The extended set consists of elements which themselves may be conventional sets, sequences, ordered sets, atoms, or even extended sets, but each element is identified by a position-identifier (numeric or mnemonic). The extended set is enclosed in square brackets [ ] while the ordered sets or sequences are in angle brackets ( ). Thus if X is an extended set consisting of elements Y and Z in positions 1 and NEXT, we have:

\[ X = [(1,Y), (\text{NEXT},Z)] \]

If we use braces { } to enclose sets, then if Y is the set of the first three integers and Z is the four-tuple consisting of "0,1,0,2" (in order), then:

\[ Y = \{1,2,3\} = [(\#,1), (\#,2), (\#,3)] \]

and

\[ Z = \{0,1,0,2\} = [(1,0), (2,1), (3,0), (4,2)] \].

Reference 16 shows that the extended set is a normal extension of set theory and that predicate calculus operations can be defined on the extended set. All normal set operations (union, intersection, difference, etc.) can also be defined, except that operations are position dependent.

Level 4 structure

The model consists of the following objects or elements: Atoms, which represent a number or character string; Sets, composed of any object; Ordered sets or sequences, composed of any object; Extended sets, composed of any object. Of course, no object may contain itself. The other model objects may all be represented as extended sets, position identifiers (which are atoms), and atoms. Commas are used to delimit objects in a sequence or set.

The basic language of the Level 4 data model consists of: predicate calculus expressions; algebraic expressions; assignment (storage) statements; retrieval expressions, with predicates to limit the response; and macros which simplify operations (e.g., Average, Sum).

In order to illustrate the operations of the extended set processor (XSP) at Level 4, a series of operations is given in Figure 7. The first ten operations are all of the "storage by assignment" type: they store ten extended sets, with synonyms (names) ME, YOU, etc. The VALUE function invokes a "copy" operation, which does not necessarily duplicate the string: the result may be represented internally by pointers. The set AUTHORS is therefore made up of the two extended sets ME and YOU.

The assignment operation for X defines a new extended set: it is a simple set containing the names of the two authors of this paper. This particular operation extracts (from the set AUTHORS) the values which have position indicator NAME. Furthermore, the summation operation (SUM) counts the elements (two) of this newly defined set X. The keyword LIST in the find instruction of Figure 7 is used to denote a set of 0 to "N" replications. The names and symbols which have not yet been described, such as PHONE, NAME, AUTHOR-TYPE and PICTURE, have no semantic meaning at Level 4. Later, some will have semantics in defining policy, but here they are merely symbols. At Level 4 the XSP is unrestricted. It will store, maintain, retrieve, or manipulate the whole or any part of any extended set. However, the XSP must have operations which can be triggered by events or conditions; these operations, applied in going from Level 4 to Level 3, are the Data Policy definitions.

Data policy definition

There are two types of XSP system applied constraints: static and dynamic. The static constraint is one that must always be satisfied: e.g., the data structure class must be hierarchic, or the access to some parts of the database are password protected. Such constraints imply a system action whenever violation occurs, and these may be implemented as special system actions ("compiled into the system") or as interpreted actions, with well defined error response. Most current systems "compile" these constraints (i.e., they "do not support other models of data" or "allow the following types of data protection . . .").

The dynamic constraint is one that is satisfied at specific times or for special operations; e.g., the validation of an element is only to be performed at input, or security is to be checked against type of user and type of operation for every access. These constraints are essentially interpreted.

