VIEW INTEGRATION

USING

THE ENTITY-RELATIONSHIP MODEL

VOL I

Mansoor Ahmed HASSAN

Doctor of Philosophy

THE UNIVERSITY OF ASTON IN BIRMINGHAM

June 1989

The University of Aston in Birmingham

VIEW INTEGRATION
USING
THE ENTITY-RELATIONSHIP MODEL

VOL I

Mansoor Ahmed HASSAN

Doctor of Philosophy

1989

The development of global conceptual schemas for very large organizations is complex, time consuming and error prone. This thesis presents a view integration methodology which integrates the various user views, expressed originally in entity-relationship model form and transformed into a specially developed language, into one global conceptual schema. The semantics of each additional user view are captured by matching the entity-relationships of that view with those of the global conceptual schema, which is updated accordingly. Two approaches to view integration have been developed: binary view integration which integrates a view at a time, and n-ary view integration which integrates all the views 'simultaneously'. Further, prototypes of n-ary view integrators called the entity n-ary view integrator and the relationship n-ary view integrator have been developed. The view integrator identifies and resolves conflicts which may take place during view integration. A particular technique called object fuzzy matching was developed to identify some of the possible synonym naming conflicts. Other types of conflicts which are called cross object type conflicts occur when the same semantics are modelled in different user views using different objects or structures. The definitions and resolutions of a number of these conflicts are presented in this thesis.

Keywords: view integration, entity-relationship model, semantic data modelling, conceptual schema, database design

Dedication

To my mother and father

and

to my wife and two children Mona and Salah

Acknowledgements

I would like to thank my research supervisor David Avison for his direct and constructive criticisms throughout this research. I have learnt a great deal from him. It gives me pleasure to thank him for his help and encouragement.

During my stay in Britain I have met many people whom I would like to thank. Thanks to my other family in the north. Thanks to Neil Simpkins for his help at the early stages of learning Prolog. Thanks to all the staff and research student from the department of Computer Science at Aston for providing a friendly and cooperative atmosphere.

From Bahrain University, I would like to thank Ebrahim El-hashimi, Nezar El-baharna and Sameer Fakhro for their support throughout and especially during my illness. Special thanks to Abdul-imam for being a good friend.

List of Contents

Volume 1

Volume 2

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Background

A *conceptual schema* (or *schema*) is a description of data and its semantic properties. In the context of databases, the conceptual schema is a description of the part of the organization which is to be represented by the data in the database (Lum *et al* 1978, Jardine 1984). This description might include:

1 The entities involved.

2 The relationships between these entities.

3 The attributes which define the entities and relationships.

4 The semantic integrity constraints which apply to attributes, entities and relationships.

A *data model* is a mechanism for specifying the structure of a database and the operations that may be performed on the data in that database (Yao 1985). Thus, a database is an instance of a specific data model. A specification of such an instance of the data model is the conceptual schema.

An early classification of data models in a database environment included the hierarchical, network and relational models (Ullman 1982, Date 1983). These are usually referred to as the *classical* or *conventional* data models. *Semantic Data Models* (or *SDM*s) were proposed because conventional data models suffer from a number of problems and limitations (Abrial 1974, Mylopoulos 1978, Biller & Neuhold 1978, Bubenko 1979, Tsichritzis & Lockovsky 1982, Brodie 1984, King & Mcleod 1985 and Hull & King 1987). The contributions of SDMs to data modelling are through:

1 The provision of a tool to enable the *database designer* (or *'designer'*) to capture and model the semantics of the organization.

2 The provision of a communication tool, which can be understood by the user and designer.

3 The separation of the implementation and the modelling issues in database design.

4 The provision of a focus for a database management system (DBMS) architecture.

The process of database design is divided into logical and physical phases. Logical database design deals with the production of the conceptual schema using an SDM. An

application includes both *static* and *dynamic* properties. Static properties are the objects, their relationships, and their attributes. Dynamic properties are the operations on the objects (De Antonellis & Zonta 1984, Branco & Yadav 1985 and Atzenit *et al* 1985). The conceptual schema may consist solely of the statics of the data or it may also include the definition of the dynamics of the application. The physical phase of database design deals with the mapping of the conceptual schema to a DBMS. This thesis deals with the development of the statics of the conceptual schema only.

