A FOUNDATION
for the
ENTITY RELATIONSHIP MODEL:
WHY and HOW

Paul C Gilmore

Technical Report 87-10
April 1987

Submitted to the 6th International Conference on
the Entity-Relationship Approach
November 9-11, 1987. New York

The research reported on in this paper has been supported by grants of the Natural Sciences and Engineering Research Council of Canada

Introduction

The appealing intuitive nature of the ER model has led to its wide use, but little has been done to provide a foundation for it as soundly based as the one offered by Codd for the relational model [Codd70]. For example, there are several query languages available for the relational model [Codd72] based on a foundation of relational algebra or mathematical logic, while no comparable language has been provided for the ER data model, except perhaps in the system ZIM [Full84].

Despite this lack of foundation it is possible to make an informal comparison of the relative strengths and weaknesses of the ER model with other models, reinforcing and adding to the points made in [Chen76,77]. The hierarchical and network models are low level in the sense that they are abstract descriptions of implemented data structures that store data. The relational model is a higher level model in that it provides abstract descriptions of the data as it is presented to the users in the form of tables. The relational model has freed users from most implementation concerns, but it forces users to be concerned with the presentation of data at the earliest stages of database design when a higher level conceptual view of information needs is more appropriate. Consequences of this insufficiently abstract view of information are evident in the unnecessarily complex normalization methods of table schema design used to ensure the absence of update anomalies, and also in the inability of the relational model to deal with referential integrity in a uniformly simple fashion. [Gil87a] The ER model has found a way to avoid both the excessive implementation concerns of the hierarchical and network models, and the restrictive presentation concerns of the relational model. The strength of the ER model is that it is neither implementation nor presentation oriented, but rather conceptually oriented, and therefore capable of taking the points of view of the other models when necessary.

But the lack of a sound foundation for the ER model hinders several essential developments:

1. The unified view of data proposed in [Chen76].
2. A more disciplined ER modelling process.
3. The description of a consistent data dictionary able to integrate schema specifications and data into one database.
4. A fully unified model of an enterprise that at the same time can give a conceptual view of the enterprise, a user's view of data as it is presented, a data administrator's view of data as it is stored, and a programmer's view of the processing of the data.
5. The application of the model to the developing theory of knowledge base systems.

In this paper the importance of each of these developments will be argued in sections 1 through 5. At the same time a sketch will be given as to how the SET model [Gil87a,87b], a formalized

semantic model based on the ER approach, can support these developments. The SET model is a semantic model in the same sense that the relational model as presented in [Codd70] is semantic. That is, from the point of view of mathematical logic, the model is a model of an unspecified theory. For the relational model, that theory has been formalized in [Reit84] with the purpose of providing a sometimes useful deductive form of the model. The SET model can also be provided with a deductive form based on the set theories described in [Gil86].

The object-oriented database system described in [LyKe86] can be seen as an alternative way of providing a foundation for the ER model. That paper, however, addresses only (4), and in particular ignores the consistency concerns raised in (3).

1. A Unified View of Data

For the unified view of data proposed in [Chen76] to be fully achieved, a database is needed that is capable of recording a high level conceptual model of an enterprise and at the same time of providing the tables for a relational database schema as a defined user view in its specification/query language. Such a management system will avoid the hand translation process that intervenes between the high level and the user table view of an enterprise [Chen77, HaMc81, Rock81, Kent83, Chen85, TYF86]. As the enterprise's information needs evolve, such a hand translation will inevitably result in either the conceptual model or the tables being out of date, or very likely the conceptual model, with all its valuable high level information, just being discarded. The SET model integrates the two stages.[Gil87a,b] In the first stage, sets are declared and recorded in a **set schema** that is a conceptual model for the enterprise; the tables needed for the second stage are just user views of the enterprise defined within the specification/query language DEFINE. As the enterprise's information needs evolve, the model is updated by the declaration of new sets, or by the removal of sets from the set schema. The users' tables are then either automatically updated or if necessary redefined. As illustrated in [Gil87a], much of the design and definition of the tables can be automated.

The SET model is based on the concept of set or class, one of the most fundamental concepts of mathematics. The concept has been incorporated into scientific and natural languages, as well as used extensively in database theory and practice. It is explicitly used in the relational and ER models, but also in the entity set [SAAF73], semantic [HaMc81], and functional data models [Ship81, LyKe86], in the system TAXIS [MyWo80], in the techniques described in [FuNe86], and implicitly used in the network data model [TaFr76]. But the concept of set used in these cited papers is an intuitive one and is combined with other related but independent concepts. The SET

model, on the other hand, uses set and ordered pair as its only fundamental concepts, with the mathematical foundation for these concepts being provided by the provably consistent set theories of [Gil86], while other needed concepts are defined in terms of these.

A distinctly computer science view of sets is taken in the SET model. No set can be presumed to preexist; each required set must be explicitly declared. All the declared sets of an enterprise form a set schema for the enterprise. A set has an **intension** and an **extension**. The intension of a set is the property that an object must possess to be a member of the set; the extension of a set is the collection of objects satisfying the intension of the set. The intension of a set is to be thought of as time invariant, while the extension of a set changes as circumstances change. The intension of a set may be expressed in a natural language statement intended for human understanding only (the **base** sets), or may be expressed in a language like DEFINE that can be understood by both humans and machines (the **defined** sets). For example, the base set E of current employees of an enterprise would have "the set of current employees" as its intension. It would be formally declared in DEFINE as

E={ E || the set of current employees }.

