A Methodology for View Integration in Logical Database Design

Shamkant B. Navathe
Database Systems Research and Development Center
512 Weil Hall
University of Florida, Gainesville, Florida 32611

Suresh G. Gadgil
Metropolitan Life
Corporate System Planning
1 Madison Avenue
New York, New York 10010

Abstract

This paper is based on a conceptual framework for the design of databases consisting of view modeling, view integration, schema analysis and mapping and physical design and optimization. View modeling involves the modeling of the user views using a conceptual/semantic data model; the view integration phase merges user views into a global model which is then mapped to an existing database environment and subsequently optimized. Here we use the data model proposed by Navathe and Schkolnick to model user views and develop a methodology for view integration. View integration issues are examined in detail; operations for merging views are defined and an approach to automating the view integration process is described. The proposed approach is being partially implemented at the University of Florida.

1. INTRODUCTION

It can be said without any doubt that one of the major obstacles to the use of database management software is the initial preparatory effort during logical database design. It is certainly a difficult task to design a "community view" of a single database which truly reflects the aggregation of views with different expectations, backgrounds and technical expertise. For realistic databases used in business, industry and government with thousands of potential users, an individual user or user group cannot be expected to be aware of the needs of the rest of the user community and a designer cannot be knowledgeable enough to comprehend the requirements of a spectrum of users.

A natural way to start, therefore is by collecting the views of user groups or application areas individually. The problem of coping with thousands of data elements/attributes or hundreds of data objects is very difficult to deal with manually [12]. In this paper we present an approach to alleviating this problem. It is assumed that user views would be explicitly represented using some data model. The View Representation Model of Navathe and Schkolnick [11] is used as a representative model and a methodology for integrating such views is discussed.

1.1 THE DATA BASE DESIGN PROCESS

The conceptual framework for the design of data bases as described in some of Navathe's previous work [11,16] and as generally accepted at the 1978 New Orleans data base design workshop [7] may be described in terms of four steps. The input to these four steps stems from a Requirements Analysis step which provides a specification of data and processing requirements. The four steps are:

A. View modeling: in this first step the user's view of the real world is abstracted and represented.

B. View Integration: the user views are combined into a global model of the data and any conflicts in the process are presented for resolution.

C. Schema analysis and mapping: the global model is mapped to the logical structure of an existing database management system.

D. Physical schema design and optimization: the logical structure is analyzed with respect to physical design alternatives and an "optimal" physical schema is constructed.

This conceptual framework is illustrated in Figure 1.

1.2 OBJECTIVE AND SCOPE

A major problem in database design is the lack of a structured design methodology and lack of automatic aids for developing complex databases. Research has been conducted with the

goal of structuring this process [12,15,17]. Further, a few automated techniques which address parts of the database design process and others oriented towards particular database management systems are available; examples are PSL/PSA [14] used primarily for requirements specification; and Database Design Aid (DBDA) [12] used for a specific database management system. We had proposed in [16] the framework for a structured database design methodology consisting of the above 4 steps and surveyed research done to date related to those steps. In [11] we described how the first step of view modeling may be accomplished.

This paper addresses the View Integration step. It assumes the use of the specific data model namely, the Navathe and Schkolnick model [11], for representing views and analyzes the process of view integration. The paper discusses the different problems related to the view integration process and proposes an approach to the development of a software system for automating the construction of integrated views. We are presently implementing these ideas in the database design aid in conjunction with a data dictionary system being developed at the University of Florida. Efforts are also under way to consolidate the research on view integration by collaborating with the University of Rome where similar work is being pursued on the extended E-R model [2]. To our knowledge, very little work has been done on the view integration problem, barring a few exceptions [5].

2. MODELING USER VIEWS

For ease of discussion and because of some of its desirable features we have chosen the model proposed by Navathe and Schkolnick [11] (henceforth abbreviated as the N-S model) as a vehicle for modeling user views. A brief description of the model and its advantages follows.

2.1 SUMMARY OF THE NAVATHE-SCHKOLNICK MODEL

The N-S model includes the two aspects of user requirements identified by Kahn [6]: "the Information Structure perspective" describing data by means of entity types and association types, and "the usage perspective" giving a description of processing and insertion/deletion of instances of these types. The discussion of the view integration process itself is mostly invariant to the model selected and will be done in a general way. The N-S model allows the modeling of user views in explicit terms. The model uses two type constructs: objects and connectors. The object type is divided into two subtypes: entity type and association type. Instances of an entity type are called entities. Entities refer to physical things, persons, concepts. An entity type may either be self-identified or externally identified. An association type refers to a collection of associations. Associations are n-ary relationships defined over entity types and association types. They are subdivided into two subtypes: simple associations and identifier associations. The subtypes of a simple association type are: categorization, subsetting and partitioning types.

Connecter types in the N-S model connect an association type to some object type which participates in the association type. They are divided into two subtypes: directed and undirected. A directed connector type implies certain rules of insertion and deletion in the context of an association type. In general, connector types are used to show three types of dependencies among object types: ownership, deletion and null dependency. They are illustrated in Figure 3.

The N-S model representation conventions allow view diagrams to be drawn. These diagrams are supplemented with assertions to state complicated interdependencies among instances. An assertion language may be defined to state these assertions. We are presently investigating the adequacy of a language based on the first order predicate calculus. The language must be capable of expressing the following kind of assertions: For the view diagram in Fig. 18 -

> "Procedures performed by the SERVICE instance called Hospital-trust are always performed free" is an assertion in natural language. The same in an assertion language could be stated as -
>
> s.Service-Name = Hospital-trust &
> <s,p> ε PROCEDURE-IDENTIFIER &
> <p,c,r> ε PERFORMED ==>
> <p,c,r> ε PERFORMED-FREE.

where: s,p,c,r are respectively instances of entity types SERVICE, PROCEDURE, SCHEDULE and PERSONNEL.

Assertions pertaining to a single view are termed intra-view assertions whereas those pertaining to multiple views are termed inter-view assertions.

For a further clarification of the N-S model the reader is referred to the Appendix in the paper and to [11].

Note: The words 'type' and 'instance' will be used in the subsequent discussion only when they seem necessary.

2.2 ADVANTAGES OF THE N-S MODEL

a) It allows association types to be defined over entity types, over association types, or a combination of both unlike the E-R model [2].

b) Models the existence-dependencies among entities separately (in terms of identifier associations) from the relationships (or associations) among entities (unlike Smith/Smith [13]). It thus defines identification paths clearly. (e.g. PROCEDURE in Fig. 19 is existence-dependent on SERVICE by virtue of identifier association PROCEDURE-IDENTIFIER).

c) Describes a view of terms of object type (which are either entities or associations). The internal hierarchy of descriptor elements (attribute types) within an object type is kept separate from the object type interaction since it is less important. For example the internal structure of A in Figure 4 namely, A(a,b,(c,d),(e,(f))) is a further level of detail about entity type A.

d) Models data relationships as seen by user both at schema and instance level. Properties of views with respect to insertion/deletion rules etc. are clearly distinguished (unlike Chen [2] and Smith/Smith [13]) by use of different subtypes of association types and by employing directed connector types.

2.3 GLOSSARY OF VIEW MODELING AND INTEGRATION TERMS

A user view is a representation of information structure and processing requirements as seen by a group of users. A subview is a subset of a user view. The subsequent discussion about user views is equally applicable to subviews.

The View Integration process has as its objective the development of a conceptual model supporting all users in the organization. This conceptual model will be called as a Global View. The Global View may in fact contain a number of views that cannot be further integrated. The software system to accomplish view integration will be called a View Integrator.

The Enterprise View forms a nucleus for development of the Global View. It describes the basic entities and associations appropriate for the organization and is based on the assumption that some characteristics of data structure and usage can be deduced from the nature of the organization.

Equivalent Views are defined as views which have the same information content but different structures. The term information content refers to the functional and non-functional associations among attributes. Information content has been defined formally for relational databases in terms of functional dependence and direct table look-up associations [1]. In this paper we will assume that an acceptable measure of information content has been defined for the user community in question.

A target of integration is an object type or a connector type which is compared among different views for possible integration.

A view integration operation is a process that is applied to the targets of integration giving rise to new object types, connector types, attribute types etc. These operations are discussed in greater detail in a later section. A preferred view is that view which is selected from among two or more equivalent views for inclusion in the Global View over a less preferred view. The integration policy is defined as the set of rules which dictate how a number of local views must be integrated into an Enterprise View. The policy contains several rules; examples of these rules are:

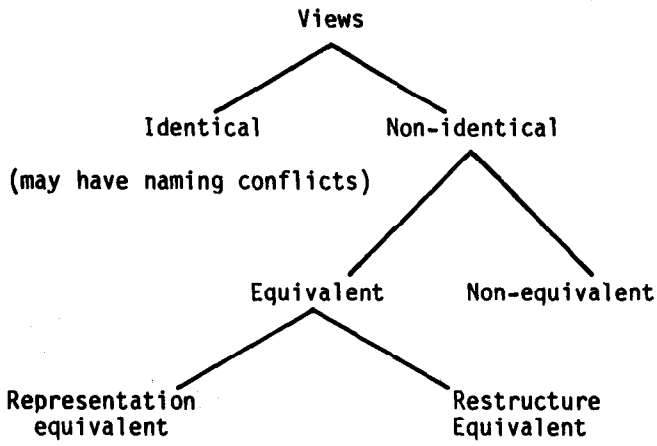
> "The user views in Finance Dept. are to be preferred over user views in Marketing Dept. at all times."
>
> or "In categorizing patients, select those categorizations which correspond to the physical allocation of wings in the hospital (e.g. Psychiatric, Obstetric, Intensive Care, etc.)."

