VIEW INTEGRATION IN DATABASE DESIGN

by Christian Wagner

Diplom-Ingenieur, Technical University Berlin, 1984

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
Faculty of Commerce and Business Administration

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA
April 1989

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Faculty
~~Department~~ of Commerce and Business Administration

The University of British Columbia
Vancouver, Canada

Date April 24, 1989

ABSTRACT

The purpose of this research is the formalization of a method of bottom up database design known as view integration.

View integration is one of the main steps of an acknowledged database design procedure, the New Orleans Database Design Workshop procedure. This procedure develops a global database (global schema) for an organization from small partial databases (user views). Individual user views are representations of the data relevant to the users' organizational tasks. Views will overlap since users will share data to some extent. View integration has to merge views without duplicating the information presented in multiple views. The task of merging views without duplication is complicated by the fact that users have different perceptions of the world which lead them to represent the same data differently, the most simple form of different perceptions being naming conflicts such as the occurrence of synonyms.

Within the last 13 years a variety of approaches to solve the integration task has been reported. Many of the approaches have neglected the problem of conflicting views altogether, leaving its solution to the database designer. Integration methods that performed conflict resolution did it in an

unsystematic and incomplete fashion. Often these methods dealt with conflict situations only if information for their resolution was conveniently available.

This research fills that gap. A conflict analysis procedure is outlined which considers all possible conflict conditions and transforms them into conditions that can be merged by means of previously developed techniques. The research proceeds in two steps. First, a conflict analysis procedure is developed that ignores the information requirements problem by assuming complete information. This simplification allows the concentration on completeness of the procedure, since one does not have to be concerned with the difficulties involved in gathering the required information. The second step relaxes the assumption of complete information. Difficult information requirements are identified and replaced by more easily satisfied ones.

Main contributions to knowledge are (1) a complete understanding of the factors causing conflicts between views, (2) detection of substitutes for difficult information requirements. Other contributions are (3) suggestions for the development of a semantic data dictionary, (4) an alternative method for the design of knowledge based systems, and (5) suggestions for efficient bottom up systems design strategies.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENT

I thank my supervisor, Professor Robert C. Goldstein, for his guidance as well as for his ongoing encouragement. My thanks go to Professor Yair Wand for his often very critical and always very stimulating comments. To Professor Wolfgang Bibel I am grateful for providing many new perspectives on the nature of this research.

I also wish to acknowledge the funding given for this research by the World University Service of Canada, dem Deutschen Akademischen Austauschdienst, and the University of British Columbia.

Finally, I thank my parents, Helmuth and Iris Wagner, for their love and support.

1. OVERVIEW

The database designer's task, converting users' casual data descriptions into a database design is time consuming, error prone, and requires substantial expertise. This argument is still valid, even though the separation of logical and physical design considerations has simplified the design effort (Curtice and Jones, 1982). Consequently, there exists strong interest in the development of techniques to improve the database design process, particularly the hardware independent logical database design process.

One approach that has been taken is the further decomposition of the design process. Frequently, database designers begin with a graphical representation of the database to be built, i.e. an entity-relationship model or Brown diagrams (Brown, 1982), before they design the actual database relations or record and set types. As DeMarco (1979) mentions in the context of structured analysis, graphical representations are a tool that provides a concise representation, allows easy consistency checking and is very maintainable. Another form of design composition focuses on the development of individual user views for small task domains and subsequent integration of user views into a complete schema. The rationale for this approach is simplification due to a more narrow focus, as well

as improved validity of the views. If every user describes only the data of her task domain--the data she is most familiar with--the resulting representation promises to be more correct than one that is done by a person only remotely familiar with the domain. However, since each each view describes data structures as perceived by the individual users, differences in user perceptions--conflicts between user views--are to be expected. These conflicts have to be settled, before views can be aggregated to form a global database structure. The purpose of this research is the formalization and solution of the conflict resolution problem. Even though a variety of integration methods are presently available, existing view integration methods are incomplete, frequently neglecting the conflict resolution problem (Batini et al., 1986, p. 348). Conflicts arise when different users model the same real world concepts differently, or different real world objects identically.

This research bridges the gap by developing a conflict classification and resolution scheme, and based on this scheme a computer program that integrates user views, grounded in rules and heuristics of database design.

2. VIEW INTEGRATION

2.1. Database Design Philosophies - Top Down vs. Bottom Up

Independent of any particular database design approach there exists the question whether database design, like any other form of systems design, should proceed top down or bottom up. Bottom up and top down represent the extreme points in a spectrum of design alternatives.

In general, top down design has the advantage over bottom up design that it is oriented towards overall goals and that it allows stepwise refinement of those general goals. Bottom up design requires integration of the elements of the overall system and will almost certainly result in conflicts between the elements and in the necessity for the redefinition of system elements. Despite this disadvantage, bottom up approaches are frequently used (Martin, 1984, McFadden and Hoffer, 1988). Their major advantage is that they do not demand the existence of an overall design before the design of particular elements can take place. Thus, no overall understanding of the system is required, or at least not to the extent necessary for the top down approach. In addition, bottom up design facilitates

the use of existing information from previous designs and thus is a better approach for incremental development.

Given that both approaches have advantages and disadvantages, designers will typically apply both design approaches, namely using a top down focus for the initial design, to partition the system into manageable subsystems which are conflict-free. Thereafter, they will apply a bottom up approach in the detailed design of these subsystems, taking into consideration the necessity for conflict resolution and trading it for ease of design.

The major database design techniques described in this paper, those using view integration, will appear to be bottom up approaches, since the integration process is based on individual user views. However, the procedure laid out at the New Orleans Database Design Workshop (New Orleans, 1979) which presents a framework for view integration approaches, recommends a database design procedure that introduces organizational goals and high level information requirements by means of Enterprise Modelling in the step preceding view integration. In other words, this widely accepted design strategy also applies a mixed top down and bottom up approach.

2.2. Database Design based on the New Orleans Database Design Workshop Procedure

In this section the focus will be on the common elements of all view integration procedures as well as on their differentiating characteristics. In short, all integration approaches can be perceived as procedures for view aggregation and schema optimization. One feature of all (comprehensive) approaches will be the ability to resolve differences between views. To permit this, the methods' data models will have to be able to represent objects and object associations. Dissimilarities among view integration procedures will arise primarily from the differences in procedure, the differences in abilities to deal with conflicting information, variations in information requirements, and on the restrictions placed on the initial schema.

View integration is an element of any bottom up database design strategy. This strategy, whose initial input are user requirements and whose final outcome is the implemented (physical) database, has been segmented by various authors (New Orleans, 1979, Teory and Fry, 1982) into the following steps:

1. Requirements Analysis

   to obtain information from users on information and processing requirements, and to analyze this

information in order to resolve conflicts and inconsistencies with the enterprise view. The analysis and incorporation of (global) business constraints adds a top down focus to this otherwise bottom up oriented technique.

2. View Modelling and Modification

   to generate application views and information access requirements.

3. View Integration

   to merge individual views into a global schema.

4. Implementation Design

   to handle issues of integrity, consistency, recovery, security and efficiency.

5. Physical Design

   to ensure functioning and efficiency of the database with a particular database/file system.

In other words, view integration takes as its inputs individual user views (and possibly processing/query requirements) and produces as its output a global database schema.

The most trivial form of view integration is an aggregation of all individual views without alteration of any of them. However, instead of generating a system of interconnected database objects, this method creates merely a lump of individual views. View integration has to go beyond aggregation, it has

to include the reorganization (optimization) of the global schema. The task is to eliminate redundancies and inconsistencies that result from combining overlapping views of users who all may have different conceptual models.

Reorganization of the global schema is intended to increase the descriptive adequacy of the global schema [1]. In addition, it may include the consideration of query requirements which has been a concern in some earlier studies, especially in non-relational database environments (for example Batini et al. (1984a) or Yao et al. (1982, 1985)). For network or hierarchical databases, consideration of processing requirements might result in a trade-off that introduces duplication of database objects to improve processing efficiency.

Even though a variety of researchers choose the same approach to database design, namely view integration, differences exist in the data modelling language used to carry out the integration process. Tightly connected to the data model is the "integration philosophy", alternatives of which have been pointed out by Yao et al. (1982) as (1) view integration based on item level synthesis using frequency information, (2) synthesis using functional dependencies among items and (3) merging of object level structures.

---

[1] Descriptive adequacy is understood as the precision with which the data model describes the world it attempts to model.

The first category is a form of "statistical" view integration, in which frequency information serves as a substitute for cohesion or functional dependency of data items (Dyba, 1977, Sheppard, 1977).

The second category builds database objects, i.e. relational data structures, based on information on functional dependencies. Proponents of this category can be found for instance in Bernstein (1976), Raver and Hubbard (1977), Yao et al. (1982), Casanova and Vidal (1983), and Biskup and Convent (1986, 1985). Most of these approaches attempt to build databases purely based on functional dependencies (and possibly other forms of dependencies) and try to avoid the consideration of the meaning of data objects as much as possible during integration. Later, these approaches will be referred to as syntactic approaches.

The third group of approaches is probably best represented by Batini et al. (for instance Batini and Lenzerini, 1984) and Navathe et al. (for instance Navathe and Elmasri, 1986). Both techniques are based on the E-R model, enhanced by some additional information (generalization/specialization). The fact that these techniques operate on an object level does not imply that functional relationships are not relevant for them.

However, in E-R models, dependencies are represented in the association of attributes to entities or relations.

Since the late seventies, the literature has moved away from statistical approaches to view integration. The main problem of statistical approaches is that they attempt to capture dependency information between data items by means of relative frequency of common use in applications or coexistence in the same file structure. This substitute may often be correct, since experienced file designers will have a good understanding of which data items should belong together (see for instance Weber, 1986 on "intuitive" normalization), but the technique is inferior to ones that concentrate on the actual data dependencies. Thus, within this research, the focus will be on the latter two groups of integration methods only. For these two groups, prototypical integration methods (together with their data models) are presented in the following list.

SYNTACTIC (attribute-level) INTEGRATION

Based on Functional Dependencies only

* Martin (1983) - "Bubble Charting"
* Bernstein (1976) - Relational Model
* Yao et al. (1982) - Functional Data Model
* Raver and Hubbard (1977) - "Bubble Charting"

* Al-Fedaghi and Scheuerman (1981) - Relational Model

Based on FDs and other Dependencies

* Casanova and Vidal (1983) - Relational Model
* Biskup and Convent (1986) - Relational Model

SEMANTIC (object-level) INTEGRATION

* Batini et al. (1983) - Entity-Relationship Model
* Navathe et al. (1986)[1] - Entity-Category-Relationship Model
* Mannino and Effelsberg (1984) - Generalization Assertions
* Teory and Fry (1982) - Semantic Hierarchical Data M.

Not all of these techniques shall be discussed in detail since there exists considerable overlap among them. The following techniques will be discussed: Martin, Bernstein, Yao et al., Casanova and Vidal, Navathe et al., Batini et al. Martin

[1] The method put forward by Navathe and others has gone through various stages and has involved various researchers. An earlier method is described by Navathe and Gadgil (1978) or Navathe and Schkolnick (1978). Other versions include Navathe, Elmasri, and Larson (1986). The method referenced here is the latest published form. It has been extended into database integration by Elmasri et al. (1986).

contributes a not particularly detailed, yet popular integration method. Bernstein presents the first algorithmic and purely syntactical view synthesis method. Casanova and Vidal introduce the first syntactic integration method that includes a richer set of dependencies. Navathe et al. put forward a semantic integration method with a large set of integration cases. Finally, Batini et al. present the (semantic) integration method that best deals with conflicting views.

2.2.1. Syntactic Approaches

Syntactic approaches are design methods in which the view integration procedure does not rely on a designer's understanding of the data during the integration process (nor on "understanding" by the algorithm)[1]. Instead, the algorithms reorganize the initial schema in a purely structural manner independent of the meaning of objects or attributes involved, once certain information requirements about functional dependencies are satisfied. These information requirements are assumed to be satisfied at the outset of the integration procedure. They are not part of the technique.

The syntactic approaches introduced below, give a complete algorithm for view integration and show the "optimality" (author's terminology) of the resulting design. Optimality (Casanova and Vidal) is not a particularly well chosen term, since the design is not optimal in all criteria a database designer might think of. "Optimal" is meant as "achieving the goals set for the design at the start of the integration process" which in particular means the generation of a valid database, i.e. one that satisfies all previously established integrity

---

[1] Ideally the techniques do not rely at all on the designer's understanding. However, at least one method (Biskup and Convent) consults the designer, when the integration algorithm is in a deadlock. Other methods (e.g., Yao et al.) require designer understanding for more complex integration cases, such as removal of redundant functions.

constraints and is free of undesirable data dependencies. We will call the resulting designs from now on "feasible" rather than "optimal". Three main proponents of different syntactic approaches are Bernstein (1976), Casanova and Vidal (1983), and Biskup and Convent (1986). Two additional syntactic approach shall also be mentioned in this context, although they differ from the above three in not being as purely synthetic, not providing a complete algorithm, and in using other data models ("bubble charts" (Martin) and the *Functional Data Model (Yao et al.)*). All approaches, other than Biskup's and Convent's, will be discussed. Biskup and Convent's technique is rather similar to Casanova's and Vidals. Hence, a separate discussion will not be necessary.

Bernstein's approach does not particularly address the view integration problem, but instead the problem of synthesizing a minimal number of 3NF relations from a set of functional dependencies. Nevertheless, its approach is applicable to view integration, since the algorithm does not mind whether the schema descriptions used for relation synthesis stem from one view or from many views. However, the procedure has obviously no means to unify conflicting perceptions of the same data. Contrary to more recent integration approaches such as Casanova's and Vidal's, Bernstein's method relies only on functional dependencies to carry out the relation synthesis procedure.

Martin's approach, Canonical Synthesis, attempts to develop a "canonical data representation"[1]. This method, like Bernstein's, has no formal means for dealing with conflicts between views, not even for naming conflicts. In addition it is much less detailed and much less algorithmic than Bernstein's.

Casanova's and Vidal's technique assumes the existence of user views and complete knowledge of dependencies (integrity constraints) for the collection of user views. It can be summarized by the following integration plan. Given a set of user views and a set of integrity constraints, define as a valid ("proper") database scheme (= global schema) one that satisfies all desirable integrity constraints. Then apply an algorithm that reorganizes the collection of user views into a valid schema by removing all undesirable data dependencies through changes in relation schemes.

Yao et al. require for their approach complete information on entities ("entity nodes"), functional relationships between entity nodes, plus assertions describing true facts about the data model which are not represented in form of entity nodes or relationships. All views are combined in one representation

---

[1] The notion of a canonical representation in data models has been put forward by Raver and Hubbard (1977) and is used to describe schemata which are redundancy-free (no nonessential associations), complete, and correct. Thus a canonical synthesis technique not only integrates user views, but can also extend them to add necessary further details.

which is thereafter subject to removal of redundant functions and redundant nodes. A proof of correctness of the integration result is not given for this approach.

One major limitation of the syntactic strategies, especially of Casanova's and Vidal's, is their extensive information requirements. They assume at least the availability of information on functional, if not also on union functional dependencies, inclusion and exclusion dependencies. It has to be questioned whether it is feasible to generate this information during the view integration process, and how reliable the information will be. With respect to the amount of information, one has to keep in mind that not only intra-view but also inter-view constraints have to be defined. This requirement can increase the number of constraints substantially, it also demands from the designer the comparison of each relation scheme from each view against all other relation schemes, to detect those dependencies. Any incorrect assessment by the designer will potentially result in an incorrect global schema.

A second limitation of these approaches is the restrictions they place on the initial views to make the integration a computationally solvable problem (i.e. only functional dependencies on the key for the initial collection of views).

A third limitation is caused by the purely syntactic treatment of data dependencies. The procedures cannot differentiate between dependencies that are of the same type and involve the same attributes, even if their meanings are different. For example, the functional dependency Employee# -> Department# might in fact represent two different relationships, first, every employee *works for* one particular department, and second, every employee *is located* in one particular department. Thus, while for example employee 6750 works for the information systems department, he resides in the offices of the accounting department. This difference in roles (here, roles of department) has to be incorporated into the attribute names, to allow the syntactic approaches differentiate between the two relationships. I.e., there has to exist a Located_in_Dept and a Employed_by_Dept.

2.2.1.1. Bernstein's Relation Synthesis

This method is described in Bernstein (1976). An implementation of Bernstein's algorithm can be found in Ceri and Gottlob (1986).

The goal of Bernstein's method is the creation of a schema containing the smallest number of 3NF relations for a given set of functional dependencies. Since the procedure does not concern itself with the origin of the functional dependencies, it does not object to the fact that the set of dependencies is taken from more than one schema. Therefore the method can be considered a view integration procedure. The method not only provides a synthesis algorithm, but also demonstrates that the set of resulting relations is minimal and probably in 3NF. The creation of 3NF relations is typically the goal and final outcome of a decomposition process in which larger tables are split into smaller redundance-free components (for example, Ullman, 1980 or Date, 1981). Bernstein, in contrast, generates 3NF relations by means of composition. This makes Bernstein's approach a view integration technique.

The goal of Bernstein's integration procedure is to find the

smallest set of 3NF relations that incorporates all pre-defined functional dependencies that have been defined.

The algorithm developed by Bernstein consists of three main parts. The first part (involving steps 1 and 2 of "Algorithm 2", see below), has the purpose to generate a new set of functional dependencies (FDs) from an arbitrary set of functional dependencies characterizing the data relationships. These new dependencies form the input to the synthesis part. Synthesis (steps 3 and 4 in Algorithm 2) first partitions the set of FDs into groups with identical left sides [1] and then merges the FDs in these groups. The last part of the procedure (steps 5 and 6 in Algorithm 2) constructs relations which are free of transitive dependencies, based on the FDs synthesized in the previous steps.

Algorithm 2:

(1) Elimination of extraneous attributes to produce a set F' of functional dependencies.

(2) Finding of a non-redundant covering C for the set F' of functional dependencies.

---

[1] "Left side" means the set of determining attributes. In contrast, the "right side" consists of the determined attributes.

(3) Partitioning of the covering C into groups of functional dependencies with identical left sides.

(4) Merging of equivalent keys.

(5) Elimination of transitive dependencies.

(6) Construction of relations.

Bernstein's approach does not differentiate among different cases of integration, based on different dependencies within the data at hand. All functional dependencies are treated by the same integration procedure. This is a positive feature of Bernstein's approach, since it simplifies the procedure. In addition, this approach has less information requirements than the two following ones, which also require information on other forms of dependencies.

One major problem of the technique, pointed out by Bernstein himself, is the purely syntactic character of the approach which is the source for the "uniqueness *assumption*". The uniqueness assumption says that only one functional dependency can exist between any two identical sets of attributes. In other words, if two FDs existed, because of a difference in roles of either set of attributes, the technique were not able to pick up the difference. In order to allow the technique to differentiate among different roles, role names have to be introduced as attribute names.

This point leads to another shortcoming of the technique, namely the significance of names. The purely syntactic technique operates on attribute names, being therefore subject to all problems caused by attribute name synonymy and homonymy. However, the technique was not conceived as a view integration technique, which justifies this weakness to some extent. Furthermore, Bernstein's approach does not rule out the development of a pre-integration procedure which could take care of such conflicts and then supply the integration procedure with conflict-free views.

2.2.1.2. Martin's Canonical Synthesis

See for example Martin (1983).

Canonical Synthesis is Martin's approach to view integration. Martin integrates views by first depicting all functional dependencies between data elements (attributes) and then overlaying any two views to generate a third new one. The main focus of his approach is on the elimination of transitive dependencies generated by the integration process. Martin stresses the use of bubble charts, showing data items (attributes) and their functional dependencies.

The procedure integrates views pairwise and consists of seven integration steps for the logical database design.

1. The designer is asked to eliminate any duplicate functional dependencies between any two data items.

2. The designer has to identify candidate keys.

3. All transitive dependencies have to be removed. The purpose of this step is to find and to remove any hidden primary keys, and finally to achieve a 3NF data structure.

4. Introduce so called "concatenated keys". The purpose of this step is to extend the data model to allow the representation of data items that are dependent on the key of more than one already existing data structure, i.e. Price is dependent on Supplier# and Part#.

5. Allocate intersection data to data items. This step deals with relationships that have attributes. If relationships have attributes, they are transformed into record structures [1].

6. Remove M:N relationships[2].

7. The technique transforms structures in which one attribute is owned by two or more primary keys. If such an "intersecting" attribute exists, the data structure is changed to give the attribute a simple owner.

Martin's method has three major limitations. First, the method is not concerned with the removal of conflicts between views,

---

[1] These record structures are similar to entities. Yet Martin does not use the terms entity or relationship to describe data constructs.

[2] Martin suggests that M:N relationships in database, aside from being supported by only few DBMSs, are an unstable data construct, one that is typically replaced by two 1:M structures as part of the design or implementation process. His technique therefore disintegrates any M:N structure into 1:M structures.

and second, it uses attributes as the atomic building blocks of the global schema. Third, the "algorithm" presented is not precise and thus does no, contrary to Martin's statement, allow immediate automation of the process.

Conflict resolution is mentioned only briefly (Martin, 1983, p. 265) referring to the problem of homonyms. All other view conflict possibilities are ignored. For example, Martin is not concerned about relationships or entities modelled incorrectly as attributes. A consequence of neglecting conflict resolution is that Martin's approach cannot be automated, given that conflicts have to be expected in real world applications. Martin has to assume that all conflicts were eliminated by the database designer prior to the integration process. Thus, like Bernstein's method, this one is a view merging procedure, but not a conflict resolution procedure.

The use of attributes as the atomic building blocks generate at least two problems. First, the modeling process based on attributes operates at a very high level of detail. In fact, it might be viewed strictly as a bottom-up approach to database design. The detail in view descriptions creates large amounts of information the designer has to process. Even with a small number of views, an evaluation of the resulting schema becomes very complex and very difficult in terms of redundancies (transitive dependencies). The entity-relationship

approach, in comparison, allows to hide part of this information, namely associations between an entity and its attributes. In the E-R model, only entities or relationships are able to form relationships to other entities or relationships. In Martin's model, every attribute can be related to any other attribute, prior to redundancy elimination. Secondly, the synthesis of attributes to higher level objects is not based on the user's semantic objects (objects meaningful to the user), but instead on functional dependency. The resulting higher level data structures (records, segments, or relations) are therefore expected to have less meaning for the user than data structures based on objects the user chooses to describe his data world (e.g. entity MANAGER). In other words, the results of canonical synthesis may lose some of its descriptive adequacy of real world objects and associations.

This comment is not meant to imply that database design based on functional dependencies is wrong. Yet, the aggregates should represent the real world view as faithfully as possible. There exists more than one possible way to describe a real world object in the data model, canonical synthesis might not allow a representative of this object in the form the user would prefer (i.e., semantic relativism, Brodie, 1984).

Finally, due to its lack of precision, this technique should only

be viewed as a guideline to integration. It still will require substantial designer interaction and designer insight.

2.2.1.3. Casanova's and Vidal's Method

See Casanova and Vidal (1983) for a description of the method, as well as Bishop and Convent (1986, 1985) for extensions.

Casanova's view integration method is a formal approach to view integration based on four types of dependencies existent in a global database schema. Goal of the integration process is the generation of an "optimised" (feasible) schema, optimised with respect to elimination of redundant information and reduction in size, as measured by number of relations [1] in the global schema.

The four types of dependencies (also referred to as integrity constraints) in this approach are: functional dependencies (FDs),

---

[1] In Casanova's language, which is based on Ullman (1980, p. 75), a "relation scheme" refers to the structure of a relational database object, while a relation is an instance of that structure, that is the actual data. Ullman defines *relation scheme* as the list of attributes for a relation.

inclusion dependencies (INDs), exclusion dependencies (EXDs), and union functional dependencies (UFDs).

A *functional dependency* fd, expressed as R:X->Y, is valid iff for any t,u ∈ r, if t[X]=u[X] then t[Y]=u[Y] [1]. For example, in a relation scheme STUDENT[Stud#,Name], if t[X] and u[X] are identical student numbers, they both have to identify the exact same student name.

An *inclusion dependency* ind is expressed as R1[X] c R2[Y], with X and Y being sequences of attributes of equal length. This dependency is valid iff r1[X] is a subset of r2[Y]. For example, UNDERGRAD[Stud#] c STUDENT[Stud#], means that the set of undergrad students is a subset of the set of all students.

An *exclusion dependency* exd is expressed as R1[X] | R2[Y], X and Y again being sequences of attributes of same length. This dependency is valid, iff r1[X] and r2[Y] are disjoint. For example, the set of graduate students and the set of undergrad students would be such disjoint sets of students.

A *union functional dependency* is a functional dependency stretching over the boundaries of one relation. It is expressed

---

[1] R refers to a relation scheme, r is an instance of that relation scheme, X and Y are sets of one or more attributes, and t and u are tuples.

in the form <Ri1:X1->Y1, ... , Rim:Xm->Ym>, as a set of functional dependencies over relation schemes Ri, where all X and Y are sequences of attributes of same length. A UFD is valid, iff a FD that holds in one relation holds in all relations included in the UFD. For example, a UFD <STUDENT:Stud#->Name, UNDERGRAD:Stud#->Uname> means that a student number '83959818' occurring in STUDENT will identify the same student name 'Jones' as the student number '83959818' in UNDERGRAD.

The last example gives some indication of the purpose of the above dependencies. They will be used to identify and eliminate sources of redundancies. Given complete information on the above dependencies, a procedure is defined that will transform the combination of all views into an integrated global schema. Complete information on dependencies necessitates complete information on all attributes in all relations of all views, plus complete information on domains of attributes. Given this information, the problem of homonymy or synonymy does not arise, because the names of relations or attributes are almost irrelevant. All the above information is assumed to be unambiguous. In other words, there will be for instance no disputes between different views concerning dependencies or domains of attributes. Hence, conflicts are ruled out by definition.

A view integration based on Casanova's and Vidal's method involves the following steps. First, for every view, define the above described dependencies. Second, combine the views by lumping them together and by defining additional constraints of the above types, to describe the relationships between the elements (relations) of different views. Third, integrate ("optimize") this schema by removing redundancies in the combination of views.

The first major problem of this integration method, as stated by the authors, is that it is computationally hard. The problem is PSPACE complete (it fits into finite computer memory space, but can run indefinitely). Casanova points out that the optimization problem may not be decidable, even if nothing but FDs and INDs are considered (see also Casanova and Fagin, 1982).

Another major problem concerns the information requirements of this technique. The approach requires large amounts of ambiguity-free information. Since it cannot deal with partially incorrect user views (wrong perceptions of data), it cannot be used to resolve conflicts caused by inconsistencies in user views.

A further limitation on Casanova's and Vidal's approach results

from its applicability to only so called "restricted" schemas.

The following restrictions apply to the input of the view integration procedure.

(1) All functional dependencies apply only to the (single) key. Thus, there are no transitive dependencies existing.

(2) Any inclusion dependency applies only to the key attributes of the relations involved.

(3) Any union functional dependency must apply to the key attributes (as the left argument of the dependency) for all relations involved and can only describe a dependency of a single attribute on the key (" ... if

<Ri1:X1:->Y1, ... ,Rim:Xm->Ym> is in C, then X1=...=Xm=Ki1=...Kim and |Yj|=1, j∈[1,m]").

(4) Any attribute of any relation can appear in at most one union functional dependency (" ... for any Ri∈S and any attribute A of Ri, A occurs in at most one UFD in C"). Note that this restriction is violated in Casanova's example. The restriction may only refer to dependent attributes, not to key attributes.

(5) All exclusion dependencies apply to only key attributes.

Especially restriction (4) seems like a significant limitation to the integration problem. Real world databases will have to serve as an indicator of how strong this limitation is.

2.2.1.4. Functional Data Model Based Integration

For references to the method, see Yao, Waddle and Housel (1985, 1982).

In contrast to many other syntactic integration methods, Yao et al. present a view integration approach based on Shipman's (1979) Functional Data Model. Within the Functional Data Model (FDM), data can be described in form of two constructs, nodes (to represent entities and value sets) and functional relationships. Nodes can be either *simple* nodes (value sets), or *tuple* nodes (cartesian product of n>1 value sets). Functions, mappings from a domain into a range, can be *functional* (n:1), one-to-one, or identity (1:1 mapping into identical value) and can be partial

(lower degree 0), or total (lower degree 1). Assertions are added as a further means for describing data, to increase the descriptive power of the model. Assertions describe true facts about data, i.e. that one set of data is the subset of another.

Views are depicted in form of nodes and relationship constructs (in a graphical representation). Therefore, complete information on entities and attributes, their domains and their relationships has to be available. Aside from this information, the approach also compiles information on the queries to be issued on the database. Database transactions, represented by means of a *Transaction Specification Language* (TASL) are kept together with the views and are updated whenever view updates require query modifications. One further piece of information is collected, namely information describing the physical data in terms of quantities of members of a set, i.e. the number of students, professors, courses in a university database. Quantity information is later used in heuristics to identify non-redundant functions in the model. The treatment of transaction and quantity information will not be subject of the following discussion.

The technique incorporates two integration operations: the removal of redundant nodes, and the removal of redundant

functions. According to Batini et al. (1986, p. 343), Yao's technique performs view integration on all views in parallel ("one-shot n-ary"). However, this is true for the integration of redundant functions only. Integration of nodes is performed on a single pair of nodes at any point in time.

A node is redundant if it represents the "same set of values" as some other node. Note that the "same set of values" (Yao et al., 1985, p. 338) does not mean the two sets are in fact identical. It is sufficient that one is a subset of the other or that they are overlapping. If two nodes represent the same set of values, they will be merged. The integration can only be performed if any existing functions between the two nodes are identity functions. Nodes A and B are merged by creating a new node C which is the union of A and B. All functions that had A or B as domain or range will be redefined to have C as domain or range.

In addition, if A and B are not identical, a *separation node* SEP will be created that stores information to differentiate between the two original nodes, given the new node C. If a separation node has to be created, also a new functional dependency will be created with C as its domain and SEP as its range. A separation node can be viewed as a set of indices that indicates, by means of pointers to the new combined set C, the origin of each value in the new set.

The second integration operation removes redundant functions. The goal is to remove a functional relationship A->C, if it can be replaced by other functions, i.e. the two functions A->B and B->C.

The authors point out that the redundancy of a function can only be decided upon analysis of data semantics. In other words, the meaning of functional relationships has to be known to decide on its redundancy. This is one of the criteria which differentiates Yao's et al.'s technique from the previously discussed completely syntactic approaches.

The method proposed by Yao et al. has a number of limitations. First, the method is incomplete. View integration is restricted to only three cases of node integration and one case of function integration. Hence, the technique will not be able to adequately represent all possible types of set relationships between view objects (for example, two nodes are not overlapping but have a common superset).

A second weakness concerns the integration procedure. The procedure is not defined exactly. For example, does function removal always precede node removal? Does the procedure perform

node merges always on single pairs of nodes, or on an arbitrary number of nodes at the same time.

Thirdly, the technique does not show the transformation from FDM into database objects, i.e. relations, or more likely, network constructs.

Fourthly, use of physical database information in logical database design is not particularly useful (i.e., record quantities).

Finally, the method has no means for dealing with conflicting information, i.e. with naming conflicts or with type conflicts.

2.2.2. Semantic View Integration Approaches Based on the E-R Model

Semantic approaches use data objects that are meaningful to the user. Since they require a higher level of understanding of the meaning of objects, these approaches are

typically interactive, that is, they demand designer intervention during the integration process. Designer intervention is for instance necessary to settle certain naming or type conflicts, and even more important, to interpret the meaning of data objects or object relationships.

Since semantic integration approaches focus more on the meaning of the data objects than on only structural information, the data models used to represent views have to be able to capture data semantics. In this section, integration techniques based on the Entity-Relationship (E-R) model, will be introduced. The E-R model itself is not particularly rich in its ability to represent data semantics. Therefore, the methods discussed below (both Navathe et al. and Batini et al.) use an extended E-R model which for instance provides the capability to model *categories* which are generalizations of entities[1].

Interactive approaches take advantage of having access to the database designer during the integration process for conflict settlement or information clarification. In consequence, they permit the integration of less restricted data models and to perform a larger portion of the integration process i.e. include conflict analysis. On the other hand, the reported interactive approaches typically do not include a complete

[1] Not all integration methods representing data semantics have to be based on the E-R model. For example, Teory and Fry (1982) developed a method based on a semantic hierarchical model.

algorithm for the integration process and do not exactly specify the restrictions placed on the data model (such as consistency).

2.2.2.1. Navathe's and Elmasri's Approach

Description of various aspects of this method can be found in Navathe, Elmasri, Larson (IEEE 1986), Navathe and Elmasri (IEEE 1984), Elmasri and Navathe (1986), Elmasri et al. (1987).

Navathe's and Elmasri's approach concentrates on the idea of *object class integration*. The entity-relationship model is extended to an entity-category-relationship model where a category refers to a class or an object type (common role or subclass). The atomic elements of this approach are entities, categories, relationships, and attributes.

Two types of categories are used, common role categories and subclass categories. A common role category is one that represents a common property of two or more otherwise different sets, i.e. the category OWNER represents a common role for both PERSON and COMPANY, who may both be owners of a vehicle.

A subclass is a specialization of an entity set, i.e. the VEHICLE entity set has subclasses CAR and TRUCK. Common role and specialization will have an impact on inheritance of attributes.

The procedure consists of three steps: pre-integration, object integration and relationship integration.

Within pre-integration three tasks are performed. First, naming correspondences are established, resolving the problem of inter-view homonymy and synonymy. Synonymy and homonymy refer to the problem of different names designating the same real world object or identical names designating different real world objects (concepts [1]). The second task is the identification of candidate keys for object classes. The third task is the definition of domains for object classes. Domains play an important role in Navathe's technique. The purpose of defining them within the pre-integration step is to gather information for the recognition of identical or related real world objects. I.e. if two objects have the same domain, it may be suspected that these objects are identical.

Integration of objects (entities or categories) is the second phase of Navathe's scheme. In this phase, information on

---

[1] Navathe uses the term "concept" to refer to a real world object, while Batini uses the term "concept" for a data model element such as an entity, attribute or relationship.

domains is used to determine similarities or dissimilarities among view objects. Navathe analyses the following cases: **identical domains, contained domains, overlapping domains,**and **disjoint domains.**

INTEGRATION OF OBJECTS

The integration of relationships follows the object integration step. Navathe points out that for relationship integration both structural and semantic considerations are important. Relationships are classified according to three criteria: **degree** (which is not the mapping ratio but the number of objects involved in the view (construct)), **roles** of object classes involved in the relationship, and **structural constraints**, such as mapping ratios.