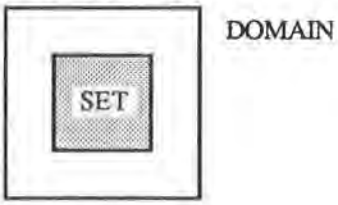
The "type" or ultimate value set INT, the set of all integers recognized by the management system supporting the declarations, is on the other hand a defined set that would be declared as follows:

INT={ x:INT | x system defined | the set of integers }.

Here the intension of the set "x system defined" is an assertion of DEFINE that can be read and understood by both humans and machines. The phrase "the set of integers" appearing after the second | is, like the intension of E, a comment that need only be understood by humans.

The extension of a set is drawn from its **domain** as illustrated in figure 1.1.

FIGURE 1.1



The domain of a set S is the extension of the cartesian product of one or more previously declared sets. To avoid an infinite regression of domains, it is necessary to assume that some sets are declared without domains; these are the **primitive** sets.

The two sets E and STR are examples of primitive sets, for there is no preexisting set from which their extensions are drawn. The fact that they are primitive is indicated by the repetition of their name in their domain declaration, the portion of the set declaration following { that appears before the first |. The fact that INT is a defined set can be inferred from the fact that its domain declaration also declares the variable 'x' used in the intension of the set.

Another example of a defined set is given by the value set VE# from which employee numbers might be drawn:
VE#= [x:INT | $1000 \leq x \leq 9999$ | a value set for employee numbers }.
Here it is assumed that $\leq$ is the less than or equal association on the integers, and that it is system defined. VE# is not a primitive set since its domain INT has been previously declared.

The ER model distinguishes between sets like E and VE#, and relationships between them, such as the one that associates an employee number with an employee. Since this association is just a set of pairs, it is declared as a set in the SET model:
E#= { ExVE# | <1,1>, <0,1> | associates a unique member of VE# with each member of E }.
E# is a nonprimitive base set. It is nonprimitive since its domain ExVE# is the cartesian product of two previously declared sets E and VE#; these sets are called the **immediate domain predecessors** of E#. It is base since its intension is expressed as an informal comment.

The advantage of treating E# as a set in exactly the same fashion as the sets E and VE# is that the domain integrity constraints implied by the domain declarations can be maintained in a uniform fashion. It is therefore never necessary to know at the time of its declaration whether a set is an entity set or an association.

The machine readable portion "<1,1>, <0,1>" of the declaration of E declares the **degrees** of the base set E#, with the first of the pairs being the degrees of E# on E and the second being the degrees of E# on VE#.

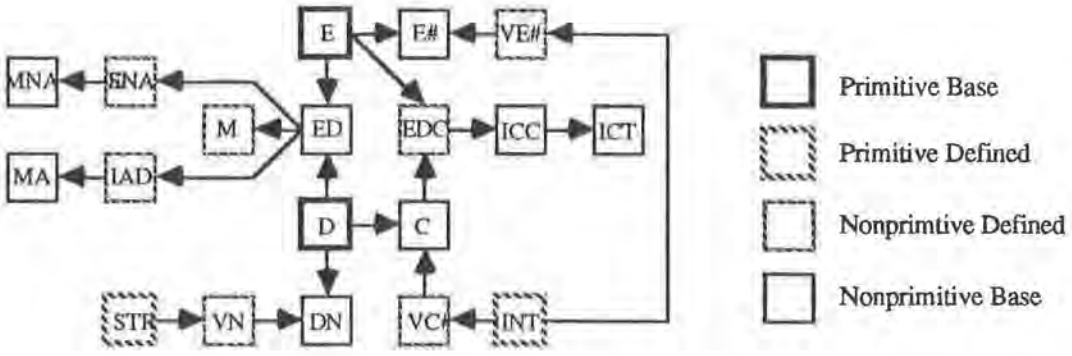
The first element of a pair of degrees is the **lower degree** and may be 0 or 1, while the second is the **upper degree** and may be 1 or *. The lower degree of the first pair <1,1> of degrees states that for every member of E there is at least one member of VE# associated with it; that is, each employee must have an emplyee number. The upper degree of the first pair states that with a member of E no more than one member of VE# can be associated; that is, each employee must have at most one employee number. The constraints expressed by the first pair <1,1> of degrees ensures, therefore, that each employee has a single employee number.

The lower degree 0 of the second pair <0,1> of degrees expresses that not every member of VE# need be associated with a member of E; that is, not every member of VE# is necessarily an employee number. The upper degree 1 of the second pair expresses that a member of VE# can be associated with at most one member of E; that is, the same employee number is never assigned to two different employees. The constraints expressed by the second pair <0,1> of degrees ensures, therefore, that if a member of VE# is used as an employee number, then it is used as the number for a single employee. The constraints expressed by the two pairs of degrees <1,1> and <0,1> ensures that each employee has a single number that is unique to the employee.