The integration policy, after all, is subject to interpretation by the designer. It should be couched in the form of a scale of preferences of views and intra-view and inter-view assertions.

2.4 EQUIVALENCE OF VIEWS

A comparison of two views can be made to determine whether they are identical. Identical views have objects and connectors that match completely. In some cases the identical nature of two views is made apparent only after naming correspondences are established.

Views which are not identical are referred to as non-identical and may be classified further into Equivalent and Non-equivalent views. A pairwise matching of views can lead to the grouping of a set of views into subsets shown schematically in the following hierarchy:



Equivalence of two views arises from two sources: the use of different modeling constructs and conventions resulting in Representation Equivalence; genuine differences in how the same data are viewed by the modeler/designer resulting in Restructure Equivalence. Restructure Equivalence is so named because each of the equivalent views can be transformed into the other by a set of restructuring operations such as compression, expansion, assembly merging and assembly partitioning [10]. The distinction between these two types of equivalences is motivated by the fact that the complexity of view integration operations is greater in dealing with Restructure Equivalent views. Since the boundary between these two is at best arbitrary, some convention must be adopted to separate the two. In this paper our convention will be as follows: if two views have objects and connectors which match except for differences in prime and candidate keys, they will be considered representation equivalent. Figure 5 shows two Representation equivalent views. Figure 6 shows two Restructure equivalent views where View 2 has entity type DEPTS categorized into categories MEDICAL, SURGICAL, NON-MEDICAL-NON-SURGICAL but contains the same information as in View 1. The equivalence is not apparent unless it is specified.

Restructure equivalence among views to be integrated generally requires identification of the preferred views(s) and gives rise to the development of mapping rules during integration. The preferred view is incorporated into the Global View with as little change as possible. Less preferred views are accommodated by means of mapping rules which enable the non-preferred view to be reconstructed from the Global View.

3. A MODEL FOR THE VIEW INTEGRATION PROCESS

A Global View is developed from the input views and assertions. The input views consist of the Enterprise View and local views; the Inter-view assertions describe certain characteristics of views which need to be taken into account in the integration process.

Other outputs of the View Integration process are - Mapping Rules and a Statement of Conflicts. The Mapping Rules describe how individual local views can be generated and how user processing requirements can be satisfied. The nature of mapping rules depend upon the way in which processing requirements are specified. E.g. if an access path specification is used to describe processing requirements, a mapping rule may state that the original access path is obtained by a combination of access paths in the global view. The statement of conflicts presents conditions which could not be resolved during the integration.

The view integration model is presented schematically in Figure 7.

The following assumptions and statements further define the scope of view integration in the present paper:

a) a single result: the Global View which really consists of multiple views is to be produced. The model does not consider development or presentation of alternative Global Views.

b) due to errors or inadequate definition of input views or assertions the integration step produces a statement of conflicts wherever necessary after designer intervention.

c) it is assumed that a "normalization of input user views" is carried out corresponding to the first normalization of the relational model [4] to eliminate all Identifier Associations. The normalization process propagates keys within the data structure so that all objects become self-identified. Figure 8 illustrates how the identifying association linking LABORATORY and CHIEF is removed by propagating LabName into CHIEF, i.e., expanding the key of CHIEF from ChiefName, to Labname, ChiefName to make it self-identified. Navathe [9] has discussed an intuitive procedure to normalize network structured data which can be applied when a network of identification paths is present.

3.1 INPUTS TO THE VIEW INTEGRATOR

The inputs to the View Integrator are the Enterprise View, the local views, assertions and processing requirements. In this paper we do not elaborate on how processing requirements within each view are specified.

A. THE ENTERPRISE VIEW

The Enterprise View is constructed by the designer based on his knowledge of the organization. It will include many of the entities in the final database design. For example, in a database for a Medical Center an Enterprise View may consist of entity types DOCTORS, PATIENTS, TESTS, MEDICATIONS, NURSES and PROCEDURES.

B. LOCAL VIEWS

Local Views are views of data seen by the individual users or user groups in the organization. They are modeled using the N-S model with entities and associations. They may be supplemented by intra-view assertions. A preferred local view and its intra-view assertion(s) are carried forward unchanged as far as possible during integration; some local views undergo changes during integration. However, unless deliberately deleted, no information from any view is supposed to be lost.

C. INTER-VIEW ASSERTIONS

Inter-view Assertions circumscribe the view integration process by specifying views which are equivalent. They state in what way views are equivalent and the data correspondence from one view to another. Data correpsondence is stated in terms of names of entities, associations and attributes and, where appropriate, the restructuring operations to convert one view to another. An example of an inter-view assertion for Restructure equivalent views shown in Figure 6 is:

1. View 2 = RSTR[View 1]
2. Preferred View = View 2
3. View 2 [MEDICAL, SURGICAL, NON-MEDICAL-NON-SURGICAL] ==> CATEGORIZE [View 1 [DEPARTMENT]/TYPE]

The first two statements above are self-explanatory. Statement 3 specifies the restructuring operation called CATEGORIZE which transforms entities from one view to another. (see assembly partitioning in [10]). In this example the entities MEDICAL, SURGICAL, NON-MEDICAL-NON-SURGICAL are categories of the DEPARTMENT entity of View 1 by the attribute type called Type.

Inter-view assertions are also used to specify prime and candidate keys in representation equivalent views. The inter-view assertions for Figure 15 are:

1. View 1 = REP[View 2]
2. Prime key: ServiceNo.; Candidate key: ServiceName.

3.2 OUTPUTS OF THE VIEW INTEGRATOR

A. THE GLOBAL VIEW

A Global View is a conceptual model of data which subsumes the Enterprise View and all pertinent user views within an organization. The Global View is characterized by a minimum of data redundancy and inclusion of all access paths for data retrieval of object instances from all user views. A Global View is the most significant output of the View Integrator; it is input to the next phase in logical database design namely, Schema Analysis and Mapping, which maps the database design to a target database management system.

B. MAPPING RULES

Mapping rules state the access paths for data and include the deletion and insertion rules for views which are equivalent. Mapping Rules are derived from inter-view assertions. They represent the means by which data specified in a non-preferred view may be obtained from the Global View. The complexity of mapping rules depends upon the number and type of restructuring operations required to transform one view into its equivalent.

Mapping rules may be illustrated with reference to Figure 6 which shows two restructure equivalent views. Inter-view assertions for these views have been stated earlier. A mapping rule for these views would be:

DEPTS ==> DEPTS + MEDICAL + SURGICAL
+ NON-MEDICAL/NON-SURGICAL

where + indicates a union operation for access paths.

The mapping rule signifies that references to the entity DEPTS of the non-preferred view - View 1 requires access to all the entities named on the right hand side of the mapping rule in the Global View. For Fig. 10, a mapping rule to obtain View 1 would indicate that accessing DEPARTMENT by Deptname would involve a table-look-up operation of looking up a Deptno for Deptname first.

3.3 A GENERAL PROCEDURE FOR VIEW INTEGRATION

In a semi-formal manner the view integration process may be described as follows.

Inputs

Assume an enterprise view E and local views V_1, ... V_n as given.

P = given integration policy
I = a set of intra-view assertions
J = a set of inter-view assertions
R = a set of processing requirements

Outputs

G = global view set
M = a set of mapping rules
S = a set of conflicts
A modified set of assertions

A step-by-step procedure for view integration:

Step 1: Divide views V_1, ... V_n into classes C_1, ... C_m such that each class contains either

- a set of equivalent views (either restructure or representation equivalent), or
- a set of identical views, or
- a single view (which is not identical nor equivalent to a view in another class).

Verify the classes by checking with the designer.

Set class index i to 1.

Step 2: Perform an integration on class C_i.

If the class contains identical views, select one as the integrated result: IV_i.

If a class contains equivalent views, attempt to generate an integrated view IV_i,

- by applying P
- by treating views in descending order of preference
- by reporting back conflicts to the designer and asking the designer to make a choice (of names, of keys etc.) or making new assertions or modifying old ones.
- by considering I and J during integration.

If a single integrated view cannot be generated, multiple intermediate views may be generated and passed to the next step.

Step 3: Provide feedback to the designer on the intermediate results of integration IV_1, ... IV_p. (These include local views which were in a class by themselves). Ask designer to provide a preference ordering of these semi-integrated views.

Step 4: Integrate views IV_i into E in descending order of preference in a way similar to step 2.

- Modify assertions in I and J on the basis of how the integration process affects the components of an assertion.
- Report conflicts back to the designer and ask the designer to make a choice, add or modify an assertion etc. If a conflict cannot be resolved add it to set S.
- Repeat step 4 as long as view integration operations are applicable. The application of specific operations during integration is extremely difficult to figure out automatically on the basis of naming, structural similarities and assertions. The designer may be required to control this process (see [2]).

The result of step 4 is the set G of global views. Ideally G should contain a single view but practically it may not. Step 4 may need to be reapplied after some modifications.

Step 5: Generate M by determining how each of the requirements in R is satisfied by using G. (Generation of M may alternately be carried out during Steps 2 and 4).