The relationship integration process evaluates the above information in the following sequence of importance: degree information (same/different degree), role information (same/different roles), and structural vs. domain constraints resulting in 8 integration cases.

The main points to be learnt from Navathe's approach are the strength of domain information and category information for view integration, the possibility of simultaneous n-object integration (in some instances), and the relevance of particular

pieces of information during the relationship integration phase.

2.2.2.2. Batini's Approach

For references see for instance Batini et al. (1984a, 1983), or Batini and Lenzerini (1983).

Batini's approach performs integration on the atomic elements of the entity-relationship model, entities, relationships, and attributes. View integration is presented as an iterative process which aggregates views pairwise. Whenever conflicts arise between the two views, a conflict resolution process is invoked and carried out interactively with a database designer. The technique starts out with a name conflict analysis, identifying intra-view homonyms and synonyms and removing them. These can be naming conflicts for the same concepts (e.g. entity) or for different concepts (e.g. entity vs. relationship). This step is followed by a type conflict analysis which results in the same real world object being represented by the same concept in different views (e.g. MARRIAGE always an entity) and in an adjustment of cardinalities (mapping ratios) and optionalities of attributes and relationships in different views to make them identical.

Finally, merging and redundancy analysis superimposes the adjusted views and removes redundancies such as redundant cycles[1].

Batini's method builds a global schema iteratively , integrating two views into a temporary global schema and adding additional views to this schema until all views have been consolidated. The two main elements of the technique are Conflict Analysis (together with merging) and Redundancy Analysis, with the main focus on Conflict Analysis. Unlike other authors such as Martin, Batini et al. address the problem of inconsistencies between different users' perceptions of the world and different naming conventions systematically (but not completely).

The goal of Conflict Analysis is to detect and solve all existing conflicts between two representations (views) of the same classes of objects. Two types of conflicts are tackled, naming conflicts and type conflicts. Naming conflicts arise if the same data model concept (entity, attribute or relationship) is labelled differently (synonyms), or if different concepts are labelled with the same name (homonyms). Type conflict analysis determines whether objects have compatible concepts (types) and adjusts them if necessary.

---

[1] The technique also includes quantitative and procedural aspects to arrive at a procedurally more adequate schema where frequent database operations can be carried out more efficiently.

To define homonymy and synonymy, Batini et al. refer to the view representation of real world objects. If a view S1 represents two different real world objects with the same concept (name), this is called an intra-view homonym [1]. Accordingly, synonymy refers to the same real world object being represented by two different concepts within one view. Given these view inconsistencies, Batini identifies a number of possible scenarios and solution alternatives. Interesting in Batini's procedure is the focus on only intra-view inconsistencies. Inter-view inconsistencies are, at least in this step, ignored.

A second step in the naming conflict analysis is the so called analysis of concept likeness or unlikeness. The attempt in this step is to find out whether a concept that has the same name in two different views possesses different "neighbor properties" (concept unlikeness), or whether concepts have different names but some common neighbor properties (concept likeness).

The next step in Batini's approach is the Type Conflicts Analysis. Its purpose is to assign the same concepts to identical real world objects in different views. I.e. if MARRIAGE were a relationship in one view, but an entity in the

---

[1] Usually one would expect inter-view homonymy to be the more important issue, two views supplying the same name to two different real world objects.

other one, at least one of these representations would be change to let MARRIAGE be represented by only one concept. The conversion of concepts is restricted to only atomic concepts (entity, attribute relation) and results in two views using same names and same concepts to describe real world objects.

The second part of type conflict analysis is compatibility checking, a process which analyzes, among the now quite similar views, whether cardinalities (mapping ratios) are identical. Compatibility checking also discovers differences in the optionality of attributes and relationships. According to Batini et al., differences in cardinalities point to errors in one of the views, or alternatively to a containment relationship.

Once all conflicts have been resolved, Merging and Redundancy Analysis follow. In merging, the conflict-free views are superimposed. Redundancy analysis removes redundant alternate paths between objects. Redundancies can occur because multiple paths are semantically equivalent.

Batini's technique concludes with an update of the individual views to make them consistent with the newly generated global schema and with an alteration of the global schema to include procedural and quantitative aspects.

Batini's approach provides a procedure for the integration process together with some exact conflict resolution algorithms, yet, based on its description in the literature, it cannot be automated. The method does not clarify when a particular integration rule has to be applied, or which information has to be available (Navathe is more exact in this matter, basing his resolution scheme on information on class membership).

2.3. View Integration Cases

The investigation of the above view integration techniques found considerable overlap among techniques with respect to their integration capabilities. When techniques differ, they typically deviate in their conflict resolution capabilities and in aspects of the integration method related to their individual data models. The more recent techniques typically provide a richer set of cases for conflict resolution. Consensus exists with respect to the integration cases for sets (of entities or relationships) whose connection to each other is known, as represented in the following eight cases.

Object Class Integration:

(1) Identical object classes

(2) Contained object class

(3) Overlapping object classes with a common superset

(4) Disjoint object classes with a common superset

Relationship Integration:

(5) Relationship identity

(6) Relationship containment

(7) Relationship overlap with a common superset relationship

(8) Disjoint relationships with a common superset relationship

The table below depicts which of the above cases are supported by the techniques presented in the chapter ('y' indicates the technique's ability to deal with the case, a blank indicates that no reference has been made to how this case would be solved).

	Cases							
Technique	1	2	3	4	5	6	7	8
Martin		cases do not apply						
Bernstein		cases do not apply						
Casanova and Vidal		y	y	y	y	y	y	y	y
Yao et al.	y	y	y					
Navathe et al.	y	y	y	y	y	y	y	y
Batini et al.			y	y	y			

2.4. Conclusion

This section shall point out the comparative strengths and weaknesses of syntactic and semantic integration approaches.

Syntactic approaches

Restricted Data Models

Syntactic approaches place considerable restrictions on the data model with which views are represented. For example, Biskup's and Convent's model is restricted to only *proper* database schemes which impose restrictions on the fields to which constraints can apply. Typically, all dependencies have to involve the key or a key attribute. Bernstein refers in his technique to the *uniqueness assumption* which dictates that only one functional dependency may exist between any pair of fields. He also points out that this restriction may lead to the necessity to bury semantics in data item names[1].

---

[1] For instance that two fields Emp# and Dept# may be related by the functional dependency "employee is located in department" or by another dependency "employee is employed by department". Syntactic models require a renaming of at least one of the Dept# fields in this case.

No Conflict Analysis

The syntactic approaches operate under the assumption that the data required for integration is complete and correct. Therefore, conflict analysis is not part of the techniques. The techniques can deal with simple conflicts, for instance with synonymy, if identity is established by means of constraints.

No Ability to Deal with Incomplete or Inconsistent Data

Again, the ability to deal with incomplete or inconsistent data is outside the scope of syntactic integration techniques. At least one technique, Biskup's and Convent's, will, when an unresolvable problem is encountered, interact with the designer to resolve the problem in order to allow a continuation of the integration process. However, this form of exception handling is not a planned form of conflict analysis, but a measure to let the technique continue when none of the integration cases is considered performable by the technique.

Extensive Information Requirements

The major information requirement of syntactic approaches is knowledge of dependencies between data items. Since all dependencies are defined on the attribute level, this information requirement exceeds that of semantic approaches which represent dependencies on the entity level only. Furthermore, the

requirement to also define inter-view constraints can lead to an exponential explosion of constraint definitions.

Computationally Hard

Casanova and Vidal and Biskup and Convent point out the computational requirements of their techniques.

Provide Integration Algorithm

One major advantage of syntactic approaches is the completeness of procedures. The approaches, instead of outlining only particular integration cases, typically present a procedure that upon termination has produced an integrated database schema.

Show Optimality (Feasibility) of Design

Another major advantage of syntactic approaches is their ex-ante specification of design objectives and their proof of achievement of these design objectives.

Semantic approaches

Require Designer Interaction

Based on the fact that semantic approaches operate on objects meaningful to users but often not meaningful to the integration mechanism, these approaches require designer interaction for interpretation of objects and for conflict analysis.

Cover Larger Portion of the Integration Process

In addition to the operations contained in syntactic approaches, semantic approaches include also conflict analysis procedures, and pre-integration procedures (see Batini et al., 1986) which are concerned, among other factors, with data gathering.

State/Solve More Integration Cases

Semantic techniques identify and solve more integration cases since they include not only the simple eight cases based on set inter-relationships as explained above, but also cases involving conflicts.

Allow Less Restricted Data Models (i.e., non-similar keys)

Semantic methods perform integration based on the meaning of objects, not (exclusively) based on structural similarities. Therefore, a semantic approach can possibly integrate two object classes in which one is a subset of the other, even when the object classes have different keys.

Less Complex

Semantic approaches simplify the integration process in two ways. First, the amount of detail is much less than that of syntactic approaches, since the focus is on entity-level items. Second, semantic data items are more meaningful to humans than arbitrary collections of fields held together only by dependencies.

Deal with Database Objects Meaningful to Designers and Users

The outcome of the design process also is more profound for the database user, since the database objects are meaningful to database users. A syntactic integration, based purely on dependencies, may derive database objects that are not suggestive to the user. One of Batini et al.'s (1986) criteria for goodness of a design is understandability.

Do not Provide Complete Procedures

One of the major weaknesses of the semantic approaches is the limited description of complete procedures for integration. Even though a variety of integration cases is outlined, the description of sequences of integration steps and possible re-iterations is, if not missing, at least very terse. In addition, when dealing with conflict analysis, semantic approaches are not complete in their analysis, nor do they show the missing elements of the analysis.

Do not Present Proof of Optimality of the Design

A consequence of the incompleteness of semantic integration procedures is their inability to demonstrate the optimality of the final design. No semantic procedure states a point at which the procedure terminates and has achieved a final design. Also, the objectives of semantic approaches involve the criterion of understandability which cannot be measured as easily as, for instance, adherence to normal forms. Yet, even for the criteria that can be shown more easily, semantic approaches typically do not provide any proof of optimality or feasibility.

Overall, conflict analysis and resolution is the common weak point in all integration techniques. Three causes of this deficiency are:

(1) syntactic techniques cannot deal with conflict analysis at all. They ignore conflicts in general.

(2) if conflict analysis is done, it is often done unsystematically. Batini et al. (1983) perform the most thorough analysis by separating naming conflicts from type conflicts and then analyzing them separately. This analysis is still not sufficient to identify, let alone solve, all possible causes of conflicts.

(3) conflict analysis is biased by information requirements considerations. Only cases are considered for which information is easily available (i.e. mapping ratios), which are most prominent (i.e.

synonyms), or which are of particular concern due to the data model chosen (i.e. semantic relativism, or mapping ratios). In contrast, a more systematic procedure should be aware of all possible conflict cases and then should determine the information requirements to solve them. Thus, even if the technique is not able to resolve all conflicts due to lack of information, it is at least aware of the possibility of existence of a certain conflict, and thus of its own limitations!

Batini et al. (1986) summarize the lack of research in the area of conflict analysis as follows:

> ... Simple renaming operations are used for solving naming conflicts by most methodologies. With regard to other types of conflicts, the methodologies do not spell out formally how the resolution process is carried out; however, an indication is given in several of them as to how one should proceed. ... (p. 348)

And further:

> ... It is interesting to note that among the methodologies surveyed, none provide an analysis or proof of the completeness of the schema transformation operations

> from the standpoint of being able to resolve any type of conflict that can arise. ... (ibid.)

The solution to these problems will therefore form the core of this research project.

3. SYSTEM FOR VIEW INTEGRATION

3.1. Research Question and Contribution to Knowledge

Research question 1:

1.1 Can a view integration process be formalized which transforms any collection of conflicting views into a complete and consistent global schema?

1.2 Which conflict cases have to be solved in the process?

The purpose of this research question is to solve the conflict analysis problem, initially neglecting information requirements. Assuming sufficient information, a mechanism is to be developed that allows the detection and solution of all view conflicts. The view integration mechanism shall be able to convert a collection of views into a complete and consistent global schema, using the previously introduced group of 8 simple integration cases for set-subset relationships, as well as others to be defined later.

Based on the sufficient information assumption, conflict cases can be described and solved without concern for the difficulty

of data gathering. Instead of mixing the conflict problem with the information requirements problem, question 1 deals only with the former one.

The first step in answering this research question will be the identification and solution of a complete set of conflict cases. The second step will focus on the development of a procedure to carry out the integration, based on the set of cases.

Research question 2:

2.1 What information can be used for the integration of user views into a global database schema when the necessary information is not explicitly available?

2.2 How can this information be gathered in a process that limits designer interrogation to a feasible level?

The basis for the second question is the assumption that in all practical situations the necessary information about views is not unavailable, or too difficult or too costly to gather. Therefore, even though the answer to question 1 reveals which information is necessary to perform view integration, all this information cannot be expected to be present. Hence, substitutes

have to be found for the missing information; substitutes that can be either known by the program (program's knowledge base) or which can be easily gathered through a minimum of interaction with the database designer.

The term "substitutes" may be better phrased as "operationalizations" of information on some database concept. For example, given sufficient information, the system will know that two relationships have identical meaning, even if their names differ. A system with insufficient information has to rely on operationalizations of the "meaning" concept to assess the identity of such relationships. Domain identity and identity of neighbour entities may be such operationalizations.

The intention behind the second question is not to find "tricks" to solve the limited information problem, but to identify substitute information; information items that allow the assessment of concepts such as "meaning", which are difficult to grasp by a computer. The knowledge of these substitutes will teach us also about alternative information requirements of data modelling techniques.

Even though availability of integration information is an important concern, the apparent lack of substitute information should not limit the comprehensiveness of the integration mechanism. Conflict analysis, at least in principle, should

not be based on the convenience with which relevant information items can be produced. On the contrary, question 2 should ideally attempt to find information sources for all requirements raised in question 1. In other words, question 1 aims at stating and solving the integration problem in a sufficient information environment, question 2 aims at solving that integration problem in a limited information environment.

In order to decide on the best information substitute in the limited information environment, questions have to be raised on the suitability of certain pieces of information. The following list gives suggestions the selection should be based on. The term "concept" refers to the information concept to be used as a substitute:

1. how well does the concept represent the underlying information that is necessary for database design?
2. when does the concept fail as a surrogate for the underlying information?
3. can the user/database designer provide the information, or can it be gathered from some other source?
4. how easy can the information be gathered during the integration process?

The last point brings up the issue of developing a process for view integration which requires the least amount of interaction by using as much inferred information as possible. Given sufficient information, designer interaction is ideally not necessary [1]. Given limited information, designer interaction will be necessary. Therefore, a process developed to answer research question 1 may require redesign to increase its usefulness. For example, a useful design change would be a modification that enabled the technique to apply previously gathered information to later stages of the integration process. One has to keep in mind that a program will quickly lose its appeal as a productivity tool, if it repeately asks the designer trivial questions. Such redesign does not change the integration cases, but the sequence of the analysis, as will be demonstrated later in the context of heuristics.

So, while the primary interest within this research is the discovery of an exhaustive set of conflict cases and resolution principles, the secondary interest is the development of an efficient integration procedure through choice of surrogates for certain pieces of information and through choice of a

---

[1] The integration mechanism which assumes information availability is implemented in form of a programmed procedure that directs all questions concerning information requirements back to the designer (user of the mechanism).

sequence that allows to make inferences from the data already gathered.

Contribution to Knowledge:

A main result of the study is prescriptive knowledge, knowledge on how view integration should be carried out. The starting point for this knowledge is the set of integration cases identified by the consensus of previous integration approaches. This research develops a systematic framework which encompasses the available integration knowledge (see chapter 2) as well as a set of additional cases for conflicting views. The research also demonstrates the framework's completeness.

Another result of the study is a set of heuristics for efficient execution of the integration process with limited information. The assumptions underlying these heuristics will be clearly stated. For example, suppose, the following heuristic is implemented. "IF object A is identical to object B and object A will have the same construct (i.e., be both entities). Heuristics are accompanied by explanations concerning their generalizability and effects of their failure.

Prescriptive knowledge encompasses knowledge on integration laws and integration process rules while descriptive knowledge

encompasses process and information substitution rules. At the end, this research presents a set of information requirements and a set of integration rules which together are sufficient to perform the integration process including conflict resolution as well as an efficient integration process.

Another contribution to knowledge can be derived from this research. It is an extension of the relational data model regarding data semantics. It is well known that the relational data model in its current form is not well suited for capturing data semantics. One step towards capturing data semantics is the data dictionary which keeps information on database items, either in computer or human interpretable form, i.e. on data types, or the meaning of the data in the relation tuples. A large amount of the dictionary information can be generated, virtually effort-free, as part of the design process. Thus, the outcome of the design process may not only be set of relations, but also a data dictionary. The view integration approach suggests information that should be captured in data dictionaries but has not been captured yet. This information may include data concerning the meaning of database objects. Future database management systems could have facilities to interpret this data in order to support the users and the system itself, for instance to improve the integrity of the database (fully integrated semantic dictionary) or at least to improve user understanding of database data. For example, the

database could explain to the user that MANUFACTURER is a subclass of SUPPLIER which supplies parts and also manufactures these parts or that SUPPLIER is a person or organization that in the present is supplying parts or in the past has been supplying parts.

3.2. Approach to the Problem

3.2.1. Overview

The problem solving approach chosen for this research is determined by the ill-structured nature of the view integration process and the previous research in the area. Previous research has identified several conflict cases and their solutions without assuring us that the problem has been solved in its entirety. With the first research question, the attempt is made to develop a complete conflict resolution method. This task is simplified by the information availability assumption. To answer this research question, an analytical problem solving approach was chosen. This approach identifies

all possible conflict cases for any pair of objects[1] from different views and shows that the list of conflict cases is complete. The list contains 17 general conflict cases with various subcases.

Completeness has to be shown for this list. The demonstration of completeness rests on the assumption that all criteria which differentiate any two views or parts thereof (i.e. different names for the same object type, different meaning of two object types) have been identified here. Once all criteria are known by which objects can be distinguished, all possible combinations of criteria can be easily generated. The latter part of the argument has to justify why some of the possible combinations are irrelevant or why they are similar to other, already identified ones.

3.2.2. Outline of the Problem with Available Information

Even though some of the previous integration approaches have dealt with the conflict analysis (conflict recognition) problem in a systematic manner, their conflict

[1] Pairwise integration has been the procedural choice for most previous integration methods (see Batini et al., 1986). Only recently, some researchers (i.e., Navathe) have demonstrated parallel integration techniques for more than two views, applicable in certain conflict situations.

classification schemes were not suitable to identify all possible combinations of object differences. Consequently, they have failed to identify some conflict cases. In this section, a categorization is presented which overcomes this weakness.

The cases discussed below represent an exhaustive list of possible conflicts between any two objects from different views. It will be argued that any possible conflict case is covered by the technique and that after resolution of conflicts, views are in a form in which they can merged. It will also be argued that there exists a "causal ordering" (compare Simon and Ando, 1963) of conflict resolution cases which determines the sequence of steps within the integration process. Hence, an integration procedure following this ordering will be outlined.

Object comparison

Object comparison focuses on the detection of any identity or difference between two objects from different views. Objects may be of type entity, relationship, attribute. For example, a designer arbitrarily picks one object from each of two view and wants to determine their identity or difference. To do this, he should choose four general criteria by which to evaluate objects:

(1) Name - are the objects' names identical?

(2) Construct - are both objects represented by the same construct?

(3) Meaning - do the objects have the same meaning?

(4) Context - are the objects associated with the same neighbor objects in both views?

The name criterion is a straightforward one and well described with in the literature. In short, identical objects should have the same name, different objects should have different names. Otherwise, the object pairs are synonyms or homonyms.

Construct refers to the object type, i.e., entity. Identical objects should have the same construct, to allow their merging. Previous research has given many examples of construct mismatches and their resolution.

Meaning is the most difficult criterion. Instead of a lengthy discussion about the interpretation of "meaning", at this point the following working definition will be used: two objects have the same meaning if they both represent the same real world object. Database design is a form of modelling. Real world objects are represented by database items. If two database items are both models the same real world object, they have the same meaning. In previous research, meaning has not been explicitly discussed as discriminating criterion,

possibly because the meaning criterion is very difficult to assess. For instance Navathe and Elmasri (for example, 1986) have frequently used domains or mapping ratios as discriminating criteria instead. We may think of domain information and mapping ratios as operationalizations capturing part of the meaning concept. Explicit representation of the meaning dimension will result in a simple and clear distinction of conflict cases[1].

Context refers to the objects that are immediate neighbors of an object. By definition, an entity will always have an empty context[2]. A relationship's context are all entities it is associated with. An attribute's context is the entity or relationship it belongs to. Context also has not been explicitly represented in previous research, even though previous researchers were aware of the importance of context, as their conflict recognition and resolution examples show.

Based on the four criteria and two states of each criterion (identity or difference), a 2 x 2 x 2 x 2 matrix with 16

---

[1] The main difficulties of meaning representation are completeness of the representation and differences in user perspective. For example, when asked about the meaning of "lion", most people may reply "dangereous animal", while a lion tamer would probably reply "livelihood". These are two different, incomplete interpretations which are both acceptable. For a discussion of the meaning concept consult Russell (1946).

[2] Even though entities have no context by definition, it may be useful later to think of an entity's context as the relationships it is involved in.

general cases of identity and difference of object pairs can be constructed. To exemplify the principles of conflict recognition and resolution, only the first three criteria, name, construct, and meaning, will be discussed in more detail and represented graphically in this section (see Figure 1). For now, the conflict problem can be simplified by assuming that whenever two objects have identical meaning, their contexts will be identical. Whenever their meanings are different, their contexts may be different or identical. The subsequent sections will deal with the full integration problem, allowing variations in context, even if meaning is identical.

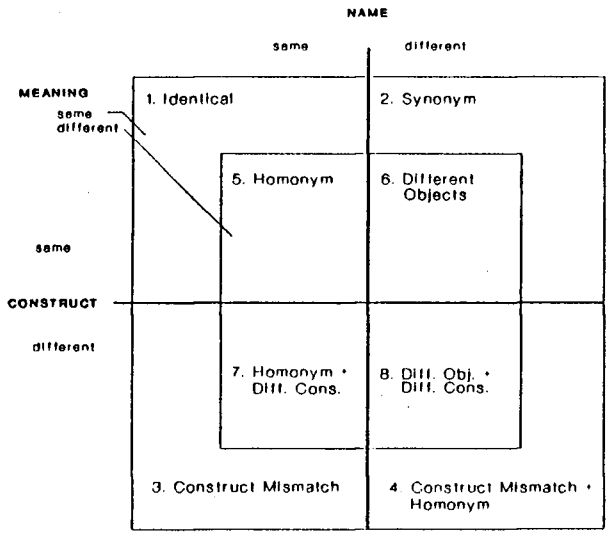


Figure 1: Object Comparison Matrix

Each of the cases depicted in Figure 1 will be briefly presented below. The focus of this discussion shall be on the cases,

not on their detailed solution. Unless solutions are simple or necessary for the discussion, they will be postponed to subsequent chapters. Note that not all cases below describe conflicts. For instance, if two objects are identical (Case 1), they can be merged without modifications. Other cases, such as synonymy (Case 2) require an object change.

Case 1: [Name:same; Meaning:same; Construct:same]
This condition corresponds to cases 1 and 5 from previous research (see chapter 2). Two objects are identical in all factors.
Example:
View 1: CUSTOMER (entity)
View 2: CUSTOMER (entity)
both describing the same customer object type.
The notion of identity is not only meaningful for entities, as exemplified, but also for identical relationships linking identical entities, and for identical attributes of identical entities (identical context).

Case 2: [Name:different; Meaning:same; Construct:same]
This is the case of a synonym. Both objects are identical but carry different names. Note that both objects have the same construct (i.e., entity).
Example:
V1: CUSTOMER (entity)

V2: BUYER (entity)

both describing the same real world customer object type.

Case 3: [Name:same; Meaning:same; Construct:different]

This case describes a situation where the same object is represented by different modelling constructs. This case will be referred to as construct mismatch. Brodie (1984) refers to this difference in construct as "semantic relativism", e.g., when the same object is represented as an entity in one view and as a relationship in another view.

Example:

V1: MARRIAGE (entity)

V2: Marriage (relationship)

Both views describe marriage objects. Both views use the same name, but a different construct. For view 1, a marriage is an entity (probably with husband and wife attributes), for view 2, a marriage is a relationship (probably linking two person entities). The solution to this case is a change in one of the constructs, either making the entity a relationship or vice versa. At the end, each object should be represented by the same construct in all views.

This example describes only one of many possible construct mismatch scenarios.

Case 4: [Name:different; Meaning:same; Construct:different]
This case is closely related to the previous one. Again, both objects have the same meaning, but this time they not only have different constructs, but also different names. Therefore, identity of objects is disguised even further, by name differences on top of construct differences.
Example:
V1: MARRIAGE (entity)
V2: Married_to (relationship)
While both views use almost similar names, to a syntactic processor, the names will be different.

Case 5: [Name:same; Meaning:different; Construct:same]
This case marks homonyms. The objects carry the same name, but have different meaning. The objects have the same construct (i.e., entity).
Example:
V1: SUPPLIER (entity)
V2: SUPPLIER (entity)
Here the same name SUPPLIER is used for both suppliers (currently supplying the product) and for manufacturers (who manufacture the product and may be potential suppliers).

Case 6: [Name:different; Meaning:different; Construct:same]
This case may refer to a trivial situation in which two objects are different in meaning and name, but have the same construct.

On the other hand, it may refer to a number of more complex situations of non-identical but related (i.e., superset-subset relationship) objects.

Example 1: trivial situation

V1: EMPLOYEE (entity)

V2: DEPARTMENT (entity)

Example 2: related objects

V1: STUDENT (entity)

V2: UNDERGRAD (entity)

The entities in the first example refer to two different real world objects which are not related [1]. The objects represented in the second example are related, namely through a superset-subset relationship. Whenever there exists such a connection between two items they cannot be treated as independent. The eight cases extracted from previous research provide solutions for such non-identical but related sets.

Case 7: [Name:same; Meaning:different; Construct:different]

This case captures homonyms. Again, the name of two objects is the same, but they differ both in meaning and in construct used. Note that this case may also contain objects that have different meaning but are related to each other (as in Case 6). Example:

V1: SUPPLIER (entity)

---

[1] "Related" is used here to express that two object classes are either overlapping or are contained by a common object class.

V2: Supplier (attribute)

The name supplier is used for both an entity and for an attribute, and the attribute does not refer to the same supplier object (i.e., refers to a manufacturer object).

Case 8: [Name:different; Meaning:different; Construct:different]

This case describes objects which are different in every respect, meaning, name and construct.

Example:

V1: SUPPLIER (entity)

V2: Department (attribute)

Supplier and department are different objects altogether, with no similarities between them. Again, this exemplifies the trivial form of the case. But again, objects may also be related.

The above eight cases fall into 2 main groups: objects that will be ultimately completely identical and objects that are different. Whether an object belongs to the first or the second group is determined by their meaning dimension. The first group consists of cases 1,2,3, and 4. The second group is represented by cases 5,6,7,and 8. In either group, certain cases describe stable states. In the first group for example, case 3 (semantic relativism) becomes a case 1 (identical items), once different constructs are eliminated. Case 4 becomes a case 3, once objects are renamed. Within the group

of different objects there exist two stable states. If objects are related (i.e., one is a subset of the other), they will ultimately belong to case 6, i.e., after renaming from case 5. If they are unrelated, they will belong to case 8 or case 6 [1]. The complete pattern of transformations into stable states is shown in Figure 2. The figure shows depicts comparison cases and transformations from one case into another. The transformation arrows show the direction of transformation during the integration process.

2 -> 1 convert true synonyms into identical items through renaming.

3 -> 1 convert construct mismatch into identical items through change of different constructs.

4 -> 3 convert construct mismatch and synonym into just semantic relativism through renaming, or

4 -> 2 convert construct mismatch and synonym into synonym through construct change.

5 -> 6 convert homonyms into different items (possibly related) through renaming.

8 -> 6 convert different items with different constructs into different items with same constructs (only if items are different but related) through construct changes.

7 -> 5 convert homonymy with different construct into

---

[1] If the objects are unrelated, case 8 is a stable state, requring no changes during conflict resolution. If objects are related, ultimately, the objects will be transformed into state 6.

homonymy through name change (only if objects are related).

7 -> 8 convert homonyms into different items through renaming.

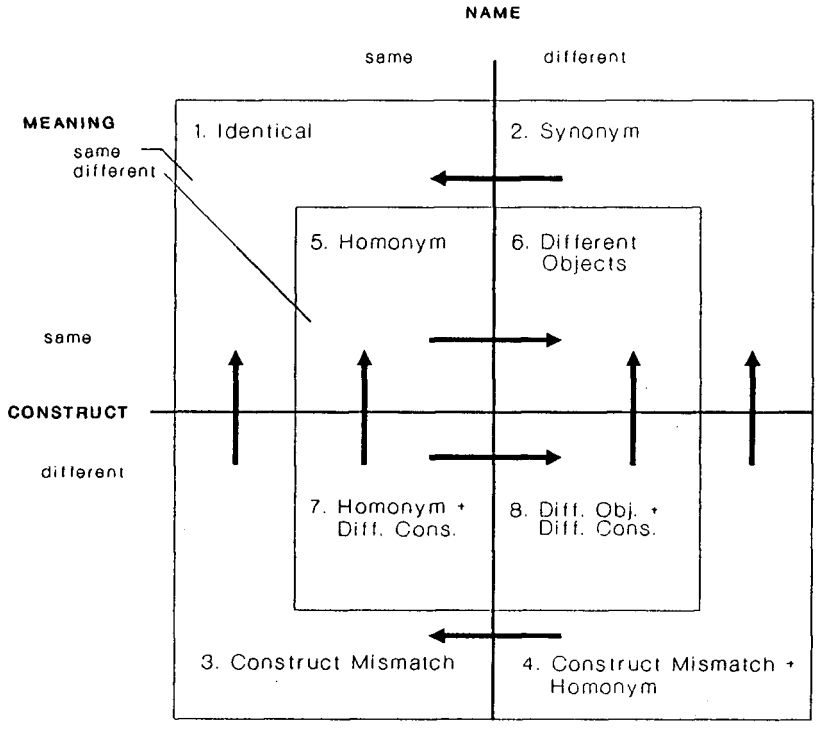


Figure 2: Case Transformations during View Integration

The transformation sequences have three end points, Case 1, Case 6, and Case 8. Case 1 is the end point for all objects with same meaning. It is captured by cases 1 and 5 extracted from previous research. Case 8 is the end point for all items

which are different in all aspects and not related. Its solution is trivial. All these non-identical items will be included in the global schema. Case 6 is the end point for non-identical, unrelated items with same construct (trivial solution) and for different but related objects. If objects are related, cases 2 to 4 and 6 to 8 from previous research (chapter 2) will apply.

The case transformations (Figure 2) are free of circularities. This makes it possible to postulate an ordering of conflict recognition and resolution. Figure 3 illustrates one possible ordering. The operations to be carried out first are construct changes (4->2, 3->1, 7->5, 8->6) for identical and for related objects. This is followed by the change of names for synonyms (2->1), and homonyms (5->6 for related objects, 7->8 for unrelated objects). The termination points of the procedure are cases 1, 6, and 8. The other possible ordering would attend to name changes prior to construct changes. For now, both sequences are equally good, even though the first one is preferable, as will be explained later.

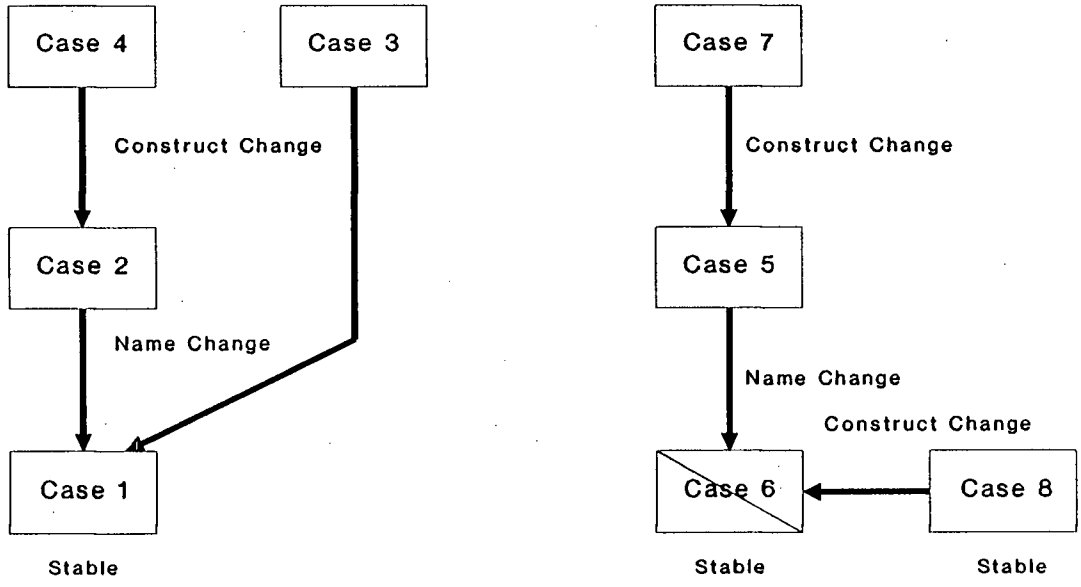


Figure 3: Ordering of View Integration Steps

3.2.3. Changes in the Integration Method when Necessary Information is not Directly Available

The integration method discussed so far is based on the assumption that necessary information to carry out the integration process is directly available. For the required information to be available, it either has to be specified ex-ante, or has to be elicited during the view integration process. Since information specification requires designer effort and represents a cost, it is desirable to reduce information specification requirements for the database designers. Hence, while previously the focus was on the design of a complete method for integration, the focus will now be on a human-oriented complete method for view integration.

The new goal will be to dtermine object identity, difference and relatedness with a small number of intelligent (i.e., non-redundant) questions. Obviously, the method should base future questions on answers to previous ones. This is a minimum requirement. The following list of questions outlines other areas in which the procedure can be improved.

1. How many objects shall be included in the object comparison?

2. Which objects should be compared?

3. What is the sequence of conflict diagnosis and therapy?

4. How shall identity or difference be decided?

How many objects?