The need for degrees has been widely recognized. In [Knt83a, LyKe86] they are called respectively the least and maximum participation, and in [LeSa83] the minimum and maximum cardinalities. In [LyKe86] the upper degree * is denoted by m. In [TYF86] the upper degrees are referred to as the cardinalities of the connectivity of a relationship, while the lower degrees are described as optional or mandatory connectivity.

An entity-relationship diagram for an enterprise has aways been stressed as an important part of the ER model for the enterprise. The comparable diagram for the SET model is an illustration of the domain graph for a set schema. In the next figure an illustration for the domain graph for the set schema of Simple University is taken from [Gil87a].

FIGURE 1.2



In this figure each box corresponds to one set declared in the set schema. For example, E, E#, and VE#, each labels a box at the top of the figure, and INT labels a box at the bottom right. Each of the boxes represents a node of a directed graph, with an edge directed from an immediate domain predecessor of a set, to the set. For example, arrows are directed from E and VE# to E# because

they are the immediate domain predecessors of E#, and an arrow is directed from INT to VE# because it is the domain predecessor of VE#.

An illustration of a domain graph, such as the one shown in figure 1.2 has features in common with data structure diagrams [Bach69], as well as ER diagrams.

The illustration of a domain graph is never a substitute for the declarations of sets in a set schema, but only provides an overview of the declarations. They are of course a very useful overview, but there is much information contained in the declaration of a set that is not represented in an illustration of the domain graph.

Each edge of a domain graph has a pair of degrees associated with it. If the edge is directed towards a base set such as E#, then the degrees stated in the declaration of the set can be associated with the edges. If the edge is directed towards a defined set such as VE#, then no degrees are declared for it, but the degrees can in general be calculated.

The domain graph of a set schema, with degrees associated with its edges, plays a central role in the domain graph method of table design described in [Gil87a]. The method automatically constructs from such a domain graph a table schema that can present all the data of the set schema without anomalies of any kind. Each table in the schema is a defined set in exactly the same sense that VE# is a defined set. For example, from the domain graph of figure 1.2, the following table schema is constructed:

TE={ u:VE#, v:VN | [**For some** x:E, y:D] (<x,u>:E# **and** <x,y>:ED **and** <y,v>:DN) | },
TD={ v:VN, u:VE# | [**For some** x:E, y:D] (<x,u>:E# **and** <x,y>:M **and** <y,v>:DN) | }, and
TC= { v:VN, w:VC# | [**For some** y:D] (<w,y>:C **and** <y,v>:D) | }
TICC={ u:VE#, v:VN, w:VC# | [**For some** x:E, y:D] (<x,u>:E# **and** <y,v>:DN **and** <x,<y,w>>:ICC) | }, and
TICT={ u:VE#, v:VN, w:VC# | [**For some** x:E, y:D] (<x,u>:E# **and** <y,v>:DN **and** <x,<y,w>>:ICT) | }.

The only significant human decision in the calculation of this schema was the calculation of degrees for defined sets. The domain graph method of table design that created these tables can be regarded as a generalization of the synthetic method of design described in [Bern76].

The SET model therefore provides the fully unified view proposed in [Chen76]. In as much as the

SET model is object-oriented in the sense of [Ditt86], at the same time it contributes to the requirement for a high quality database design method called for there.

Since queries are simply a request for the listing of the membership of a defined set, the language DEFINE can be used as a query language by declaring defined sets like those declared for the table schema.

2. Guiding Principles for ER Modelling

The process of constructing an ER model for an enterprise is a difficult exercise requiring an ability to abstract from a large amount of information of varying quality the essential entities of the enterprise. In constrast to the construction of a table schema from a set schema, the process makes use of more art than science. It will never be possible to remove the art entirely from the process, but it is possible to provide guiding principles that reduce the art.

It is evident from the domain graph illustrated in figure 1.1 that the primitive sets of a set schema are fundamental to the schema. The only entities about which information can be recorded for a database are the members of primitive sets, and tuples or nested tuples of such members. Since the primitive value sets are assumed to have been declared and be available before an information needs analysis is begun, their choice is out of the hands of an analyst. But the choice of the primitive base sets is entirely in the hands of the analyst, and the success of an ER model for an enterprise is critically dependent upon the choice.

The primitive base sets for the set schema of the domain graph of figure 1.1 are E, the set of employees, and D the set of departments. These are typical of primitive base sets. The members of the sets are entities that are not machine readable and writeable strings, but can be distinguished by a human being from each other member. Further no two of the sets have any members in common. Thus the most elementary entities encountered in ER modelling are recognized only as members of primitive sets, and each is a member of exactly one primitive set.

The members of the set E are identified by members of the value set VE# through the **identifier** E#. Since members of E cannot be recognized or processed by a machine, it is necessary to associate with each of them a unique string that can be used for man-machine communication. In the set schema illustrated in figure 1.1, the identifier for D is DN, department name. It is assumed that every primitive base set of a set schema is provided with an identifier. Although for most primitive

base sets, a single attribute will suffice as an identifier, for some a pair, or even a triple of attributes may be needed. But no primitive value set and no nonprimitive set need have an identifier declared for it.