4. THE MATCHING PROCESS IN VIEW INTEGRATION

The view integration operations to be applied to views are determined by comparing the views to be integrated. At the object type level, similarity of objects is determined by comparison of attributes and keys; at the view level, the comparison is between objects and connectors from one view with another. The matching process is used in Step 1 of the view integration procedure to divide views into classes. There are three possible outcomes of

the matching process: complete match, partial match or mismatch.

It is assumed that two completely matching object types will have completely matching instances. The reason behind such an assumption is that at the logical database design stage no conclusions can be drawn regarding actual values in instances of data. If details about the values are available, they may be supplied by means of inter-view assertions:

e.g., Value-set (View 1 ' PLACE-OF-ORIGIN)
= 'US','OTHER' ,

Value-set (View 2 ' PLACE-OF-ORIGIN)
= 'US','EUROPE','ASIA','OTHER'

4.1 OBJECT SIMILARITY

Object similarity is determined first by comparing the names of the object types and then by comparing the attribute names and definitions of the object types. This matching results in either a complete match or a partial match or a mismatch. The conditions of complete match and complete mismatch are easy to visualize. Matching names are reported to the designer for confirmation that they have identical meanings. The partial match conditions arise out of a combination of complete match, partial match, and mismatch when key attributes and non-key attributes from different objects are compared.

The range of outcomes of the matching process is illustrated in Table 1.

The general rules for resolving partial matches and mismatches are described below.

Let O be an object type being matched; O_1 is from a preferred local view, O_2 is from a non-preferred view and O_3 is the result of integrating O_1 and O_2.

Let $Key(O_i)$ = set of key attribute types in O_i.

$N\text{-}Key(O_i)$ = set of non-key attributes types in O_i.

a) Partial match on key -

$$Key(O_3) = Key(O_1)$$

$$N\text{-}Key(O_3) = N\text{-}Key(O_1) \cup N\text{-}Key(O_2) \cup \{Key(O_2) - Key(O_1)\}$$

b) Mismatch on key -

$$Key(O_3) = Key(O_1)$$

$$N\text{-}Key(O_3) = N\text{-}Key(O_1) \cup \{N\text{-}Key(O_2) - Key(O_1)\} \cup Key(O_2)$$

The union operation on the attribute sets above is supposed to eliminate duplicate attributes although their names may not be identical. The duplication is inferred from assertions.

Figure 9 illustrates a partial match on key; the entity type DEPARTMENT has key attribute DivNo, DeptNo in view 1 which is preferred. After integration, the same key applies to DEPARTMENT. Figure 10 illustrates a mismatch on key. Deptno from the preferred view 2 is selected as a key in the target view. Deptname remains as a non-key attribute type. Figure 11 illustrates a case with no match on name between entity types NURSE-STATION and RECEPTION-STATION. However, an inter-view assertion (possibly input by the designer after ascertaining that they mean the same) forces a match among them. Matronname and Head-nurse-name are also supposed to be declared as identical attribute types. The names, key etc. from the preferred view are carried to the integrated view.

Table 1: Outcomes of Object type matching

Match Type	Match On		
	Name	Key attributes	Non-key attributes
Complete	Complete	Complete	Complete
			Partial
			No Match
		Partial	Complete
			Partial
			No Match
		No Match	Complete
			Partial
			No Match
Partial	No Match	Complete	Complete
			Partial
			No Match
		Partial	Complete
			Partial
			No Match
		No Match	Complete
			Partial
Mismatch	No Match	No Match	No Match

4.2 SUBVIEW OR VIEW SIMILARITY

Assume a situation in which local views are matched by extracting subviews made up of one or more unary, binary or n-ary associations and matching them among themselves. A subview thus has at least two objects and may have ownership and deletion dependency characteristics by virtue of directed connectors. Subview similarity may be determined by a comparison of object types and a comparison of dependencies. At a minimum we can consider two object types and the ownership and deletion dependency, whichever are applicable, between them. Object matching has already been discussed in terms of name, key attributes and non-key attributes of the object

types. As regards different dependencies, we have three states, viz., ownership, deletion dependency and null dependency in each of two views, giving a total of nine combinations. In the matching process we consider the combination [Ownership (View 1), Deletion Dependency (View 2)] the same as [Deletion Dependency (View 1), Ownership (View 2)]. Therefore, we have to consider only the following six cases:

Table 2: Different combinations of dependencies

Combination #	Type of Dependency View 1	View 2	Match Outcome
1	Ownership	Ownership	
2	Deletion	Deletion	Complete Match
3	Null	Null	
4	Ownership	Null	Partial Match
5	Deletion	Null	
6	Ownership	Deletion	Conflict

Combinations 1, 2 and 3 represent conditions of complete match and the merging process can continue. Combinations 4 and 5 represents a partial match and require resolution in one of two ways: maintain both relationships with redundant object types or provide mapping rules so that data for each of the two views can be obtained from the Global View without any redundancy. The latter method is chosen in keeping with one of the objectives of integration namely, to minimize redundancy. This is in contrast to El Masri and Wiederhold's approach [5] where they require that partially matched objects from equivalent views in the Global Model be maintained AND consequent insertion and deletion of instances of these objects be handled consistently. The approach in [5] has the advantage of preserving the same view of data as originally defined by each user in the Global View but results in a heavy data redundancy as well as increases processing overhead.

As a general policy, the integrated view for Combinations 4 and 5 will include the more constrained of the views being integrated namely, View 1 in Table 2. Combination 6 shows a conflict in two views which cannot be resolved automatically. This condition requires designer intervention followed by a specification as to the preferred view.

4.3 VIEW INTEGRATION OPERATIONS

View Integration operations permit the merging of user views to develop a Global View. Operations are performed on targets of integration namely, object types and connector types. One operation on a target of integration may trigger several operations with the object and connector types which were involved in the former operation as targets. View integration operations can be defined at two levels: schema (type) operations and instance operations. Instance operations would apply to individual instances of objects (e.g., delete instances where age > 75). A view integrator need not really support such operations during design. They may be provided in the form of modifications of assertions.

Schema operations consist of operations on associations and connectors. The former operate on the three kinds of special associations: categorization, partitioning and subsetting. The latter operate on connector types.

4.3.1. ASSOCIATION TYPE OPERATIONS

Categorization Addition is the additon of a new categorization of an entity-type. Figure 12 shows the integration of two views with different categorizations of the same object PATIENT. The integrated view is obtained by merging the PATIENT entity and the addition of categorization ADMISSION-STATUS to an existing categorization AGE-GROUP. Assume that mutual exclusivity was given as an intra-view assertion. The implication on instances is that a given patient has one instance of type PATIENT, one of either type INPATIENT or type OUTPATIENT and one of either type PEDIATRIC or type ADULT.

Category Enhancement is the addition of a new category type supplementing an existing categorization. Category enhancement is illustrated in Figure 13. The two input views show different categorizations of the PROCEDURE entity type. The integration of these views results in one categorization association of the PROCEDURES entity type into four entity types. This operation is similar to the categorization addition except that (i) no new categorization association is added here; only new category types are added to an existing categorization. If mutual exclusivity is to be enforced, categorization addition is preferred.

Partition Enhancement involves the addition of partitions to an existing partitioning association. Partition enhancement is illustrated in Figure 14 where View 1 and View 2 have a partitioning of the EMPLOYEE by EMPLOYEE TYPE. The entity type called EMPLOYEE-TYPE is shown in terms of its instances, e.g., NURSE and DOCTOR in View 1. Each of these instances is supposed to be associated with a partition of the set of instances of EMPLOYEE. In the integrated view the partitioning association includes EMPLOYEE-TYPE partitions from View 1 enhanced by EMPLOYEE-TYPE partitions from View 2. Mutual exclusivity must be obeyed, otherwise partition enhancement is disallowed. In this example an EMPLOYEE instance must map into one of the four partitions defined by the four instances of EMPLOYEE-TYPE.

Partition Addition is the addition of a new partitioning association for a given entity type. In the above example of EMPLOYEE-TYPE partitions, if the partitions from two views cannot be disjoint (e.g. a Nurse can also be a Surgical-assistant), then the two EMP-TYPE-SEL partitionings need to be kept separate in the integrated view. From any single preferred view's standpoint, this amounts to adding a new partitioning association to that view.

Subset Addition is the addition of one or more subsets to an existing association to allow a different subset of association instances to be a named subset. Figure 15 illustrates subset addition. The integrated view shows the five association types which are subsets of the association type DRUG-SCHEDULE from View 1 and View 2.

4.3.2 ATTRIBUTE TYPE OPERATIONS

Attribute Enhancement is the addition of new attribute types to an object type to account for the partial match or mismatch of that object type with other views. Figure 10 shows how attribute enhancement is applied to the entity type DEPARTMENT. Considering that each view contributes instances of an entity type, the attributes which are not present in a particular view are supposed to be given null values in the integrated view.

Attribute Creation is an operation where a new attribute is created in an object type to convey the same information as is contained in another view by means of a categorization or partitioning association. Figure 16 shows an example where object-type STUDENT is being matched in two views. To convey the categorization information about a STUDENT, a new attribute called Category is created in the integrated view. The value set for Category will be assigned (by the designer); e.g., Category = 0 for graduate, 1 for undergraduate. Note that the attributes, Degree and Rank, which are a part of the entity type GRAD-STUD and UG-STUD are used for an attribute enhancement of STUDENT.