The previously outlined procedure always compared object pairs, i.e., "is entity E1 identical in meaning to entity E2?" This type of question can always be answered with "yes" or "no", but for n objects in view 2 this form of questioning requires n questions . By asking, "is E1 identical to one of {E2, E3, ..., Em}", the number of questions can be reduced to 1. The question can be answered either with the object's identifier, or with "no". This form of questioning drastically reduces the questioning effort. The questioning format will always be 1:n instead of 1:1. An m:n format will not be used, since the answers become awkward (a list of tuples of identical objects).

Which objects?

The procedure would not behave intelligently, if it included objects in the comparison that should not be included. For instance, if E21 from view V2 was found to be identical to E11 from view V1, the procedure should never again inquire whether E21 is identical any other object from V1. Other rules which are described in the results chapter, reduce the group of

relevant objects even more. Furthermore, heuristics (also rules, but not always true) were found to reduce the group of objects even further. For example, once two entities are found to be identical, and both participate in relationships, one may expect to find identical pairs of relationships within these smaller groups.

Which sequence?

So far, sequences of object modifications have been outlined which resulted in stable state cases, (Case 1) identical objects, (Case 6) different, but related objects, and (Case 8) different and unrelated objects. For instance, a case of construct mismatch (Case 3) was transformed into Case 1 through a construct change. The question is whether the method should operate by searching actively for conflict cases such as Case 3 or Case 4? The answer is "no". A human-oriented integration procedure will alter the sequence of tests. Following the assumption that in absence of information to the contrary, two views are assumed to be identical, the procedure will always attempt first to find matching objects, not object mismatches. For example, typically the assumption at the outset of the object comparison will be that for an object O1 in view V1 there exists an object O2 in view V2 with an identical construct, i.e., both are relationships. Figure 4 briefly outlines the basic sequence of tests.

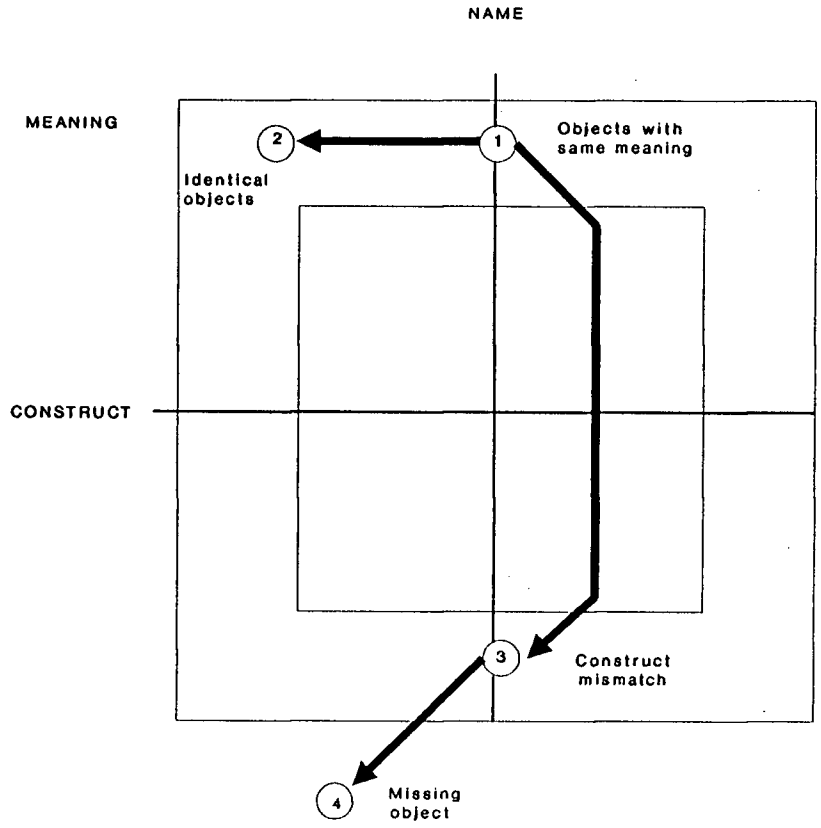


Figure 4: Conflict Recognition Procedure (abbreviated)

For any object O1 from view V1 and any set of objects {O2} from view V2, the first test is a test for identity of meaning. If it fails, a test for construct mismatch follows. If there is no construct mismatch, an object is assumed to be missing. Note that name and context difference or identity are ignored

at first. The test for relatedness which begins with the assumption of relatedness is separated from the test for identity of objects. Tests for relatedness are postponed until all tests for identity are carried out.

How to decide on identity or difference?
For all object characteristics, identity or difference have to be asserted. While this is simple for construct and name, it is not for meaning and context. Only people can ultimately judge whether two objects have the same meaning, but an intelligent integration procedure should help as much as possible in making this decision. In short, the procedure will help by filtering out objects that are not identical to the object in question. Rules to filter out these non-corresponding objects are defined.

3.2.4. View Integration Conflict Cases

Previously, only 8 of the 16 general types of cases were discussed, when context was held constant. The case list below describes all possible cases for the comparison of two arbitrary objects from different views. Cases are identified by name (N), construct (object type T), meaning (M), and context (C) <N,T,M,C> of the involved objects. Object O1 is

denoted through <N1,T1,M1,C1>, object O2 through <N2,T2,M2,C2>. For every case the equality or non-equality of parameters is stated.

The overview table below shows for each case identity or difference along the four dimensions. For example, a '=' under N means that both objects have identical names, an 'x' means they are different. For the meaning dimension, 'r' means the meanings are different but related.

Case	N	T	M	C	
1	=	=	=	=	Identical objects
2	=	=	=	x	Identical objects with different context
3	x	=	=	=	Synonym
4	x	=	=	x	Synonym with different context
5	=	x	=	x	Construct mismatch (semantic relativism)
6	x	x	=	x	Construct mismatch and synonym
7	x	=	x	=/x	Different and unrelated objects
8	=	=	x	=/x	Homonym
9	x	x	x	x	Different objects with different constructs
10	=	x	x	x	Different objects with different constructs, but homonyms
11	x	=	r	=	different but related objects
12	=	=	r	=	different but related homonyms
13	x	=	r	x	different but related objects with different context
14	=	=	r	x	different but related homonyms with different context
15	x	x	r	x	different but related objects of different type
16	=	x	r	x	different but related homonyms of different type
17	-	-	-	-	missing object O2

Note that if two objects are of different type, their context will be different, due to the definition of context. Note also that identity or difference of context is irrelevant for objects with different meaning.

A more detailed list of view conflicts can be found in the Appendix. The list in the appendix breaks each general case down into subcases based on differentiation according to the constructs of participating objects. I.e., a construct mismatch exists between an entity and a relationship as well as between an entity and an attribute. The extended list has been left out here for the purpose of readability. The Appendix also provides a brief description of the solution for all case scenarios. The general conflict resolution rule for object identity and difference is to have all other dimensions follow the meaning dimension. If two objects have identical meaning, all other dimensions will have to be made identical. If two objects have different meaning, the name dimension has to reflect this. Cases of object relatedness are solved through representation of the superset subset relationships.

Omitted from this solution description is the technique for re-allocation of attributes when relatedness is detected. The general rule is to allocate those attributes that belong to both the superset and the subset to the superset, and to

allocate to the subset only the attributes that are unique to it (see for instance Navathe and Elmasri (1986)).

3.3. Expert System Methodology

An implemented solution for the view integration problem requires an adequate problem representation and solution mechanism. So far, cases of potential integration problems and a procedure have been identified, yet no implementation mechanism has been suggested. Before any further discussion of an adequate mechanism, here a short reminder of the problem situation.

Correcting the conflicts in a set of user views is clearly a problem solving task. Within this research, view integration is treated as a diagnosis and therapy task (note that Hayes-Roth et al. mention diagnosis and therapy ("repair") as generic tasks of knowledge engineering applications). Characteristic of a typical diagnosis task is the goal to find out "what's wrong" in the actual state. Thus, the purpose of the diagnosis part of view integration is the identification of the discrepancy or mismatch between a pair of views. Once the conflict case has been identified, the therapy or "fixing" phase will adjust one or both views to remove an existing conflict. Therapy creates the new, desired structure. Diagnosis and therapy tasks are prototypical tasks for expert systems or knowledge based systems. The integration method discussed here was not built by extracting diagnosis and therapy rules from expert designers. Hence it is not truly an expert system. However,

it will represent conflict recognition and conflict resolution knowledge.

Database design rules for conflict recognition and resolution can be easily formulated as sets of production rules. In simplified form, one may want to think of each production rule as describing one of the cases. For each object comparison, the rule matching the conflict situation would fire and transform the case into another one, until one of the stable state cases were reached (for a description of the production system reasoning mechanism see for instance Barr and Feigenbaum, 1981).

The most appealing property of the production system mechanism is the modularity of the resulting systems. Rules can be added, deleted or changed without directly affecting other rules. Figure 5 illustrates this fact. Figure 5 (taken from Vessey and Weber, 1986) depicts a decision table with cooking instructions for vegetables to exemplify the convenience of rule editing. Each instruction (column) corresponding to one production rule. The list can be easily expanded through addition of new columns. By the same token, the deletion of a column does not affect any other column (or rule) in the table. Furthermore, each column can be changed, thereby affecting only the instructions for one particular dish. The cause for this simplicity of the rule based system lies in the

design of the condition list. Each condition stub is specified with the utmost detail, not referring to conditions which are aggregates of more than one fact. I.e., the decision table does not create intermediate results (aggregates of truth values) such as "juicy and crispy and leafy but not tall", which could appear later as a single condition in the condition list for both "fry" and "steam". In other words, condition items are decoupled as much as possible. Consequently also the rules (i.e., the dishes) are decoupled.

Juicy	Y	Y	Y	Y	Y	Y	N
Tall	Y	N	N	N	N	N	—
Crispy	—	Y	Y	Y	N	N	—
Leafy	—	Y	Y	N	—	—	—
Red	—	Y	N	—	—	—	—
Hard	—	—	—	—	Y	N	—
Fry		X					
Steam			X				
Grill				X			
Peel					X	X	
Boil					X		
Chop	X					X	
Roast							X

Figure 5: Decision Table Illustration

The modularity of production rules makes their implementation very forgiving. If a case is left out in the beginning, or is specified incompletely at first, additions can be made with very little effect on the already existing rules.

One disadvantage is the inefficiency of the production system approach, due to duplication of identical condition elements. This result is the cost induced by complete decoupling of conditions. Every condition list has to be created and tested in detail without being able to make use of established intermediate results. A more sensible design approach should compromise between complete decoupling of conditions and processing efficiency. A heuristic for aggregating conditions would group those conditions together that form a meaningful unit (are highly cohesive). Meaningful stands in contrast to purely accidental coincidence of conditions. I.e., "juicy and crispy and leafy, but not tall" is not a particularly meaningful grouping, because it does not identify a certain well-known group of food items. Therefore, this aggregate should not be chosen as a grouping, even though it could simplify the decision table in the example.

A second disadvantage of production systems is the fact that they disguise the control flow. It is difficult for a designer to understand the control flow in the production system. In

so called "procedural" programming languages, i.e. Pascal or Fortran, the control flow is determined by the ordering of the language statements, if branching statements are neglected for the moment. In production systems, the sequence of rules has much less importance. I.e., the "chop" rule will not be applied first even though it is the first rule in the decision table in Figure 5, unless its conditions are true. If the last rule in the system is the one whose conditions become true first, it will be the first to fire. Hence, production systems in general require substantial re-thinking by systems designers who are used to procedural languages. In a Prolog implementation this problem is alleviated to some extent since the language's interpreter interprets rules still in sequential order.

In conclusion, even though it has some disadvantages, a production system seems to be a suitable representation mechanism for the implementation of this research. The case description already provides many guidelines for the definition of conflict resolution rules. Also, the maintainability advantage of production systems becomes important when subsequently heuristics have to be added to the integration method to improve its operation with insufficient information.

A different, apparently more elegant approach to view integration could perform the integration process as a theorem proving

task. Similar to other theorem proving tasks (see for instance Nilsson, 1980) the program would be given a set of views and the question "does there exist a conflict free global schema which contains all the information of the individual conflicting views"? If the answer to that question were "yes", the global schema would be produced as a "by-product". Using Robinson's resolution principle (1965), the program would solve the problem by creating a new goal "there exists no global schema" and by falsifying this statement through a counter example. This approach is elegant because it is based on a very general problem solving mechanism, the theorem proving mechanism. However, definition of the integration rules, especially the procedural rules of conflict recognition and resolution is more difficult than in the production system approach.

Two other reasonable representations for the task are *frames* and *semantic networks* (Waterman, 1986). They will be discussed below.

Frames (Minsky, 1975) are complex data structures containing both factual and procedural knowledge. Frames have slots which can contain data concerning frame properties. Related to slots can be procedures which are invoked when a slot is filled. Slots that are not filled can take initially defined default values. This default capability is one of the advantageous features of frame based knowledge representations.

Mylopoulos and Levesque (1984) for instance stress their ease of dealing with incomplete knowledge. Frames have been used as knowledge representations in a variety of expert systems (see Waterman, 1986 or Hayes-Roth et al., 1983). Barr and Feigenbaum (1981) state that frames "have problems", yet do not mention where these problems lie.

Semantic nets represent knowledge in a network in which properties are inherited from other objects along the arcs of the network. Waterman states that semantic nets have also been used in expert systems, in fact he argues that semantic nets and frames are similar. Mylopoulos and Levesque (1984) emphasize as qualities of semantic nets their data organization and the provision of good access methods. As a disadvantage they state the lack of formal semantics and standard terminology. The problem of formal semantics becomes clear, when the interpretation mechanism for semantic nets is investigated.

All approaches are feasible. However, for its forgivingness in the maintenance of the knowledge base, the production system approach has been chosen for this research. The integration method has been implemented in Prolog. The program is called AVIS, for Automatic View Integration System.

4. RESULTS

4.1. Rules Guiding View Integration

View integration as a problem solving task is guided by a set of rules which allow the problem solver to define the problem environment, identify the particular problems (conflicts) and to solve them. In this section, the general rules underlying the process are presented, exemplified and justified. The rules can be divided into two major groups: base rules and heuristics. Base rules are believed to be always true. Heuristics are support rules. The beliefs expressed in them are known to be wrong sometimes but are expected to be true in most cases.

Especially in its conflict recognition part, this view integration method relies to a large extent on asking the right questions. If the method can ask the right questions, it can perform a large segment of the integration without user interaction. When user interaction cannot be avoided, a selection of the right questions can simplify the user's answering task. Furthermore, the method will not appear to be stupid, if it can avoid asking trivial or redundant questions. To help in the question formulation process, heuristics were included which for instance change the content and sequence of questions.

Base rules are separated into four groups of rules. The first three groups are static modelling rules. The fourth group contains process rules:

1. General Modelling Rules
2. Rules of the Modelling Language
3. Rules of Database Design/View Integration
   - 3.1 General Database Design Rules
   - 3.2 Rules Concerning the Test for Identity of Objects (Conflict Recognition and Reconciliation Rules)
   - 3.3 Rules Concerning the Relatedness [1] of Objects (Rules for Recognition and Modelling of Inter-Schema Relationships)
4. Process Rules
   - 4.1 Process Rules for Conflict Recognition and Reconciliation
   - 4.2 Process Rules for the Recognition and Modelling of Inter-Schema Relationships

General modelling rules are valid not only in the database context. For example, "each relevant real world object[2] shall be represented by exactly one object in the model" is such a rule. Rules of the modelling language, here the E-R modelling language, describe true statements about the E-R language that are relevant to the view integration task. Rules of database

---

[1] The term "relatedness" is used to signify superset-subset relationships such as all managers are employees, MANAGER--Isa--EMPLOYEE. The term "relationship", unless occurring in the form "subset/superset/containment relationship", is used to denote associations between entities.

[2] Throughout the chapter, the terms object and object type will be used interchangeably to describe object types. Particular instances are referred to as object instance or object occurrence.

design are separated into rules to guide the database designer's (or the method's) test for the identity of objects and rules to guide the uncovering of inter-schema (superset-subset) relationships. Process rules describe the sequence in which tests (i.e., conflict recognition) and corrective measures (i.e., conflict resolution) shall be carried out.

The discussion will begin with a description and explanation of the base rules, followed by an analysis of the heuristics.

Base Rules

General Modelling Rules:

1. Each relevant real world object type shall be represented by exactly one object type in the model (redundancy-free representation).

All model building tries to create a representation of the real world that contains all relevant information in the most concise form. Not all the information of the real world can be represented. Most of the detail may not even be required for the tasks at hand. Hence, some real world object types

will not find their way into the model. If a real world object type is represented more than once in the data world, update anomalies can occur. Each new object instance of the real world has to be inserted more than once into the data model. Should the real world object type itself cease to exist, more than one data model object type has to be removed. This creates extra processing effort and the possibility of inconsistency. One of the purposes of database design is to avoid exactly these problems.

2. An integration of multiple models shall not result in the loss of information from any of the models.

Any bottom-up modelling approach attempts to build a large global model through the combination of smaller models. Each of the small models represents the real world facts that one model-builder perceives as relevant. Omission of any of these facts out in the global model would result in an incomplete global model. Hence, the rule demands that all individual models are correct and that the collection of models is in itself consistent (Biskup and Convent, 1986).

Rules of the Modelling Language:

3. Every object in a view is represented with exactly one of following three constructs: Entity, Relationship, Attribute.

The view integration method models databases based on Chen's Entity-Relationship model in which only Entities, Relationships and Attributes exist. Categories which are represented in some extended forms of the E-R model will be depicted as special (Is-a) relationships.

4. Entities are autonomous objects. They can exist without the existence of Relationships and without the definition of Attributes.

Entities are things or individuals. As things or individuals can exist even if they have no associations with other things or individuals, so can entities. For example, an entity SUPPLIER can exist <u>without</u> an association to another entity, such as BUYER.

5. A Relationship cannot exist without the existence of at least one Entity.

Relationships represent associations between entities. They map instances of one entity to instances of some other entity.

In the most restricted case, one entity is associated with itself. For example, the entity PERSON is associated with itself through a Supervisor or a Parent relationship. Typically, more than one entity will be involved in a relationship, but never less than one.

6. An Attribute cannot exist without the existence of the Entity or Relationship it belongs to.

Attributes represent associations between an entity and a value set, or a relationship and a value set. For example, the Person_name attribute associates the PERSON entity with a value set of names which itself is a set of strings containing valid person names. The attribute cannot exist without the existence of the entity or relationship it refers to (value sets are not part of the E-R model).

General Database Design Rules:

7. Two types of Attributes exist. "Property" Attributes which describe the object (Entity or Relationship) in more detail (i.e., color, name) and "Interconnection" Attributes which describe the

association of the object (Entity or Relationship) to some other object (Entity or Relationship).

Attributes are always associations between entities or relationships and value sets. However, sometimes attributes are not used to describe an innate property of the entity or relationship they belong to, but instead, to describe an association between the entity or relationship and some other object. For example, the attribute Person_name describes a property of a PERSON entity, their name. PERSON could also have an attribute Savings_acct_no. This attribute even though associated with PERSON, is not a property of a person. In fact, the attribute implicitly states that things called savings accounts exist and that a person is or may be related to such a savings account. Thus, the attribute describes not a property, but an association. PERSON possesses SAVINGS_ACCT (PERSON is_associated_with SAVINGS_ACCT). While in the example the difference between a property attribute and an interconnection attribute was distinct, it will not be as clear in all cases.

8. Interconnection Attributes are shortened forms of Entities (if the A is a Relationship-Attribute), or

of Entity-Relationship constructs (if the A is an Entity-Attribute).

In the above example, PERSON had a Savings_acct_no attribute which indicated the existence of savings accounts and a person's possession of such an account. Obviously, savings_account could become an entity, since it is a thing in the real world. In that case, a relationship such as Has_account would represent a person's possession of such an account. Also, being an Entity, a savings account could have attributes itself, such as Account_balance, or Date_opened. The model builder may not need all this extra information. If the account number information is sufficient, there is no reason to describe savings accounts, or other real world objects, in more detail. After all, a model should contain only the relevant information about the system it is modelling.

In the example, an entity attribute (Savings_acct_no) which was an association between an entity (PERSON) and a value set (of account numbers) took the role of a relationship (Has_account) between PERSON and another entity SAVINGS_ACCT. The attribute thus represented both a relationship (Has_account) and an entity (SAVINGS_ACCT) through the account number value.

All attributes of SAVING_ACCT other than its number, as well as any potential non-key attributes of Has_account are not

represented. Hence, interconnection attributes are a compressed form of information representation.

This compression has the undesirable side effects of deletion and insertion anomalies. I.e., savings accounts do not exist, until people exist that possess the accounts. Accounts also cease to exist with the person owning them.

9. If Attributes are multi-valued, they are interconnection Attributes.

This rule helps in the detection of interconnection attributes. If a multi-valued attribute is found, it is considered to be a interconnection attribute. For example, if the Address attribute of an EMPLOYEE requires multiple entries it should better be represented by a new entity RESIDENCE, related to EMPLOYEE through a relationship such as Resides_at. Storey (1988) deals with multi-valued attributes in this manner during view creation.

10. A Relationship is a less fundamental object than an Entity.

Since relationships cannot exist without the existence of at least one entity, their continuing existence is based on two

factors. First, it is based on the existence of the objects underlying the entities, and second, on the existence of the association between those real world objects? Should either one not exist, then the relationship has to be removed. For entities, on the contrary, it is unimportant whether any formerly existing association between them is still in place. They will only disappear once the real world objects underlying them disappear. The same is true for entity and relationship instances. For example, if a database contains the entities EMPLOYEE and DEPARTMENT as well as the relationship Employed_by, individual instances of Employed_by, such as [1005, Manufacturing] are only meaningful if employee 1005 still exists, the manufacturing department exists, and the employee in fact still works for the manufacturing department (referential integrity).

11. Each object has four relevant dimensions: Name, Construct (Entity/Relationship/Attribute), Meaning, and Context.

One of the basic assumptions underlying this view integration method is that there exist only four relevant differentiation criteria for objects in a view: name which is the name of an object, such as SUPPLIER, construct or object type, such as relationship, meaning, and context. Meaning encompasses all

the relevant knowledge conveyed by the object. For example, meaning includes all the information that is known, once it is known that a particular entity is a SUPPLIER. I.e., supplies parts, will be paid for parts. Meaning is the most important of all four dimensions. It will have absolute precedence over the other dimensions. If two objects have the same meaning, they refer to the same real world object and therefore all other dimensions will have to be adjusted accordingly. Context identifies the set of objects an object is associated with. An attribute's context is the entity or relationship it belongs to. A relationship's context are the entities associated by it. Entities are defined as having no context. Entities are the only objects able to exist without any other type of objects.

12. Along each dimension, any two objects can be either "same" or "different", i.e. same name, same construct.

Another major assumption of the view integration method refers to the variations in each dimension. It is more important to find out whether two objects are identical (same) or different in each of the relevant dimensions rather than to find out the actual values for each dimension. In order to merge two objects, they have to match, they have to be completely identical. If they are even slightly different a change is required. The magnitude of dissimilarity does not matter,

since a change is required nevertheless. For example, the entity names SUPPLIER and SUPPLIERS are only slightly different. Nevertheless, they are different and will require a name change if the entities are to be merged. The same is true for the other dimensions. Two relationships may have "almost" the same context, that is, most of the entities associated by them are the same. Despite this fact, these relationships have a different context and cannot be merged unless the context of one or both of them is changed.

13. Two objects with different meanings can be related in meaning.

Meaning is the only dimension where identity or difference are not the only two relevant values. For example, the entities EMPLOYEE and PART_TIME_EMPLOYEE have obviously different meaning, yet they are not completely independent. EMPLOYEE refers to a type of individuals which includes the type of individuals referred to by PART_TIME_EMPLOYEE. Hence, when two objects are different in meaning, any superset-subset relationships between them are nevertheless relevant. Objects with such relationships will be called related in meaning.

14. Two related objects O1 and O2 will display one of the following set relationships between them:

1. O1 and O2 have a common subset (yes/no); and
2. O1 and O2 have a common superset (yes/no);

resulting in the following possible combinations:

(a) one object contains the other object;
(b) both objects have a (meaningful) common superset and a common subset, yet the superset is not one of O1 or O2;
(c) both objects have a common superset, but they do not overlap;
(d) both objects have no common superset and do not intersect; virtually no relatedness, no need for representation in a database.

Set relationships and their treatment within view integration have been discussed at different levels of completeness by all previously reviewed integration techniques, most completely by Navathe and colleagues, Elmasri and Navathe (1984), Navathe and Elmasri (1983).

This rule lists all relevant relationships between two sets. The qualifier "meaningful" for supersets or subsets implies that any such superset or subset has to be a cohesive group from the point of the users. For example, the entities EMPLOYEE and CUSTOMER have a common superset requiring implementation, the entity PERSON. Consequently, both EMPLOYEE and CUSTOMER would inherit the attributes of PERSON and all instances of EMPLOYEE and CUSTOMER would be instances of PERSON. Another, less meaningful superset would be an entity EMPLOYEE&CUSTOMER. The choice of an appropriate common superset, i.e.,

EMPLOYEE&CUSTOMER vs. PERSON, has to remain with the user[1]. While there are no fixed rules to what constitutes a "good" entity, there are indicators for less good entity choices. For instance, if the user cannot provide a good name for the object, it may not be a (good) entity. I.e., EMPLOYEE&CUSTOMER is not a good object name. Hence, the object is not expected to be very meaningful. Or, if the objects attributes are identical to an already existing object's attributes, the object may not be a (good) entity.

Examples for the forms of relatedness are:

(a) EMPLOYEE contains PART_TIME_EMPLOYEE;

(b) PRODUCT_TEAM_MEMBER and PROJECT_TEAM_MEMBER are both subsets of EMPLOYEE, their intersect is PRODUCT&PROJECT_TEAM_MEMBER;

(c) PART_TIME_EMPLOYEE and FULL_TIME_EMPLOYEE are both subsets of EMPLOYEE, but they do not overlap;

(d) CUSTOMER and DEPARTMENT do not intersect.

The relatedness in (d) is so weak that it shall be ignored. Even though it represents some extra knowledge about the world, the knowledge is negative knowledge. Since negative

---

[1] Throughout the text, the term "user" refers to a database designer who employs the integration method. This "designer user" represents the interests of the end users of the database. The end users are assumed to have provided the original views.

knowledge is so much more abundant than positive knowledge, its representation typically becomes infeasible.

15. Two unrelated objects O1 and O2 may share a common role.

Two entities, for example PERSON and COMPANY can be different and unrelated, but they still can have a common role such as the role of shareholder. Neither view may contain a shareholder object, even though both may contain a STOCK entity. Goldstein and Storey (1988) discuss unrelated objects sharing a common role ("W-relationship") and the proper representation of this situation in a generalization lattice.

16. Two objects are identical, if they are identical in all dimensions.

Only the previously discussed four dimensions are relevant to judge whether objects are identical. Objects have to correspond in all dimensions. For example, an entity EMPLOYEE and an entity WORKER are known to mean the same. Thus they are identical in meaning, construct (entity), and context (empty). Nevertheless, the objects are identical only after their names have been made identical too.

17. Each object is related to itself (contains itself and is contained by itself). This relatedness shall not be represented in any view.

This rule guides and limits the search for between-view set relationships. For example, if an entity EMPLOYEE has been found to be identical to another object EMPLOYEE from some other view, each of the entities is also a superset of the other one. They also share a common subset, the entity set itself. The representation of this finding bears no extra information. It would also result in an infinite expansion of the global database, since if every object is related to itself, also the object expressing this relatedness is related to itself which has to be expressed through yet another object, and so on.

18. An object can be related to between 0 and n other objects.

It is important to remember that one object can be related to more than one other object. The search for related objects from another view is not completed after one related object has been found. However, it is also possible that no related objects can be found in another view.

19. Each object in one view can have a maximum of one identical object in another view (call this object also the "corresponding" object).

This rule follows from the general rule of modelling that no real world object shall be represented more than once in a model. A view is a model. Hence, if two objects of one view are identical to another entity from some other view, the two objects must be identical. This rule implies that once a pair of identical objects has been found, there is no need to search for further identical objects.

20. Two views are the same, if all their objects are identical.

The goal of the conflict recognition and resolution procedure is to correct omissions and conflicts so that at the end two previously different views are identical. Then they do not have to be merged, one of them can be removed, since all its information is also contained in the other view. This rule states when the identity condition is achieved.

21. Each individual view is complete and consistent and minimal.

A view is complete if it represents all the individuals, things, and associations between them, relevant to the user. A view is consistent if none of the facts stated concerning the relatedness of sets are contradicted by others in the view. For example, if the view states that the entity PART_TIME_EMPLOYEE is a subset of the entity EMPLOYEE, no other fact in the view may present contrary information, such as PART_TIME_EMPLOYEE and EMPLOYEE have no members in common, (see Casanova and Vidal (1982), Biskup and Convent (1983)).

Minimality of a view entails that each real world object is only represented once in a view. For example, if one view contains two entities, SUPPLIER and DEALER, these entities have to be different; they have to refer to different objects in the real world.

The completeness assumption clarifies the role of the integration method as a method that finds omissions or conflicts in views based not on within-view (intra-view) analysis but based on between view (inter-view) comparison.

22. The collection of views before integration is consistent.

The view integration method assumes that not only views individually are consistent, but also that the collection of views is consistent as a whole. In other words, facts stated concerning relatedness of sets in one view cannot contradict facts stated in another view.

This rule clarifies the purpose of the conflict recognition and resolution method as a method that corrects omissions and conflicts (i.e., differences in opinion on name, context) but not contradictions. For instance, if view V1 states that all managers have to be full-time employees, while view V2 states that also part-time employees can be managers, the views contradict. Both statements cannot be true at the same time. The method assumes that such contradictions do not exist.

Rules Concerning the Test for Identity of Objects:
(Conflict Recognition and Resolution Rules)

23. If for an object O1 from view V1 an identical object O2 cannot be found in view V2, then O2 is either missing or represented through an object that has the same meaning but is different along its other dimensions.

Ideally, an identical object O2 from V2 exists for each object O1 from V1. Both objects are identical if they are identical in all relevant dimensions: name, construct, meaning, and context. The most crucial dimension is the meaning dimension. If two objects have the same meaning, they refer to the same object in the real world. Hence, if an object O2 with the same meaning as O1 exists, there may remain a name, construct or context conflict between O1 and O2 to be taken care off, but O2 is not missing. If no O2 exists that refers to the same real world object as O1 does, then that O2 is truly missing.

24. No change of a view during integration shall result in the loss of information.

This rule provides a guideline to the direction of change in cases of construct mismatch as described by one of the following alternatives:

Object in view 1:	Object in view 2:
Entity	Relationship
Entity	Attribute
Relationship	Attribute

Mismatches between an attribute on one hand and an entity or relationship on the other hand will result in a change of the object with the attribute construct. This adjustment rule follows from the rule on interconnection attributes.

A mismatch between an entity and a relationship, results in a change of the object with the relationship construct, based on the rule concerning object permanence. Relationships are less fundamental than entities. Relationship instances cease to exist when the entity instances they refer to cease to exist (referential integrity), as illustrated below:

View 1: SUPPLIER--Sup_con--CONTRACT--Cus_con--CUSTOMER

View 2: SUPPLIER--Contract--CUSTOMER

Both views have suppliers in a contract situation with customers, yet in view 1, the contract itself is an entity, in view 2 it is a relationship. In view 2, a disappearing customer (instance) destroys all records of a contractual agreement between him and the supplier. No historic data remains. In view 1, contracts have a life of their own and survive the disappearance of a customer instance. Hence, the less permanent character of a relationship potentially leads to information loss in the database extension. Consequently, a construct mismatch between

an entity and a relationship should result in a change of the relationship construct into an entity construct.

25. If two unrelated objects share a common role, the common role object and specific role objects have to be represented as well as Isa relationships between the original objects and the specific role and between the specific roles and the common role.

This rule is based on Goldstein and Storey (1988). For example, in:

V1: PERSON--Holds--STOCK

V2: COMPANY--Holds--STOCK

PERSON and COMPANY have the same role. Therefore, a common role object SHAREHOLDER is needed to describe the situation. Furthermore, specific role objects, PERSON_SHAREHOLDER and COMPANY_SHAREHOLDER are needed. Then, PERSON_SHAREHOLDER is a PERSON as well as a SHAREHOLDER. SHAREHOLDER here will be the object associated with STOCK through the Holds relationship.

Rules Concerning the Test for Relatedness of Objects:
(Recognition and Modelling of Inter-Schema Relationships)

26. Any Object O1 from V1 which is not an entity and which is related to an object O2 from V2 shall become an entity.

Any object O1 that is not an entity is either a relationship or an attribute. Neither of the two may be associated with other objects by means of a relationship. Relationships involved in relationships are not permitted, nor are relationships involving attributes. However, if two objects are related, they will have to be connected by an Isa relationship. Thus, this construct change is necessary. For example, an attribute Supplier belonging to entity PART in view 1 is related to entity DEALER from view 2. The relatedness is such that all suppliers are dealers but not all dealers are suppliers. In this case, the Supplier attribute in view 1 will become an entity, which will be associated with part through a Supplies relationship. Supplier in view 1 was an interconnection attribute which is now more adequately represented through an entity. For a more detailed illustration of construct changes compare section 4.3 on conflict therapy.

27. If an object O1 contains an object O2, the containment shall be represented by an Isa relationship. If the Isa relationship does not exist, it must be added.

The contained object will possess all attributes of the containing object.

This rule on the E-R representation of containment is taken from Elmasri and Navathe (1984).
For example, if EMPLOYEE contained PART_TIME_EMPLOYEE, the connection between the two would have to be represented by an Isa relationship, stating that PART_TIME_EMPLOYEE is an EMPLOYEE. PART_TIME_EMPLOYEE would inherit all attributes of EMPLOYEE.

28. If two objects O1 and O2 overlap, and neither object contains the other, the overlap shall be represented by an overlap object O3. If the overlap object does not exist, it must be added. The overlap object O3 will inherit the union of the attributes of O1 and O2. The connections O1-O3 and O2-O3 shall be represented by one Isa relationship each. If either of the Isa relationships does not exist, it must be added.

This rule states how the method handles relatedness of the form common subset (overlap). The following example will illustrate the rule:
View 1: PROJECT_EMPLOYEE[Emp#,Proj#,Yrs_experience,Title]
View 2: PRODUCT_EMPLOYEE[Emp#,Prodname,Function,Title]

the common subset PROJECT&PRODUCT_EMPLOYEE inherits the attributes Emp#, Proj#, Yrs_experience, Prodname, Function, Title and contains all instances of employee contained in PROJECT_EMPLOYEE and in PRODUCT_EMPLOYEE (intersect). Furthermore, the following relationships are added:

PROJECT&PRODUCT_EMPLOYEE--Isa--PROJECT_EMPLOYEE

PROJECT&PRODUCT_EMPLOYEE--Isa--PRODUCT_EMPLOYEE

The creation of overlap objects is explained in detail in Yao et al. (1982).