A primitive value set requires no identifier since its members represent themselves. A nonprimitive set requires no identifier since it inherits one from its immediate domain predecessors. For example, members of the set ED are identified by a pair consisting of an employee number and a department name.

The first guiding principle to be applied in ER modelling is:

1. Primitive base sets should be chosen to be the largest sets that contain only entities of interest to the organization, and that can be easily identified and distinguished.

For example, although in the set schema for Simple University no set of students is declared, in some extension of the model it will have to be declared. It would be reasonable, therefore, to declare a set PERSON as follows:
PERSON= { PERSON || students and employees of Simple University },
especially if an employee may also be a student and a common identifier for the two sets can be declared. But it would not be reasonable to declare employees and departments as a single primitive base set since they are unlikely to have attributes in common, and it would be difficult to declare a natural common identifier for them.

A set may be declared to be base or defined. The second guiding principle for ER modelling is:

2. Don't declared a set as base if it can be defined in terms of previously declared sets.

Assume, for example, that an attribute SEX is declared for employees with value set VSEX with members the strings 'M' and 'F'. Then the set MALE can be declared as follows:
MALE={ x:E | x:SEX:'M' | the male employees }
It would be a serious mistake to declare MALE as a base set. For the membership of a base set must be maintained by humans, while that of defined sets can be maintained by the system. If MALE were declared as base along with the base set SEX, the membership of both these sets would have to be maintained by humans, and furthermore they would have to be maintained so as to ensure that MALE had the membership determined by the given declaration.

The third guiding principle for ER modelling involves defined sets as well. It often occurs that the membership of a base set is restricted in some fashion. For example, assume that Simple University maintained a soccer team among its employees, and that it restricted the membership of the team to male employees. The set SOCCER of soccer players is clearly a base set with domain the defined set MALE:
SOCCER={ MALE || the employees playing on the soccer team }.
It is of course possible to declare SOCCER with E as its domain and rely on human understanding of the restriction on soccer players to ensure that only male employees were made members of SOCCER. But again it is better to rely upon the system's capacity to maintain defined sets:

3. Whenever possible declare a defined set as the domain of a base set to enforce restrictions on the membership of the base set.

The fourth guiding principle again involves the choice of primitive base sets:

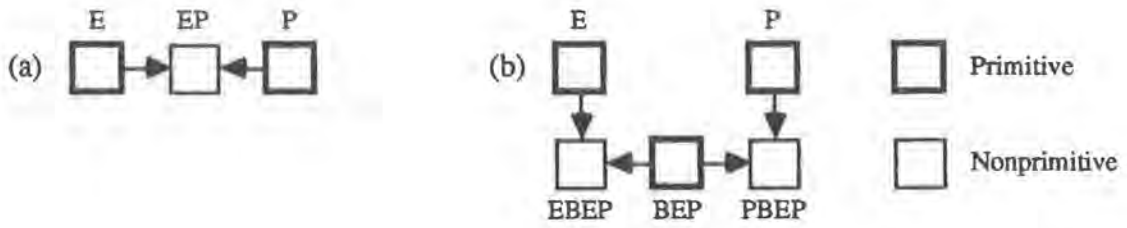
4. Sets of arity greater than 1 should not be declared as primitive sets of arity 1.

This principle may appear curious at first hand since it may be difficult to understand how a set of arity greater than 1 could be declared as a set of arity 1. But assume that Simple University has frequent need for special committees or projects to address particular problems. These projects are of a temporary nature, can be of any number, and any employee can be assigned to them. Approriate declarations would therefore be:
P={ P || ongoing projects }
EP={ ExP | <0,*>, <0,*> | employee project assignments }.
Here the upper degree of * means that an employee can be assigned to any number of projects, and that a project can have any number of employees.

It is possible to declare EP as a primitive base set:
BEP={ BEP || the pairs <e,p> where employee e is assigned to project p }
along with two other base sets:
EBEP={ ExBEP | <0,*>, <1,1> | selects e from a pair <e,p> that is a member of BEP}
PBEP={ PxBEP | <0,*>, <1,1> | selects p from a pair <e,p> that is a member of BEP}.
The sets EBEP and PBEP are the typical fansets of a network implementation of a database. They are the "roles" of E and P in the association BEP. [Bach77]

The domain graphs for the two declarations are illustrated in figure 2.1.

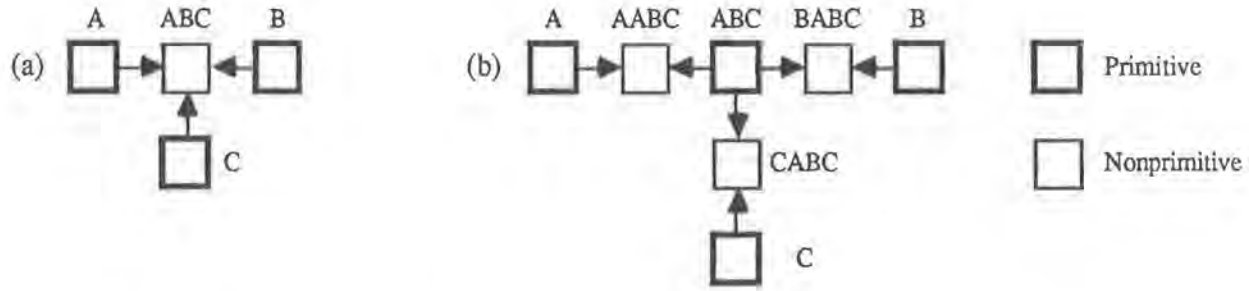
FIGURE 2.1



The principle states that EP should be declared, not BEP, EBEP, and PBEP. If for some reason the role sets EBEP and PBEP are needed, for example, in a network implementation of the database, they can be declared as defined sets.