An important consideration in database design is whether to design a single or large global database to meet the information needs of the users. McFadden & Hoffer (1985) consider the problems of the global approach:

> With the global approach, the designer attempts to design a single integrated database to meet the organization's present and future information needs. This approach is basically the "total systems" approach that was widely advocated during the 1960s. However, the global approach has seldom proved successful. The design task is so complex and the time and resources required are normally so large that the global approach becomes quite risky. Any benefits from database implementation are delayed for months or years, so that the project is in danger of loosing organizational commitment and momentum.

1.2 Motivation

The production of a schema for a very large organization is complex, time consuming and error prone. For such organizations, considerable demands are placed on the designer, who has to identify thousands of data elements, their semantic connections and their integrity constraints, and combine these to make a unique schema. Despite the use of an SDM to model the organization's semantics, the designer is expected to make both difficult and routine decisions in order to produce the final schema.

A large organization comprises many different users. Each user is concerned with a particular section of the overall data, and consequently may view this data in a different way to other users. Therefore, schema modelling using an SDM must allow the different user views to be reflected in a *Global Conceptual Schema* (or *GCS*) which represents the semantics of the whole organization.

The enormity of the task expected from the designer inspired many researchers to develop tools to help in the database design process. Database design tools have been developed for both the logical and the physical phases. However, the majority of these tools have concentrated on the logical design phase. There is much published research concerning

database design tools. Whilst these tools have contributed to helping the designer, he is still expected to know about the overall semantics of the organization and make all the decisions needed to develop the schema.

A different approach to developing the schema of a large organization is to model each section of the organization separately and then integrate these sections to form the schema. A section for a particular user in the organization is called a *view*. Breaking the organization into a number of views, and modelling these views individually using an SDM is called *view modelling*. Since each view is modelled separately, the designer is not at that stage burdened with the details of how the semantics of one view would map on to the semantics of the other views. Thus the view modelling task can be achieved quickly. The modelled views are fed to a *view integrator*, which integrates them into a GCS. The integration of the views by a view integrator is called the *view integration process*. This process can either be achieved manually by the designer or more effectively by a view integrator. Efforts to develop the schema using the view modelling and view integration approach are reported in Raver & Hubbard (1977), Elmasri & Wiederhold (1979), Navathe & Gadgil (1982), Batini & Lenzerini (1983 and 1984), Navathe *et al* (1984), Elmasri *et al* (1984), Elmasri & Navathe (1984), Batini *et al* (1982 and 1985b), Yao *et al* (1982 & 1985) and Navathe *et al* (1986).

The view modelling and view integration approach to database design relieves the designer from having to deal with numerous and complex semantics simultaneously. The view integrator is therefore expected to integrate the semantics from the views into the GCS, and to ensure that the resultant GCS is correct and complete. To achieve this, the view integrator must do the following:

1 Transport the semantics from the views to the GCS.
2 Identify and resolve the conflicts which may occur during integration.
3 Check the completeness of the GCS.

The transportation of the semantics from the views to the GCS requires the view integrator to 'comprehend' the definition of the SDM used to model the views, and the syntactic and semantic structures of the views. The level of 'comprehension' of these two aspects by the view integrator depends on a number of factors, described in sections 2.4 and 2.7.3.

Integration conflicts (or *conflicts*) take place when the same entities, relationships, entity-relationships or attributes are modelled differently in different views. These conflicts are either caused by different interpretations of the same semantics by the designer(s), or by genuine naming conflicts. *Naming conflicts* can either be *synonyms* (the same object is modelled under different names) or *homonyms* (different objects modelled under the same

name). The view integrator must identify and resolve synonyms and homonyms. Although these naming conflicts are reported in the view integration literature (see, for example, Batini & Lenzerini 1984), no method has been developed for their identification and resolution.

Another type of conflict occurs when two objects of different SDM types are modelled under the same name. This type of conflict is called the *cross object type conflict* (or *COT conflict*). In the entity-relationship model (ERM), described in Chen (1977 and 1985), for example, the same name could be shared by an entity and an attribute, an attribute and a relationship, an entity and a relationship, and so on.

Conflict identification and resolution is one of the most important functions of a view integrator. The view integrator can identify and possibly resolve all conflicts, provided the SDM used to model the views is formally defined, and strict view modelling rules are applied. There are many SDMs developed, which are reviewed in Bubenko (1979), Brodie (1984), Tsichritzis & Lockovsky (1982), King & Mcleod (1985) and Hull & King (1987), each claiming to be the most formally defined, the most semantically rich, the easiest to use, and so on. The most popular of these SDMs is ERM. A great deal of research has been reported about ERM in a series of conferences which have been held every two years since 1979 (Chen 1981, Chen 1983b, Davis *et al* 1983, Ferrara 1985b and Spaccapietra 1987).