29. If two objects O1 and O2 have a common superset, and neither object contains the other, the superset shall be represented by a superset object O3. If the superset object does not exist, it must be added. The superset object O3 will possess the intersect of the attributes of O1 and O2. If they are not identifier attributes, these attributes will have to be removed from O1 and O2. The connections O1-O3 and O2-O3 shall be represented by one Isa relationship each. If either of the Isa relationships does not exist, it must be added.

This rule states how the method handles relatedness of the form common superset. The following example will illustrate the rule:

View 1: PROJECT_EMPLOYEE[Emp#,Proj#,Yrs_experience,Title]

View 2: PRODUCT_EMPLOYEE[Emp#,Prodname,Function,Title]

the common superset EMPLOYEE receives the attributes Emp#,Title. The non-key attribute Title are removed from PROJECT_EMPLOYEE and from PRODUCT_EMPLOYEE:

EMPLOYEE[Emp#,Title]

PROJECT_EMPLOYEE[Emp#,Proj#,Yrs_experience]

PRODUCT_EMPLOYEE[Emp#,Prodname,Function]

EMPLOYEE contains all instances of employees included in PROJECT_EMPLOYEE or in PRODUCT_EMPLOYEE (union). Furthermore, the following relationships are added:

PRODUCT_EMPLOYEE--Isa--EMPLOYEE

PROJECT_EMPLOYEE--Isa--EMPLOYEE

The creation of overlap objects and attribute relocation is explained for instance in Navathe et al. (1986).

30. If two objects exclude each other, the exclusion shall be represented through an integrity constraint.

No new objects are added in the case of an exclusion. However, an integrity constraint can be added to prevent any object instance from accidental insertion into the non-overlapping sets. For example:

View 1: FULLTIME_EMPLOYEE

View 2: PARTTIME_EMPLOYEE

describe two non-overlapping sets. An integrity constraint could be formulated to permit insertion of instances into either object only if after the insertion a join of both objects still returns the empty set.

If the model (and the DBMS) can support integrity constraints, this restriction can improve the data quality. The representation of exclusion integrity constraints is suggested by [Casanova and Vidal, 1983] and [Biskup and Convent, 1986].

31. Containment is transitive. If A contains B and B contains C, then A contains C. The transitivity shall not be explicitly represented in any view. An Isa relationship between A and C is assumed to exist, if an Isa relationship exists between A and B and between B and C.

This rule prevents the generation of new redundant Isa relationships in multi-level hierarchies. If for example, PERSON, EMPLOYEE, and PART_TIME_EMPLOYEE entities exist in a view, and EMPLOYEE--Isa--PERSON, as well as PART_TIME_EMPLOYEE--Isa--EMPLOYEE has been expressed, there is no need to also express PART_TIME_EMPLOYEE--Isa--PERSON.

32. If an Isa relationship hierarchy implies another Isa relationship hierarchy because of transitivity, the implied Isa relationship shall be removed.

This rule assures the removal of already existing redundant Isa relationships in multi-level hierarchies. If for example view 1 states that PART_TIME_EMPLOYEE--Isa--EMPLOYEE--Isa--PERSON, while view 2 expresses that PART_TIME_EMPLOYEE--Isa--PERSON, expressed, the transitive Isa in view 2 contains both Isa's in view 1 and is redundant. It has to be removed.

33. Creation of a new superset or subset object will result in relocation of relationships if these relationships were previously linked to entities at an incorrect level of generalization.

Whenever a new superset-subset relationship is introduced into a view, the possibility exists that existing relationships may have to be relocated. Consider the following example:

V1: DEPARTMENT--Employs--FULLTIME_EMPLOYEE,

V2: FULLTIME_EMPLOYEE--Isa--EMPLOYEE.

In V1, Employs refers to FULLTIME_EMPLOYEE, because no more general EMPLOYEE object exists. Once the new EMPLOYEE becomes

part of V1, the Employs relationship will be relocated to associate DEPARTMENT with EMPLOYEE.

V1/V2: DEPARTMENT--Employs--EMPLOYEE--Isa--FULLTIME_EMPLOYEE.

Process Rules:

34. In view integration, the test for identity (conflict recognition and reconciliation) shall precede the test for relatedness.

The test for identity and the test for relatedness are two independent phases of view integration. The test for identity detects or creates identical pairs of objects in the involved views so that finally for each object in view V1 exactly one identical object exists in view V2. The test for relatedness has the purpose to detect currently missing forms of relatedness (set relationships) between views. Its purpose is not to detect within-view relatedness. All occurrences of within-view relatedness are supposed to be already represented in the individual views (completeness assumption). An example may illustrate this fact. V1 has employees working in departments, V2 assigns employees to projects.

View 1: EMPLOYEE--Works_in--DEPARTMENT

View 2: EMPLOYEE--Assigned_to--PROJECT

The completeness assumption postulates that no forms of relatedness exist within either of the views, because none are explicitly stated (no knowledge is interpreted as negative knowledge). For example, it is known that EMPLOYEE is not a subset of DEPARTMENT. Consequently, the search for inter-view relatedness has to focus only on those objects that originally exist in one view but not in the other. I.e., if EMPLOYEE were identical to EMPLOYEE, Works_in identical to Assigned_to, and DEPARTMENT identical to PROJECT, then no undetected inter-view relatedness could exist. In order to know which views originally existed only in one view but not in the other, the test for identity has to be carried out first. Thus, the sequence of the two independent view comparisons, for identity and for relatedness, is determined by the fact that a previous test for identity can reduce the number of comparisons for relatedness.

Process Rules for Conflict Recognition and Reconciliation:

35. For each object O1 from view V1, try to find an identical object O2 in view V2.

The purpose of the method is to either find that two views are identical, or to make them identical. Once two views are identical, one of them can be eliminated because all its information is represented in the remaining view. As defined earlier, two views are identical, if all their objects are identical. Hence, the test for identity begins with an attempt to find an identical object O2 in V2 for each object O1 from V1.

36. If no identical object O2 from V2 can be found for O1 from V1, try to find an object that has the same meaning as O1 and change the dissimilar dimensions of O1 and O2 so that they become identical.

Earlier, complete identity of objects was defined. This rule describes the action to be taken if two objects are only partially identical, if they have the same meaning. The meaning dimension as the most important dimension determines the direction of change. If the entity SUPPLIER in view 1 has the same meaning --refers to the same real world object-- as the attribute Dealer_no in view 2, both objects finally have the same name and the same construct.

37. If no object O2 with same meaning can be found, add a new object O2 to V2 where O2 is identical to O1 from V1.

If no object O2 with same meaning as O1's can be found, then O1 has no corresponding object in V2. Hence an object identical to O1 has to be added to V2.

38. For each object O2 from V2 which is different in meaning to O1 from V1 but has the same name, change the name so that no two objects with different meaning carry the same name.

This rule forbids the existence of homonyms in the database. If a homonym is found, a name change is required based on this rule. Again, name follows the more important dimension meaning. If meanings are different, names have to be different. The other dimensions, construct and context can remain as they are.

39. For each O2 in V2 that remains without an identical object from V1, after all objects in V1 have been matched with an identical object in V2, add a new object O1 to V1 which is identical to O2.

View V2 may contain objects that are not part of V1. Hence, after all of V1's objects have been assigned an identical object in V2, some of the objects in V2 may be left without an identical object in V1. Consequently, these objects have to be added to V1.

Process Rules for the Recognition and Modelling of Inter-Schema Relationships:

40. Compare each object O1 from V1 which was originally unique to V1 (before addition of missing objects during identity test) against all objects (O2) formerly unique to V2, to find out whether O1 contains O2, or O2 contains O1. Represent each identified containment.

Purpose of the analysis is only the addition of missing inter-view superset-subset relationships. Therefore, the containment test applies only to objects that were originally unique to one of the two views. For example:

View 1: PART--Last_ordered_from--SUPPLIER

View 2: PART--Carried_by--DEALER

Here PART is the same in both views and therefore is not unique. Hence, only Carried_by, Last_ordered_from, DEALER, and SUPPLIER, are potentially related to objects from the other view. I.e., DEALER could be related to Last_ordered_from or to SUPPLIER, Last_ordered_from could be related to DEALER or to Carried_by. If, for instance all SUPPLIERs are DEALERs but not all DEALERs are SUPPLIERs, then DEALER contains SUPPLIER. Consequently, an Isa relationship between SUPPLIER and DEALER would have to be created.

The comparison summarized in this rule is the first test for relatedness, because it the most special case of relatedness and requires the least change in the existing views. The comparison A contains B is a special case of common containment (A contains A and A contains B), as well as a special case of common subset (B is a subset of A and B is a subset of B). In this special case, only an Isa relationship is added to the views. In the general case, the common superset and the common subset have to be added too. Thus, if this test is the first one, the subsequent steps are simplified.

41. For all pairs of originally unique objects O1, O2 in which neither object contains the other, investigate whether O1 and O2 are contained by a common object

different from O1 and O2. Represent the common containment.

This rule summarizes the procedure for a common containment where the containing object is different from O1 or O2. Only those objects are compared that were originally represented in one view only. All object pairs in which one object contains the other are not considered.

42. For all pairs of originally unique objects O1, O2 in which neither object contains the other and which have a common superset, also investigate whether O1 and O2 intersect. Represent any existing common subsets. Represent the lack of a common subset through an integrity constraint.

This rule summarizes the procedure for a common subset where the intersect object is different from O1 or O2. Only those objects are compared that were originally represented in one view only. Also, only objects that have a common superset (different from O1 and O2) are compared. Objects without a meaningful common superset cannot have a meaningful common subset.

43. For all object pairs O1, O2 originally unique to one view, investigate the existence of a W-relationship (common role). Represent any existing W-relationships.

Even though the test for related objects may not find any relatedness among the objects themselves, objects can have a common role, which requires the addition of objects to represent the common role and the objects' special role. I.e., both a company and a person can be car owners. Even though company and person are not related (i.e., have no meaningful common superset in the database), their common role car owner requires representation, as do their special roles person-car-owner and company-car-owner.

Heuristics

Heuristics are rules that are generally true, but not true in all cases. The use of these rules during the view integration process will simplify the process for the user in cases where the heuristics are true and will slightly inconvenience or prolong the process when the heuristic fails. The use of incorrect heuristics will not result in an incorrect database design, but it may prolong the database design process.

Heuristics improve the integration process by helping the user to find objects with similar or related meaning. If object O1 is compared to a set of objects {O2} from view 2 and that set is large and diverse (large number of objects including entities, relationships and attributes), the selection problem may be difficult for the user. If the set {O2} is small, the selection problem becomes simple or even trivial. Heuristics help to simplify the selection problem by including only those objects in the set that are likely to be identical or related to the object O1.

The list below shows only some heuristics, it cannot be complete. It is always possible to formulate further assumptions to simplify the search procedure. Furthermore, some of the heuristics shown may be too stringent for a particular design, others may be too loose. Heuristics that are too stringent are a particular problem, since they can result in decision errors which require lengthy recovery procedures. This problem is exemplified in the next section which shows alternative view integration procedures, one without any heuristics, one with only one heuristic implemented.

The following heuristics have been identified:

1. Two objects with identical or related meaning will have some common context.

This rule says that identical or related objects will be found in the vicinity of identical objects. For example, if it has been found that there exists an entity EMPLOYEE in views V1 and V2, and EMPLOYEE in V1 participates in a relationship Employment, then it is reasonable to assume that EMPLOYEE will participate in a similar association in V2, even though that association may not be called Employment in V2 and even though it may not be a relationship.

The heuristic is based on the assumption that people describing the same environment will have the same perception of the environment. Since both views have common elements, both views describe at least partially the same real world environment. In the absence of information to the contrary, the method therefore that all users regard the same real world objects and associations as relevant.

In the example, the heuristic fails if the Employment association is not relevant in V2 and therefore missing. Note however, that the Employment association may not be missing, but be more difficult to find, if in V2 it is not represented as a relationship, but as an entity attribute or as an entity.

Even though entities are defined to have no context it is useful to treat the relationships they are involved in as their context, to permit the application of this valuable heuristic.

2. Two objects with identical or related meaning will have the same construct.

This rule states that even before conflict resolution, two object with identical or related meaning will be of the same type. Thus, the rule leads the integration method to look for a matching object only among those with the same construct. If EMPLOYEE is an entity in V1, the matching object in V2 will also be an entity.

This heuristic is based on the assumption that if two people describe the same object or association from the real world, they will agree in their assessment of the construct that the object or association should be represented with. Depending on the real world item, this assumption is more or less reasonable. One would assume that almost anyone considers an employee or a customer to be an individual, but a customer's order may be perceived as a thing (entity), or as an association (relationship) between a customer and a company.

The heuristic fails in all cases of construct mismatch (semantic relativism), i.e., where one real world object is represented as an entity in one view and as a relationship in the other view. For cases in which the rule fails, the integration procedure has to backtrack and look at objects with different constructs to find a match.

3. If no two objects with identical or related meaning and identical construct can be found, the construct mismatch will be of the following type:
   - If O1 is an entity or a relationship, then O2 will be an entity attribute.

This heuristic suggests which construct mismatch to investigate first. Storey (1988) found that a very common error in database design was the representation of an entity-relationship construct as an interconnection attribute. Since this "mistake" is very frequently made, checking for its occurrence when an identical object was not found is useful. In combination with the common context heuristic, this heuristic is expected to reduce the set {O2} to a manageable size.

Some attributes can under no circumstance be interconnection attributes, while others are more likely to be interconnection

attributes. Two support rules help in identifying these groups:

- a single attribute object key cannot be an interconnection attribute.
- attributes in a multi-attribute object key (composite key) are assumed to be interconnection attributes.

For example, Employee# is the single attribute key of an employee. It does not represent the relationship between EMPLOYEE and some other object. In contrast, the key of an ORDER entity, Customerid+Product# identifies links to two other objects, a customer object and a product object. Both are potential interconnection attributes.

Since more forms of mismatches other than the interconnection attributes exist, the heuristic can fail. To recover from this failure, the system will then search according to the following rules:

- If O1 is an entity and O2 is not an entity attribute then O2 will be a relationship attribute.
- If O1 is a relationship and O2 is not an entity attribute then O2 will be an entity.

These are the only other alternatives for construct mismatch, aside from the interconnection attribute assumption. However, any of these rules may fail too, if an object is missing.

4. Objects with identical meaning will have identical names (consider a name in singular identical with its plural).

This heuristic assumes that a particular application uses a standardized language to label its objects. In absence of information to the contrary, members of the same organization are expected to use terms to label the same objects. For instance, terms such as "department" or "job classification" or "account" are expected to be used consistently. If this were true, synonyms and homonyms would not exist. Hence, this assumption is expected to have very limited reliability. Nevertheless, it provides a good starting point in the search for matching pairs of objects at the outset of the integration procedure.

When this heuristic is applied, two objects are treated as having the same name even if one is in singular form while the other one in the plural (i.e., employee vs. employees).

If the heuristic fails, the search for a matching object has to continue among all objects with different names.

5. Objects with related meaning will have names with identical word stems.

In the search for related objects, the word stem can be a very strong filter to identify those objects that are likely unrelated. For example, FULLTIME_EMPLOYEE and EMPLOYEE have the same stem employee, GRADUATE_STUDENT and UNDERGRADUATE_STUDENT have the same student stem. Thus, they are likely to be related. An even stronger interpretation of the word stem phenomenon may conclude that if one object's name is the word stem, it will be the superset of the other object, while two object with different prefixes have a common superset.

Again, since synonyms and homonyms are frequent, this rule will be of only limited use. Nevertheless, in a computerized procedure, it requires no user effort and is therefore a desirable feature, even if its benefits may be marginal.

6. Two objects with identical or related meaning will have some attributes with identical names (for entities and relationships only).

Especially in the search for identical objects, this rule can be used to eliminate those objects that are very unlikely candidates for identity. Two different views describing the

same EMPLOYEE entity are expected to use at least some identical attributes to specify employee properties. In particular, identical or related objects are assumed to have the same key attributes (with the same key attribute names).

Obviously, homonymy is a problem in this context. Attributes may be identical, but attribute names may be not.

7. Objects with identical or related meaning will belong to the same pre-defined meaning category.

In a subsequent section, a hierarchy of object categories will be introduced which provides a structure for the categorization of database objects according to their meaning, i.e., as an "animate object". If each object's meaning is pre-defined, in terms of the category it belongs to, then two objects from different categories cannot be identical. Again, this heuristic provides a filter to eliminate non-identical objects.

4.2. Diagnosis Procedure

The conflict and omission recognition procedure consists of two parts: the test for identity of objects (object types), and the test for relatedness of objects. The test for identity is concerned with the identification of identical objects in the observed views; the test for relatedness is concerned with the detection of inter-view set relationships (object relatedness).

Even though an object from one view can have at most one corresponding object in any other view, more than one object of another view can be related to it. Relatedness means that there exists a set relationship between the objects. The relatedness question has to be approached independently. It is impossible to conclude the relatedness or non-relatedness of objects from the existence of a pair of identical objects, or vice versa.

The first question will refer to the identity of objects. In order to restrict the test for relatedness only to inter-view relatedness, the relatedness test has to be preceded by the test for identity. Inter-view relationships can only exist between objects that are originally unique to one view. To find out, which objects have no corresponding objects in the

other view, the test for object identity has to be performed[1].

Test for Identity of Objects

The purpose of this test is to answer the question "does there exist an object O2 in V2 which is identical to O1 from V1?", i.e. if view 1 contains an entity SUPPLIER, does view 2 also contain an entity with same name and same meaning. Again, "same meaning" can be interpreted as "both objects refer to the same object in the real world". Obviously, finding a perfect match will be the exception. It is more likely that objects will be found that are somewhat similar, but not identical. In such cases, adjustments have to be made. The general rule is to make objects completely identical if they refer to the same real world objects (have same meaning). In such cases, possible mismatches in name, construct or context will be adjusted. If objects refer to different real world objects, then a possible, but undesirable, match in their names (homonym) has to be corrected.

The test for identity is carried out incrementally, with a comparison of the involved objects along one dimension at a time. All tests compare one object from view 1 to a set of

---

[1] The test procedures will frequently mention therapy procedures to resolve conflicts or to reflect inter-view relationships, without going into much detail. Detailed solution descriptions will be given in the subsequent section.

objects from view 2, to find the ones that fulfill the condition of the test. Objects are identical if their four dimensions are identical. Since the meaning dimension is the most important one--other dimensions are adjusted accordingly--it presents a good starting point for the analysis. The main problem with this approach is that an object O1 from view V1 is compared to all objects O2 from V2, independent of their name, construct or context, even though only one object from V2 can be identical to O1. This may require that the user check a long list of irrelevant objects. The heuristics introduced in the previous section can be used to alleviate the problem. Therefore, a second procedure will be shown which includes the heuristic "objects with identical meaning will have identical constructs", to exemplify the effect of heuristics. This second procedure begins with a search for objects with constructs identical to that of O1.

While it is important to begin with the meaning dimension in the first procedure, the analysis sequence for other dimensions may vary. The order chosen here is: construct, context, name. Construct analysis has to precede context analysis, because every test for identity may result in a change in that dimension. For example, a test for identity of construct will cause a construct change, if constructs are not identical. But a construct change will also result in a context change. In contrast, context changes do not affect the construct. Thus,

no test for identity of context should be executed until constructs have become identical. Name identity analysis should follow construct analysis, because the user may decide to give objects different names, which are based on their construct. The complete procedure is depicted in flowchart form in Figure 6 (with abbreviated notation).

To illustrate the whole procedure with an example, it will be assumed that an object O1 from view V1 is selected at random, i.e., the entity type SUPPLIER which denotes the set of current suppliers of a company. With this object held fixed, the following tests are carried out:

The procedure begins with the goal to find an object O2 with identical meaning to O1. To find the object, the procedure generates the hypothesis H1 "there exists an object O2 from V2 such that O2 is identical in meaning to O1". Directed towards the user, it results in the question "which object from view V1 is identical in meaning to O1?" The use can then either identify an object, or reply with a "none". For example, view V2 may contain an entity MANUFACTURER which is used in V2 to describe all suppliers. If a matching object is found, the system state s'=s1 is reached. If not, s'=s5. In contrast to the subsequent hypotheses H2-H4, this test compares O1 to a set {O2} from view V2 rather than to a single object. {O2} contains all objects from V2 which so far have not been

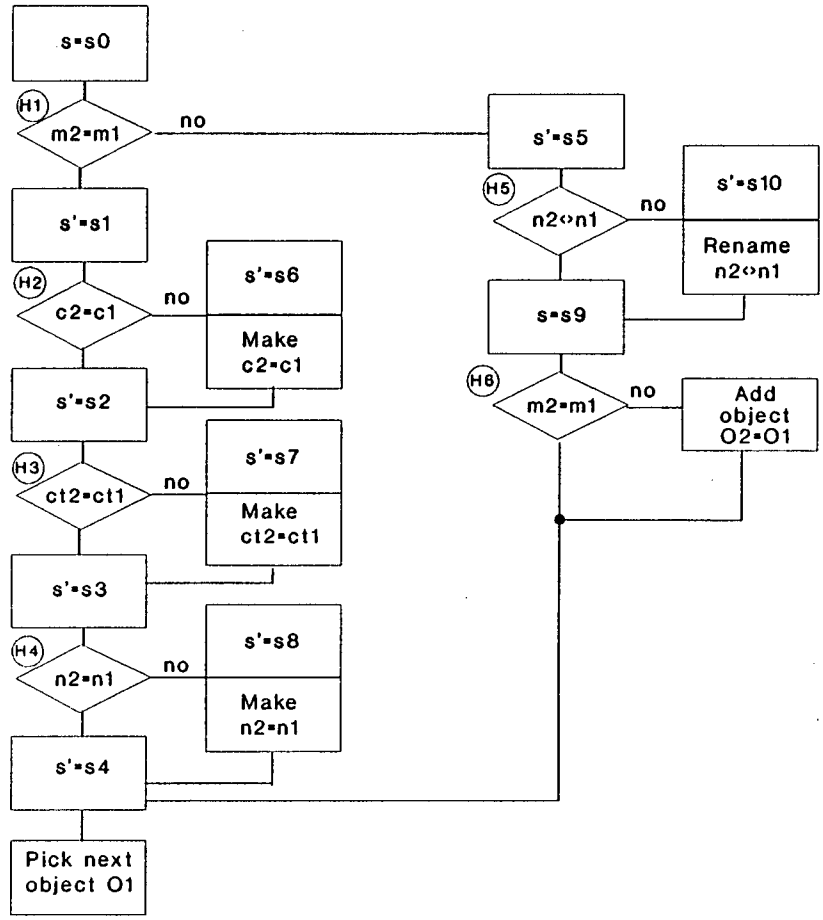


Figure 6: Test for Object Identity, Procedure without Heuristics

matched up with an object from V1. As a result of H1, either one of these objects will find a matching object in V1, while the remaining n-1 objects will be in state s5, or all objects from {O2} will be in state s5. In other words, for most, if not all objects from V2, the result of this test will be state s5. Thus, in the flowchart in Figure 6, for most if not all objects in {O2}, the outcome of H1 will be the "no" path, while at most one object will follow the "yes" path.

If a matching object is found, the method continues with hypothesis H2 which states that O1 and O2 will have the same construct, i.e., that both are entities. The method issues the question, "do O1 and O2 have the same construct?" In a computerized view integration system, the integration procedure will look up the information to answer this question from the view definitions. Should both objects have different constructs (s'=s6), a construct change would have to occur. If the constructs are identical, state s'=s2 is reached. In the example, SUPPLIER and MANUFACTURER are both entities and thus have identical constructs.

Subsequent to s2, the system checks for identical context. Are O1 and O2 associated with identical objects? For entities, the answer to this question is always positive, since their context is an empty set. If O1 and O2 are relationships or attributes and not all their context objects have been matched

to objects in the other view yet, then the identity test for O1 and O2 is suspended, until the context objects are matched to objects in the other view. If the result of the context test is that O1 and O2 have different contexts (s'=s7), the contexts have to be made identical (s'=s3). In the example, both object are entities. Thus, both have identical (empty) contexts.

If state s3 has been reached, the remaining test is the test for name identity of the objects. The method's hypothesis is that both objects have identical names. If they do not share the same name (s'=s8), their names are made identical (s'=s4) through a change of at least one of the names. The new name will have to be different from the names of all other objects in V1 and V2 to avoid homonymy. In the example, at least one of the entities would require a name change. The name chosen should be such that it is not identical to the name of another object.

Once the pair of objects is identical in all four dimensions, the identity test is completed for this pair. The method continues by selecting a new object O1 from view V1, and subjecting it to the same analysis. The procedure terminates when all objects have a matching object in the other view.

The set of all objects {O2} from V2 that, as a result of H1, are known to be different in meaning from O1 (s'=s5) is subject to further analysis. H5 tests whether all of the objects have names different from O1's name. All objects with same names (s10) require renaming to make their names unique (s9). In addition, if none of the objects {O2} was identical in meaning to O1, a new object O2, completely identical to O1, has to be added to achieve the state s4.

The use of heuristics results in changes to the view integration procedure. To exemplify such changes, a procedure will be discussed below that includes only one heuristic: "objects with identical meaning will have identical constructs." This heuristic is in fact one of the heuristics implemented in the view integration program AVIS. Again, the procedure begins by picking one object O1 from view V1. It again will attempt to find an object in view V2 that is identical to O1.

The procedure (see Figure 7) begins with the goal "find the set of objects {O2} from V2 that have the same construct as object O1". Since the procedure assumes that all objects with same meaning have the same construct, it decides to only consider those objects O2 for further identity testing that have the same construct as O1. A number of objects from V2 will qualify and thus be in state s0, while the objects of

different type will be in state s5. Since in the example SUPPLIER is an entity, all entities from V2 would be considered for further identity testing. One may want to think of the use of construct as a "filter" which can reduce the number of objects to be considered, hopefully without being too stringent a condition.

For those objects with same construct, the procedure then investigates whether there exists an object O2 which has the same meaning as O1 from V1. I.e., it is looking for an entity in V2 identical in meaning to SUPPLIER. Again, at most one object of V2 is allowed to fulfill this condition. That object will be in state s1. All objects with different meaning will be in state s6. If an object with same meaning is found, the procedure continues with the context (H3) and name (H4) tests, similar to the tests above. However, if no object in V2 is found to have the same meaning as O1, the procedure continues differently, to verify one of two possible interpretations of the situation. The first possibility is that the heuristic is wrong. Thus, an object O2 with same meaning but different construct exists in V2. The second possibility is that no object with identical meaning exists in V2, regardless of construct. The procedure has to find out which alternative is true, to avoid the creation of a non-minimal global schema.

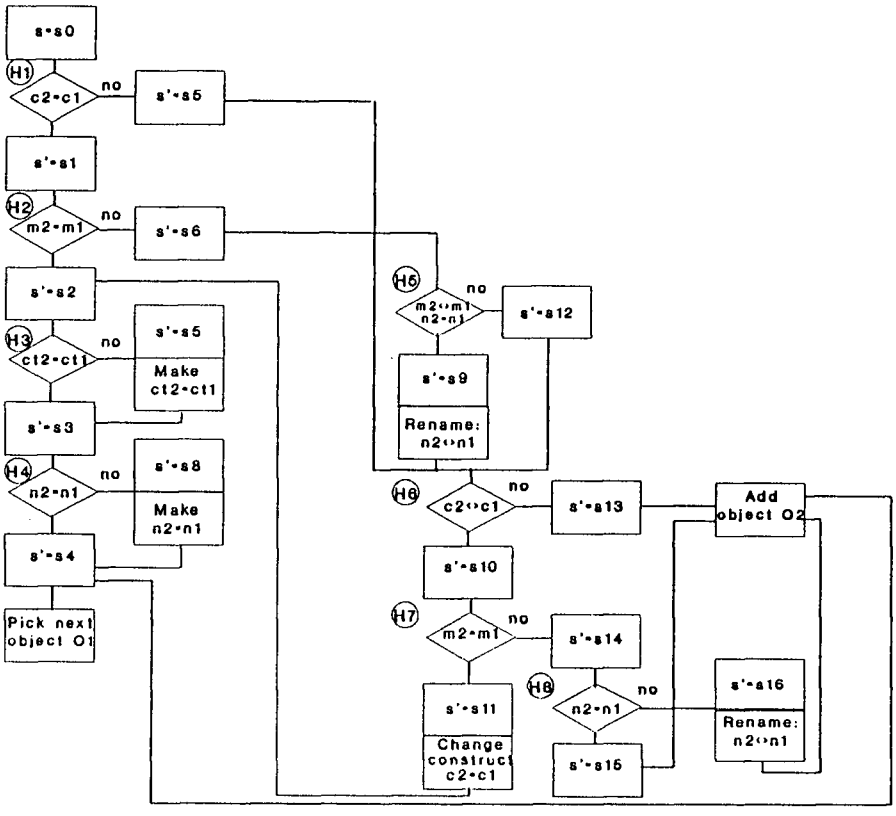


Figure 7: Test for Identity with Heuristic

Thus, after taking care of homonyms (H5), the procedure continues with a test to identify those objects with constructs different from O1's construct. In the figure, this test is shown in abbreviated notation as c2<>c1. Its correct interpretation is "are there any objects in V2 that have a different construct?" This question may appear redundant for the objects in s5, because they failed the "same context" test. However, the set of objects in state s5 may be the empty set. Thus, they would qualify for the answer "no" to question H6 (s13), requiring the addition of a new object.

If there are objects in V2 with constructs different from O1's, the procedure checks whether any of them have the same meaning as O1 (H7). If an object with same meaning is found (s11), its construct has to be changed. If no such object exists (s14), a test for homonymy follows (H8), resulting in a name change for all homonyms. Subsequently, the missing object is added.

In this procedure variant, the main effect is a sequence change with respect to the tests for meaning identity and construct identity. It results in a prolongation of the procedure if the heuristic is wrong.

The procedure could be varied further, for instance by a switch in the sequence of meaning identity and context identity test. Therefore, the test for meaning identity would follow the test for construct and context identity. Consequently, only those objects with same construct and same context would initially be considered for the meaning identity test. This procedure change would reflect the heuristic "identical objects are in the vicinity of identical objects." The procedure would look in the neighborhood of matching objects to find further matching objects. This heuristic is, in modified form, also implemented in AVIS. AVIS requires only part of the context to be identical.

The test for meaning identity could even be moved past the test for name identity to reflect the heuristic that objects with same meaning will have same names. Since this heuristic is expected to be frequently wrong, it has not been implemented in AVIS.

Test for Relatedness of Objects

The purpose of this test is to find out whether aside from being identical, objects from one view are related to objects from another view through set relationships. I.e., an entity

(type) SUPPLIER in V1 is a subset of an entity DEALER in V2. Such a case would exist in a situation where SUPPLIER referred to all current suppliers of the company, while DEALER refers to all present and all potential suppliers of the company. If those relationships are not made explicit, anomalies can occur. I.e., if a member is dropped from the entity set DEALER, it should also be automatically dropped from the entity set SUPPLIER. Furthermore, attribute inheritance can be derived from set relationships.

The procedure described below is a generic procedure without the use of heuristics (see Figure 8). It begins with a test for containment (H1 and H2). Subject of the test is whether one of the objects is contained by the other object, i.e., SUPPLIER is contained by DEALER. The procedure first determines the set {O2} of objects contained by O1, and then, for those objects not contained by O1, the set {O2'} containing O1. The way the question is raised to the user is "Which of the objects (in V2) are contained by O1", and vice versa "which of the objects (in V2) contain O1?" It is possible that O1 contains some objects in V2 while being itself contained by others. I.e., SUPPLIER (V1) is contained by DEALER (V2) but may contain another object SMALL_QTY_SUPPLIER from V2. In such a situation an Isa relationship between DEALER and SMALL_QTY_SUPPLIER would have existed which now would have to be removed because it is a transitive Isa.

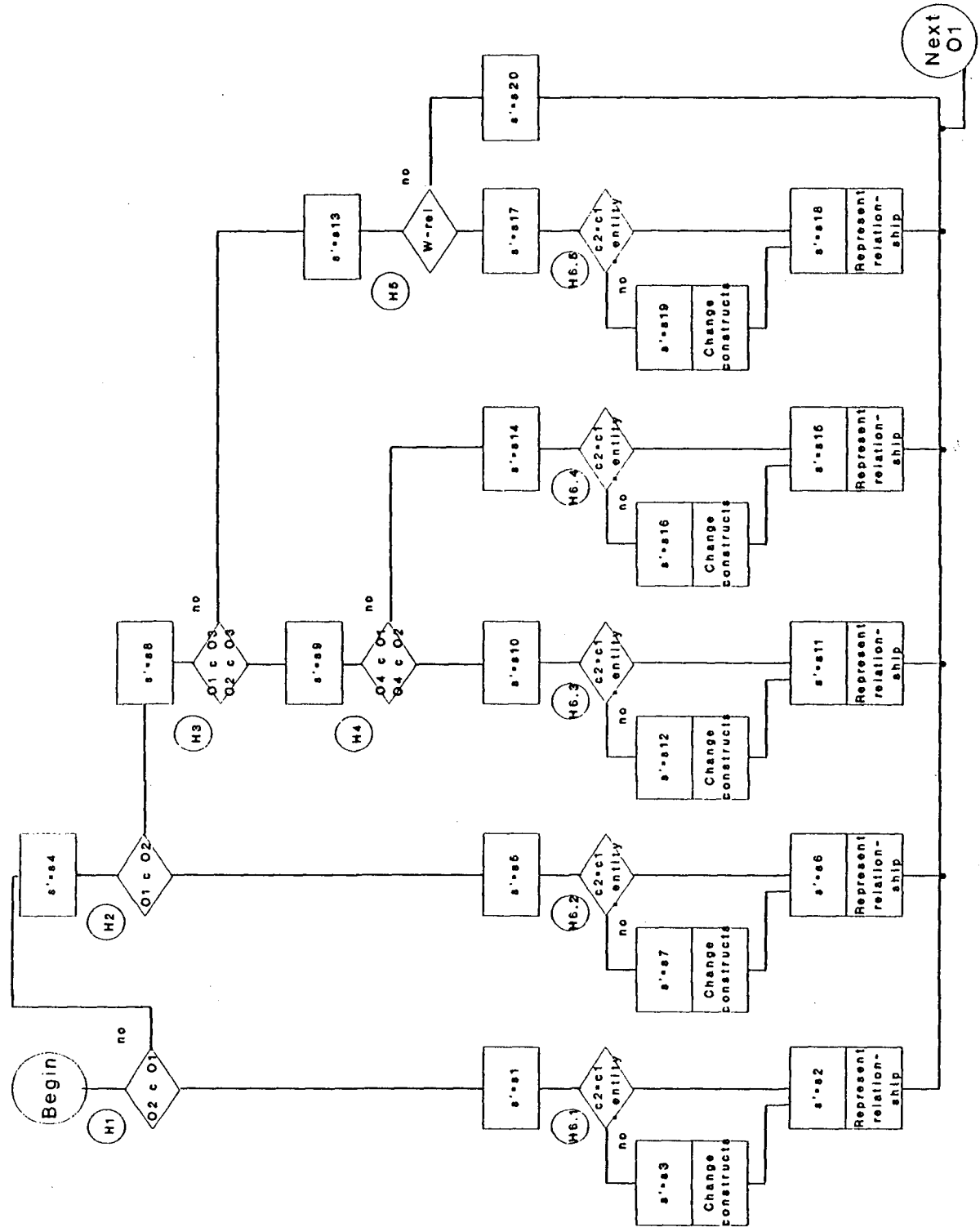


Figure 8: Test for Relatedness of Objects

The containment test is the first one issued, because it is the most specialized form of common containment and common superset, requiring the least amount of additions to the existing views. Only one Isa relationship has to be established between the objects. The insertion of an Isa between the objects requires, however, that both objects are entities. If they are not, all of them which are not entities have to be converted into entities. The test H6.1 is executed to determine whether both objects are entities.