The principle applies equally well to sets of arity greater than two. Even though it is tempting to declare sets of arity greater than 2 as sets of arity 1, because then only sets of arity 1 and 2 need be declared. In figure 2.2 (a) is illustrated a ternary association ABC with domain AxBxC. In (b) is illustrated how it might be declared using only sets of arity 1 and 2.

FIGURE 2.2

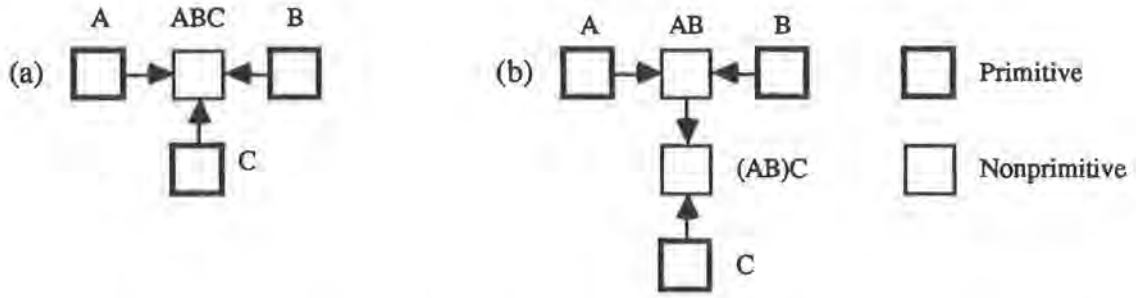


In (b) ABC is declared as a primitive base set along with A, B, and C, and has therefore arity 1. Again, however, the declarations ilustrated in (b) are not recommended because they require declaring as base sets the associations AABC, BABC, and CABC that in case (a) can be declared as defined sets. The fourth guiding principle asserts that ABC should be declared as illustrated in figure 2.2 (a), rather than as illustrated in (b).

A fifth guiding principle asserts, however, that (a) may not be the correct declaration for ABC.

Figure 2.3 (a) repeats the illustration of the declaration of the ternary association ABC of figure 2.2 (a), while (b) illustrates how it could be declared as a binary association (AB)C, once the binary association AB is first declared.

FIGURE 2.3



The association AB consists of those pairs <a,b> for which <a,b,c> is a member of ABC for some c, while (AB)C consists of those pairs <<a,b>,c> for which <a,b,c> is in ABC. The lower degree of (AB)C on AB is necessarily 1.

Whether (a) or (b) is the correct declaration depends first upon the upper degree of (AB)C on AB. When that upper degree is 1, then (b) is clearly the correct declaration, while if it is *, then (b) is more rarely the natural declaration.

The fifth and final guiding principle presented here is therefore:

5. A base set ABC of arity 3, should be declared as a base set (AB)C of arity 2 whenever its upper degree on AB is 1.

Again the principle can be generalized to sets of greater arity.

3. A Consistent Integrated Data Dictionary

A third reason for needing a mathematical foundation for the ER approach involves the very nature of conceptual models of enterprises. A conceptual model for an enterprise requires the capture of large amounts of what is called meta-data that records information about the common data processed in the enterprise's daily activities. As that information is gathered, the meta-data expressing it must be recorded somewhere so that it can be queried and updated like any other data.

Traditionally data dictionaries have been used for this purpose but they beg the question "where is the meta-data for a data dictionary to be stored?". To avoid an infinite sequence of data dictionaries it is necessary to begin with a repository for information capable of recording information about the repository itself. Although the need for such self-describing databases has been recognized before [Mark85, Lyke86, TrLo87], a concern that they raise has gone unnoticed: If such a repository is not to fall prey to the paradoxes and contradictions of set theory that shook the foundations of mathematics early in this century, then it must be provided with a provably sound foundation. The necessity for this has been demonstrated again more recently; the semantic networks described in [Sowa84] are subject to the same contradictions as naive set theory [Blac85]. The SET conceptual model, on the other hand, is based on the provably consistent set theories described in [Gil86].

"Consistency" is used here in the sense employed in logic and mathematics: A theory is consistent if it is not possible to conclude that both a sentence and its negation are true for the theory. The meaning given in [Date83] is included in the meaning of "integrity" used here.

In a consistent database in which all integrity constraints have been satisfied, it is not possible to conclude that a sentence and its negation are both true. But a consequence of the high level of abstraction required of an integrated data dictionary, is that such a database might satisfy very strong integrity constraints but neverthless be inconsistent, unless care is exercised in its implementation.