4.3.3 CONNECTOR TYPE OPERATIONS

Restriction is defined as the selection of the most restrictive connector specification for representation in the integrated view. Connector types show three types of dependency: ownership, deletion and null dependency. When merging two views the combinations of dissimilar connector types may be integrated as follows:

View 1	View 2	Integrated View
Ownership	Null	Ownership
Deletion	Null	Deletion
Ownership	Deletion	CONFLICT

Conflict in connector types calls for designer intervention.

5. CONFLICTS IN VIEW INTEGRATION

Conflicts in the integration process arise during the matching and integration of views for several reasons. They are presented to the designer for resolution. If the designer is able to resolve them (possibly with the help of users) integration continues with the new information; otherwise the conflict is added to the set of unresolved conflicts. Conflicts may partly be caused by incomplete or erroneous specification of input and partly due to differences in modeling. Conflicts may be classified by considering their source into the following classes:

a) Object descriptions, including naming and dependency conflicts
b) Equivalent View conflicts
c) Modeling conflicts

5.1 OBJECT DESCRIPTION CONFLICTS

These can occur in the form of synonyms (different names referring to the same objects) or homonyms (one name referring to different objects). In the proposed matching scheme synonyms will cause a complete match on key and non-key attributes but no match on name; and homonyms will be shown up by a condition where two objects have the same name but the key and non-key attributes do not match completely. In either case, our current implementation scheme requires designer confirmation before any merging of object types is attempted. A designer should allow two object types, whether with the same name or different names, to be integrated (merged) only after ascertaining that they have the same set of instances. A wrong choice of name could cause the reporting of more conflicts as additional views are integrated.

As discussed previously, several dependency conflicts may occur. These conflicts are generally resolved by adopting the most restrictive specification out of those in different views and then providing mapping rules wherever necessary for individual views. One case where the

conflict cannot be resolved (see Restriction Operation) is when the same object is an owner of association A in view 1 and is deletion dependent on association A in view 2.

5.2 EQUIVALENT VIEW CONFLICTS

The integration process assumes that equivalent views will be accompanied by inter-view assertions stating equivalence. When equivalent views are not accompanied by inter-view assertions, the integration process cannot detect equivalence and hence yields redundancies of data. In addition, partial match conditions may be detected and the designer warned of possible conflicts. Figure 6 shows two views which are equivalent. However, without an inter-view assertion about equivalence, the integrator will report that object-type DEPTS from View 1 partially matches with DEPTS as well as with MEDICAL, SURGICAL, etc. from View 2. The conflict may arise whenever no equivalence is stated by an assertion, yet there is a total match or a partial match among either key or non-key attributes which indicates a possibility of equivalence. This signals an equivalent view conflict and warrants designer intervention.

5.3 MODELING CONFLICTS

Differences in representation of views can potentially lead to redundant data after integration. However, genuine cases exist where the same situation may be modeled differently by different users. Figure 17 illustrates a modeling conflict involving the use of partitioning and categorization associations. The instance implications of Views 1 and 2 are drastically different. Consider the following situation. Entities of type PATIENT are partitioned in two ways in View 1: by admission status and by age-group. ADMISSION-STATUS is an entity type with two instances and so is AGE-GROUP. In View 2 the information about patients is more detailed - there is an instance of one of the category types INPATIENT or OUTPATIENT for every PATIENT; similarly there is an instance of category types PEDIATRIC or ADULT for every PATIENT too. To support both views, the designer must allow the categorization association as well as the partitioning associations to be preserved.

Between the conditions of a complete match and a complete mismatch of objects we have a range of partial matches. Table 1 shows the match conditions when matching on Object names, key attributes and non-key attributes. An integrator must provide for a set of actions in each of those cases.

6. CONCLUSIONS

We are currently implementing a design aid which allows user views to be input using the N-S model, builds a dictionary of views and performs view integration using the approach described above. Emphasis is being placed on designer interaction and our aim is to produce a workable solution for dealing with large problems of integration which designers cannot cope with manually.

The conclusions of our investigation into the view integration process are as follows:

a) The success or failure of view integration is largely dependent on getting as explicit a statement of user views as possible. The job of extracting all relevant information from each user (or user group) is a major challenge to the designer. It is not clear whether the designer should do an ad hoc analysis of user views to detect equivalences (e.g. restructure equivalence) and then try to specify them on his own or whether he should expect different user areas to know of the similarities and differences in their data needs.

b) From the detailed discussion of object and connector matching it is clear that the task of matching and integrating views represented by objects and connectors is non-trivial. Decisions made in the automatic View Integrator should be confirmed at every step and the designer/user should bring in his instance level knowledge about data to evaluate the matches or mismatches. User help may be sought from time to time to guide this process.

c) In the conflict resolution area, considerable human involvement is necessary. It is possible to classify conflicts and help the user in an understanding of the conflict, but the ultimate repsonsibility to resolve them will be up to the users and management who set policies.

d) Given that users are kept actively/interactively involved in the view integration effort, the main advantage a machine can provide is to deal with a large number of integration alternatives and present them to the user. An automated or semi-automated approach is still beneficial when the volume of comparisons of data names and data characteristics is too high to deal with manually. The actual view integration operations at the object and connector level were shown to be straightforward and can be easily automated.

The overall conclusion from the above, therefore, is that a View Integrator can be a

valuable aid in design for realistic "integrated" databases used in most organizations. It is however futile to expect that such a system could solve the problem completely on its own and produce a tailored design of the database. The system has to be an interactive one where the user/designer team will navigate the course to an acceptable design.

ACKNOWLEDGEMENT

The authors wish to thank Melanie Smith for her comments on an earlier version of the paper. Discussions with Maurizio Lenzerini of the University of Rome have been useful. This work was supported in part by DOE grant No. DE-AS05-81ER10977.

APPENDIX:

View modeling in the Navathe-Schkolnick Model

This appendix describes the essentials of the View Representation model [11] which was introduced as a vehicle for expressing the views of data held by user groups or individual users of a database prior to its design. Most of the terms used below are identical with those in [11]. Some terminology has been changed to improve claritiy and understandability. The reader is referred to [11] for a comprehensive discussion about the model.

General Description

The model expresses a user's view of data by employing the following type constructs: entity, association, attribute, connector. These types are divided into subtypes as shown in Figure 2. Assertions are used additionally to state interrelationships and constraints which cannot be otherwise expressed. The term object type is used to stand for either an entity type or an association type. In general, the model allows the user to start with the entity types of interest, describe each entity type with a nested list of attribute types and build any number of levels of association types. No instance information is captured in a view diagram besides that in the form of assertions.

The model provides conventions for a visual description of a user view. This visual description, called a view diagram, is in the form of a graph with object types as nodes and connector types as edges. Assertions are not graphically represented.

The semantics of the model include rules of insertion and deletion at both the type and instance level. Some of them are covered below and in Fig. 3. In the following we define each important concept in the model and provide some explanation. References are made to the examples used in the body of the paper.

Terminology

An entity type refers to a physical thing, object, event, person, document, etc., whose existence is of interest to the user. An association type is an n-ary relationship among n object types. An object type is either an entity type or an association type. An association type is a fact, a concept or a relationship among its components.

Each object type has a descriptor set which is a list of attribute types used to describe the object type. An attribute type may be atomic in which case it is a property, quality or characteristic of the object types which it can describe. Otherwise, an attribute type may be defined as a list of other attribute types. The attribute type structure within an object type can thus be hierarchically organized and is shown by a nested parenthetical notation (e.g. see Figure 4).

A connector type shows the linkage or the logical access path among two object types where one of them is an association type and the other is a participant in that association type. A connector type, if undirected, is represented by a two tuple. When directed it is represented by an ordered two tuple. An assertion is a statement (in some assertion specification language) which describes a specfic relationship among instances of object types from one view or multiple views; the term intra-view and inter-view assertions are used correspondingly. Assertions are used when the semantics of the directed connectors is inadequate to model instance interdependencies.

A list of key attribute types of an entity type provide a total identification to it when each instance of the entity has a unique value list of the key attribute types. Such entity types are called self-identified. Partial identification of an entity type by its key attributes implies the need for external identification. External identification is defined as the process of augmenting the partial internal identifier (PID) of an entity with the key attributes of other objects. E.g., in Fig. 8, CHIEF has a key attribute Chiefname (key-attributes are always underlined in view diagrams) which is a partial identifier. CHIEF is externally identified from entity type LAB. The model allows an object type to have several external identifications.

Association types are divided into two subtypes: simple and identifier. An identifier association is an n-ary association which

provides external identification to uniquely identify an instance of a partially identified object type from the remaining (n-1) objects types. An association is called a simple association if it is not an identifier association. (See [8] for a detailed discussion of identification of data).

Diagrammatic Notation

The visual description of a view is realized in the form of an acyclic graph called a view diagram where object types are nodes and connector types are edges. All of the concepts mentioned above, except intra-view assertions, are used in the view diagram. Object types (except identifier association) are represented as ovals. In the view diagram, association types are grouped on one side of the diagram and entity types are grouped on the other. Object types are described by descriptor sets within the diagram underneath the object type if possible. A self identified object type is represented by placing a # symbol next to the object type in the view diagram (e.g. SERVICE in Fig 18). An identifier association is represented as a rectangle with undirected connectors from the identifying object type(s) to the identifier association and a directed connector from the identifier association to the identified object type.