The entity test (H6) is issued for each pair of objects after their relatedness has been discovered. There is no need to test for object type earlier, since only related objects that are not entities will require construct changes. Unrelated objects will keep their original constructs. Since the object type test (H6) is identical for all forms of relatedness (H6.1 - H6.4), it will not be discussed further in the procedure.

Should neither object contain the other one (s8), the procedure inquires whether both objects have a common superset (H3). If they do, the procedure further inquires whether a common subset exists between them (H4). The common superset question precedes the common subset question, because objects that have a (meaningful) common subset and are themselves meaningful sets have to have a (meaningful) common superset. Although it

is possible to construct sets such as the set of "all green things" and the set of "all edible things" which have a common subset in the set of "all green edible things", while having no meaningful superset other than "all things", the rule is nevertheless valid when only meaningful sets are considered. In the example, especially the set "green things" is not a meaningful set as it has no clearly defined attributes (rather than green color) which we expect for an entity or relationship type.

If objects have both a common superset and subset (s10), two new objects will be created to represent the superset and the subset. Also, new Isa relationships will be created to represent the relatedness. If the objects have a common superset but no common subset (s14), only a common superset entity and the corresponding Isa relationships will be added. In addition, an integrity constraint may be defined to identify that the objects are not overlapping.

Objects without a common superset (s13) are tested for the existence of a W-relationship (Goldstein and Storey, 1988). If no common superset exists, the objects are in fact not related. Yet the objects may still require the creation of inter-view relationships if they have a common role. If the objects have a common role, i.e., both a PERSON and a COMPANY entity may be car-owners, a new object describing the common

role (CAR_OWNER), plus objects describing the special roles (PERSON_CAR_OWNER, COMPANY_CAR_OWNER) have to be created. Furthermore, Isa relationships have to be added to represent the associations between the objects.

If not even a W-relationship exists between the objects, they are unrelated and require no addition of inter-view relationship objects.

4.3. Conflict Therapy

As soon as a conflict is detected by the diagnosis procedure, the integration method will correct the problem. Thus, while there exists a diagnosis procedure to recognize conflicts, there exists no therapy procedure per se. Instead, for each conflict case, a case solution is defined. All case solutions are based on a set of 11 elementary solution operations which were formulated earlier as rules guiding view integration:

1. Relationship becomes an entity.
2. Relationship attribute becomes an entity.
3. Entity attribute becomes an E-R construct.
4. Association of an entity to a relationship.
5. Relocation of a relationship after creation of new superset or subset classes.
6. Representation of containment.
7. Representation of a common role (W-relationship).
8. Representation of common superset without overlap.
9. Representation of common superset with overlap.
10. Renaming of homonyms and synonyms.
11. Addition of missing objects.

One or more of these elementary therapy measures may have to be carried out during conflict reconciliation. Each of them

will be described in detail. Appendix 2 will show which groups of elementary solutions will be applied to specific conflict cases and their sub-cases.

Relationship becomes an entity (S1)

Whenever necessary, a relationship is transformed into an entity. If a relationship becomes an entity, the linkages between the relationship and the entities it associated become relationships themselves (see Figure 9).

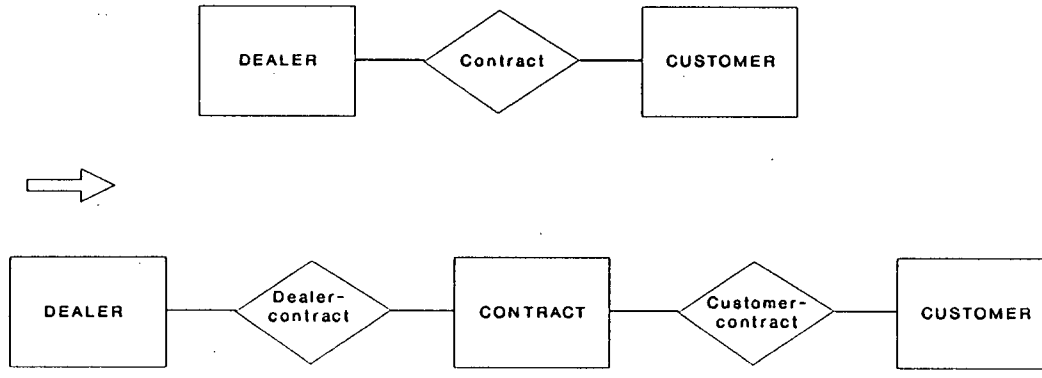


Figure 9: Relationship Becomes an Entity

The entity construct is the more fundamental one. Furthermore, an entity can be associated to other entities by means of a relationship, i.e. an Isa relationship. Consequently, for the newly created entity set relationships to other objects can be represented within the E-R modelling language. In the example in the figure, the relationship Contract between DEALER and CUSTOMER becomes an entity itself and two new relationships, Dealer_contract and Customer_contract are created in addition.

Relationship attribute becomes an entity (S2)

When necessary, relationship attributes are converted into entities and a linkage is expressed between the relationship and the newly created entity (see Figure 10).

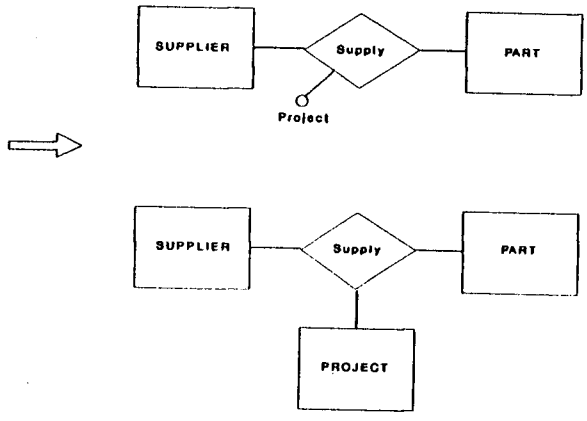


Figure 10: Relationship Attribute Becomes an Entity

Relationship attributes that have to be transformed into entities are interconnection attributes. Interconnection attributes represent entities (or E-R constructs) in shortened form. If the database requires that an interconnection attribute be associated with another object, it first has to be converted into an entity (or an E-R construct). In the illustration, SUPPLIER is associated with PART through the Supply relationship which has an attribute Project. This attribute subsequently becomes an entity.

Entity attribute becomes an E-R construct (S3)

Similar to relationship attributes, entity attributes may have to be transformed, if they require association with other objects, or if another view represents them differently. An entity attribute which is an interconnection attribute represents an entity-relationship construct in shortened form. Therefore, it will be converted into an entity-relationship structure (see Figure 11).

Typically, the newly created entity will refer to the same real world object that the original attribute referred to. However, the user may think of the newly created relationship as the object that corresponds to the original attribute. In

fact, the attribute corresponds to both the entity and the relationship. In the example, the PART entity has an attribute Supplier which in fact represents a Supply relationship and a SUPPLIER entity in shortened form.

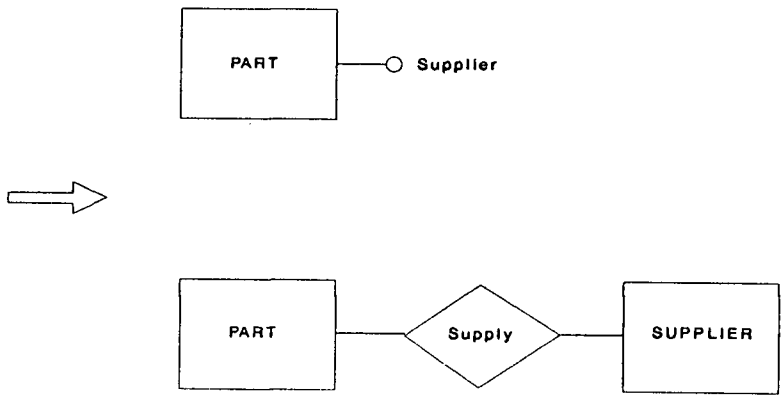


Figure 11: Entity Attribute Becomes an Entity-Relationship Construct

Association of an entity to a relationship (S4)

A conflict situation may require the association of an already existing entity with an already existing relationship. The new element added to the view is the association link (role) between the entity and the relationship (see Figure 12).

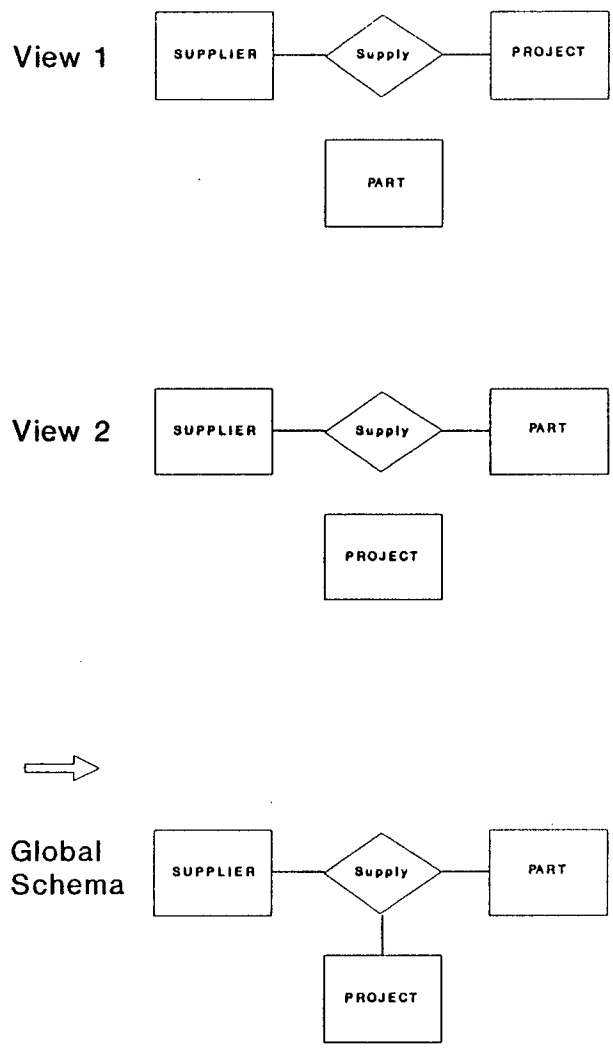


Figure 12: Association of an Entity to a Relationship

Such a situation arises when two relationships are similar, even though one involves only a subset of the entity types associated by the other relationship, i.e. one is a binary, the other a ternary relationship. The figure shows a Supply relationship, involving only the SUPPLIER and PART in the first relationship. Subsequently, the PROJECT entity is also tied into the relationship.

Relocation of a relationship after creation of new superset or subset classes (S5)

Whenever a new superset-subset relationship is introduced into a view, the possibility exists that existing relationships may have to be relocated. Figure 13 shows such a case. In view V1 DEPARTMENT Employs FULLTIME_EMPLOYEE, while view V2 reveals that every FULLTIME_EMPLOYEE is an EMPLOYEE. Once the views are combined, it becomes evident that the Employs relationship should associate DEPARTMENT with EMPLOYEE rather than with FULLTIME_EMPLOYEE. Hence, the Employs relationship is relocated.

Relocation becomes necessary whenever the original relationship, i.e. Employs, should have referred to either a more general object, i.e. EMPLOYEE instead of FULLTIME_EMPLOYEE, or to a more specific object.

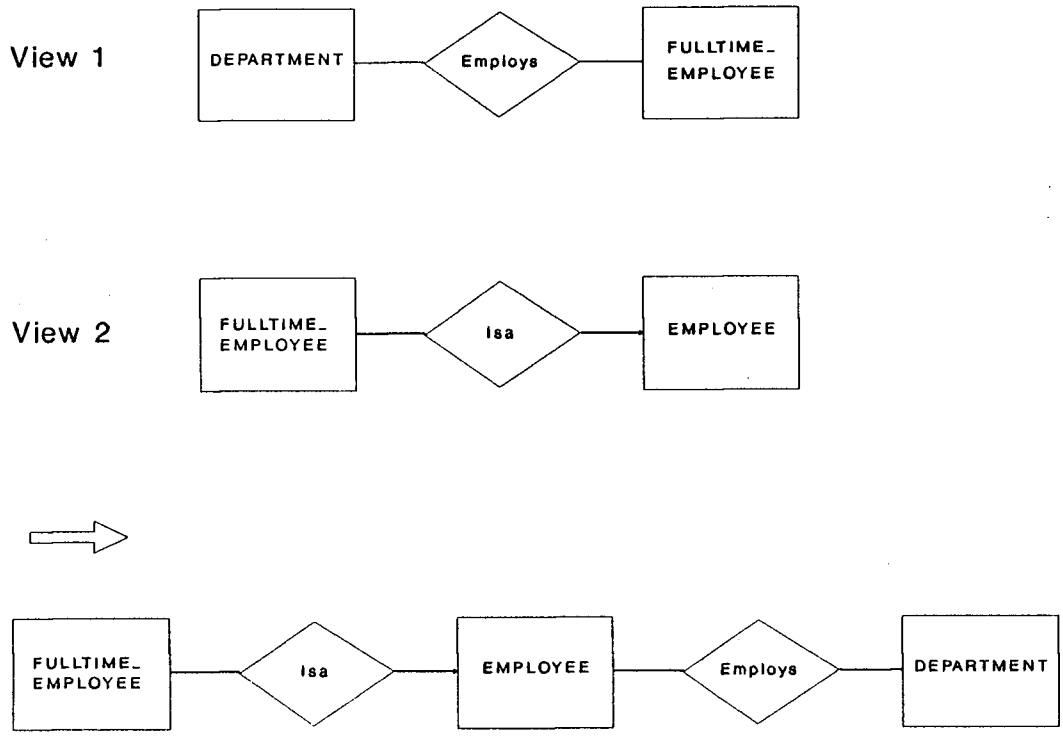


Figure 13: Relationship Relocation

Representation of containment (S6)

Whenever one object (class) represents the superset of another object and this superset-subset relationship is meaningful for

the database, it has to be represented by an Isa relationship between the two objects (see Figure 14).

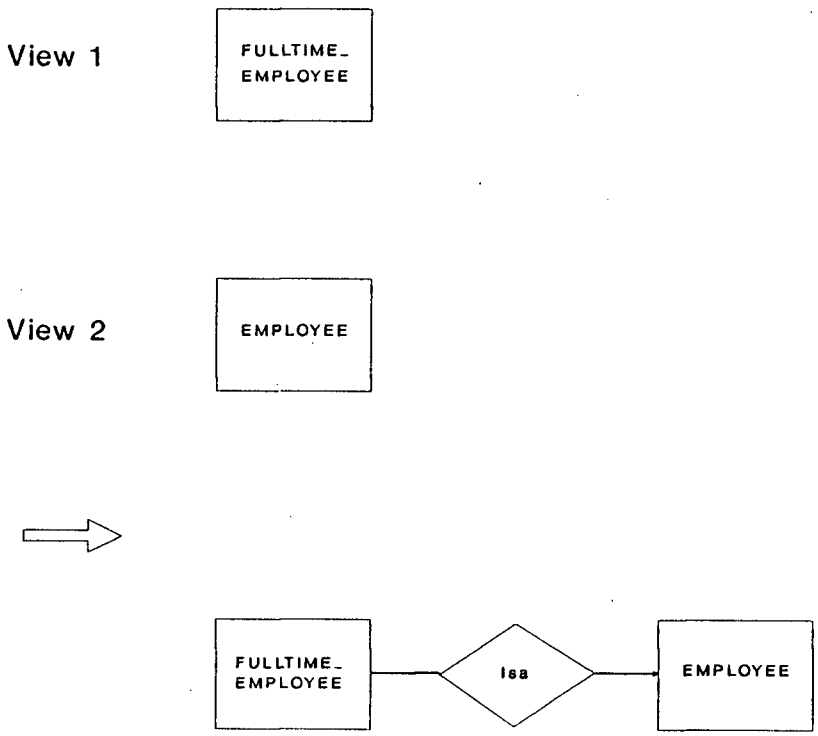


Figure 14: Representation of Containment

The illustration in the figure shows the creation of an Isa relationship between an EMPLOYEE and a FULLTIME_EMPLOYEE entity.

Representation of a common role (W-relationship) (S7)

Two objects can be unrelated but nevertheless have some affinity to each other, if they assume a common role. Goldstein and Storey (1988) identify this affinity as a W-relationship. Figure 15 depicts two entities, COMPANY and PERSON, as unrelated but both assuming the role of a car owner. Both people and companies can be car owners.

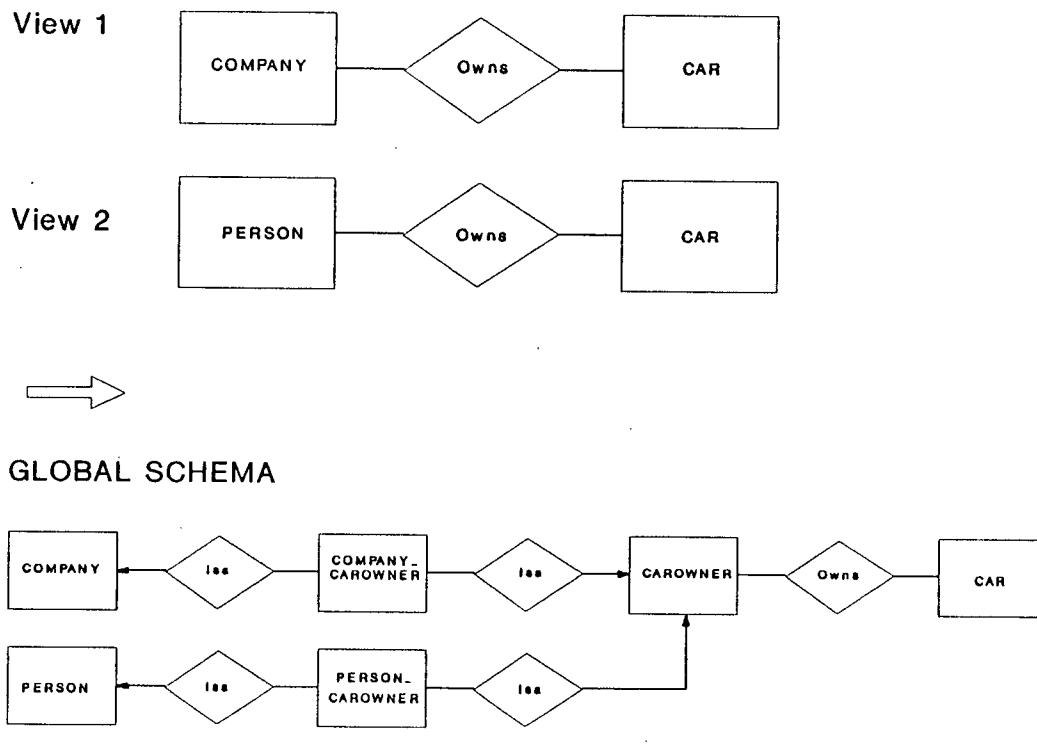


Figure 15: Representation of a Common Role

In such a situation, new objects have to be created to represent the common role, i.e. STOCKHOLDER, as well as to represent the specific roles, i.e., COMPANY_STOCKHOLDER and PERSON_STOCKHOLDER. Each object representing a specific role will be contained by one of the original objects, i.e. COMPANY or PERSON, as well as by the object representing the common role. Whenever a common role is represented, relocation of relationships may have to take place.

Representation of common superset without overlap (S8)

A Superset but no overlap describes objects that exclude each other, such as FULLTIME_EMPLOYEE and PARTTIME_EMPLOYEE. Figure 16 illustrates such a scenario and shows the creation of a new superset object EMPLOYEE, connected to the original objects through two Isa relationships.

The example in Figure 16 is based on the assumption that the EMPLOYEE entity has not previously existed in either of the view. Whenever a common superset is represented, relocation of relationships may have to occur.

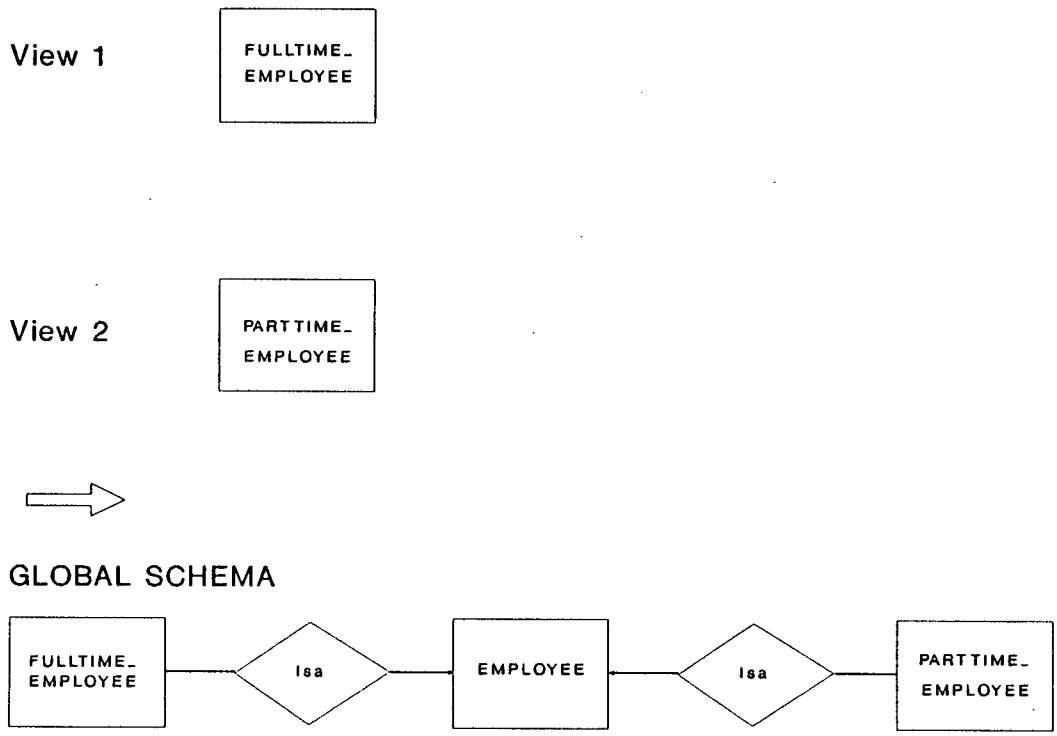


Figure 16: Representation of a Common Superset without Common Subset

Representation of common superset with overlap (S9)

In situations where two objects not only have a common superset but also a common subset (overlap) both the superset and the subset have to be represented by additional objects and Isa relationships between the original objects and the superset and subset objects (see Figure 17).

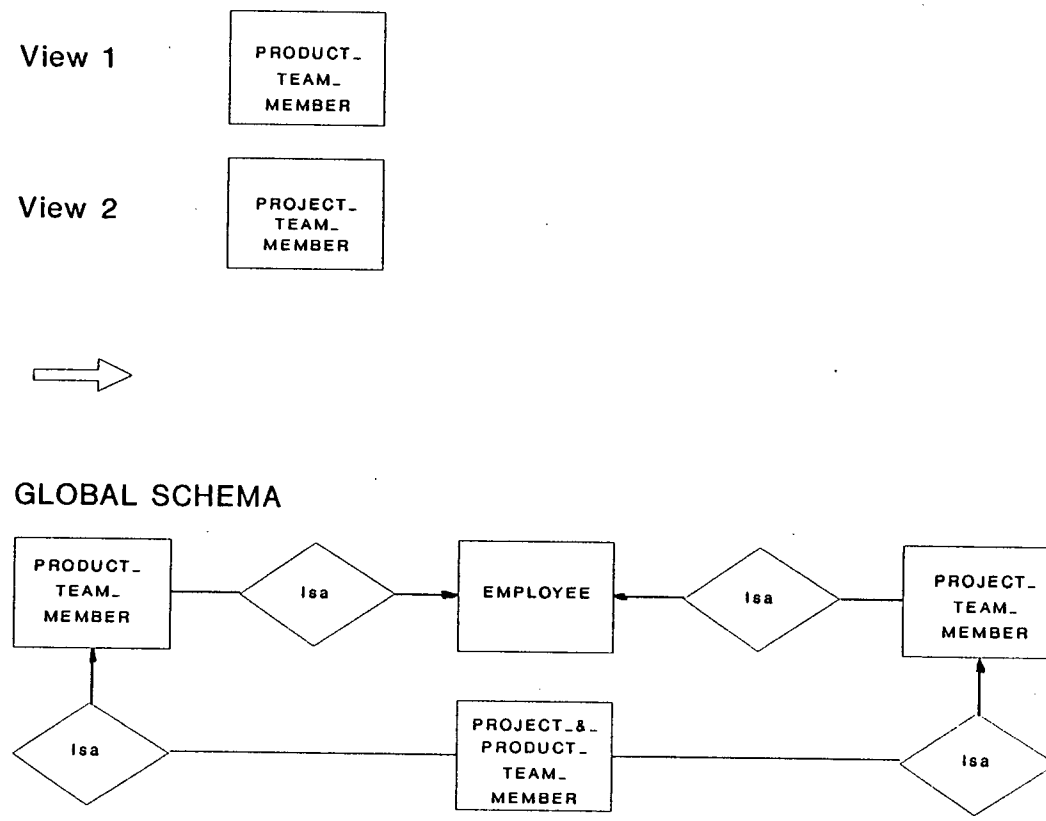


Figure 17: Representation of Common Superset and Common Subset

Figure 17 depicts PROJECT_TEAM_MEMBER and PRODUCT_TEAM_MEMBER entities. Both have the common superset EMPLOYEE and the common subset PROJECT&PRODUCT_TEAM_MEMBER. The Isa relationships represent that all team members are employees and that the members of the project&product team belong to both the project and the product team. Again, any previously existing superset, subset or Isa relationships will not be reduplicated. Whenever a common superset or a common subset is represented, relocation of relationships may have to occur.

Renaming of homonyms and synonyms (S10)

Renaming becomes necessary when otherwise identical objects carry different names (synonym), or when different objects carry the same name (homonym). Once synonyms are treated, the objects should have the same name. That name should also be different from the name of any other object in either view. Once homonyms are treated, the involved objects should carry names that are different from each other and different from all objects they are not known to be identical to.

Addition of missing objects (S11)

Objects can be missing. Most views will overlap only partially. Hence, for any two views, all objects that exist in one view but not in the other have to be added to the other view in order to make the views identical. The addition of missing objects is part of the "view completion" strategy used in this integration method. During integration, both views that take part in the integration process are altered until finally they are identical. This strategy is different from those that create a third "integrated" view during the conflict resolution process.

Many conflict cases require the combination of several elementary therapy procedures to correct a conflict. For instance, a case of construct mismatch paired with synonymy (Case 6), requires a name change and a construct change, therapies S10 and one of S1, S2, or S3. Appendix 2 presents the conflict cases and applicable therapy procedures. Case 6 is shown below for illustration.

CONSTRUCT MISMATCH AND SYNONYM

N1 <> N2; T1 <> T2; M1 = M2; C1 <> C2;

6.1 Entity is Relationship.
Solution: *S10 and S1.*
6.2 Entity Attribute is Entity-Relationship construct.
Solution: *S10 and S3.*

6.2.1. Attribute is Entity.
6.2.2. Attribute is Relationship.
6.3. Relationship Attribute is Entity.
Solution: *S10 and S2.*

4.4. The Impact of Heuristics

The main goal of this research is the development of a complete view integration method. The secondary goal is an adaptation of this method to operate with insufficient information.

The integration method in the form described so far does not take into account the source of its information requirements. For example, if the method has to know whether EMPLOYEE in view 1 and DEALER in view 2 are of the same object type (construct), the method expects this information to be available. The source of the information is of no concern. Among the four relevant dimensions for each object, name, construct, meaning, and context, name and construct are the ones most easily assessed. Does EMPLOYEE have the same name as DEALER? Obviously not. Also the object type is observable, because object types are explicitly stated in E-R models. The assessment of meaning identity, and therefore also context identity, is a much more difficult problem. The question is whether two view objects refer to the same real world object.

Recognition or interpretation of real world objects is a task beyond most computer systems and not a concern of this research. Nevertheless, recognition of meaning identity or difference is the most crucial recognition task, since the other dimensions

follow the meaning dimension. I.e., if two objects have the same meaning, their names will ultimately be the same, if they have different meaning, their names will ultimately be different.

The following alternatives exist to satisfy the meaning information requirement:

1. user interrogation;
2. advance meaning specification;
3. method "guesses".

The first alternative to satisfy the meaning information requirement is through user interrogation. Every time two objects are compared, the system could ask the user "are these two objects identical in meaning?". This form of operation demands a substantial amount of question answering by the user, especially since for any object O1 in view 1 at most one object O2 in view 2 with the same meaning is allowed to exist.

Advance meaning specification requires an ex-ante definition of the meaning of each object in a form that allows the method to compare it to other objects and to decide on identity or difference. This requirement results in two main problems. First, meaning descriptions may have to be very detailed to differentiate between objects that are quite similar, yet not

completely identical. Thus the up-front effort required is very high. Secondly, meaning definitions have to be formulated in such a form that there can be no misinterpretations. The terms used to define meaning have to be consistent over all object definitions. These two problems virtually rule out a prior complete definition of each object's meaning.

Method "guesses" require that the integration method has strong evidence on which it can base its guesses. "Guessing" implies that whenever the method compares two objects, it makes a decision whether to believe that the objects are identical or not. This is the way in which humans operate. When we say "I know", we mean that we believe, based on evidence for the fact and no or little evidence against the fact". If evidence is not available, the method is bound to make mistakes. Unfortunately, ample opportunity for mistakes exists, since the amount of positive information --any O1 is identical to at most one O2--is so much smaller than the amount of negative information. Hence, reliance on guesses is not a desirable alternative.

Apparently, none of the alternatives by itself provides a reasonable solution to the information requirement problem. The first alternative, interrogation, provides the information, yet at high cost to the user. The second alternative, up-front definition, does not necessarily provide all the

information and it requires a lot of user effort in addition to an unambiguous representation. The third alternative requires no user effort but does not guarantee that the information requirements are satisfied correctly. Consequently, the best strategy to satisfy the requirements, is to combine the good aspects of the discussed alternatives.

User interrogation is the only method that satisfies the information requirements, therefore it is the dominant approach (if the user says that in his world two objects are identical, they are identical, unless this fact conflicts with a previous statement). The other two alternative approaches can be used to overcome or at alleviate the weakness of direct user interrogation, because they can limit and prioritize the questions to be asked.

Most of the questions of the type "is object O1 identical to ..." will result in the answer "no" or the will demand the comparison to a vast number of other objects at once. If O1 is compared to all objects in O2 in one comparison, the user has to deal with a large amount of information which may make it difficult to answer correctly. Consequently, an improved method should reduce the number of objects O1 has to be compared to. If object identity is the goal, only such O2s should be compared to O1 which could potentially be identical to O1. In other words, a filter would be used to reduce the number of

objects in the comparison. Ex-ante meaning definitions of objects, if in unambiguous form, can be used in such a manner.

If the purpose of ex-ante meaning definitions in this approach is to allow an automatic assessment of difference, meaning definitions can become much shorter. For example, the meaning definition of each object could contain just one fact, its value being either "animate object", "inanimate object" to separate all E-R model objects describing living creatures from those describing things. If all database objects were correctly classified, the method could automatically decide that EMPLOYEE and DEPARTMENT are different, because the former one is a living object, the latter one not. A few general categories can be chosen which can allow sufficient specification and differentiation of meaning without the need for an excessive up-front definition effort. Ein-Dor (1987) discusses the use of such "common sense knowledge" in reasoning. Grounded on such a common sense knowledge based classification, the integration method could quickly eliminate those objects O2 that are not identical to object O1. The user would only have to decide among the remaining objects.

A further reduction in the number of objects involved in the comparison can be initiated through the use of other available information, in combination with the use of heuristics, as discussed previously. Instead of guessing which objects are

identical, the method could use any additional evidence to further reduce the number of objects under consideration. The following two views shall exemplify this approach which utilizes context information:

View 1: EMPLOYEE--Employed_by--DEPARTMENT

View 2: EMPLOYEE--Works_in--XYZ--Engaged_in--PROJECT

Suppose, it is already known that EMPLOYEE in view 1 and EMPLOYEE in view 2 are identical. Now, the next task would be to find out whether the relationship Employed_by is identical in meaning to any object in view 2. One reasonable assumption would be to expect that an object identical to Employed_by would also be a relationship in view 2. This does not have to be the case but is quite likely (hence, a heuristic). This simple assumption reduces the number of contenders in view 2 to the objects, Works_in and Engaged_in. Another reasonable assumption would be to expect that the object sought in view 2 is also associated with that view's EMPLOYEE entity. Again, this does not necessarily have to be the case, information could be missing in view 2, yet it is an assumption likely to be true. The second assumption leaves only Works_in as a potential candidate to have the same meaning as Employed_by. Consequently, instead of asking the user "is the relationship Employed_by identical in meaning to one of the following: Works_in, XYZ, Engaged_in, PROJECT?", it can more intelligently ask, "is the relationship Employed_by identical in meaning to

the relationship Works_in?", thus simplifying the decision task for the user.