To declare a set is to state that it is a member of the set DSET of declared sets. Necessarily an integrated data dictionary for the SET model will have the set DSET as one of its declared sets, and necessarily therefore DSET will be declared to be a member of itself. Since self-membership is at variance with the classical methods for avoiding the paradoxes in set theory, some attention must be paid to the consequences of allowing it.

Consider, for example, the following declared set:
R={ x:DSET | **not** x:x | };
the members of R are those declared sets that are not members of themselves. The set E for example, is a member of R, while the set DSET is not.

There is nothing inherently wrong with the declaration of R, and it is even conceivable that it might prove useful for some purposes, but the set has a notorious past: It is the basis for the Russell paradox that shook the foundations of mathematics at the turn of the century. The paradox arises when one asks whether R is a member of itself. Using naive reasoning it can be concluded that R

is a member of itself, and that it is also not a member of itself. Thus a database system that permits the declaration of R, or some similar set, and that permits naive reasoning will be inconsistent, even though it enforces very strong integrity constraints. This is what [Blac85] demonstrated for the semantic networks described in [Sowa84].

Since the discovery of the Russell and other paradoxes, a number of set theories have been invented that maintain consistency by restricting the declaration of sets in a variety of ad hoc fashions. In [Gil86] it is argued that it is the reliance on naive reasoning, rather than the declaration of sets, that is the source of the paradoxes.

An assumption employed in naive reasoning is that every sentence is either true or false. A careful examination of this assumption shows, however, that it can only be made for atomic sentences, or what is called in theorem proving, ground sentences. The truth value for a complex sentence must be reduced to the truth values of simpler sentences, and be ultimately expressed in terms of the truth values of atomic sentences. If it is not possible to ground a complex sentence in atomic sentences, then the sentence receives no truth value. For example, the complex sentence R:R cannot be grounded in atomic sentences and therefore receives no truth value. It is this resolution of the paradoxes, proved sound in [Gil86], that is used to maintain the consistency of the SET model.

This simple resolution of the paradoxes has an important consequence. On the basis of the semantics, the correct answer is "no" to a query as to whether R is a member of itself, since R:R has not been assigned the truth value **true**; it is also the correct answer to a query as to whether R is not a member of itself, since R:R has not been assigned the truth value **false** either. Negation cannot therefore be understood in the sense of "negation as failure" as suggested in [Clar78] and criticized in [Flan86]; the negation of an assertion is assigned a truth value if and only if the assertion has been assigned a truth value, and it then receives the opposite truth value. In this sense a negated assertion derives its meaning from the assertion that can be obtained from it by driving negations down to the level of assertions of membership in base sets. All assertions expressing membership in defined sets must be replaced with their intensions, with membership in recursive sets requiring "recursive" treatment. Once a query has been grounded in this sense, a truth value can be determined for it, but not before. Therefore a management system must determine the truth values of assertions in a two stage process: First it attempts to ground the assertion, and if successful it assigns a truth value to it. Therefore a management system may respond to a query with a report of failure to ground the query.

Another consistency concern that must be addressed, although more mundane, is nevertheless

serious. It is possible to declare a set schema in the SET model with degree constraints that cannot be satisfied by any extensions of the declared sets. Consider, for example, a base association R with domain AxB and degrees <1,1> and <1,1>, where
A={ x:INT | x=1 }, and B={ x=INT | x=1 **or** x=2 | }.
Clearly the degrees can never be satisfied since they require that A and B have the same number of members. The satisfaction problem for degree constraints is unsolvable when a sufficiently rich form of DEFINE is available for defining sets; for example, the halting problem for a Turing machine can be expressed as the problem of determining whether two defined sets have the same number of members. But in [Gil87b] it is shown that a simple condition on set schemas is sufficient for satisfiability. The condition does not limit ER modelling in any serious fashion.

A fuller treatement of these topics is provided in [Morr].

4. A Unified Model of an Enterprise

A fully unified model of an enterprise is one that at the same time can give a conceptual view of the enterprise, a user's view of data as it is presented, a data administrator's view of data as it is stored, and a programmer's view of the processing of the data. The construction of such a unified model of an enterprise is ambitious, but is nevertheless necessary. For without such a unified model there will always remain the need for moving from one model to another when dealing with different aspects of information and its processing, with the probable consequence of the different models becoming inconsistent with one another. The need for such a unified model, raised in [Kent78] and repeated in [LyKe86], is argued forcefully in [TrLo87].

A commonly encountered perception of programs is that the dynamic nature of data processing prevents the representation of a programmer's view in a "static" model such as SET. In one sense the perception is correct, but in a wider sense it is not.

Once a set schema has been declared, a user must be able to add members to any declared set, and a command must be available for doing this. The form of a suitable command is:
Add tup **to** S **where** assert
where the variables occurring in the tuple tup occur unbound in the assertion assert; the latter provides an appropriate description of the entity to be added to the declared set S. A companion command removes an entity from a set:
Drop tup **from** S **where** assert.