The relational model of data

An example of the definition of allowable data structure classes for a relational model is given in the restrictions of Figure 8. This definition states that all sets are made up of ordered sets which contain "attribute-value" pairs (e.g., the elements ME and YOU in Figure 7 are ordered sets of three pairs). Moreover, the position identifiers shall be
ME = [NAME, SIBLEY>, <SS#, 017|32|7992>, <PHONE, (301) 262-7138>]
YOU = [ <PHONE, (301) 937-7726>, <NAME, KERSCHBERG>, <SS#, 294|36|4321>]
IT = [ <TITLE, DATA ARCHITECTURE ETC.>, <AUTHOR, {VALUE(ME), VALUE(YOU)}>, <PHONE, (301) 262-7138>]
AUTHOR-TYPE = [NAME-TYPE, PHONE-TYPE, SS#-TYPE]
NAME-TYPE = {PICTURE X(25 MAX)}
PHONE-TYPE = {PICTURE '(' '999')'999'-'9999}
SS#-TYPE = {PICTURE 999|'99'|'9999}
ARTICLE-TYPE = [<1, TITLE-TYPE>]
TITLE-TYPE = {PICTURE X (UNRESTRICTED)}
AUTHORS = {VALUE(ME), VALUE(YOU)}
PRINT (NAME-TYPE)
X = {NAME | SET = AUTHORS}
Y = SUM(X)
Z = SUM ({AUTHOR | SET = IT})
X1 = {NAME | SET = AUTHORS AND SS# NOT='017|32|7992'}
AUTHORSHIP-TYPE = [AUTHOR, LIST (AUTHOR-TYPE), ARTICLE, ARTICLE-TYPE]
AUTHORSHIP = {VALUE(IT)}

Figure 7—Some operations of an XSP at Level 4
Data Architecture and Data Model Considerations

POLICY RESTRICTIONS OF XSP.

OBJECTS: ATOM, ORDERED-SET, SET.
MEMBERSHIP: MEMBER (ORDERED-SET) IS <POSITION-ID, ATOM>,
MEMBER (SET) IS <#, ORDERED-SET>.
CONSTRAINT: ALL POSITION-ID (ORDERED-SET) IS MEMBER (POSITION-ID
(ORDERED-SET-DEFINITION)).

LEVEL 3: DEFINITION WITHIN RESTRICTION.

SET OBJECT. AUTHORS.
ORDERED-SET OBJECT. AUTHOR-TYPE.
ATOM OBJECT. NAME-TYPE, PHONE-TYPE, SS#-TYPE.

TIMING.

APPLY PHONE-TYPE, SS#-TYPE ON INPUT.
APPLY NAME-TYPE ON OUTPUT, INPUT.
APPLY AUTHOR-TYPE ALWAYS.

An Invalid DEFINITION would be:

SET OBJECT. AUTHORSHIP.
ORDERED-SET OBJECT. AUTHORSHIP-TYPE.

... (See Figure 4.1: this is not a first Normal Form definition)

Figure 8— Policy statements and level 3 definition for an XSP: Example 1: Relational model
found within a definition. Thus, if we consider the Level 3 definitions of Figure 8: AUTHOR-TYPE defines the position identifiers (NAME, PHONE, SS#), then ME and YOU conform to this type of ordered set. Moreover, the set AUTHORS in Figure 7 now conforms to the definition AUTHORS in Figure 8.

It is now obvious that Figure 7 contains definitions of extended sets which can apply (or be applied) with semantics at Level 3. We class the elements by statements in Figure 8 which show the definition that AUTHORS contains elements (tuples) which comply with the (Figure 7) AUTHOR-TYPE definition, while the (validation) criteria of the atoms SS#-TYPE, etc., are applied on input, with NAME-TYPE checking also on output. Moreover, because some extended set operations might allow generation of invalid sets during valid operations, the AUTHOR-TYPE criterion is applied on all operations (this may be a duplicative statement, depending on the overall constraint . . . ALL POSITION-ID . . . etc., being universally applied, or "compiled into the system").

If we gave XSP, the definition for AUTHORSHIP-TYPE, an error must occur, because the definition in Figure 7 allows non-atomic elements in the ordered set (due to LIST).