Not only context and construct can be used to make assumptions about the identity of objects. Other available information, such as names can be used too. Figure 18 provides an overview of potential sources of evidence for meaning identity. The first aspect, meaning representation, has already been discussed.

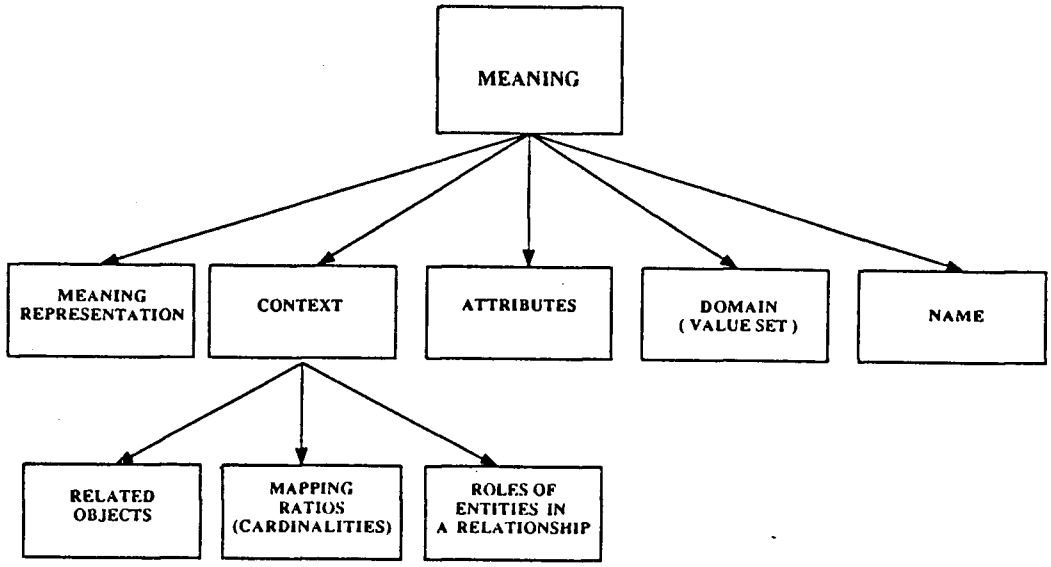


Figure 18: Sources of Evidence for Meaning Identity

The second aspect, context, is broken down into three observable facts: related objects, cardinalities, and roles of entities in a relationship. "Related objects" denotes the general definition of context. Cardinalities refers to the context of relationships. If two relationships do not only associate the same entities, but also with the same mapping ratios, the evidence for the relationships' identity is even stronger. When a view contains multiple relationships associating the same set of entities, a differentiation by cardinalities can be useful. The use of roles applies only when roles are defined. If names are given to the associating link between an entity and a relationship, then these role names can be used for comparison.

Third, attributes can serve as an indicator for identity. The problem is that attributes are objects in themselves and therefore subject to the same difficulties with respect to identity assessment. One aspect of attributes, however, is easily found out, their names. Thus, two objects may be speculated to be identical, if their attributes have identical names. As for all previous indicators, there has to be room for interpretation. The requirement should not be that all attributes have to be identical, yet at least some. Alternatively, the key attribute(s) could be the focus of

attention. Identical objects are likely to have identical key attributes.

Fourth, identical domains can be an indicator for identical meaning, if domains can be defined unambiguously. For attributes, domains are the value sets from which the attribute values are drawn, i.e. "Social Security Number". For other objects, an object's superset defines its domain. I.e., EMPLOYEE--Isa--PERSON specifies the domain of EMPLOYEE as being a person. If the other view contains also the PERSON entity, then the EMPLOYEE entity could exist only among its subsets.

Finally, the name of an object as an indicator for its meaning can be another relevant piece of evidence. Especially if name identity is not defined as strict identity of the character strings, but if it also allows for singular/plural differentiation, as in EMPLOYEE vs. EMPLOYEES. Both objects could be expected to be the same, even though their names are, strictly interpreted, different. For the analysis of relatedness of objects, this interpretation flexibility could be widened, allowing for comparison of objects that only differ in their names' prefixes. For example PART_TIME_EMPLOYEE, EMPLOYEE, and FULL_TIME_EMPLOYEE could be expected to be identical or at least related, since they all their names contain the root word employee.

It is unlikely, that for any given object all these aspects point into the same direction, that is, identify the same object. Often, it may not be known what the context of a particular object is, naming preferences will differ, and different tasks may require different object attributes. The approach to be taken is to use these indicators as a filter of variable density. At first, the filter should be tight, to suggest only the most likely candidate(s) for a meaning match, i.e., only the objects of the same type with same context and of the same meaning category. Should this filter be too wide still, i.e., for a database with many entities of the people category, partial overlap of attribute names, or identity of key attribute names can be used to restrict the number of objects. Upon failure, i.e., if none of the suggested objects resulted in a proper match, the technique could remove one or more of the earlier applied restrictions, i.e., look for all objects of the same meaning category, regardless of object type and context.

There exists no single best rule for the application of meaning indicators. The only indicator which is always applicable and correct in its prediction, should the information be available, is the meaning <u>category</u> indicator. By definition two objects cannot be identical in meaning unless their meanings belong to the same category of meaning. I.e., EMPLOYEE and DEPARTMENT

cannot have the same meaning because one is an animate object, the other one an inanimate object. Hence, this indicator is the only one that can eliminate objects with certainty. The other indicators can only suggest that an object may have different (or same) meaning.

Only empirical data generated under a variety of conditions can provide stronger evidence on which meaning indicators work better than others. For instance, if the same systems analyst produces all views (based on different users' information requirements), one may expect that object type may be a reasonable indicator (filter) for meaning identity; the underlying assumption being that a single database designer will be more consistent in what he models as a relationship vs. an entity or attribute than a multiplicity of designers. If all views specifications and designs are done by the same person (user designer), one should expect names to be used consistently throughout the views. Hence, names could provide a good basis to judge meaning identity.

4.5. Generalization Hierarchy for Database Objects

The previous section introduced the idea of ex-ante meaning definitions according to predefined meaning categories. Here, the concept of a generalization hierarchy shall be introduced to facilitate the categorization.

The difficulty in developing such a classification scheme is the fact that it has to be acceptable to all people involved in the database design process. In order to fulfill this goal, the generalization hierarchy should be:

1. complete;
2. consistent;
3. discriminative;
4. concise.

Criteria 1 and 2 are minimum criteria. First, a classification scheme that does not allow the user to classify all his objects in accordance with it is insufficient to capture that user's knowledge. Second, if the scheme induces the user to classify the same object under different categories, it violates the purpose of the scheme, namely to identify similarity or difference of object meanings.

Criteria 3 and 4 are based on Leibniz's Minimality Principle (Leibniz, 1956, pp. 198-199). This principle postulates that a representation is superior to another one, if it requires a shorter explanation to explain the same phenomena. Correspondingly, a generalization hierarchy that can differentiate among a larger number of object classes than another one with the same number of differentiation criteria is superior. What is undesirable is a classification scheme that is very fine-grained for a subset of object classes but very coarse for the remainder of object classes. Similar to an unbalanced binary tree, the too fine/too coarse generalization hierarchy would waste too many levels of specialization on too few phenomena.

Unfortunately, choice of the "right" generalization hierarchy will consequently depend on the knowledge domain and on the way in which the person who classifies objects differentiates among them. For example, a generalization hierarchy which contains only one class for all "people objects" will deal poorly with a database that stores only data for different people roles (i.e., employee, investor, saver, tax payer). Consequently, validation of the quality of a generalization hierarchy is possible only within the context of a particular knowledge domain and a specific person who classifies objects. Hence it is necessary to include the creation of such a generalization hierarchy in the requirements analysis effort.

The database designer has to develop a hierarchy which can represent the application domain and has the above mentioned desirable properties.

If no such specialized categorization hierarchy exists, a domain-independent categorization hierarchy could be used. The hierarchy created as part of this project, is rather flat, incorporating only few levels of specialization.

A flat generalization hierarchy has the obvious disadvantage of limited discriminative ability. However, object classifications are used to identify difference in meaning, not meaning identity. Object classification is only one of the identifiers used by the integration method, and the method will always interrogate the user, if in doubt. Since the focus is on difference in meaning, even a flat generalization hierarchy has reasonable discriminative ability, as the following example may illustrate.

Consider a generalization hierarchy that can differentiate among 20 classes, such as Person, Animal, Organization. Object EMPLOYEE is classified as a Person. The question to be answered is "is object XYZ different in meaning from object EMPLOYEE?". Without further knowledge about XYZ, XYZ has equal probabilities to belong into either class, and thus a .05 chance of belonging into the class Person. Thus there

exists a .05 chance for the classification mechanism to suggest that EMPLOYEE and XYZ are not different in meaning. In this situation (1 out of 20 cases), the user would have to be consulted, if not other indicators were able to answer the question. An increase of the number of classes to 40 would reduce the probability to .025, an increase to 200 classes would result in a .005 probability, requiring user interrogation only in 1 out of 200 cases. The reductions in probability have to be weighed against the classification effort which is an ex-ante investment.

A generalization hierarchy for the categorization of object classes shows similarities with the attempts to represent common sense knowledge in artificial intelligence. The classification hierarchy discussed here is, however, less ambitious, since the task, judging whether two objects are different in meaning, is simpler than the task presented in the artificial intelligence applications (i.e., Schank's and Rieger's restaurant scripts (1974) or Hayes' naive physics (1979)). Ein-Dor suggests concept clusters for common knowledge in the business environment (1987). His categories are:

1. exchange,
2. time,
3. location,
4. measurement,
5. media of exchange,
6. obligations and commitments,
7. types of businesses,
8. behaviors,
9. naive economics,

10. employment,
11. people who engage in business.

This classification clarifies the difference between a common knowledge representation and a generalization hierarchy. Ein-Dor's classes are not mutually exclusive. For example, the employment situation can be classified as group 10 as well as group 6. These classes represent areas in which a common sense computer program should have knowledge in.

The categorization that can be used in absence of any more domain oriented hierarchies, is structured as follows:

1. Objects
1.1. Living objects (even if now dead)
1.1.1. Plants (flora)
1.1.2. Animals (fauna)
1.1.3. Persons
1.1.3.1. Person (generic, not person roles)
1.1.3.2. Person roles
1.1.3.2.1. Person roles in person-person interaction (i.e., parent)
1.1.3.2.2. Person roles in person-thing association (i.e., car owner)
1.1.3.2.3. Person roles in person-person-thing interactions (i.e., manager)
1.2. Inanimate objects
1.2.1. Abstract objects
1.2.1.1. Abstract objects that are organized (have structure)
1.2.1.1.1. Hierarchies (i.e., a business company)
1.2.1.2.2. Markets (i.e., the real estate market)
1.2.1.1.1. Other Structures
1.2.1.2. Heaps, lumps and atomic abstract objects (i.e., a dream, a theory)
1.2.2. Concrete objects ("things")
2. Object characteristics (i.e., color, size)

According to this categorization scheme, each view object can have a meaning list containing up to 5 elements, such as [object,living,person,role,person-thing] for category 1.1.3.2.3.

Objects classified as belonging to different categories cannot be identical in meaning. If the meaning list for an object is incompletely specified, i.e., category 1.1.3. it may not be different from an object classified as 1.1.3.2.3. and therefore user interrogation may be necessary. Objects belonging to different categories but belonging to the same higher category may be related in meaning. More domain specific categorization schemes will have more and better fitting categories but will use the same reasoning mechanism to interpret the results of categorization.

4.6. Assessment of the Method

In an earlier chapter, the strengths and weaknesses of previous integration methods were assessed. The same evaluation criteria will now be used to highlight the capabilities and limitations of the method presented here.

Similar to previous semantic integration methods, the one introduced in this research requires designer interaction during the integration process. The designer has to be consulted to settle questions concerning identity or difference in meaning. However, the method employs heuristics to reduce the number of questions that must be asked.

View integration, as discussed here, covers a larger part of the integration problem than most other techniques. It performs conflict resolution, view merging and addition of inter-set relationships. Batini et al. (1983) cover additional aspects of the conceptual design process, including correctness and completeness tests for individual views before the integration process (pre-integration). These tests, however, are not an essential part of the integration process; rather, they are elements of the view creation task.

This research exceeds all preceding approaches in the number of conflict cases covered. Less important than the number of

cases, however, is the fact that the conflict list is exhaustive, based on all relevant object differentiation criteria.

Similar to other semantic methods, this one reduces the complexity of the integration task by focussing on high level objects entities and relationships. The method also separates the test for relatedness from the test for identity. Heuristics further reduce the task complexity. The question "is object O1 identical in meaning to one of the objects {O2}?" can be simplified through reduction of the size of the set {O2}. Heuristics are used to eliminate unlikely candidates from {O2}.

This research also investigated whether the integration problem could be described by an even smaller set of conflict categories than the 17 general cases identified in section 4.1. To simplify the description of conflicts, a graph notation was chosen which represents every object, whether entity, relationship, or attribute, as a node, and every association between objects (entity role, attribute association) as an edge. Based on this notation, view conflicts take the form of missing nodes or edges, or inconsistently labelled nodes (name mismatch). A mismatch between types of nodes, i.e. entity-attribute vs. entity-relationship construct, can be characterized as a graph contraction. A graph contraction is the removal of an edge which results in the merging of the two objects linked by the edge into one new object. I.e., an E-R

construct is merged into one new object, an entity attribute. Similarly, a relationship replaces a relationship-entity-relationship structure, when two edges are contracted in the latter one. Both types of contraction are depicted in Figure 19.

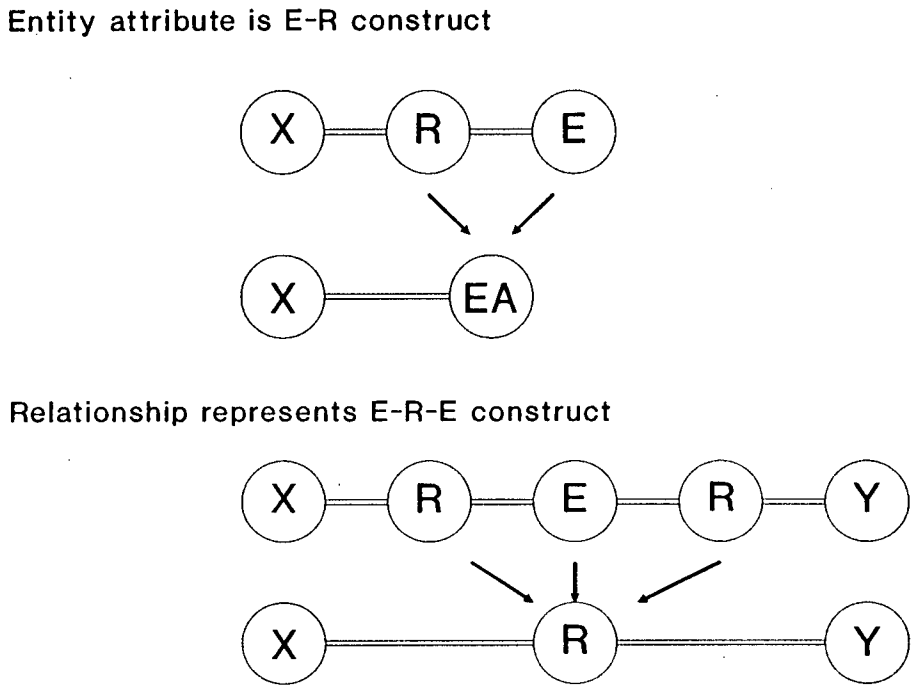


Figure 19: Construct Mismatch Shown as Graph Contraction

The examples illustrate that the graph notation is able to describe the construct mismatch conflict, in addition to the missing object conflict and the context mismatch conflict, based on only two criteria: missing nodes and missing edges. A missing object translates into a missing node, context mismatch translates into missing edges (plus potentially

missing nodes), and construct mismatch translates into missing edges and graph contraction. Since the notation can describe the same conflict phenomena as the E-R model using fewer mechanisms, it is a more powerful description tool.

The AVIS view integration program developed as part of this research employs the graph approach. In AVIS, views are described in the form of nodes and edges. Nodes represent objects, and edges, roles. Each object (node) is defined by the same set of properties: type (i.e., attribute), view, object identifier, object name, and object meaning (plus one more property not relevant for this explanation). Each role (edge) contains the identifiers of the two objects it is connecting. Both are explained in more detail in the subsequent chapter.

Even though the graph notation is more powerful as a description tool than the E-R model, integration cases have been discussed within this research using E-R concepts. The E-R model is widely used as a conceptual modelling language in database design, while the above graph notation is not. Thus, conflict cases and solutions described by means of the E-R model are more easily understood and thus presumably more useful to the database designer than ones based on a graph notation. The differences between the internal graph representation in AVIS and the external E-R representation require that AVIS frequently

translate between these two representation forms. Nevertheless the internal representation in the form of graphs is very useful because it allows the system to easily compare objects of different types along their relevant dimensions. For instance,the question "do object O1 and object O2 have identical meaning?" can be easily phrased in the graph notation, shown in Figure 20 in its Prolog equivalent. This simple example illustrates that the integration method can compare objects of any type in the same manner. I.e., T1 may be "attribute", while T2 is "entity" [1].

```
identical_meaning(O1,O2) :-
   object(T1,V1,O1,N1,M),
   object(T2,V2,O2,N2,M).
```

Figure 20: Identical Meaning Query in Prolog Graph Notation

An additional strength of the method discussed in this research is the use of meaningful data objects. The E-R model allows the description of objects that are meaningful to database users. The integration method further allows the representation of some data semantics.

---

[1] However, the example in the figure shows an over simplification of the meaning comparison problem. AVIS does not use Prolog's pattern matching mechanism in this simple form to assess meaning identity. Meaning comparison is described in more detail in the subsequent implementation chapter.

Unlike other semantic integration methods, this one includes an algorithm for the identity and for the relatedness tests, which explicitly specifies the steps of the procedure. For example, the identity test without heuristics contains a four-step procedure in which identity or difference of the four relevant object criteria is assessed. Due to the form in which meaning identity and relatedness questions are stated, namely as a 1:N comparison ("Is object O1 identical to one of {O2}?"), the computational effort grows linearly with the number of objects. The procedure terminates when the initially different views have become identical. To be identical, both views have to contain the same objects. Objects are identical if they are identical in all four relevant dimensions (meaning, context, construct, and name).

To judge the value of the method, the questions of correctness and completeness of the resulting views have to be addressed. (The working prototype only demonstrates the workability of the method for specific cases.) Based on the earlier description of the integration algorithm, it is known that the procedure always terminates if the initial views contain a finite number of objects. The procedure performs the integration task through an adjustment of both initially different views. When the procedure terminates, for each object in one view, an identical object exists in the other view. Hence, the completeness question depends on whether objects can be "lost"

during integration so that the final views do not contain all objects from the initial views. The correctness question concerns whether objects from the initial views may be misrepresented in the final view. Furthermore, it has to address whether the order in which views are integrated and/or the sequence in which objects within a view are considered have any impact on the outcome of the integration process.

In this integration method, objects cannot be lost. Every object represented in at least one initial view will also be represented in the global schema. This does not imply that each object will appear in its original form. The object meaning will be preserved, but the object representation in name, construct and context may change. A relationship may be relocated, a name may be changed, or an object's construct may be changed. After a construct change, an object will in most cases be represented through more than one new object, i.e., a relationship will become a relationship-entity-relationship group. The only exception is the change of a relationship attribute into an entity, where the construct change replaces one old object by one new object. Due to the direction of change in cases of construct mismatch, an old object is always replaced by at least one new object. Hence, objects cannot be lost during the integration process.

Although objects cannot be lost, the resulting view may still be incorrect, if objects are mis-represented or objects are added arbitrarily. An object is mis-represented if the knowledge represented in its post-integration form contradicts with the knowledge representation in the pre-integration form. This includes name changes that result in names which do not convey the meaning of the object, construct changes which compress the information content of an object, meaning changes which result in incorrect meaning descriptions, and context changes which connect objects to objects they should not be connected to. The integration method performs none of these invalid operations, nor does it add objects arbitrarily.

Objects are only added if this addition is suggested by one of the views, that is if at least one of the views contains an object that is not part of other views. Name changes occur only when synonyms or homonyms are detected. The choice of suitable names to overcome these conflicts is a task for the designer who uses the method. Construct changes never result in the loss of information, since the construct chosen is always the one which is able to convey the most information. Meaning changes are never made by the system (database designer). Meaning is specified by the users of the system and can only be changed by the users of the system. Context changes occur for three reasons. First, construct changes cause context changes, as depicted in Figure 10 in the conflict therapy

section. Second, an association of an entity to a relationship results in a context change (exemplified in Figure 12). Third, relationship relocation results in context change (shown in Figure 13). All of these changes make the representation of data object in one view compatible with that of another view. In the first two of these cases, an object O1 will only be connected to an object O2, if at least one view states that the two objects should be connected. If all views are correct prior to integration, this operation cannot result in incorrect context. Relationship relocation takes place only if during the integration process, the database designer identifies that the relationship is applicable to the superset object rather than to the subset object (Figure 13).

Finally, we must consider whether the same outcome, that is, the same global structure, will be achieved independent of the sequence in which views are integrated. In a two-view integration problem, sequence refers to the order in which objects compared. For example, is O1 from V1 compared to all objects from V2 first, followed by O7 from V1, or does O7 precede O1? In a multi-view integration problem, sequence also addresses the order in which views are compared. I.e., if three views, V1, V2, and V3 have to be integrated, will V1 be integrated first with V2 and the result of this integration be integrated with V3, or will the integration begin with V2 and V3?

In both the two-view and the multi-view integration problems, the following operations are performed: objects existing in all views become part of the global schema, objects existing in at least one view become part of the global schema, objects represented differently in different views are adjusted and become part of the global schema. In addition, inter-view relationships are added to the global schema. Objects that exist in all views will not be affected by the sequence of the integration process. They will appear in the same form in the global schema. Objects that originally did not exist in all views will also be added to the global schema, independent of the integration sequence. Inter-view set relationships are similarly missing objects, however missing in all views. They also will be added, independent of sequence. In fact, they are added after all tests for identity of objects are completed. The critical element for this assessment of the view integration procedure is the adjustment of views when conflicts are detected.

In the two-view situation, the sequence in which objects are compared may vary. Does this change affect the outcome of the integration? This question translates into two more basic questions, namely first, does the sequence in which objects are compared result in differences in the diagnosis of conflicts, and second, does a potentially different diagnosis result in a different global schema?

The conflict diagnosis procedure uses as its most important criterion the meaning dimension. Once objects with identical meaning are found, conflicts are detected based on differences in the remaining dimensions, name, construct, and context. For each object in each of the views, at most one object with identical meaning can exist in the other view. This is true, independent of the sequence in which object are compared. Furthermore, with the exception of name changes for homonyms, the remaining dimensions of an object are not changed before meaning identity with another object has been established. Therefore, for any two objects from different views, the object comparison will yield the same result, independent of the sequence of comparisons, unless the database designer using the method is inconsistent in renaming objects when homonyms are found.

One other potential source of error exists, but it is also in the domain of the database designer. The designer may find it difficult in certain situations to decide whether two objects are identical in meaning. Therefore, if both objects O1 and O2 from view V1 appear to the designer as if they could match the meaning of object O3 from V2, then the order of comparison may bias the designer to decide for O1 in one situation and for O2 in some other situation. This is a particular problem in cases of construct mismatch, where, for instance, an entity

attribute in one view corresponds to an entity-relationship construct in the other view (see Figure 11). In this example, the database designer has to decide whether the attribute Supplier corresponds to the entity Supplier or to the relationship Supply. But even though the designer may have some discretion in deciding which of the objects is the matching one (entity or relationship), the conflict will be resolved in exactly the same way. The attribute will be replaced by an E-R construct. The same is true for other forms of construct mismatch.

In summary, as long as the designer is consistent in his assessment of meaning identity of objects, the diagnosis will always be the same, independent of sequence. If the designer is inconsistent in his assessment of meaning identity, the procedure will still produce identical outcomes for cases of construct mismatch.

In the multi-view situation, invariance of the outcome (global schema) to changes in the order of view comparisons is the concern. Can objects end up in the global schema with different names, different constructs, or different contexts, based on the order in which views are processed. Again, this is not the case. The integration method prevents those variations for all but naming decisions which are in the designer's domain. For construct changes, there is only one direction of

change, to avoid loss of information. For example, if out of n views, n-1 represent an object as a relationship attribute and only one view represents it as an entity, the object will still become an entity in the global structure. In all cases, the most information rich object representation will be the one chosen for the global structure. Context changes are dealt with in a similar manner. For example, if a relationship R in view V1 has as its context the set of entities {E1}, in view V2 the set {E1, E3}, and in view V3 the context {E1, E2}, the global schema will show {E1, E2, E3} as R's context, independent of the sequence in which the views were integrated. The same is true for attributes. Entities and relationships in the global view have attribute sets which are the union of the attribute sets of the corresponding objects from the original views (except, of course, when an attribute is converted to another construct).

In conclusion, even in a multi-view situation, the method will produce the same global schema, independent of sequence, if the designer is consistent in his decisions on meaning identity.

5. IMPLEMENTATION - THE AVIS PROGRAM

5.1. Overview

An implementation of the view integration method is available in form of the AVIS (Automatic View Integration System) program. AVIS is written in Prolog.

The purpose of the program is not to show correctness of the conflict resolution method. Correctness of the method should be judged based on its underlying assumptions, the rules guiding view integration, and the conclusion drawn from them concerning the diagnosis and therapy procedure. The program can only serve as a testbed to show mistakes or omissions in details of the resolution procedure. Furthermore, it can show the feasibility of an automated view integration procedure. Appendix 3 contains the screen displays of a view integration session with AVIS to illustrate the operation of the system and its role as a testbed.

5.2. Function and Structure of the AVIS Program

To fulfill its purpose as a testbed and an indicator for feasibility, the program is an implementation of the diagnosis and therapy procedure outlined in earlier sections. The

program always operates on a set of two views which are to be integrated. Such a set of two views has to be loaded into the system at the outset of the integration session. The program proceeds by checking conflict hypotheses. For each hypothesis that is checked, one eligible object from view 1 is chosen and compared to all eligible objects from view 2. Hypothesis tests are carried out in the sequence established by the integration rules and heuristics. Depending on the outcome of a test, an appropriate therapy activity is performed, followed by another test. A therapy can be "do nothing" if objects do not have to be changed, or any of the other therapy actions discussed previously. The program terminates when both views have become identical. The program structure which achieves this function is depicted in Figure 21.

Following the typical architecture of knowledge-based systems, the program is designed in highly decoupled form. For instance, the sequence in which hypotheses are tested is not fixed (programmed), but determined by the sequence in which they occur on the OBJECT COMPARISON AGENDA (box 8 in the figure). Therefore, an "urgent" hypothesis test (typically performed during a therapy operation consisting of more than one therapy action) can pre-empt tests that would normally have occurred next. Another form of decoupling separates the step which recognizes that an object is missing (box 4), from the step that actually adds the object to the view (box 5).

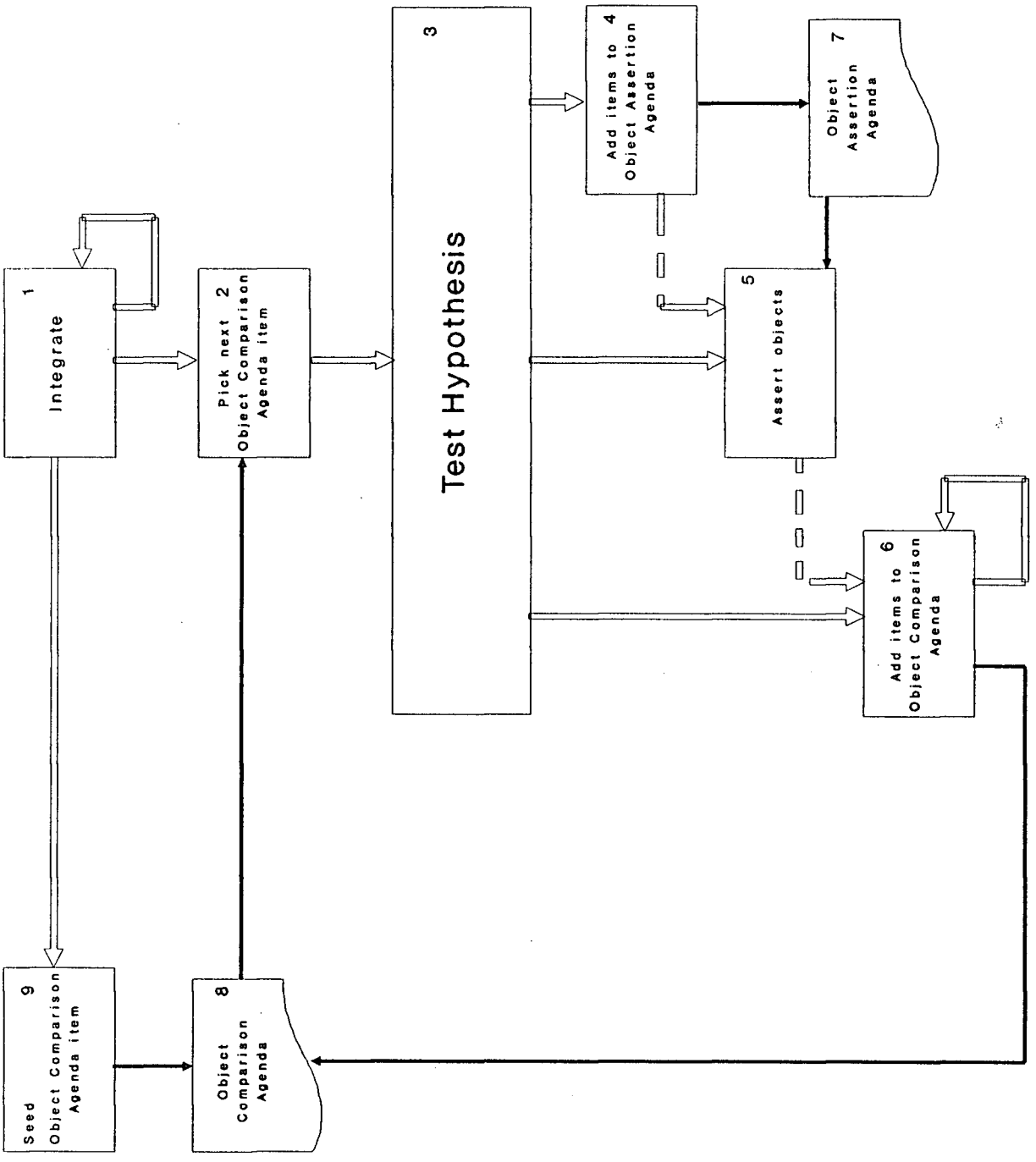


Figure 21: AVIS Program Structure

Hence, if the program realizes that an object is missing, it reports this finding in the OBJECT ASSERTION AGENDA (box 7). Then, in an independent step, the program will try to assert (add) the object. If this is not yet possible, due to the fact that some other pre-conditions are not fulfilled, the missing object will simply remain in the OBJECT ASSERTION AGENDA until those preconditions are satisfied.

Overall, the program operates as follows. It repeatedly calls the predicate INTEGRATE (box 1), which initiates object comparisons. Object comparisons are carried out as specified by the entries in the OBJECT COMPARISON AGENDA. Every such entry will consist of the (generic) hypothesis to be tested, for instance SIMILAR ENTITY (same meaning), and the objects involved in the test, i.e., SUPPLIER for view 1 and DEALER, BUYER, INVENTORY for view 2. The generic hypothesis together with the objects to be tested form a specific hypothesis. This hypothesis is then tested. The program will attempt to find an answer to the hypothesis first on its own, before asking the user.

To find an answer without user interaction, the program will first check whether results of previous tests can help in deciding the question. For example, if all entities in view 2 already had corresponding entities in view 1, the program could answer the question with "no", because each object can

have only one matching object in the other view. If previous tests cannot help in deciding, structural information may help. For example, a necessary condition for two relationships to be related is to have at least two common entities. If no two common entities exist, the program can assert that the relationships in question are not related. If structural knowledge cannot help, the program may be able to use any semantic knowledge it possesses concerning the application domain. Currently this option is not implemented in a form where the program is able to make such inferences (the information is only passively available). If the program cannot decide by itself whether a hypothesis is true or false, it will ask the user.

Following the hypothesis test, the program will place an entry into the OBJECT ASSERTION AGENDA, if objects have to be added as a consequence of the test. In a next step, objects are added to a view if all preconditions for their creation are fulfilled (box 5). Finally, and also based on the outcome of the hypothesis test, new specific hypotheses may be placed on the OBJECT COMPARISON AGENDA (boxes 6 and 8).

At points during the integration procedure, the OBJECT COMPARISON AGENDA may be empty, even though the integration has not been completed. Such a point occurs for instance at the outset of the integration process. To "boot-strap" itself in these

situations, the program will activate the SEED predicate (box 9), which places a first entry on the agenda. The integration process ultimately terminates, if the agenda is empty and no more seeds can be generated.

5.3. Knowledge Representation

5.3.1. Representation of views

To allow the operation on arbitrary views, the program stores views separate from the procedural knowledge. A set of two small views is shown in Figure 22.

```
object("entity",1,3,"dealer",["sells","supplies"],[])
object("entity",1,4,"branch",["alternate_location","subsidi-
ary"],[])
object("entity",2,1003,"dealer",["sells","supplies"],[])
object("entity",2,1004,"customer",["buys","pays","orders"],[])
object("entity",2,1005,"contract",["agreement"],[])
object("relationship",1,502,"supply",["delivery","goods_trans-
fer"],[])
object("relationship",2,1502,"dealer_contract",["dealer_cont-
ract"],[])
object("relationship",2,1503,"customer_contract",["customer_cont-
ract"],[])
object("attr[1] ",1,600,"contract",["identifier"],[])

role(502,3)
role(502,4)
role(1502,1003)
role(1502,1005)
role(1503,1004)
role(1503,1005)
role(600,3)
```

Figure 22: Representation of Views in AVIS

Each object is stored as an atom of the form object(Type,View#, Object#,Name,Meaninglist,Replacelist). Type is one of entity, relationship, or attr(ibute). View numbers are arbitrary, but

---

[1] "attr" is used instead of attribute because attribute is a restricted term in the programming language.

objects from the same view carry the same view number and objects from different views carry different view numbers. Object numbers are unique identifiers for objects within the view they belong to. The object name is the user defined name for the object. Meaninglist is a list of strings that give some indication of the nature of the real world object represented by the database object. Replacelist is a list of object numbers for objects that have been replaced by the object at hand. For example, if a relationship attribute becomes an entity, the new entity retrains a reference to the former relationship attribute through the number in the Replacelist.