In the most elementary form of the **Add** and **Drop** commands, S must be restricted to being base, and the meaning of the commands when S is defined must be reduced to the elementary form. For humans are responsible only for the membership of base sets, while the system is responsible only for the membership of defined sets. A command to add or drop a member of a defined set must, therefore, be interpretable as one or more commands to alter the membership of base sets. But how such commands are to be interpreted requires more research; the equivalent problem for the relational model is the updating of virtual relations, or views [Date83, Kell85].

As remarked in the previous section, the declaration of a set is equivalent to adding the set to the primitive base set DSET.

In as much as the commands **Add** and **Drop** are extensions to DEFINE, the perception that a "static" model such as SET cannot capture a programmer's view of data processing is correct. For example, it is not possible to represent the addition of a new employee to the primitive base set E as an association in the same sense that the assignment of an employee to a department is represented by ED: First, the new employee, before being added to the set E, is not a member of any set, so that a domain for **Add** is not available; and second, the effect of adding the employee to E is not to change its intension, but only its extension. When an entity that is a member of the domain of a nonprimitive base set is added to the set, a change of extension is the result. Such an application of **Add** can be regarded as an association between two states of the extensions of the declared sets. But that requires treating the extensions of all the declared sets as a tuple of tuples, something theoretically possible but often impractical.

On the other hand, each defined set can be regarded as a "program" that determines the membership of the set from the membership of its immediate predecessors. For example as the membership of E is changed, so may the membership of MALE be changed also; even the membership of the base set SOCCER may be changed if a male employee is no longer employed with Simple University. In this sense, therefore, the perception of the SET model as only supporting static descriptions is not soundly based. Typical data processing requires in part the use of the **Add** and **Drop** commands, sometimes imbedded in assertions, but also a substantial number of defined sets.

The two cornerstones of object-oriented programming, encapsulation and inheritence [Cox86], are central features of the SET model. These are the features also of the object-oriented data modelling methodology for information systems described in [LyKe86]. The objects of the SET model are the members of declared sets. The **Add**, and **Drop** commands provide the procedural element, as the comparable commands do in [LyKe86]. In the LORE approach to object oriented

programming[BCP86], the procedural element is introduced through message passing, traditionally a part of object-oriented programming.

5. Knowledge Bases

Sound foundations are needed for knowledge base systems capable of dealing with incomplete information. Such systems require distinctions that can only be made precise in a formally described model. For example, it is necessary to distinguish between sets with intensions that can only be understood by humans, and sets with intensions expressed in a language like DEFINE that can be understood by both humans and machines. It is also necessary to distinguish between an internal surrogate for an entity maintained by a management system and a character string that is an identifier for the entity, in order to deal adequately with identity [Kent78, Codd79, KhCo86].

Identity of strings and integers is a system defined concept, as is order on the integers. Identity of surrogates, on the other hand, is a management system defined concept, defined in terms of the attributes of entities introduced to identify them.

To deal properly with all aspects of incomplete information, it is also necessary to distinguish between completed and incompleted update transactions. In a knowledge base system updates may not be fully specifed and therefore result in transactions that cannot be completed. It is therefore necessary to maintain records of incomplete transactions and distinguish between conclusions drawn from completed and incompleted transactions.

Details on this approach to knowledge bases is provided in [Gil87c]

BIBLIOGRAPHY

[Bach69] BACHMAN, C.W. Data structure diagrams. *Data Base* 1,2 (Summer 1969), 4-10.

[Bach77] BACHMAN, C.W. The role concept in data models. *Proc. 3rd Int. Conf. on Very Large Databases*, Tokyo(1977).

[BCP86] BENOIT, CHRISTOPHE, CASEAU, YVES, AND PHERIVONG, CHANTAL. The LORE Approach to Object Oriented Programming Paradigms. Memo C29.0, Laboratoires de Marcoussis, Centre de Recherches de la C.G.E. April 15, 1986.

[Bern76] BERNSTEIN, P.A. Synthesizing third normal form relations from functional dependencies. *ACM Trans. Database Syst.*, 1, 4 (March 1976), 271-298.

[Blac85] BLACK, MICHAEL JULIAN. Naive Semantic Networks. Final Paper, Directed Study in Computer Science, Univ. of B.C. Jan 22, 1985.

[BMS84] BRODIE, MICHAEL L., MYLOPOULOS, JOHN, AND SCHMIDT, JOACHIM W. On conceptual modelling, Springer-Verlag, 1984.

[Chen76] CHEN, PETER PIN-SHAN. The Entity-Relationship model - toward a unified view of data. *ACM Trans. Data Base Syst.*, 1, 1 (March 1976), 9-36.

[Chen77] CHEN, PETER PIN-SHAN. The Entity-Relationship model - A basis for the enterprise view of data. *AFIPS Conference Proceedings*, Vol. 46, 1977 NCC.

[Chen85] CHEN, PETER P.S. Database Design Based on Entity and Relationship. [Yao85], 174-210.

[Clar78] CLARK, K.L. Negation as Failure. H. Gallaire and J. Minker (eds.), Logic and Data Bases, Plenum, New York, 1978.

[Codd70] CODD, E.F. A relational model of data for large shared data banks. *Comm. ACM* 13, 6 (June 1970), 377-387.

[Codd72] CODD, E.F. Relational completeness of data base sublanguages. In Data Base

Systems, ed. RUSTIN, Prentice-Hall, 1972.