The DBTG data model

In dealing with the definition of a DBTG-like structure, one is faced with some prior decisions. Using the definition of Reference 25, a DBTG-set consisting of an owner record \((A_n)\) and three member records \((B_{j1}, B_{j2}', B_{j3}')\), in that order, will be represented as

\[(A_n, (B_{j1}, B_{j2}', B_{j3}'))\]

The restrictions imposed in an earlier section on DBTG-records is applied here also: no repeating groups are allowed. Thus the definition of a record in Figure 9 follows that of an ordered set in Figure 8. The definitions therefore are special only in their inclusion of membership in DBTG-sets.

Data Policy is represented by the ability to have AUTOMATIC operation of DBTG-set inclusion on storing a predefined record.

Comparisons

It has been suggested that the implementation of a relational structure in a DBTG system is a matter of:

1. Only allowing system owned sets (DBTG termed this a "SINGULAR SET");
2. Removing concepts of database procedures, inclusion and deletion properties (AUTOMATIC, MANDATORY, etc);
3. Making keys unique (using CALC with a DUPLICATES NOT ALLOWED clause);
4. Allowing new macros like JOIN and PROJECTION on (mathematical) sets of records.

It will be seen that the definitions do not truly reflect the first requirement, and that the second can be considered a triggered or system applied procedure (the result of an operation depending on the functional properties of the DBTG-set, etc.). Thus the restrictions are not correct. By further refining the models, it should be possible to determine true similarities and differences. However, it is first necessary to add the operations and their mappings—a non-trivial task.

CONCLUSIONS

Work with two GDS (Functional and Extended Set Models) leads us to the following conclusions:

1. The model of an enterprise information structure may be defined independently of the GDS used for its implementation.
2. If the GDS is "universal" it may store any information structure (both data syntax and semantics) in terms of its primitives.
3. The specialization of the GDS to a data model like that supported by most current research and commercial DBMS involves Data Policy definition. Using this, a GDS may be restricted to perform as one or more specific data models.
4. The process of mapping from an information structure within a GDS to a data structure within a traditional
DBMS follows the restrictions of the Data Policy definition. Consequently there must be a correspondence between Data Policy restriction and data modeling (i.e., passing from information structure to data structure). This process is diagrammed in Figure 10. Most information analysts already have a specialized data model (e.g., DBTG systems) in mind when constructing their information model. Thus, they take the Data Modeling route in Figure 10. We suggest that the correct (more general) process is to express conceptual information struc-

**Figure 10—The parallelism of data policy and modelling**

<table>
<thead>
<tr>
<th>SYSTEM LEVEL</th>
<th>DATA SYSTEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>DATA MODEL GENERATOR (Generalized Data System)</td>
</tr>
<tr>
<td></td>
<td>Examples:</td>
</tr>
<tr>
<td></td>
<td>Functional Model</td>
</tr>
<tr>
<td></td>
<td>Extended Set Model</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DATA POLICY DEFINITION</th>
<th>INFORMATION MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Examples: Infological Synglish (ref. 29) Conceptual Schema</td>
</tr>
<tr>
<td>Specialized Data Model</td>
<td>Data Representation</td>
</tr>
<tr>
<td>Examples:</td>
<td>RELATIONAL SYSTEM</td>
</tr>
<tr>
<td></td>
<td>HIERARCHIC SYSTEM</td>
</tr>
<tr>
<td></td>
<td>DBTG SYSTEM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Special Usage of Special System</th>
<th>DATA OPERATION DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example: A personnel implementation in a Relational System</td>
<td>DATA POPULATION AND UTILIZATION</td>
</tr>
</tbody>
</table>

| 1 | Populated Database for the Specific Usage |
tures in the GDS and to 'tune' the information structures to data structures by means of Data Policy decisions. In other words, it is advantageous to represent information at Level 4 so that all semantics of the information are retained.

Data Policy definitions impose added structure (with restrictions) on allowable data structures. Tradeoffs will then make it easier to consider the losses in representation of information structures in the supported data structures.

REFERENCES