One way to think of the meaning list is to view it as a list of thesaurus terms for the object name. Each of the terms in the list may describe some facets of the objects meaning, through slightly different labelling of the object. The meaning list can also be used to identify the categories an object belongs to in a generalization hierarchy. This list of categories does not have to be a central pool of base sets from which all object sets have to be defined, it may be simply a set of category terms which capture the language terms used in the organization under study. In either form, the meaning list helps to simplify the identification of dissimilar (or even of identical) objects.

The context of an object is stored by means of role(Object#, Object#) atoms. The first object number indicates the object, the second one the object it is associated with. By definition, only attributes and relationships have a non-empty context. Thus, roles exist only for these two object types.

5.3.2. Representation of View Integration Knowledge

The knowledge contained in the program consists mostly of hypotheses, rules for the selection of objects for subsequent tests, rules for the elimination of irrelevant tests or irrelevant test objects, and rules for the therapy of conflict cases. Hypothesis atoms serve mainly to control the sequence of the integration procedure. Selected hypotheses are depicted in Figure 23.

```
hypothesis(3,["n","n"],[4,1],[12,701,24,5],"Similar Entity","same_meaning","different_meaning")

hypothesis(4,["o","o"],[],[],"Synonym","synonym","same")

hypothesis(12,["o","o"],[],[],"Homonyms","homonyms","not_homonyms")

hypothesis(701,["n","n"],[8],[24],"Entity is Relationship Attribute","relationship_attribute","not_relationship_attribute")
```

Figure 23: AVIS Hypotheses

Hypothesis 3 formulates the test for "Similar Entity". This test investigates whether for an entity in view 1 there exists an entity in view 2 with the same meaning but possibly with a different name. The lists of integers which are part of the hypothesis atom (i.e., [4,1]) indicate subsequent activities depending on the outcome of the test. For instance, if a similar entity is found, the next hypothesis to be tested is hypothesis 4, which tests whether both entities have same names. Thereafter, hypothesis 1 would follow. If the test

result were negative, a number of other hypotheses would be invoked, i.e., 12, 701, 24, and 5. Each hypothesis shows also which knowledge will be added to the knowledge base as a consequence of the test outcome. For instance, if hypothesis 3 becomes true, the involved objects will be memorized as having same meaning. If the test outcome is negative, they will be stored as having different meaning.

Rules that select objects for subsequent tests are "make agenda" rules. One example is shown in Figure 24.

```
m_a(3,_,[O1],[O2],H):-
  H = 1,!,
  find_r(O1,Rl1),
  find_r(O2,Rl2),
  filter(H,Rl2,Rl2n),
  m_a(0,b,Rl1,Rl2n,H),!.
```

Figure 24: AVIS "make agenda" Rule

The "make agenda" rule shown in the figure prepares a new hypothesis test, after the test for similar entity succeeded. Once two identical entities have been found, AVIS searches locally, in the vicinity of these entities, to find identical relationships. The rule finds all relationships entity O1 is associated with, as well as all relationships O2 is associated with. It then filters out relationships that have been previously investigated, and formulates a test in which all relationships Rl1 will be compared to all relationships Rl2n,

to find matching pairs. If R11 contains more than one relationship, the agenda item will later be decomposed into as many items as there are elements in list R11. This is necessary, since all tests are carried out in a 1:n mode, where one object of view 1 is compared to n objects of view 2.

Rules to filter out irrelevant tests or irrelevant test objects are exemplified in Figure 25.

```
/* the attribute O1 is a key */
test_hypo([O1],O12,H,_,O12):-
  H = 14,
  is_key([O1],[O1]),
  make_agenda(H,t,[O1],O12,HN),
  do_ao(H,O1,0,'n'),!.
```

Figure 25: Filtering Rule in AVIS

The rule depicted in Figure 25 refers to the test of hypothesis 14. Hypothesis 14 states the possibility that an entity attribute may correspond to an entity-relationship construct. The test_hypo rule shown here states that if the entity attribute O1 is a key (identifier) attribute of the entity it belongs to, then it cannot correspond to the entity-relationship construct O12. Entity attributes can only correspond to entity-relationship constructs if they are interconnection attributes, i.e. the Supplier attribute of a PART entity. If the attribute is a singular identifier (not part of a compound key), it refers to the object itself, i.e. Part# refers to PART itself. Such objects can be excluded from the test. By

using filtering rules, the AVIS program can reduce information requests from the user.

Rules for the therapy of conflict cases typically become rules to create new objects. Figure 26 illustrates such an "assert object" rule.

```
asso(H,O1,O2,'y',New):-
  H = 14,
  object(relationship,_,O2,_,_,_),
  find_e(O2,E12),
  object(attr,V1,O1,_,_,_),
  role(O1,E1),
  fct(same,E1,E2),
  member(E2,E12,E1r),
  single(same_meaning,E1r,E1s),
  dup(H,E1s,V1,E1s1),
  dup(H,[O2],V1,O1n),
  append(E1s1,O1n,New),
  retract(object(attr,V1,O1,_,_,_)),
  retract(role(O1,E1)),!.
```

Figure 26: AVIS Object Assertion Rule

The figure shows one of the rules dealing with the situation where an entity attribute in one view corresponds to an entity-relationship construct in the other view. This rule replaces the attribute O1 with a relationship O1n, by simply duplicating the relationship O2 from view 2 and subsequently eliminating the attribute from view 1. Furthermore, from all entities (E12) in view 2 that are associated by the relationship O2, those that have no corresponding objects in view 1 are identified (E1s) and duplicated in view 1.

5.4. The Impact of Domain Knowledge

One of the biggest problems for knowledge based systems is the requirement to contain knowledge about a wide variety of problem domains. "Deep" knowledge is much easier implemented than "wide" knowledge. This is similarly true for the view integration program which already has to contain deep knowledge on diagnosis and therapy. This weakness limits the necessary ability to assess identity of object meanings. How can a program judge that two objects are identical in meaning if it contains no domain knowledge?

If the "true" meaning of an object cannot be assessed, then at least a number of indicators exist to help in the assessment of true meaning (see Figure 18, previous chapter). Obviously, each object could carry with it a meaning representation, a list of symbols describing the meaning of the objects. Meaning comparison would then involve the comparison of such lists. Problems could arise from homonyms and synonyms in these lists. A second indicator could be object context. If two objects are similar their immediate neighbors are likely to be similar too. Thus, the finding of similar neighbors would provide some evidence for the assumption that two objects are similar. Other forms of context comparison involve the analysis of relationship cardinalities and, if defined, roles of entities

in a relationship. Similar roles and similar cardinalities are evidence for object similarity. Third, similar attributes (or at least similar attribute names or similar key attributes) can be another indicator for similarity. Fourth, if value sets have been defined, these can be compared. Finally, the name of an object itself can be an indicator for meaning similarity.

Most of the above mentioned indicators are plagued by the problem of ambiguous representation. If names of objects, due to homonymy and synonymy problems, are not a reliable indicator for similarity, the same will be true for other indicators such as attribute names or meaning lists. The use of context may be viewed merely as a recursive restatement of the problem. For example, to know whether entities E1 and E2 are identical, one has to know whether their context R1 and R2 is identical[1]. To find out whether R1 and R2 are identical one has to find out whether the context of R1 and R2 is identical, and so on. Nevertheless, comparisons are possible. For instance, partial overlap of meaning representations can be indicated, or partial context similarity, can be indicated. The AVIS program operates in this manner, however in passive form. The program never decides whether two objects are identical. Yet the user can ask the program for the values of similarity indicators. So

[1] Note that the AVIS program recognizes context also for entities in order to make use of local search for identical objects.

far, only the indicators meaning representation (comparing meaning lists), context (comparing immediate neighbors) and name are implemented. Figure 27 shows the systems response to a user inquiry on the value of the meaning indicators.

```
                                    A V I S
Testing for hypothesis:  SIMILAR ENTITY,
involving the entity DEALER (3) and one of the following  objects:

                                  Meaning Match
Match between entity DEALER (3)  ["sells","supplies"]
and objects below:
ID    NAME     Match of: NAME----MEANING-----CONTEXT
1003  dealer              y         y         unknown
1004  customer            n         n         unknown
1005  contract            n         n         unknown

                          Response
Press <spacebar> to continue
```

Figure 27: AVIS Meaning Identity Indicators

A more advanced form of meaning indicators, is based on the meaning representation (meaning categorization) feature. While currently meaning lists for objects have no form restrictions, therefore allowing the use of any symbol to define the meaning of an object; future meaning lists will be more restricted in the choice of terms. Terms will have to be elements of a categorization hierarchy and will be therefore unambiguous.

6. SUMMARY AND EXTENSIONS

The main contribution of this research is the development of a complete view integration procedure. The research went beyond the problem of inter-view constraint representation (relatedness of objects). It systematically categorized inter-view conflicts into conflict types, based on an analysis of the sources of conflicts. The source of all conflicts is mismatches between the meaning dimension on one hand and all other relevant object dimensions, name, construct, and context, on the other hand. Whenever two objects are identical in meaning, they also have to be identical in their other dimensions. If not, a conflict arises. Similarly, if two objects have different meanings they also have to differ in the name dimension to be conflict-free. The method presented in this research can diagnose all possible combinations of mismatches and has therapy rules for all of them.

In addition to rules for recognition and resolution of conflicts, an algorithmic view integration procedure was described. It specifies the sequence of tests for object identity and object relatedness. At the termination of this procedure, two initially different views become identical and represent all relevant inter-view constraints. Thus, either of the views has become a global schema containing the two original views. The integration procedure developed here begins with a test for object identity. At the end of this step, both views contain

the same objects. The subsequent test for relatedness determines all inter-view constraints for all originally unique objects (existing in only one view). The test for relatedness may result in the addition of entities to represent superset and subset objects and in the addition of Isa relationships.

Furthermore, the research provided heuristics to simplify the integration problem for the user. Heuristics were developed to ease the user's task of identifying object pairs with identical meaning. Assumptions such as "(in absence of information to the contrary,) two objects with identical meaning will have identical constructs", reduce the number of objects among which the user has to look for a matching object. In case of information to the contrary, i.e., if no pair of objects with same meaning were found, the heuristic would fail and would require a more painstaking search for a match. The research exemplified how the introduction of heuristics alters the integration procedure.

The method was designed for use as a view integration tool, through implementation as a knowledge based system (i.e., the AVIS system). Implementation in the form of a computer program assures adherence to the sequence of conflict analysis and resolution steps. It also eases as much as possible the designer's task. Nevertheless, the conflict recognition and resolution rules which form the core of the research are valid

independent of any implementation. The rules have been developed based on rules of modelling, based on the E-R model and based on database design principles, rather than through tracing of database design expert behavior.

Future extensions to the research will focus on at least two areas. First, more heuristics will be developed. This will not only simplify the user's task further, it will also shed more light on the question of how we can assess when two objects are identical in meaning. The assessment of meaning identity is the most difficult part of the integration process. Currently, the integration method does not decide on the identity of two objects without user consultation. It would be desirable to have the method decide, at least in some cases, whether two objects have the same meaning. One possible approach to extend the method in this direction is the development of categorization hierarchies for particular application areas. In this research, a very coarse categorization hierarchy has been introduced, one which facilitates deciding whether two objects have different meanings. More elaborate, as well as more domain specific hierarchies would allow a sharper distinction between concepts and thus allow for better judgment on identity or difference in meaning. This measure would require that users be very precise and explicit in their choice of names for entities, relationships, and attributes in the pre-integration stage. Hence, use of a

categorization hierarchy may be one good source of evidence, but may not be sufficient. Ultimately a procedure will have to use more sources of evidence and will have to be tolerant of user specification errors, in order to make judgments on meaning identity that are as good as human judgments.

A second area of extension to focus on is the detection of errors in user views. The integration method in its current form assumes that views are complete (all relevant objects included), consistent (no conflicting knowledge), and minimal (each object only represented once)[1]. If views are incorrect, inconsistent or not minimal, the global schema will be incorrect, inconsistent or not minimal. For example, if one view stated (incorrectly) that "all EMPLOYEEs are FULLTIME_EMPLOYEEs", while another view stated (correctly) that "every FULLTIME_EMPLOYEE is an EMPLOYEE", the method would represent both constraints in the global schema (inconsistency), not recognizing that the only logically correct interpretation of these two statements would require EMPLOYEE and FULLTIME_EMPLOYEE to be identical. Mistakes like this one could be detected and corrected during the integration process. To permit recognition

---

[1] The constraints on input views may seem rather stringent. However, we can expect views to be in consistent and minimal form, if they have been created with a view creation system such as Storey's (1988). Completeness has to be assumed, unless evidence to the contrary exists.

All previously discussed integration approaches make similar demands on the inputs to their integration methods.

of such errors, a set of error scenarios and correction rules would have to be developed.

Another possible extension that goes substantially beyond the scope of this research is the translation of the findings for database integration to knowledge base integration. While databases contain facts, knowledge bases contain facts and rules and are therefore much more difficult to integrate. Nevertheless, with the increase in the development of knowledge based systems and corresponding efforts to improve the knowledge acquisition effort such a project may become a fruitful endeavour for the future.

7. REFERENCES

Al-Fedaghi, S. and P. Scheuermann. Mapping Considerations in the Design of Schemas for the Relational Model. *IEEE Trans. Software Engineering*, SE-7, No. 1, 1981.

Armstrong, W.W. Dependency Structures of Database Relationships. *Proc. 1974 IFIP Congress*, Amsterdam: North Holland, pp. 580-583.

Atzini, P., C. Batini, M. Lenzerini, and F. Villanelli. INCOD: System for Conceptual Design of Data and Transactions in the Entity-Relationship Model. *Proceedings of the Second Int'l Conference on the Entity-Relationship Approach*, Washington, D.C., October 1981, pp. 379-414.

Bachman, Charles W. and Manilal Daya. The Role Concept in Data Models. *VLDB 77*, pp. 464-476.

Barr A. and E. Feigenbaum. *The Handbook of Artifical Intelligence*. London: Pitman, 1981.

Batini, C., M. Lenzerini, S.B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, Vol. 18, No. 4, 1986, pp. 323-364.

Batini, C., V. De Antonellis, A. Di Leva. Database Design Activities within the DATAID Project. *Quarterly Bulletin of the IEEE Computer Society Technical Committee on Database Engineering*, Vol. 7, No. 4, 1984, pp. 16-21. (1984a)

Batini, C., B. Demo, A. Di Leva. A Methodology for Conceptual Design of Office Databases. *Information Systems*, Vol. 9, No. 4, 1984. (1984b)

Batini, C., M. Talamo, and R. Tamassia. Computer Aided Layout of Entity Relationship Diagrams. *Journal of Software and Systems*, 1984. (1984c)

Batini, C., M. Lenzerini. A Methodology for Data Schema Integration in the Entity Relationship Model. *IEEE Transactions on Software Engineering*, Vol. 10, No. 6, 1984, pp. 650-663.

Batini, C., M. Lenzerini, M. Moscarini. Views Integration. In: *Methodology and Tools for Data Base Design* by S. Ceri (ed.). Amsterdam: North-Holland, 1983.

Batini, C. and M. Lenzerini. A Conceptual Foundation for View Integration. *Proceedings of IFIP Working Conference*, Budapest, Hungary, May 1983.

Batini, C., M. Lenzerini, and G. Santucci. Computer-Aided Methodology for Conceptual Database Design. *Information Systems*, Volume 7, No. 3, 1982, pp. 265-280.

Beeri, C. and P.A. Bernstein. Computational Problems Related to the Design of Third Normal Form Schemas. *ACM TODS*, Vol. 4, No. 1, 1979, pp. 30-59.

Bernstein, P. Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Transactions on Database Systems*, Volume 1, No. 4, December 1976, pp. 277-298.

Bernstein, Philip A., J.R. Swenson, and D.C. Tsichritzis. A Unified Approach to Functional Dependencies and Relations. *Proc. ACM 1975 SIGMOD Conf.*, San Jose, California, pp. 237-245.

Biskup, Joachim and Bernhard Convent. A Formal View Integration Method. *Int'l ACM SIGMOD Conf. 1986*, pp. 398-407.

Biskup, Joachim and Bernhard Convent. A Formal View Integration Method. *Forschungsbericht 208*, Universität Dortmund, 1985.

Brodie, Michael. On the Development of Data Models. In *On Conceptual Modelling* by Michael Brodie, John Mylopoulos, Joachim Schmidt (eds.). New York: Springer, 1984.

Brown, Robert. *Logical Database Design Techniques*. Mountain View, CA: The Database Design Group, 1982.

Casanova, Marco. Theory of Data Dependencies over Relational Expressions. *Proc. ACM SIGACT/SIGMOD Symp. on DB Systems*, 1982, pp. 189-198.

Casanova, Marco and Ronald Fagin. Inclusion Dependencies and their Interaction with Functional Dependencies. *Proc. ACM SIGACT/SIGMOD Symp. on DB Systems*, 1982, pp. 171-176.

Casanova, M. and V. Vidal. A Sound Approach to View Integration. *Proceedings of the ACM Conference on Principles of Database Systems*, March 1983, pp. 36-47.

Ceri, S. and G. Gottlob. Normalization of Relations and Prolog. *Communications of the ACM*, Vol. 29, No. 1, 1986, pp. 524-544.

Chen, Peter. The Entity-Relationship Model: Towards a Unified View of Data. *ACM TODS*, Volume 1, No. 1, 1976, pp. 9-36.

Curtice, Robert M. and Paul E. Jones, Jr. *Logical Database Design*. New York: an Nostrand Reinhold Co., 1982.

Date, Chris. *An Introduction to Database Systems*. Reading: Addison-Wesley, 1981.

DeMarco, Tom. *Structured Analysis* and Systems Specification. Englewood Cliffs: Prentice-Hall, 1979.

Dyba, E. Principles of Data Element Identification. *Auerbach Data Base Management Services*, Portfolio No. 23-01-03, 1977.

Ein-Dor, Phillip. Commonsense Business Knowledge Representation A Research Proposal. Working Paper, Tel-Aviv University, February, 1987.

Elmasri, R., J. Larson and S. Navathe. Schema Integration Algorithms for Federated Databases and Logical Database Design. Technical Report, Honeywell Corporate Research Center, 1987.

Elmasri, Ramez and Sham Navathe. Object Integration in Logical Database Design. *IEEE International Conference on Data Engineering*, Los Angeles, 1984, pp.426-433.

Elmasri, Ramez, A. Hevner, and J. Weeldreyer. The Category Concept; An Extension to the Entity-Relationship Model. *Data and Knowledge Engineering*, Volume 1, No. 1, June 1985, pp. 75-116.

Elmasri, Ramez, James A. Larson, Sham Navathe, and T. Sashidar. Tools for View Integration. *Quarterly Bulletin of the IEEE Computer Society Technical Committee on Database Engineering*, Vol. 7, No. 4. 1984.

Elmasri, Ramez and G. Wiederhold. Properties of Relationships and their Representations. *Proceedings of the National Computer Conference*, AFIPS, Volume 49, 1980, pp. 319-326.

Fagin, R. The Decomposition versus the Synthetic Approach to Relational Database Design. *Proceedings of the 3rd VLDB*, 1977, pp. 441-446.

Goldstein, Robert C. and Veda Storey. Unravelling Is-A Networks in Database Design. Working Paper, University of British Columbia, November, 1988.

Hayes, Patrick. The Naive Physics Manifesto. In *Expert Systems in the Micro Electronic Age* by Donald Michie (ed.). Edinburgh: Edinburgh University Press, 1979, pp. 242-270.

Hayes-Roth, Frederick, Donald Waterman, Douglas Lenat. *Building Expert Systems*. Reading: Addison-Wesley, 1983.

Housel, Barron C., Vance E. Waddle, and S. Bing Yao. The

Functional Model for Logical Database Design. *Proceedings of the 5th VLDB*, 1979, pp. 194-203.

Hubbard, G. and N. Raver. Automating Logical File Design. *Proceedings 1st VLDB*, 1975, pp.227-253.

Leibniz, Gottfried Wilhelm. *Philosophical Letters and Papers*, Vol. 1 (english translation). Chicago: The University of Chicago Press, 1956.

Mannino, M. and W. Effelsberg. Matching Techniques in Global Schema Design. *Proceedings IEEE COMPDEC*, Los Angeles, 1984, pp. 418-425.

Martin, James. *Managing the Database Environment*. Englewood Cliffs: Prentice Hall, 1983.

McFadden, Fred and Jeffrey Hoffer. *Data Base Management*. Menlo Park: Benjamin Cummings, 1988.

Minsky, Marvin. A Framework for Representing Knowledge. In *The Psychology of Computer Vision* by P. Winston (Ed.). New York: McGraw-Hill, 1975.

Mylopoulos, J. and H. Levesque. An Overview of Knowledge Representation. In *On Conceptual Modelling* by Brodie, Mylopoulos and Schmidt (Eds.). New York: Springer, 1984, pp. 3-17.

Navathe, Shamkant, Ramez Elmasri, James Larson. Integrating User Views in Database Design. *IEEE Computer*, 1986, pp. 50-62.

Navathe S, S. Gadgil. A Methodology for View Integration in Logical Database Design, in *Proc. ACM SIGMOD*, Austin, 1978.

Navathe, Shamkant, and Mario Schkolnick. View Representation in Logical Database Design. *Proceedings Int'l ACM SIGMOD* Conference, 1978, pp. 144-156.

New Orleans Database Design Workshop Report (Summary), VLDB, Rio(1979).

Nilsson, Nils. *Principles of Artificial Intelligence*. Palo Alto: Tioga Press, 1980.

Raver, N. and G.U. Hubbard. Automated Logical Database Design Methodology and Techniques. *IBM Systems Journal*, Vol. 16, No. 3, 1977.

Robinson, J. A Machine-oriented Logic Based on the Resolution Principle. *JACM*, Volume 12, No. 1, 1965, pp. 23-41.

Russell, Bertrand. *A History of Western Philosophy*. London: George Allen & Unwin, 1946.

Schank, Roger and Charles Rieger. Inference and the Computer Understanding of Natural Language. *Artificial Intelligence*, Volume 5, No. 4, 1974, pp. 373-412.

Sheppard, D. Principles of Data Structure Design. *Auerbach Data Base Management Series*, Portfolio No. 23-01-04, 1977.

Shipman,D. The Functional Data Model and Data Language DAPLEX. *ACM TODS*, Vol. 6, No. 1, March 1980, pp. 140-173.

Simon, Herbert and A. Ando. Aggregation of Variables in Dynamic Systems. In *Essays on the Structure of Social Science Models* by Ando, Fisher, and Simon. Cambridge: MIT Press, 1963.

Storey, Veda. *View Creation: An Expert System for Database Design*. Washington: ICIT Press, 1988.

Teory, T.J. and J.P. Fry. *Design of Database Structures*. Englewood Cliffs: Prentice Hall, 1982.

Ullman, Jeffrey. *Principles of Database Systems*. Stanford: Computer Science Press, 1980.

Vessey, Iris and Ron Weber. Structure Tools and Conditional Logic: An Empirical Investigation. *Communications of the ACM*, Vol. 29, No. 1, January 1986, pp. 48-57.

Vetter, M. Database Design by Implied Data Synthesis. *VLDB 77*, pp. 428-440.

Waterman, Donald A. *A Guide to Expert Systems*. Reading: Addison-Wesley, 1986.

Weber, Ron. Data Models Research in Accounting: An Evaluation of Wholesale Distribution Software. *The Accounting Review*, Vol. 61, No. 3, July 1986, pp. 498-518.

Yao, S. Bing, Vance E. Waddle, Barron C. Housel. View Modeling and Integration Using the Functional Data Model, *IEEE Transactions on Software Engineering*, Volume SE-8, November 1982, pp. 544-553.

Yao, S. Bing, Vance E. Waddle, Barron C. Housel. An Interactive System for Database Design and Integration. In *Principles of Database Design*, Vol. 1, S. Bing Yao (ed.), Englewood Cliffs, N.J.: Prentice Hall, 1985.

APPENDIX

Appendix 1: Conflict Cases

1. IDENTICAL OBJECTS
N1 = N2; T1 = T2; M1 = M2; C1 = C2;

Solution: do nothing.

1.1. Entity is Entity.
1.2. Relationship is Relationship.
1.3. Attribute is Attribute.

2. IDENTICAL OBJECTS WITH DIFFERENT CONTEXT
N1 = N2; T1 = T2; M1 = M2; C1 <> C2;

2.1. Relationship is Relationship of different degree or associating different entities.
Solution: tie not yet associated entities to relationship(s). If entities cannot be found, test for construct mismatch (5.2.1. or 6.2.1) and missing entity (17.1.).
2.2. Attribute is Attribute of a different entity or relationship (both are possession attributes).
Solution: convert both attributes into E-R constructs or entities, similar to 6.2. or 6.3.

3. TRUE SYNONYMS (SAME OBJECT TYPE)
N1 <> N2; T1 = T2; M1 = M2; C1 = C2;

Solution: rename at least one object so that N1 = N2.

3.1. Entity/Entity.
3.2. Relationship/Relationship.
3.3. Attribute/Attribute.

4. TRUE SYNONYMS WITH DIFFERENT CONTEXT
N1 <> N2; T1 = T2; M1 = M2; C1 <> C2;

Solution: rename and make contexts identical (combine solutions 3. and 2.).

4.1. Relationship/Relationship.
4.2. Attribute/Attribute.

5. CONSTRUCT MISMATCH

N1 = N2; T1 <> T2; M1 = M2; C1 <> C2;

5.1. Entity is Relationship.
Solution: convert the relationship into an entity. Create new relationships to associate the new entity with the entities it associated as a relationship.

5.2. Entity Attribute is Entity-Relationship construct.
Solution: convert the attribute into an E-R construct (entity and relationship).

5.2.1. Attribute is Entity.

5.2.2. Attribute is Relationship.

5.3. Relationship Attribute is Entity.
Solution: convert the attribute into an entity.

6. CONSTRUCT MISMATCH AND SYNONYM

N1 <> 2; T1 <> T2; M1 = M2; C1 <> C2;

Solution: rename objects to make names identical and deal with construct mismatches as in 5.

6.1. Entity is Relationship.

6.2. Entity Attribute is Entity-Relationship construct.

6.2.1. Attribute is Entity.

6.2.2. Attribute is Relationship.

6.3. Relationship Attribute is Entity.

7. DIFFERENT AND UNRELATED OBJECTS

N1 <> N2; T1 = T2; M1 <> M2; not(related(M1,M2)); C1 = C2 or C1 <> C2;

7.1. Objects are different, unrelated and have no common role.
Solution: do nothing.

7.1.1. Entity/Entity.

7.1.2. Relationship/Relationship.

7.1.3. Attribute/Attribute.

7.2. Object 1 and Object 2 in same role (W-relationship).
Solution: create a common role object, special role objects, and Isa relationships between the role objects and objects O1 and O2. If objects are not entities, transform them into entities first.

7.2.1. Entity/Entity.

7.2.3. Relationship/Relationship.

7.2.3. Attribute/Attribute.

8. TRUE HOMONYM

N1 = N2; T1 = T2; M1 <> M2; C1 = C2 or C1 <> C2;

Solution: rename at least one object, giving it a name that is not assigned to any other object in the view. Thereafter treat common role occurrences similar to 7.

8.1. Objects are different, unrelated and have no common role.
- 8.1.1. Entity/Entity.
- 8.1.2. Relationship/Relationship.
- 8.1.3. Attribute/Attribute.

8.2. Object 1 and Object 2 in same role (W-relationship).
- 8.2.1. Entity/Entity.
- 8.2.2. Relationship/Relationship.
- 8.2.3. Attribute/Attribute.

9. DIFFERENT OBJECTS WITH DIFFERENT CONSTRUCTS

N1 <> N2; T1 <> T2; M1 <> M2; C1 <> C2;

9.1. Objects are different, unrelated and have no common role.

Solution: do nothing.
- 9.1.1. Entity/Relationship.
- 9.1.2. Relationship/Attribute.
- 9.1.3. Entity/Attribute.

9.2. Object 1 and Object 2 in same role (W-relationship).

Solution: create a common role object, special role objects, and Isa relationships between the role objects and objects O1 and O2. If objects are not entities, transform them into entities first.
- 9.2.1. Entity/Relationship.
- 9.2.2. Relationship/Attribute.
- 9.2.3. Entity/Attribute.

10. DIFFERENT OBJECTS WITH DIFFERENT CONSTRUCTS, BUT HOMONYMS

N1 = N2; T1 <> T2; M1 <> M2; C1 <> C2;

Solution: treat objects like true homonyms. Change the name of at least one object to make it different from all other object names in the same view. Treat common role objects as in 9.

10.1. Objects are different, unrelated and have no common role.
  10.1.1. Entity/Relationship
  10.1.2. Relationship/Attribute
  10.1.3. Entity/Attribute
10.2. Object 1 and Object 2 in same role (W-relationship).
  10.2.1. Entity/Relationship.
  10.2.2. Relationship/Attribute.
  10.2.3. Entity/Attribute.

11. DIFFERENT BUT RELATED OBJECTS
N1 <> N2; T1 = T2; M1 <> M2; related(M1,M2); C1 = C2;

11.1. One object contains the other (Object 1 contains Object 2 or vice versa).
Solution: create an Isa relationship between the two objects.
  11.1.1. Entity/Entity.
  11.1.2. Relationship/Relationship.
  11.1.3. Attribute/Attribute.
Solution: before creating an Isa relationship, convert attributes into entities (for relationship attributes) or into E-R constructs (for entity attributes).
11.2. Object 1 and Object 2 have a common superset (but do not overlap).
Solution: create a superset object and Isa relationships from objects O1 and O2 to the superset object.
  11.2.1. Entity/Entity.
  11.2.2. Relationship/Relationship.
  11.2.3. Attribute/Attribute.
Solution: precede general solution by transformation into entities or E-R constructs.
11.3. Object 1 and Object 2 have a common superset and overlap
Solution: combine solutions for 11.2. and 11.3.
  11.3.1. Entity/Entity.
  11.3.2. Relationship/Relationship.
  11.3.3. Attribute/Attribute.

12. DIFFERENT BUT RELATED HOMONYMS
N1 = N2; T1 = T2; M1 <> M2; related(M1,M2); C1 = C2;

Solution: rename and solve similar to 11.

12.1. One object contains the other (Object 1 contains Object 2 or vice versa).
  12.1.1. Entity/Entity.
  12.1.2. Relationship/Relationship.
  12.1.3. Attribute/Attribute.
12.2. Object 1 and Object 2 have a common superset (but do not overlap).
  12.2.1. Entity/Entity.
  12.2.2. Relationship/Relationship.
  12.2.3. Attribute/Attribute.
12.3. Object 1 and Object 2 have a common superset and overlap.
  12.3.1. Entity/Entity.
  12.3.2. Relationship/Relationship.
  12.3.3. Attribute/Attribute.

13. DIFFERENT BUT RELATED OBJECTS WITH DIFFERENT CONTEXT
N1 <> N2; T1 = T2; M1 <> M2; related(M1,M2); C1 <> C2;

13.1. Entity Attribute related to Entity Attribute of a different entity.
Solution: transform attributes into E-R constructs and solve relatedness as in case 11.
  13.1.1. Attribute 1 contains Attribute 2 (or vice versa).
  13.1.2. Common superset.
  13.1.3. Common subset and superset.
13.2. Entity Attribute related to Relationship Attribute
Solution: transform entity attribute into E-R construct, relationship attribute into entity and solve relatedness as in 11.
  13.2.1. Attribute 1 contains Attribute 2 (or vice versa).
  13.2.2. Common superset.
  13.2.3. Common subset and superset.
13.3. Relationship Attribute related to Relationship Attribute
Solution: transform attributes into entities and solve relatedness as in 11.
  13.3.1. Attribute 1 contains Attribute 2 (or vice versa).
  13.3.2. Common superset.
  13.3.3. Common subset and superset.
13.4. Relationship related to Relationship
Solution: transform relationships into entities and solve relatedness as in 11.
  13.4.1. Relationship 1 contains Relationship 2 (or vice versa).
  13.4.2. Common superset.

13.4.3. Common subset and superset.

14. DIFFERENT BUT RELATED HOMONYMS WITH DIFFERENT CONTEXT
N1 = N2; T1 = T2; M1 <> M2; related(M1,M2); C1 <> C2;

Solution: rename to avoid homonym and solve similar to 13.

14.1. Entity Attribute related to Entity Attribute of a different entity.
14.1.1. Attribute 1 contains Attribute 2 (or vice versa).
14.1.2. Common superset.
14.1.3. Common subset and superset.
14.2. Entity Attribute related to Relationship Attribute.
14.2.1. Attribute 1 contains Attribute 2 (or vice versa).
14.2.2. Common superset.
14.2.3. Common subset and superset.
14.3. Relationship Attribute related to Relationship Attribute.
14.3.1. Attribute 1 contains Attribute 2 (or vice versa).
14.3.2. Common superset.
14.3.3. Common subset and superset.
14.4. Relationship related to Relationship
14.4.1. Relationship 1 contains Relationship 2 (or vice versa).
14.4.2. Common superset.
14.4.3. Common subset and superset.

15. DIFFERENT BUT RELATED OBJECTS OF DIFFERENT TYPE
N1 <> N2; T1 <> T2; M1 <> M2; related(M1,M2); C1 <> C2;

15.1. Entity Attribute related to Entity-Relationship construct.
Solution: transform entity attribute into E-R construct and solve relatedness similar to 11.
15.1.1. Entity Attribute related to Entity.
15.1.1.1. One object contains the other.
15.1.1.2. Common superset.
15.1.1.3. Common subset and superset.
15.1.2. Entity Attribute related to Relationship.
15.1.2.1. One object contains the other.
15.1.2.2. Common superset.
15.1.2.3. Common subset and superset.
15.2. Relationship Attribute related to Entity.
15.2.1. One object contains the other.

15.2.2. Common superset.
15.2.3. Common subset and superset.
15.3. Entity related to Relationship.
15.3.1. One object contains the other.
15.3.2. Common superset.
15.3.3. Common subset and superset.

16. DIFFERENT BUT RELATED HOMONYMS OF DIFFERENT TYPE

N1 = N2; T1 <> T2; M1 <> M2; related(M1,M2); C1 <> C2;

Solution: rename at least one object to avoid homonym and solve similar to 15.