[Codd79] CODD, E.F. Extending the Database Relational Model to Capture More Meaning. *ACM Trans. Database Syst.*, 4, 4 (Dec 1979)

[Cox86] COX, BRAD J. Object-Oriented Programming. Addison-Wesley, 1986.

[Date83] DATE, C. J. An Introduction to Database Systems, Volume II. Addison-Wesley 1983.

[DJNY83] DAVIS, CARL G., JAJODIA, SUSHIL, NG, PETER ANN-BENG, AND YEH, RAYMOND T. Entity-relationship approach to software engineering, North-Holland, 1983.

[DiDa86] DITTRICH, KLAUS, AND DAYAL, UMESHWAR. (Eds) *Proc. International Workshop on Object-Oriented Database Systems. ACM and IEEE.* Sept 23-26, 1986.

[Flan86] FLANNAGAN, TIM. The Consistency of Negation as Failure. *Journal of Logic Programming*, 2 (1986), 93-114.

[Full84] FULLER, ARTHUR. Zim, Zanthe Information Inc. Software Review, *InfoAge,* Nov 1984, 59-60.

[FuNe86] FURTADO, ANTONIO L. AND NEUHOLD, ERICH J. Formal Techniques for Data Base Design. Springer-Verlag. 1986.

[Gil86] GILMORE, PAUL C. Natural deduction based set theories: a new resolution of the old paradoxes. *J. Symb. Logic*, 51, 2 (June 1986), 393-411.

[Gil87a] GILMORE, PAUL C. The SET Conceptual Model and the Domain Graph Method of Table Design. Tech. Report 87-7, Dept. of Computer Science, University of B.C., March 87.

[Gil87b] GILMORE, PAUL C. Justifications and Applications of the SET Conceptual Model. Tech. Report 87-9, Dept. of Computer Science, University of B.C., April 87.

[Gil87c] GILMORE, PAUL C. Logic, Sets, and Common Sense Reasoning. *In Preparation.*

[HaMc81] HAMMER, MICHAEL, AND McLEOD, DENNIS. Database description with SDM: a semantic database model. *ACM Trans. Database Syst.*, 6, 3 (Sept 1981), 351-386.

[Kell85] KELLER, ARTHUR M. Updating Relational Databases Through Views. Stanford Computer Science Department Tech. Report STAN-CS-85-1040. Feb 1985.

[Kent78] KENT, WILLIAM. Data and Reality, North-Holland, 1978

[Kent83] KENT, WILLIAM. Fact-based data analysis and design, [DJNY83], 3-54.

[KhCo86] KHOSHAFIAN, SETRAG N. AND COPELAND, GEORGE P. Object Identity. [Meyr86]. 406-416.

[LeSa83] LENZERINI, MAURIZIO, AND SANTUCCI, G. Semantic integrity and specifications, [DJNY83], 529-550.

[Mark85] MARK, LEO. Sef-describing database systems - formalization and realization. Technical Report - #1484. Dept. Comp. Sci. Un. Maryland. April, 1985.

[Meyr86] MEYROWITZ, NORMAN. (ed.) Proc. Object-Oriented Programming Systems, Language and Applications. *ACM Sigplan Notices*, 21, 11 (Nov 86).

[Morr] MORRISON, RODERICK. Implementating a Set Based Data Model and its Data Definition/Manipulation Language. PhD thesis, Department of Computer Science, Un. British Columbia. *In progress.*

[MyWo80] MYLOUPOLOS, J., AND WONG, H. Some features of the TAXIS data model. *Proc. 6th Int. Conf. Very Large Databases*, Montreal (1980).

[Reit84] REITER, RAYMOND. Towards a Logical Reconstruction of Relational Database Theory, [BMS84], 191-233.

[Rock81] ROCK-EVANS, ROSEMARY. Data analysis. IPC Electrical-Electronic Press Ltd, Sutton, Surrey, England (1981).

[SAAF73] SENKO, M.E., ALTMAN, E.B., ASTRAHAN, M.M., AND FENDER, P.L. Data

structures and accessing in data-base systems. *IBM Syst. J.* 12, 1 (1973), 30-93.

[Ship81] SHIPMAN, DAVID W. The functional data model and the data language DAPLEX. *ACM Trans. Database Syst.*, 6, 1 (March 1981), 140-173.

[Sowa84] SOWA, J.F. Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley, 1984.

[TaFr76] TAYLOR, ROBERT W.; AND FRANK, RANDALL L. CODASYL Data-Base Management Systems. *ACM Comp. Surveys*. 8,1 (March 1976), 67-103.

[TYF86] TEOREY, TOBY J., YANG, DONGQING, AND FRY, JAMES P. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model. *ACM Computing Surveys*, 18, 2 (June 1986). 197-222.

[TrLo87] TRYON, D.C.; AND LOYD, D.G. Information Resource Depository: History, Current Issues, and Future Directions. Pacific Bell, A Pacific Telesis Company. Presentation to Canadian Information Processing Society, Vancouver, Canada, February 1987.

[Yao85] YAO, S. BING (editor). Principles of Database Design, Volume I, Logical Organizations, Prentice-Hall, 1985.