Simple Association Type

A simple association type is a way of relating instances of one object type with instances of another object type. The content of a simple association contains tuples composed of the total internal identfiers (TIDs) of all object types involved in the association type, as well as any attribute types describing the simple association type. This type of association is represented with undirected connectors between the objects (involved in the association) and the simple association. Identification of simple association types is achieved by concatenating the TIDs of all of the object types involved in the association type. There is no restriction on the number of simple association types that an object type can be involved in.

Directed Connector Types

Directed connector types express different kinds of dependence among object types to which they relate. In an n-ary simple association there is a directed connector from an entity to the association if it plays the role of an owner of the association. If deletion of the association implies that certan entity instances get deleted, this is also indicated by a directed conector type from the association type to those entity types. Lack of directed connectors in a simple association implies that there is no specification of the ownership of the association nor of any deletion dependency. Fig. 3 shows the above possibilities and explains the insertion/deletion rules. In an identifier association type a directed connector type points to the entity type which is identified by the rest (see Fig. 8).

Three special types of simple associations are defined:

1. categorization is an n-ary relation between an owner object type and member object types called category types.

2. partitioning is a binary relation among object types; an instance of the object type which causes partitioning is associated with a set of instances (a partition) of the other object type.

3. subsetting is a unary association over an object type.

Categorization A of an object type X into object types $X_1, X_2, \ldots X_n$ is denoted by an association type $A(X_1, X_2, \ldots X_n)$ which is marked "cat" in a view diagram. Figures 12, 13 show examples of categorization. Categorization A implies that there exists a mapping f_a which maps every instance of X into a subset of categories $X_1, X_2, \ldots X_n$.

A binary partitioning association type S(X,Y) associates an instance of X with a subset of the instances of Y. Typically, X and Y are entity types but either one or both may be association types. Partitions of the instances of Y are non-overlapping, such that each partition has a different instance of X associated with it via S. A partitioning association is marked "partition" in a view diagram.

Mathematically,

let I_y be the set of instances of Y.

2^{I_y} denotes the power set of I_y,

i.e. the set of all its subsets.

Then there is a function G_s associated with S:

$$G_s: I_x \rightarrow 2^{I_y}, \text{ such that}$$

$$I_y = \bigcup_{i=1}^{m} G_s(x_i)$$

$$\text{and } G_s(x_i) \cap G_s(x_j) = \phi \text{ for } i \neq j,$$

where $x_1\ x_2 \ldots x_m$ are all instances of X.

The partitioning association EMP-TYPE-SEL in View 2 of Figure 14 implies that the instances of entity type EMPLOYEE are partitioned into 2 sets. One is associated with the first instance of the entity type EMPLOYEE-TYPE representing surgeons, the other with the second instance of the entity type EMPLOYEE-TYPE representing surgical assistants.

A unary subsetting association type T(A) provides a means of giving the name T to a subset of the instances of A. In the view diagram such an association is marked "sub". A is typically another association type (e.g., DRUG-SCHEDULE in Figure 15) but could also be an entity type. There is no mutual exclusiveness constraint among the subsets of instances of A defined by subsetting associations $T_1(A)$, $T_2(A)$, ..., etc.

View Representation using the N-S model is illustrated with the example in Figure 18. The figure shows six entity types: SERVICE, PROCEDURE, SCHEDULE, PERSONNEL, INPATIENT-PROCEDURES, OUTPATIENT-PROCEDURES; an identifier association PROCEDURE-IDENTIFIER which identifies instances of SERVICE with instances of PROCEDURE; a simple association type PERFORMED; the categorization association PROCEDURE-CATEGORIES which categorizes PROCEDURE into INPATIENT-PROCEDURES and OUTPATIENT-PROCEDURES and subsetting association types PERFORMED-FREE and PERFORMED-REPEATEDLY. The figure also shows with directed connectors that the object type PROCEDURE is identified by association type PROCEDURE-IDENTIFIER, and owns the categorization PROCEDURE-CATEGORIES, and that the entity type SCHEDULE is deletion dependent on the association type PERFORMED. PERFORMED is shown to be the owner of the subsetting association types.

REFERENCES

1. Arora, A.K. and Robert C. Carlson, "The information preserving properties of relational database transformations", Proceedings of the 3rd Very Large Database Conference, Berlin, West Germany, IEEE, 1978.

2. Batini, C., M. Lenzerini and G. Santucci, "A Computer-aided Methodology for Conceptual Database Design", Information Systems, Vol. 7, No. 3, Pergammon Press (to appear).

3. Chen, P.P.S., "The entity-relationship model: towards a unified view of data", ACM Transactions on Database Systems, Vol. 1, No. 1 (March 1976).

4. Codd, E. F., "Normalized data base structure: a brief tutorial", Proceedings of the 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, Ann Arbor, MI May 1974.

5. El-Masri, R., and G. Wiederhold, "Data model integration using the structural model", Proceedings ACM SIGMOD International Conference, Boston, MA, June 1979, pp. 191-202.

6. Kahn, B.K., "A method for describing information required by the database design process", Proceedings of the ACM-SIGMOD International Conference on Management of Data, June 1976, ACM, New York.

7. Lum, V.Y. et al., "1978 New Orleans data base design working report", Proceedings of the 5th International Conference on Very Large Data Bases, Rio de Janiero, Brazil, October 1979, pp. 328-339.

8. Navathe, S.B., "Schema analysis for database restructuring", ACM Transactions on Database Systems, 5, 2, June 1980.

9. Navathe, S.B., "An intuitive procedure to normalize network structured data", Proceedings of the 6th Very Large Database Conference, Montreal, Canada, October 1980, pp. 350-358.

10. Navathe, S.B., and J.P. Fry, "Restructuring for large databases: three levels of abstraction", ACM Transactions on Database Systems, 1, 2, June 1976.

11. Navathe S.B., and M. Schkolnick, "View representation in logical database design", Proceedings of the ACM-SIGMOD International Conference, Austin, TX, June 1978, ACM, New York.

12. Raver, N. and G.U. Hubbard, "Automated logical database design: concepts and applications", IBM Systems Journal, No. 3, 1977.

13. Smith, J.M. and D.C.P. Smith, "Database abstractions: aggregation and generalization", ACM Transactions on Database Systems, 2, 2, June 1977.

14. Teichroew, D. and E.A. Hershey, III, "PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems", IEEE Transactions on Software Engineering, Vol. S.E-3, No. 1, January 1977.

15. Weldon, J.L., "Integrating database logical views", Unpublished Working Paper, New York University, 1979.

16. Yao, S.B., S.B. Navathe, and J.L. Weldon, "An integrated approach to logical database design", Proceedings of the NYU Symposium on Database Design, May 1978, [in 17].

17. Yao, S.B., S.B. Navathe, J.L. Weldon, T.L. Kunii [eds.], Database Design Techniques I: Requirements and Logical Structures, Springer Verlag, New York, 1982.

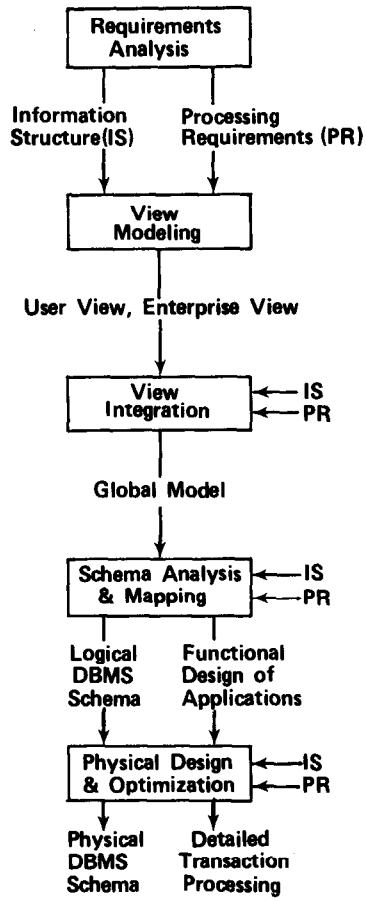


Fig. 1: Conceptual Framework for Database Design [11]

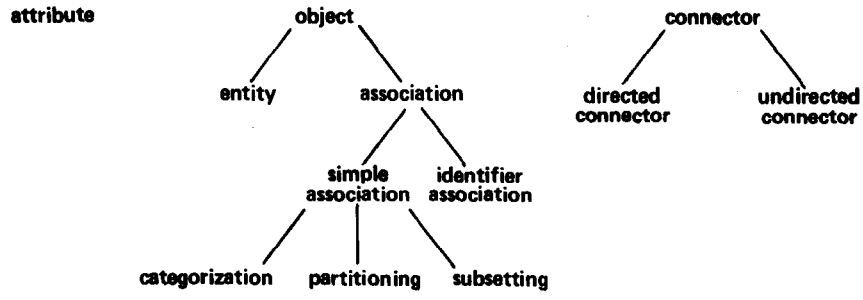
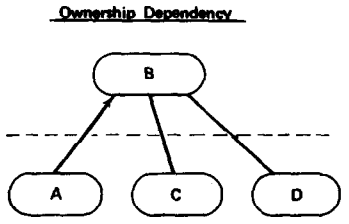


Figure 2: The Type Hierarchy in the View Representation Model

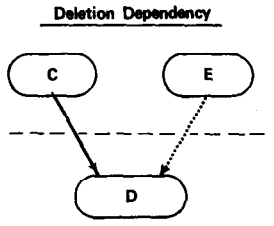


A is the owner object type in association type B.
C and D are "owner dependent" on A.