16.1. Entity Attribute related to Entity-Relationship construct
16.1.1. Entity Attribute related to Entity.
16.1.1.1. One object contains the other.
16.1.1.2. Common superset.
16.1.1.3. Common subset and superset.
16.1.2. Entity Attribute related to Relationship.
16.1.2.1. One object contains the other.
16.1.2.2. Common superset.
16.1.2.3. Common subset and superset.
16.2. Relationship Attribute related to Entity.
16.2.1. One object contains the other.
16.2.2. Common superset.
16.2.3. Common subset and superset.
16.3. Entity related to Relationship.
16.3.1 One object contains the other.
16.3.2. Common superset.
16.3.3. Common subset and superset.

17. MISSING OBJECT

Object 2 does not exist.

Solution: add missing object.

17.1 Entity missing.
17.2 Relationship missing.
17.3 Attribute missing.

Appendix 2: Conflict Solutions

1. IDENTICAL OBJECTS
N1 = N2; T1 = T2; M1 = M2; C1 = C2;

Solution: *do nothing*.

1.1. Entity is Entity.
1.2. Relationship is Relationship.
1.3. Attribute is Attribute.

2. IDENTICAL OBJECTS WITH DIFFERENT CONTEXT
N1 = N2; T1 = T2; M1 = M2; C1 <> C2;

2.1. Relationship is Relationship of different degree or associating different entities.
Solution: *S4, possibly S1 or S2 or S11*.
2.2. Attribute is Attribute of a different entity or relationship (both are possession attributes).
Solution: *S2 or S3*.

3. TRUE SYNONYMS (SAME OBJECT TYPE)
N1 <> N2; T1 = T2; M1 = M2; C1 = C2;

Solution: *S10*.

3.1. Entity/Entity.
3.2. Relationship/Relationship.
3.3. Attribute/Attribute.

4. TRUE SYNONYMS WITH DIFFERENT CONTEXT
N1 <> N2; T1 = T2; M1 = M2; C1 <> C2;

4.1. Relationship/Relationship.
Solution: *S10 and S4, possibly S1, or S2, or S11*.
4.2. Attribute/Attribute.
Solution: *S10 and S2 or S3*.

5. CONSTRUCT MISMATCH
N1 = N2; T1 <> T2; M1 = M2; C1 <> C2;

5.1. Entity is Relationship.
Solution: S1.
5.2. Entity Attribute is Entity-Relationship construct.
Solution: *S3*.
5.2.1. Attribute is Entity.
5.2.2. Attribute is Relationship.
5.3. Relationship Attribute is Entity.

Solution: *S2.*

6. CONSTRUCT MISMATCH AND SYNONYM
N1 <> 2; T1 <> T2; M1 = M2; C1 <> C2;

6.1. Entity is Relationship.
Solution: *S10 and S1.*
6.2. Entity Attribute is Entity-Relationship construct.
Solution: *S10 and S3.*
6.2.1. Attribute is Entity.
6.2.2. Attribute is Relationship.
6.3. Relationship Attribute is Entity.
Solution: *10 and S2.*

7. DIFFERENT AND UNRELATED OBJECTS
N1 <> N2; T1 = T2; M1 <> M2; not(related(M1,M2)); C1 = C2 or C1 <> C2;

7.1. Objects are different, unrelated and have no common role.
Solution: *do nothing.*
7.1.1. Entity/Entity.
7.1.2. Relationship/Relationship.
7.1.3. Attribute/Attribute.
7.2. Object 1 and Object 2 in same role (W-relationship).
7.2.1. Entity/Entity.
Solution: S7.
7.2.2. Relationship/Relationship.
Solution: *S1 and S7.*
7.2.3. Attribute/Attribute.
Solution: *S2 or S3 followed by S7.*

8. TRUE HOMONYM
N1 = N2; T1 = T2; M1 <> M2; C1 = C2 or C1 <> C2;

8.1. Objects are different, unrelated and have no common role.
Solution: *S10.*
8.1.1. Entity/Entity.
8.1.2. Relationship/Relationship.
8.1.3. Attribute/Attribute.
8.2. Object 1 and Object 2 in same role (W-relationship).
8.2.1. Entity/Entity.
Solution: *S10 followed by S7.*
8.2.2. Relationship/Relationship.

Solution: *S10 and S1 followed by S7.*

8.2.3. Attribute/Attribute.
Solution: *S10 and S2 or S3 followed by S7.*

9. DIFFERENT OBJECTS WITH DIFFERENT CONSTRUCTS
N1 <> N2; T1 <> T2; M1 <> M2; C1 <> C2;

9.1. Objects are different, unrelated and have no common role.
Solution: *do nothing.*
9.1.1. Entity/Relationship.
9.1.2. Relationship/Attribute.
9.1.3. Entity/Attribute.

9.2. Object 1 and Object 2 in same role (W-relationship).
9.2.1. Entity/Relationship.
Solution: *S1 followed by S7.*
9.2.2. Relationship/Attribute.
Solution: *S1 and S2 or S3 followed by S7.*
9.2.3. Entity/Attribute.
Solution: *S2 or S3 followed by S7.*

10. DIFFERENT OBJECTS WITH DIFFERENT CONSTRUCTS, BUT HOMONYMS
N1 = N2; T1 <> T2; M1 <> M2; C1 <> C2;

10.1. Objects are different, unrelated and have no common role.
Solution: *S10.*
10.1.1. Entity/Relationship.
10.1.2. Relationship/Attribute.
10.1.3. Entity/Attribute.

10.2. Object 1 and Object 2 in same role (W-relationship).
10.2.1. Entity/Relationship.
Solution: *S10 and S1 followed by S7.*
10.2.2. Relationship/Attribute.
Solution: *S10, S1 and S2 or S3, followed by S7.*
10.2.3. Entity/Attribute.
Solution: *S10 and S2 or S3, followed by S7.*

11. DIFFERENT BUT RELATED OBJECTS
N1 <> N2; T1 = T2; M1 <> M2; related(M1,M2); C1 = C2;

11.1. One object contains the other (Object 1 contains Object 2 or vice versa).

11.1.1. Entity/Entity.
Solution: *S6.*

11.1.2. Relationship/Relationship.
Solution: *S1 and S6.*

11.1.3. Attribute/Attribute.
Solution: *S2 or S3, followed by S6.*

11.2. Object 1 and Object 2 have a common superset (but do not overlap).

11.2.1. Entity/Entity.
Solution: *S8.*

11.2.2. Relationship/Relationship.
Solution: *S1 and S8.*

11.2.3. Attribute/Attribute.
Solution: *S2 or S3, followed by S8.*

11.3. Object 1 and Object 2 have a common superset and overlap

11.3.1. Entity/Entity.
Solution: *S9.*

11.3.2. Relationship/Relationship.
Solution: *S1 and S9.*

11.3.3. Attribute/Attribute.
Solution: *S2 or S3, followed by S9.*

12. DIFFERENT BUT RELATED HOMONYMS

N1 = N2; T1 = T2; M1 <> M2; related(M1,M2); C1 = C2;

12.1. One object contains the other (Object 1 contains Object 2 or vice versa).

12.1.1. Entity/Entity.
Solution: *S10 and S6.*

12.1.2. Relationship/Relationship.
Solution: *S10 and S1 and S6.*

12.1.3. Attribute/Attribute.
Solution: *S10 and S2 or S3, followed by S6.*

12.2. Object 1 and Object 2 have a common superset (but do not overlap).

12.2.1. Entity/Entity.
Solution: *S10 and S8.*

12.2.2. Relationship/Relationship.
Solution: *S10 and S1 and S8.*

12.2.3. Attribute/Attribute.
Solution: *S10 and S2 or S3, followed by S8.*

12.3. Object 1 and Object 2 have a common superset and overlap.

12.3.1. Entity/Entity.
Solution: *S10 and S9.*

12.3.2. Relationship/Relationship.
Solution: *S10 and S1 and S9.*

12.3.3. Attribute/Attribute.
Solution: *S10 and S2 or S3, followed by S9.*

13. DIFFERENT BUT RELATED OBJECTS WITH DIFFERENT CONTEXT

N1 <> N2; T1 = T2; M1 <> M2; related(M1,M2); C1 <> C2;

13.1. Entity Attribute related to Entity Attribute of a different entity.

13.1.1. Attribute 1 contains Attribute 2 (or vice versa).
Solution: *S3 and S6.*

13.1.2. Common superset.
Solution: *S3 and S8.*

13.1.3. Common subset and superset.
Solution: *S3 and S9.*

13.2. Entity Attribute related to Relationship Attribute

13.2.1. Attribute 1 contains Attribute 2 (or vice versa).
Solution: *S2 and S3 and S6.*

13.2.2. Common superset.
Solution: *S2 and S3 and S8.*

13.2.3. Common subset and superset.
Solution: *S2 and S3 and S9.*

13.3. Relationship Attribute related to Relationship Attribute

13.3.1. Attribute 1 contains Attribute 2 (or vice versa).
Solution: *S2 and S6.*

13.3.2. Common superset.
Solution: *S2 and S8.*

13.3.3. Common subset and superset.
Solution: *S2 and S9.*

13.4. Relationship related to Relationship

13.4.1. Relationship 1 contains Relationship 2 (or vice versa).
Solution: *S1 and S6.*

13.4.2. Common superset.
Solution: *S1 and S8.*

13.4.3. Common subset and superset.
Solution: *S1 and S9.*

14. DIFFERENT BUT RELATED HOMONYMS WITH DIFFERENT CONTEXT
N1 = N2; T1 = T2; M1 <> M2; related(M1,M2); C1 <> C2;

- 14.1. Entity Attribute related to Entity Attribute of a different entity.
  - 14.1.1. Attribute 1 contains Attribute 2 (or vice versa).
    Solution: *S10 and S3 and S6.*
  - 14.1.2. Common superset.
    Solution: *S10 and S3 and S8.*
  - 14.1.3. Common subset and superset.
    Solution: *S10 and S3 and S9.*
- 14.2. Entity Attribute related to Relationship Attribute.
  - 14.2.1. Attribute 1 contains Attribute 2 (or vice versa).
    Solution: *S10 and S2 and S3 and S6.*
  - 14.2.2. Common superset.
    Solution: *S10 and S2 and S3 and S8.*
  - 14.2.3. Common subset and superset.
    Solution: *S10 and S2 and S3 and S9.*
- 14.3. Relationship Attribute related to Relationship Attribute.
  - 14.3.1. Attribute 1 contains Attribute 2 (or vice versa).
    Solution: *S10 and S2 and S6.*
  - 14.3.2. Common superset.
    Solution: *S10 and S2 and S8.*
  - 14.3.3. Common subset and superset.
    Solution: *S10 and S2 and S9.*
- 14.4. Relationship related to Relationship
  - 14.4.1. Relationship 1 contains Relationship 2 (or vice versa).
    Solution: *S10 and S1 and S6.*
  - 14.4.2. Common superset.
    Solution: *S10 and S1 and S8.*
  - 14.4.3. Common subset and superset.
    Solution: *S10 and S1 and S9.*

15. DIFFERENT BUT RELATED OBJECTS OF DIFFERENT TYPE
N1 <> N2; T1 <> T2; M1 <> M2; related(M1,M2); C1 <> C2;

- 15.1. Entity Attribute related to Entity-Relationship construct.
  - 15.1.1. Entity Attribute related to Entity.
    - 15.1.1.1. One object contains the other.
      Solution: *S3 and S6.*
    - 15.1.1.2. Common superset.
      Solution: *S3 and S8.*
    - 15.1.1.3. Common subset and superset.
      Solution: *S3 and S9.*

15.1.2. Entity Attribute related to Relationship.
15.1.2.1. One object contains the other.
Solution: *S3 and S6.*
15.1.2.2. Common superset.
Solution: *S3 and S8.*
15.1.2.3. Common subset and superset.
Solution: *S3 and S9.*
15.2. Relationship Attribute related to Entity.
15.2.1. One object contains the other.
Solution: *S2 and S6.*
15.2.2. Common superset.
Solution: *S2 and S8.*
15.2.3. Common subset and superset.
Solution: S2 and S9.
15.3. Entity related to Relationship.
15.3.1. One object contains the other.
Solution: *S1 and S6.*
15.3.2. Common superset.
Solution: *S1 and S8.*
15.3.3. Common subset and superset.
Solution: *S1 and S9.*

16. DIFFERENT BUT RELATED HOMONYMS OF DIFFERENT TYPE
N1 = N2; T1 <> T2; M1 <> M2; related(M1,M2); C1 <> C2;

16.1. Entity Attribute related to Entity-Relationship construct
16.1.1. Entity Attribute related to Entity.
16.1.1.1. One object contains the other.
Solution: *S10 and S3 and S6.*
16.1.1.2. Common superset.
Solution: *S10 and S3 and S8.*
16.1.1.3. Common subset and superset.
Solution: *S10 and S3 and S9.*
16.1.2. Entity Attribute related to Relationship.
16.1.2.1. One object contains the other.
Solution: *S10 and S3 and S6.*
16.1.2.2. Common superset.
Solution: *S10 and S3 and S8.*
16.1.2.3. Common subset and superset.
Solution: *S10 and S3 and S9.*
16.2. Relationship Attribute related to Entity.
16.2.1. One object contains the other.
Solution: *S10 and S2 and S6.*
16.2.2. Common superset.
Solution: *S10 and S2 and S8.*
16.2.3. Common subset and superset.
Solution: *S10 and S2 and S9.*
16.3. Entity related to Relationship.

16.3.1. One object contains the other.
Solution: *S10 and S1 and S6.*
16.3.2. Common superset.
Solution: *S10 and S1 and S8.*
16.3.3. Common subset and superset.
Solution: *S10 and S1 and S9.*

17. MISSING OBJECT
Object 2 does not exist.

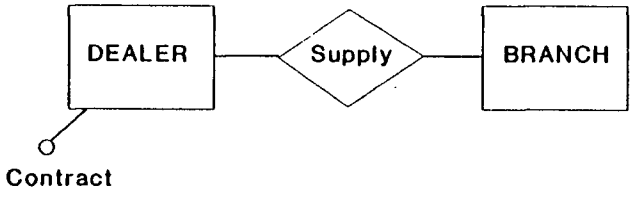
Solution: *S11.*

17.1. Entity missing.
17.2. Relationship missing.
17.3. Attribute missing.

Appendix 3: View Integration Session with AVIS

A view integration session with AVIS is illustrated through a set of 22 screen displays. The problem "c34" consists of two small views which have to be integrated. Figure 28 depicts the structure of the views.

View 1:



View 2:

Figure 28: View Integration Sample Problem

The screens shown below exemplify questions asked by the AVIS system as well as AVIS' support functions. These support functions for instance indicate to the designer what the program already knows or what the current contents of each view are. Example screens which display system questions to the user will not depict user replies. The short summary description of each screen shown below, however, states the user answers and documents the purpose of each screen.

Screen	Purpose
1	AVIS title screen, asks user to choose a problem file. Chosen here: "c34".
2	First system question. User answers "1003".

The following screens 3 - 8 exemplify support functions which can be activated at any time during the integration session when the system is ready to accept input. Some of the screens may initially have no or little content, i.e., screen 4. They are shown here to demonstrate the system status at the beginning of an integration session and to allow a comparison with later system stati. Screens 3 - 8 show the system status before the

user's answer "1003". The user gave his answer after seeing screen 8.

3	Shows "Agenda", consisting of present and future object comparison tasks (preview).
4	Shows "Old Agenda", consisting of current and previous object comparison tasks (history log).
5 & 6	Show the contents of views 1 and 2 (at the outset of the integration session).
7	Shows list of "facts", knowledge about the set of views based on previous object comparisons. Here the list is still empty.
8	Meaning comparison screen. Shows what the system knows about the match between objects. Here, best match is with "1003 - dealer".
9	System question 2. User answers "n".
10	System question 3. User answers "n", but not until seeing screen 11.
11	"Old Agenda" now shows the previous four system questions. Note that the system never asked the user for Synonym (agenda item 2) because it can assess without user help whether objects carry different names (simple string comparison).
12	System question 4. User answers "0", but not until seeing screen 14.
13	"Meaning match" support function suggests no similarity.
14	Fact list shows the knowledge asserted at this point in time. I.e., objects 3 and 1003 are identical (same).
15	System question 5. User answers "0".
16	System question 6. User answers "1005". Note that the system reports in the lower right window that in the mean time, a new object, 2013 - branch, has been added.
17	System question 7. User answers "n". The number 18 on the upper right hand corner of the screen shows that the system has internally created 18 questions, but has asked the user only 7. The remaining ones were answered by the system.
18	"THE AGENDA IS EMPTY". The system has created two identical views, without further questions to the user. Note the internal count of 30 questions (upper right corner).
19	The "Old Agenda" shows the last 12 questions, answered by the system without user interaction.
20	First part of the Fact list.
21 & 22	The adjusted views 1 and 2. All newly created objects can be identified by their object identifiers (>2000).

SCREEN 1

A V I S

```
          AAA         VV          VV       II          SSSSSSSSSS
         AA AA         VV        VV        II        SS
        AA   AA         VV      VV         II        SS
       AA     AA         VV    VV          II         SSSSSSSS
      AA       AA         VV VV            II                 SS
     AA AAAAAAA AA         VVV             II                 SS
    AA           AA         V              II          SSSSSSSSS
   AA             AA
  AA               AA AA AA AA AA AA AA AA AA AA AA AA AA AA AAA
```

- Automated View Integration Systems -

by Christian Wagner, 1988.

Response

What is the name of the file containing the views? [TEST.PRO]
c34

SCREEN 2

A V I S

Testing for hypothesis: SIMILAR ENTITY,
involving the entity DEALER (3) and one of the following objects:

1003 -- dealer
1004 -- customer
1005 -- contract

Make Agenda

Hypo Test

Response

Please type in the number of the applicable object (or 0 if none).

New Objects

Assert Objects

SCREEN 3

A V I S

Testing for hypothesis: SIMILAR ENTITY,
involving the entity DEALER (3) and one of the following objects:

Agenda

Current Agenda Item
H: 3 Similar Entity - [3][1003,1004,1005]

Response

Press <spacebar> to continue

SCREEN 4

A V I S

Testing for hypothesis: SIMILAR ENTITY,
involving the entity DEALER (3) and one of the following objects:

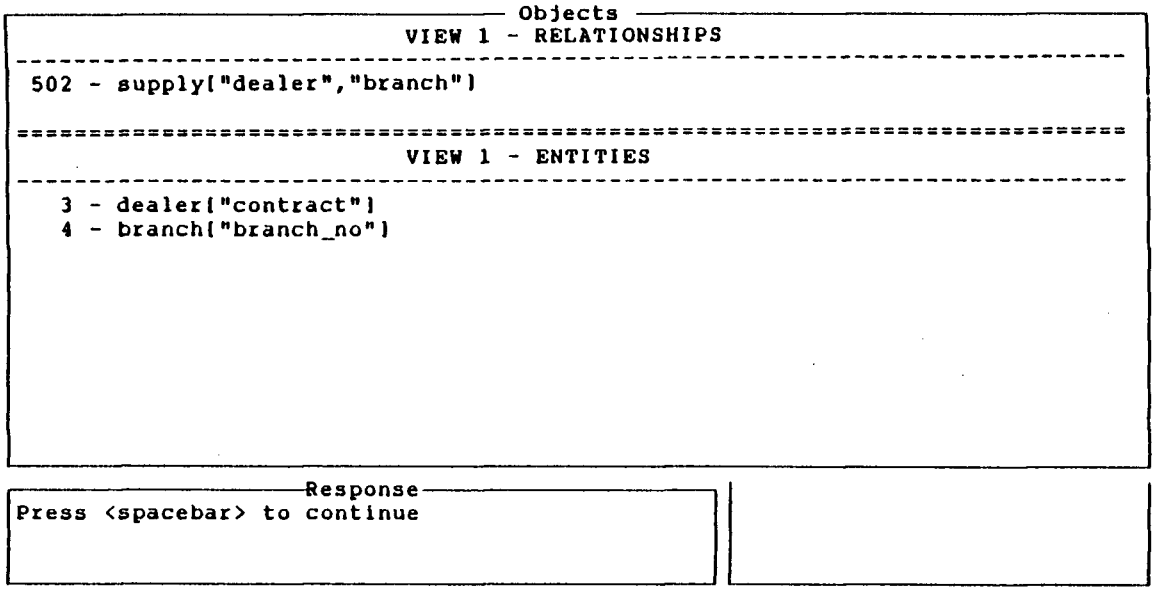
Old Agenda

1: -1-> 3 - Similar Entity - [3][1003,1004,1005]

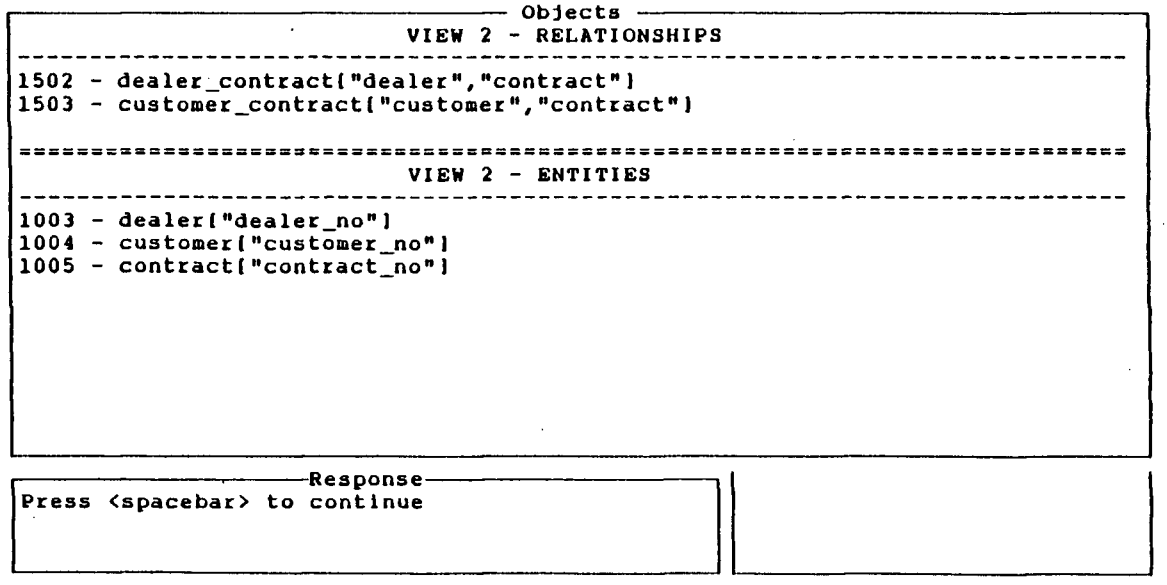
Response

Press <spacebar> to continue

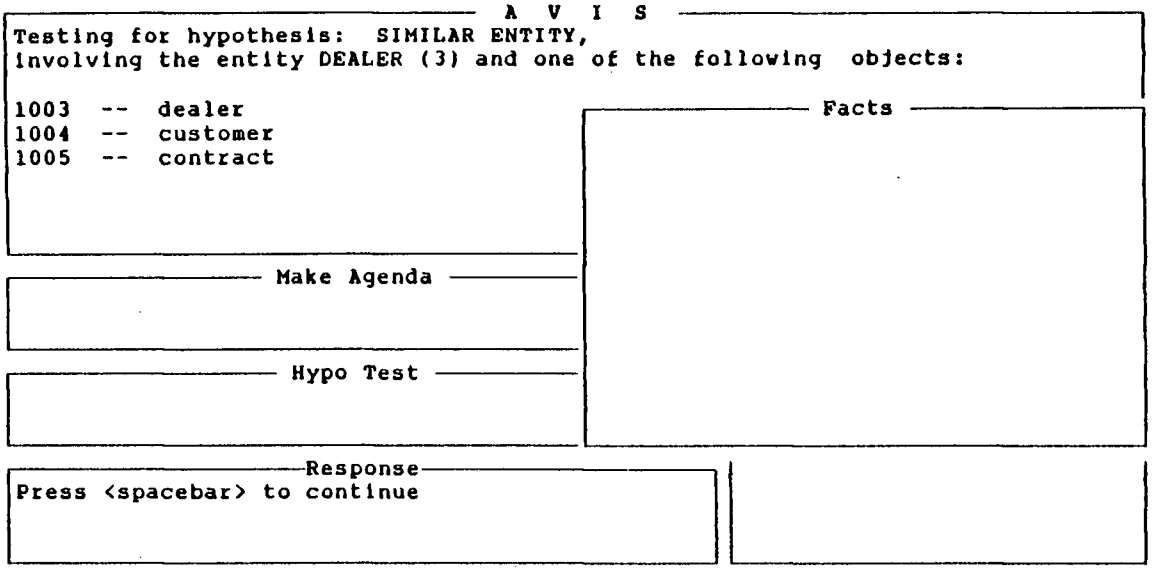
SCREEN 5



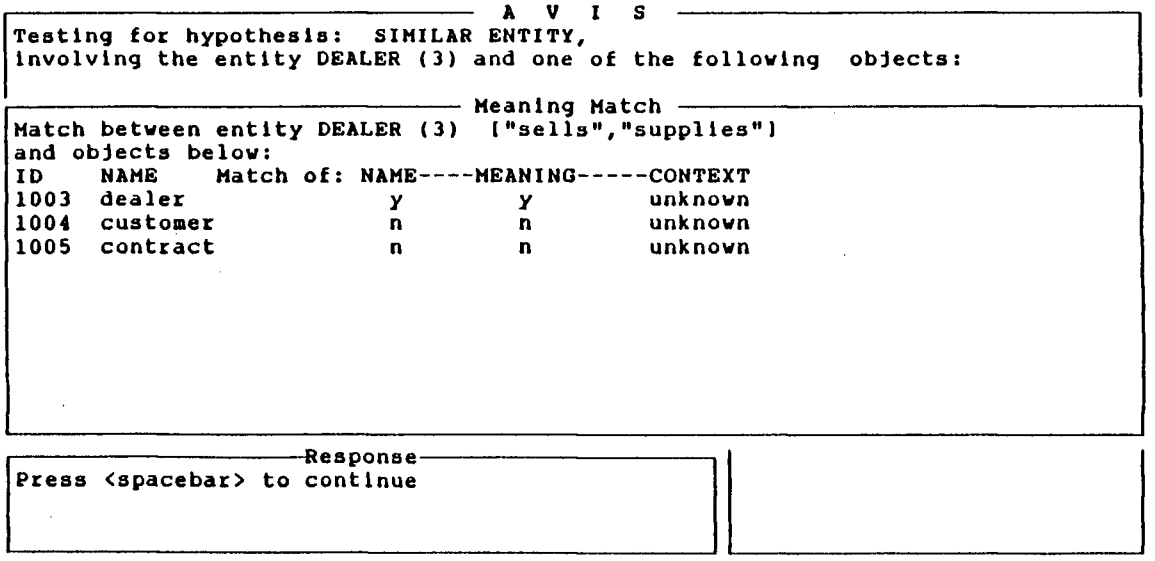
SCREEN 6



SCREEN 7



SCREEN 8



SCREEN 9

3

A V I S

Testing for hypothesis: SIMILAR RELATIONSHIP,
involving relationship SUPPLY (502) and relationship DEALER_CONTRACT (1502)

Make Agenda

3 -> agenda(similar_meaning,[3],[1003])

Hypo Test

TO BE EXECUTED:
similar_meaning([502],[1502])

Response

Please answer with y or n to indicate whether
the hypothesis is true or false.

New Objects

Assert Objects

ao(4,3,1003,n)

- test_hypo(7)

SCREEN 10

4

A V I S

Testing for hypothesis: RELATED RELATIONSHIP,
involving relationship SUPPLY (502) and relationship DEALER_CONTRACT (1502)

Make Agenda

1 -> agenda(homonyms,[502],[1502])

Hypo Test

TO BE EXECUTED:
related([502],[1502])

Response

Please answer with y or n to indicate whether
the hypothesis is true or false.

New Objects

Assert Objects

ao(1,502,1502,n)

- test_hypo(7)

SCREEN 11

4

A V I S

Testing for hypothesis: RELATED RELATIONSHIP,
involving relationship SUPPLY (502) and relationship DEALER_CONTRACT (1502)

Old Agenda

```
 1: -1->  3 - Similar Entity - [3][1003,1004,1005]
 2:  3->  4 - Synonym - [3][1003]
 3:  0->  1 - Similar Relationship - [502][1502]
 4:  1-> 13 - Related Relationship - [502][1502]
```

Response

Press <spacebar> to continue

- test_hypo(7)

SCREEN 12

7

A V I S

Testing for hypothesis: ENTITY ATTRIBUTE IS RELATIONSHIP CONSTRUCT,
involving the attr DEALER_NO (2001) and one of the following objects:

4 -- branch
502 -- supply

Make Agenda

13 -> agenda(ea_is_rc,[502],[1502])

New Objects

Hypo Test

TO BE EXECUTED:
ea_is_rc([2001],[4,502])

Assert Objects

ao(13,502,1502,n)

Response

Please type in the number of the applicable
object (or 0 if none).

- test_hypo(7)

SCREEN 13

7

A V I S

Testing for hypothesis: ENTITY ATTRIBUTE IS RELATIONSHIP CONSTRUCT,
involving the attr DEALER_NO (2001) and one of the following objects:

Meaning Match

Match between attr DEALER_NO (2001) ["key"]
and objects below:

ID	NAME	Match of: NAME	MEANING	CONTEXT
4	branch	n	n	none
502	supply	n	n	none

Response

Press <spacebar> to continue

- test_hypo(7)

SCREEN 14

7

A V I S

Testing for hypothesis: ENTITY ATTRIBUTE IS RELATIONSHIP CONSTRUCT,
involving the attr DEALER_NO (2001) and one of the following objects:

4 -- branch
502 -- supply

Facts

similar_meaning(3,1003)
same(3,1003)
dissimilar_meaning(502,1502)
unrelated(502,1502)

Make Agenda

13 -> agenda(ea_is_rc,[502],[1502])

Hypo Test

TO BE EXECUTED:
ea_is_rc([2001],[4,502])

Response

Press <spacebar> to continue

- test_hypo(7)

SCREEN 15

A V I S

```
Testing for hypothesis:  SIMILAR ENTITY,
involving the entity BRANCH (4) and one of the following  objects:

1004  --  customer
1005  --  contract
```

Make Agenda

Hypo Test

Response

```
Please type in the number of the applicable
object (or 0 if none).
```

New Objects

Assert Objects

SCREEN 16

6

A V I S

```
Testing for hypothesis:  ENTITY ATTRIBUTE IS RELATIONSHIP CONSTRUCT,
involving the attr CONTRACT (600) and one of the following  objects:

1005  --  contract
1502  --  dealer_contract
```

Make Agenda

Hypo Test

```
TO BE EXECUTED:
ea_is_rc([600],[1005,1502])
```

Response

```
Please type in the number of the applicable
object (or 0 if none).
```

New Objects

```
H-similar_meaning
  added objects:
2013  --  branch
```

Assert Objects

```
ao(301,4,0,n)


-  test_hypo(7)
```

SCREEN 17

18

A V I S

Testing for hypothesis: ENTITY ATTRIBUTE IS RELATIONSHIP CONSTRUCT,
involving attr DEALER_NO (2001) and relationship SUPPLY (502)

Make Agenda

13 -> agenda(ea_is_rc,[1502],[502])

Hypo Test

TO BE EXECUTED:
ea_is_rc([2001],[502])

Response

Please answer with y or n to indicate whether
the hypothesis is true or false.

New Objects

H-missing
added objects:
2023 -- customer_contract

Assert Objects

ao(13,1502,502,n)

- test_hypo(7)

SCREEN 18

30

A V I S

Make Agenda

13 -> agenda(ea_is_rc,[2027],[2017])

Hypo Test

PRECONDITION FAILED:
related([2027],[2017])

Response

- - - THE AGENDA IS EMPTY - - -

New Objects

H-missing
added objects:
2027 -- supply

Assert Objects

ao(13,2027,2017,n)

- asso(1301)

SCREEN 19

```
                                                                    30
------------------------------- A  V  I  S ----------------------------

------------------------------- Old Agenda ----------------------------
19:  0-> 19 - Missing Relationship - [502][]
20: -5->  1 - Similar Relationship - [2023][1503,1502]
21: -5->  1 - Similar Relationship - [502][2027]
22: -5->  1 - Similar Relationship - [2017][2027]
23: -6->  1 - Similar Relationship - [1502][2023]
24: -6->  1 - Similar Relationship - [1503][2023]
25: -6->  1 - Similar Relationship - [2027][2017,502]
26: -7-> 13 - Related Relationship - [2017][2027]
27: -7-> 13 - Related Relationship - [2023][1502]
28: -8-> 13 - Related Relationship - [1502][2023]
29: -8-> 13 - Related Relationship - [1503][2017]
30: -8-> 13 - Related Relationship - [2027][2017]

--------------- Response ---------------
Press <spacebar> to continue                 -  asso(1301)
```

SCREEN 20

```
                                                                    30
------------------------------- A  V  I  S ----------------------------

                                        ------------ Facts ------------
                                        similar_meaning(3,1003)
                                        same(3,1003)
                                        dissimilar_meaning(502,1502)
                                        unrelated(502,1502)
                                        ea_is_not_rc(2001,4)
                                        ea_is_not_rc(2001,502)
----------- Make Agenda -------------   dissimilar_meaning(4,1004)
                                        dissimilar_meaning(4,1005)
13 -> agenda(ea_is_rc,[2027],[2017])    similar_meaning(4,2013)
                                        same(4,2013)
------------ Hypo Test --------------    dup(4,2013)
PRECONDITION FAILED:                                  ... more >
related([2027],[2017])

--------------- Response ---------------
   - - -  THE AGENDA IS EMPTY  - - -         -  asso(1301)
```

SCREEN 21

Objects

VIEW 1 - RELATIONSHIPS

```
 502 - supply["dealer","branch"]
2017 - dealer_contract["dealer","contract"]
2023 - customer_contract["customer","contract"]
```

VIEW 1 - ENTITIES

```
   3 - dealer["contract"]
   4 - branch["branch_no"]
2015 - contract["contract_no"]
2021 - customer["customer_no"]
```

Response

```
Press <spacebar> to continue
```

```
- asso(1301)
```

SCREEN 22

Objects

VIEW 2 - RELATIONSHIPS

```
1502 - dealer_contract["dealer","contract"]
1503 - customer_contract["customer","contract"]
2027 - supply["dealer","branch"]
```

VIEW 2 - ENTITIES

```
1003 - dealer["dealer_no"]
1004 - customer["customer_no"]
1005 - contract["contract_no"]
2013 - branch["branch_no"]
```

Response

```
Press <spacebar> to continue
```

```
- asso(1301)
```