To overcome the problems caused by the occurrence of conflicts in view integration, researchers associate special *assertions* with the views at view modelling and view integration. *Modelling assertions* are identified by the designer to complement the semantics of the views because these semantics cannot be modelled using the chosen SDM. *Integration assertions* are provided by the designer to direct the view integrator in resolving anticipated conflicts (Navathe & Gadgil 1982). The integration assertions are an indication of the inability of the view integrator to 'comprehend' the semantics of the views. The need for modelling assertions can be reduced by choosing a more semantically rich and formally defined SDM. However, integration assertions require research into ways of identifying and resolving conflicts automatically. Although some efforts to identify conflicts in view integration have been reported, the following points are still valid:

1 Not all possible conflicts have been identified.

2 The resolutions of some of the identified conflicts would not work for more general cases.

3 The methods devised for conflict resolution are manual, and many of these cannot be automated.

The problem of identifying synonyms is addressed in this thesis by implementing a method for calculating the level of similarity between objects of the same type. A method is also presented for the identification and resolution of COT conflicts.

A large organization consists of many views. Each view consists of many *entity-relationships* (or *E-R*) (assuming ERM as the SDM). The desired sequence in which the views, and consequently their E-Rs, are chosen for integration is not well argued in the literature. However, two ways in which views can be chosen for integration have been recommended (Navathe & Gadgil 1982 and Batini & Lenzerini 1984). The *binary* approach advocates that one view at a time is considered for integration with the GCS. The *n-ary* approach advocates that all the views are simultaneously integrated. Most of the literature stresses the binary approach because it is simpler to design the algorithm for the view integrator. However, no proposals are made about which view or E-R should be chosen next and why. With regards to n-ary view integration, no view integrator based on this approach has yet been reported. This thesis presents descriptions of both types of integration approaches and their implementation prototypes.

The view integration methodologies reported in the literature concentrate on different aspects of view integration, but no algorithms for the view integration process have been published (Batini *et al* 1986). Algorithms for all the activities of both the binary and n-ary view integrators are presented in this thesis.

1.3 Contributions

The number of view integration methodologies reported in the literature is limited. These reported methodologies have left many questions unanswered and some aspects unresearched. The objectives of this research were to:

1. Review view integration research.
2. Propose a method of integrating two E-Rs modelled in ERM.
3. Propose a method of identifying synonyms. This method is called *object fuzzy matching*.
4. Define and propose solutions to all the possible COT conflicts which could occur in view integration, and to implement a sample of these.
5. Develop a language called the *view definition language* (or *VDL*) to represent the views textually.
6. Implement a prototype of a *binary view integrator* (or *BVI*), which does not need any modelling or integration assertions.
7. Implement prototypes of two types of *n-ary view integrators* (or *NVI*).

a) An *entity n-ary view integrator* (or *ENVI*).

b) A *relationship n-ary view integrator* (or *RNVI*).

8 Develop a simple case study to test BVI and NVI prototypes.

1.4 Organization of the thesis.

Chapter 2 This chapter discusses the background of view integration and reviews the relevant research. Aspects of view integration discussed include the relationship between the SDM and the view integrator, the disadvantages of view integration based on dependencies, and the role of assertions. Each view integration methodology reported in the literature is reviewed.

Chapter 3 This chapter shows a method of integrating two E-Rs modelled in ERM. It is shown that integrating two E-Rs without the consideration of attributes, roles or cardinalities, results in 32 different situations. The integration of entities and relationships are shown separately.

Chapter 4 The first part of this chapter presents a method of carrying out object fuzzy matching of ERM objects to help identify synonyms. It is argued that only object fuzzy matching of entities is useful. The second part of the chapter defines a number of COT conflicts, and discusses their resolutions.

Chapter 5 A method for breaking a large organization into its views and subviews is presented. The result is a *view modelling tree*. Each of the views or subviews in the tree can then be modelled using an SDM. Further, a textual language called the *View Description Language* (*VDL*) is defined. The VDL is used to represent the semantics of the pictorial views to the view integrator textually.

Chapter 6 This chapter describes the implementation of BVI. It also shows how the views and the GCS are represented internally.

Chapter 7 This chapter describes two types of algorithms of N-ary view integration.

Chapter 8 This chapter discusses the results of a simple case study concerned with integrating sixteen views of the Department of Computer Science at Aston University. It shows the results of the object fuzzy matching of entities, the statistics of the conflicts which took place, and the resultant GCSs.