(i) instances of A can be freely inserted.

(ii) instances of C or D cannot be inserted unless associated with some A.

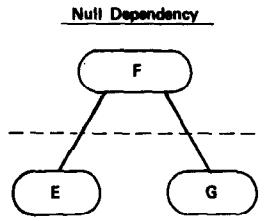
(iii) if an instance aa of A is deleted, its its association instances in B e.g.<aa, bb, cc> containing aa are also deleted; instances of C and of D which are not referenced in any other instance of B are also deleted.



D is the member object type in association type C.
D is "deletion dependent" on C and E.
Deletion of an instance cc of C implies automatic deletion of the associated instance dd of D, subject to two conditions:

(i) dd is not a component of any other instance of C and

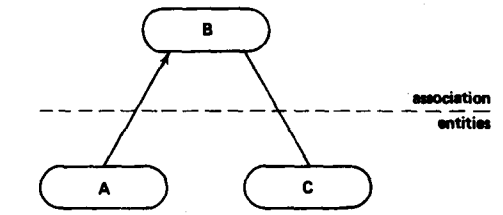
(ii) dd is not a component in another association instance (different from C, such as E)



No ownership and no deletion dependency for association type F and among object types E and G.

Implies that E and G can be freely inserted and deleted.

Fig. 3: Different Types of Dependencies



A = (a, b, (c, d), (e, (f)))

a, b, c, d, e, f are attribute types. The notation on the left shows their nested structure within object-type A. Clusters such as (c,d) may have their own names.

Fig. 4: View Representation with N-S model showing separation of entity-types from association types and a hierarchy of attribute types within an entity

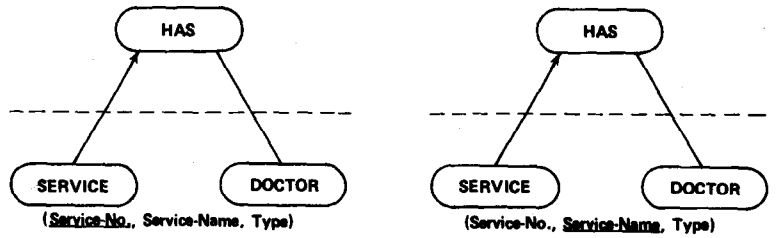


Fig. 5: Representation Equivalence - Same objects in two views but with different prime keys.

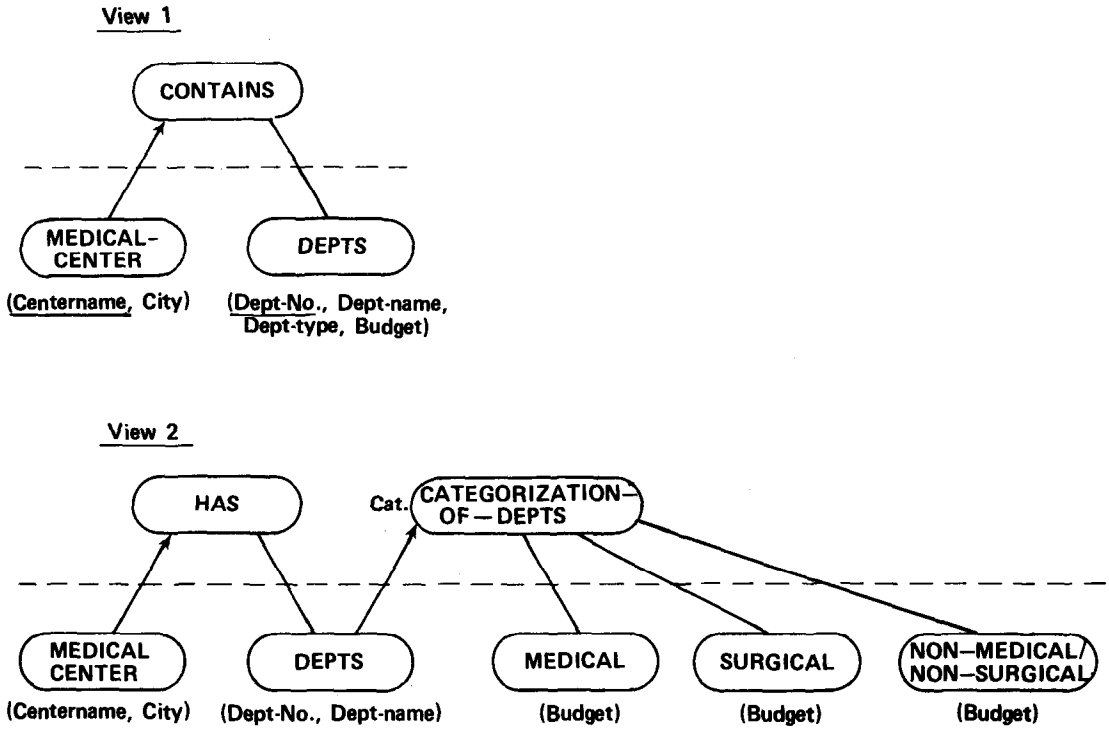


Fig. 6: Restructure Equivalence

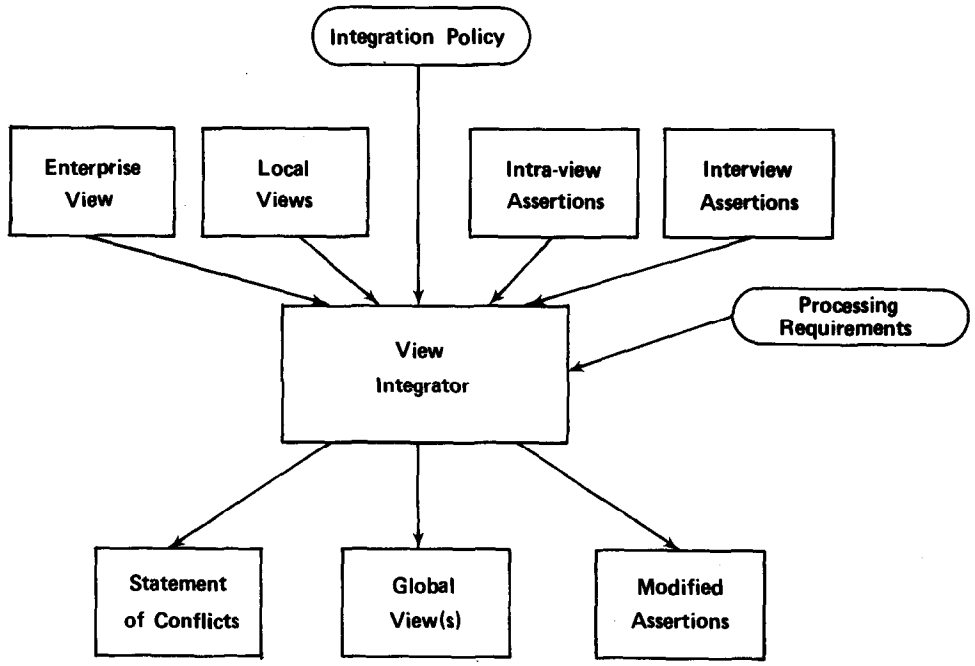
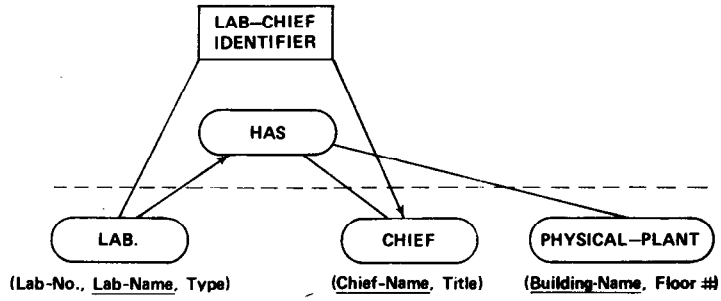


Fig. 7: A Model for View Integration



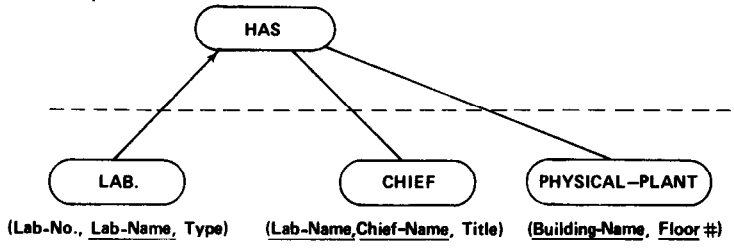
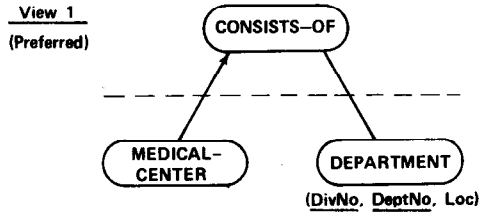
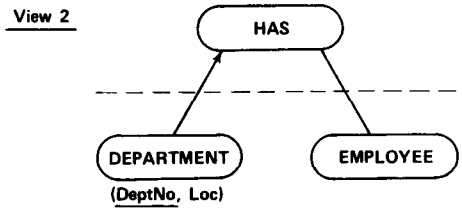


Fig. 8: Normalization of Identifier Associations





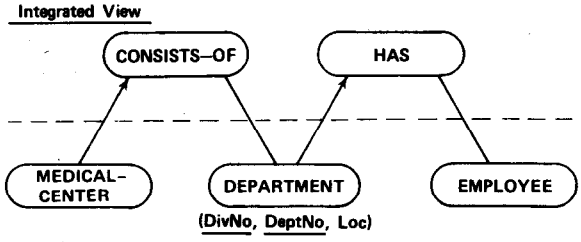


Fig. 9: Integration of views when there is a complete match on name, partial match on key attributes and complete match on non-key attributes.

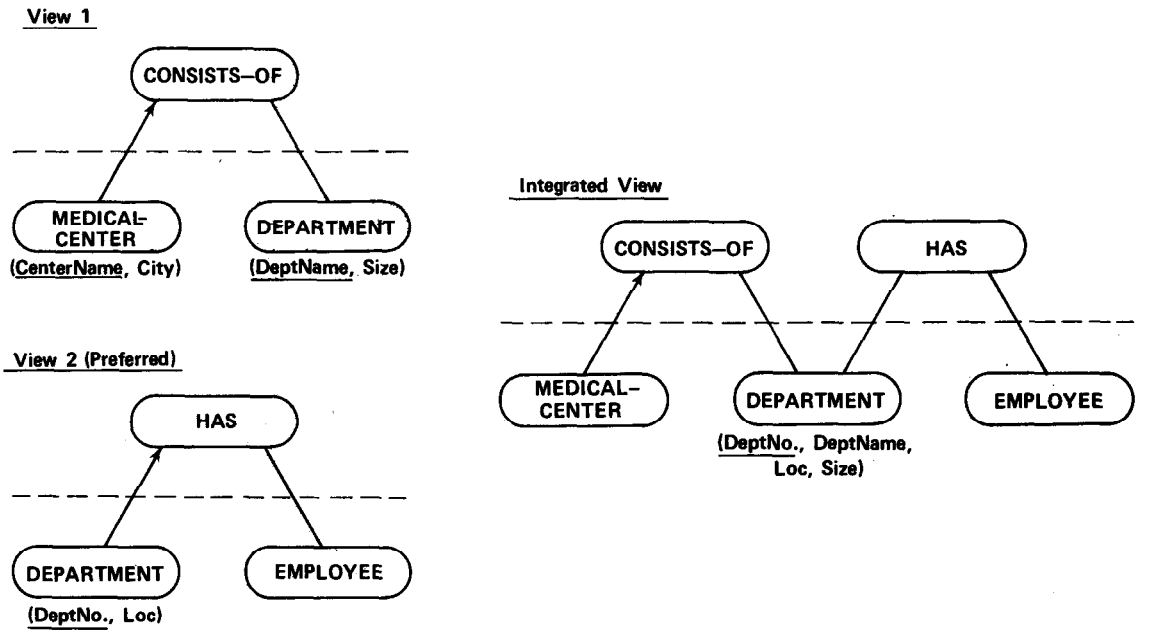
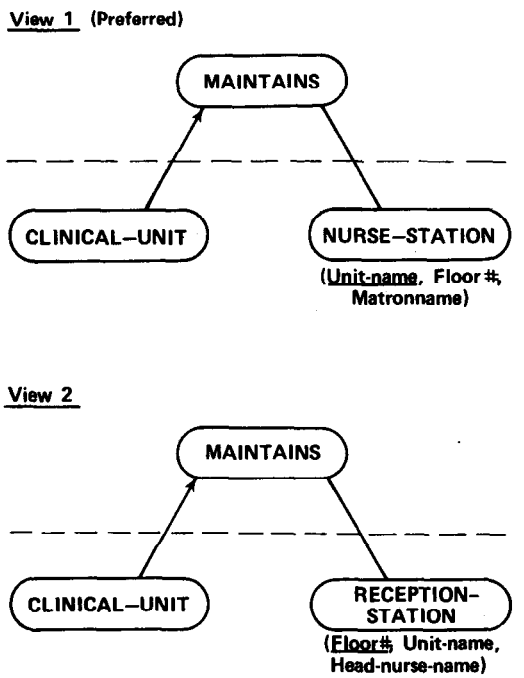


Fig. 10: Integration of views when there is a complete match on name, no match on key or non-key attributes.



Integrated View: Same as View 1

Fig.11: Integration of views with object types having different names.

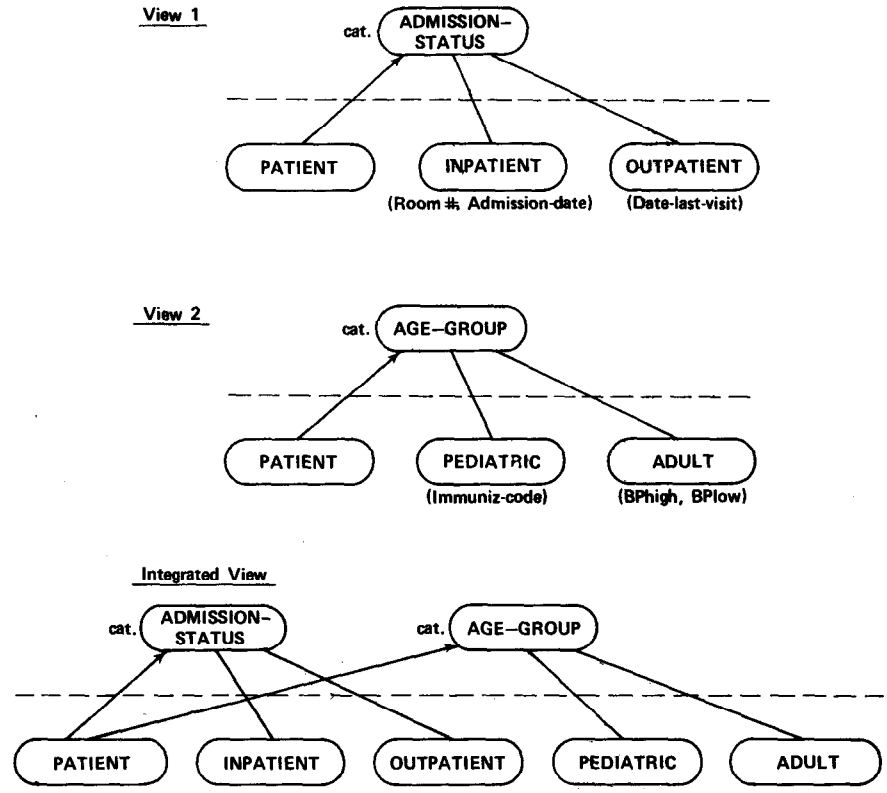


Fig. 12: Categorization Addition

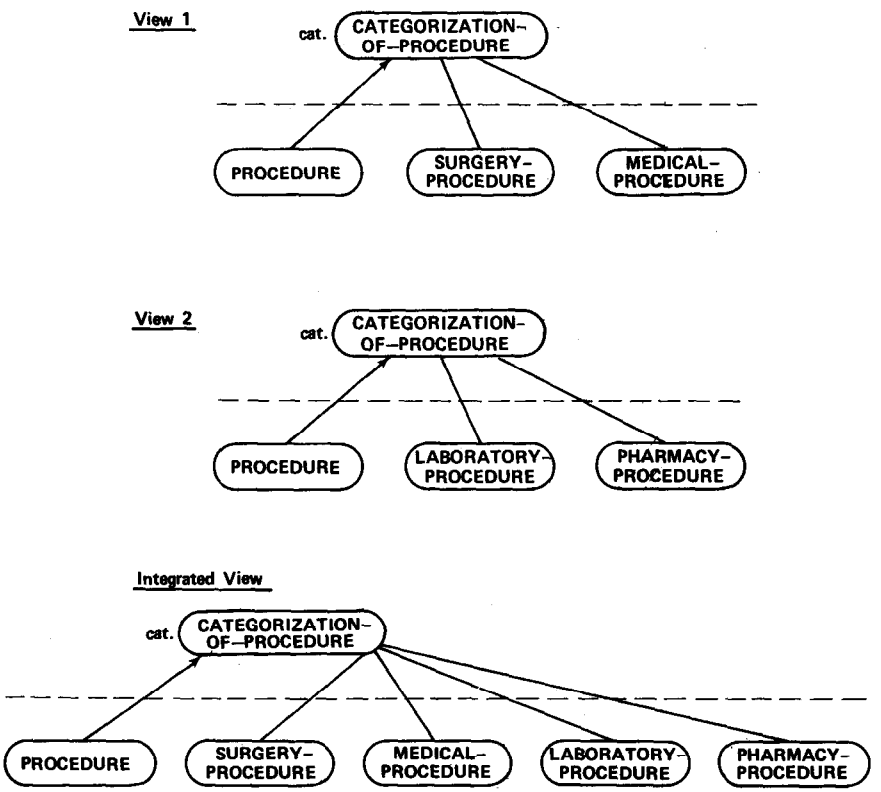


Fig. 13: Category Enhancement (type diagram)

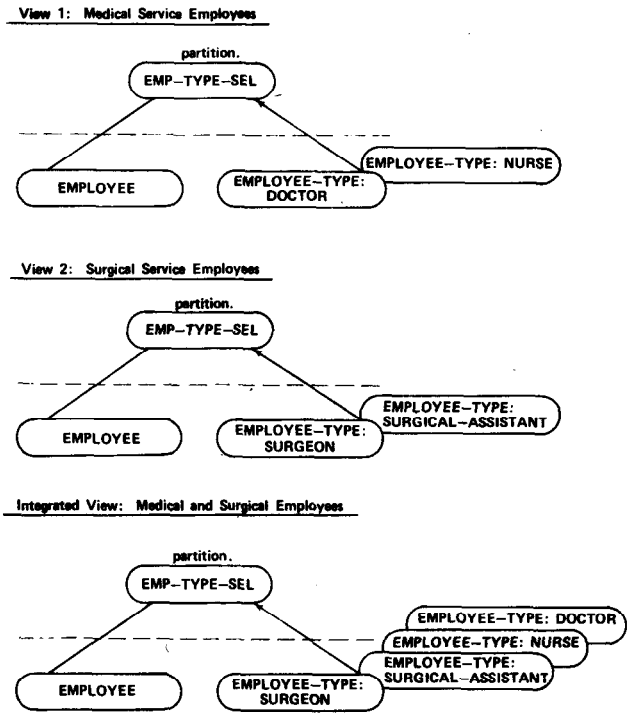


Fig. 14: Partition Enhancement (shows instances of an entity type called EMPLOYEE–TYPE)

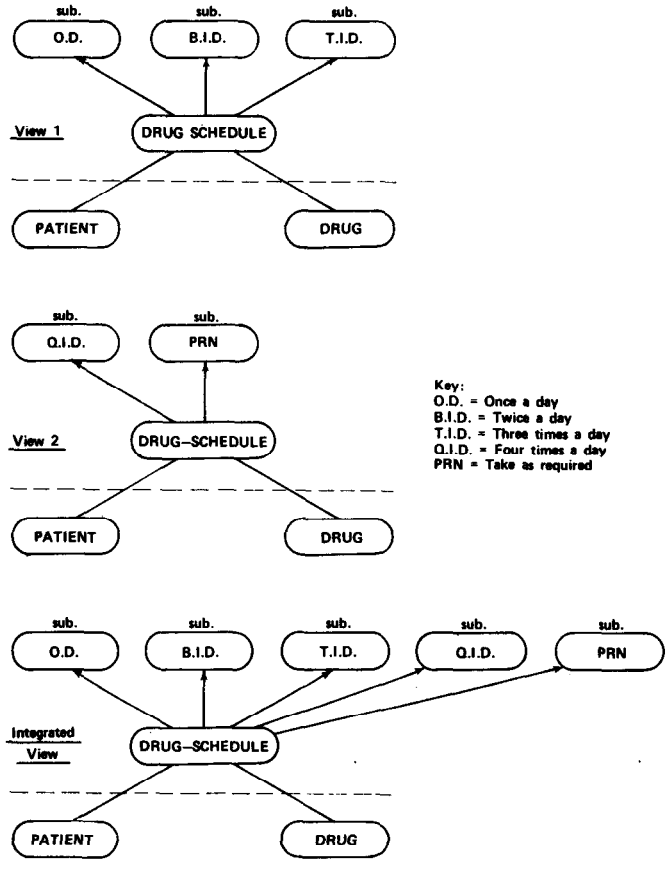


Fig. 15: Subset Addition (type diagram)

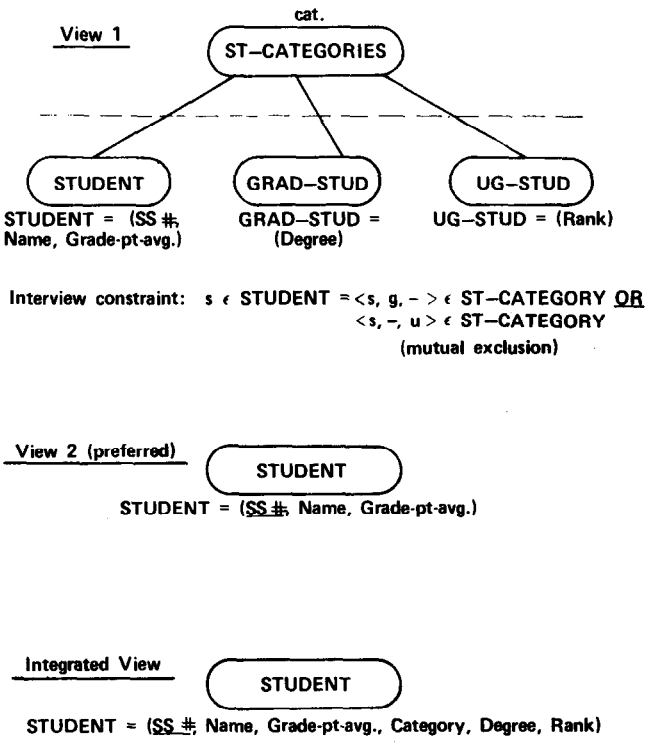


Fig. 16: Creation of a new attribute type called 'Category' during integration.

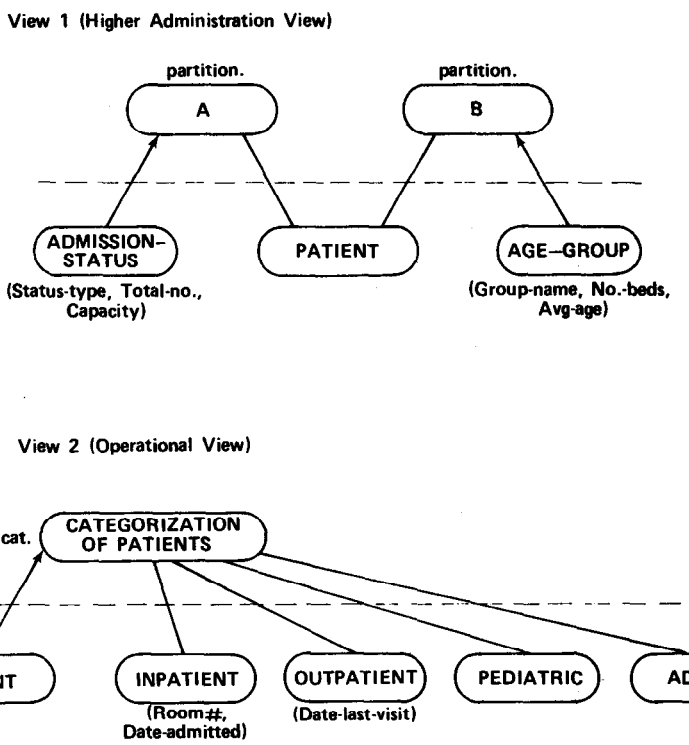
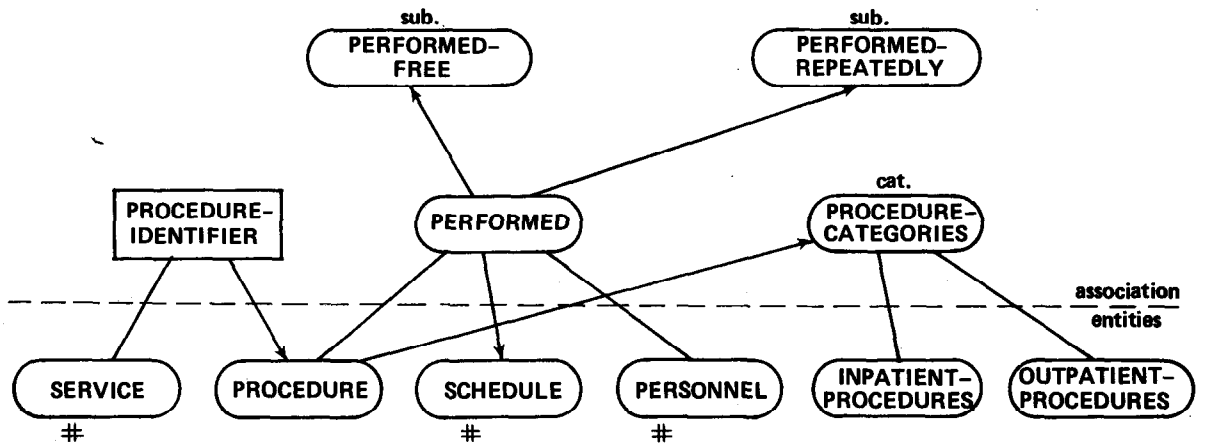


Fig. 17: Two different views of patient information, which may coexist.



Entity Types

SERVICE (Service-Name, Location)
PROCEDURE (Procedure-No., Procedure-Name, Type)
PERSONNEL (Employee-No., Employee-Name, Title, Sex)
SCHEDULE (Date-Number, Time-start, Time-finish)
INPATIENT–PROCEDURE (Labname)
OUTPATIENT–PROCEDURE (Outpatient-unit-no.)

Association Types

PROCEDURE–IDENTIFIER (identifier-association type)
PERFORMED
PROCEDURE–CATEGORIES (categorization-association type)
PERFORMED–FREE (subsetting-association type)
PERFORMED–REPEATEDLY (subsetting-association type)

Fig. 18: View Representation using the N-S model.