Chapter 9 This chapter presents the conclusions of the research. It proposes further research needed to enhance the prototypes and suggests further research into view integration as a whole.

CHAPTER 2

VIEW INTEGRATION - DESCRIPTION AND RESEARCH

2.1 Background

Raver & Hubbard (1977), Elmasri & Wiederhold (1979), Navathe & Gadgil (1982), Batini & Lenzerini (1984), Navathe *et al* (1984), Elmasri & Navathe (1984), Yao *et al* (1982) and Navathe *et al* (1986) divide the logical database design into the following four phases:

1 Requirements analysis.
2 View modelling.
3 View integration.
4 Schema structuring and optimization.

The main difference between this approach and the ordinary logical database design approach (also known as the global approach, McFadden 1985) is that the latter does not include the view modelling and view integration phases, the GCS being produced directly from the requirements analysis phase. Whilst a database design tool may help the designer, knowledge of the overall semantics of the application area of the organization is still required. The view modelling and view integration approach allows the designer to concentrate on one section of the organization at a time, without being concerned about how this section maps to the GCS.

The view integration process deals with two major tasks:

1 Integrate the views to form the GCS.
2 Identify and resolve all types of conflict.

To integrate the views, the view integrator must make decisions. These could include choosing the view to be integrated next, choosing the next object from the current view being integrated and considering the effect that the integration of the current view or the current object have on the GCS. To identify and resolve conflicts, the view integrator must have prior knowledge of all the types of conflicts which may take place. Some of these conflicts can be automatically resolved by the view integrator, but others may need the designer's intervention.

The desire by organizations to centralize their databases means that the need to integrate live data bases, distributed or otherwise, is now a necessity. *Database integration* is therefore defined as the process of integrating live databases. This has been studied by

Motro & Buneman (1980), Motro & Buneman (1981), Smith *et al* (1981), Landers & Rosenberg (1982), Mannino (1983), Motro (1983), Dayal and Hwang (1984), Mannino and Effelsberg (1984), Breitbart *et al* (1986), De Souza (1986), Czedo & Embley (1987), Motro (1987), Marinos *et al* (1988) and Marinos and Papazoglou (1988).

Smith *et al* (1981), describe how databases based on different models can be transformed into one unique model and then integrated. Breitbart *et al* (1986) discuss the integration of a distributed heterogeneous database system. Marinos and Papazoglou (1988) discuss the problem of providing a global view for a collection of independent heterogeneous databases. Mannino (1983) and Mannino & Effelsberg (1984) discuss the matching of objects of the schemas describing the local database to be integrated. Mannino & Effelsberg (1984) discuss attribute and entity matching and, to a certain extent, this approach can be related to that of Elmasri and Navathe (1984). De Souza (1986) describes an integration methodology which applies a mathematical function, based on fuzzy set theory (Kaufmann 1975 and Negoita & Ralescu 1975), to determine the resemblance between two schemas modelled in a schema definition facility presented in Saha (1983) and Stocker & Cantie (1983).

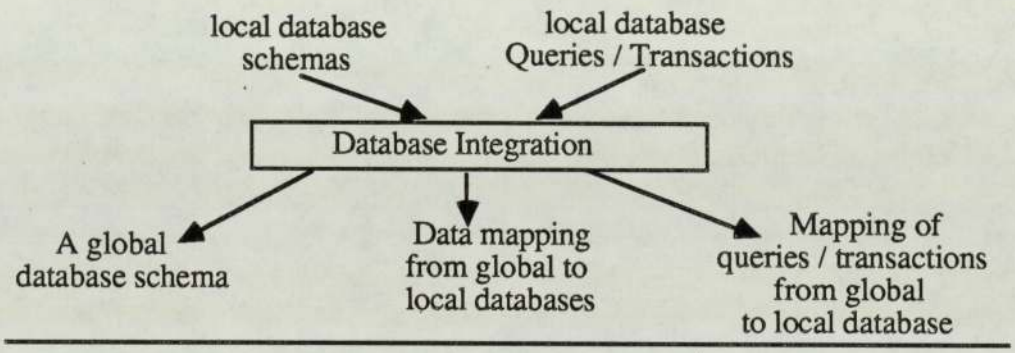


Fig. 2.1 Database Integration (Batini *et al* 1986)

Although database integration shares the same basic concept as view integration for conceptual schema development, it presents a number of other problems. In the case where the databases to be integrated are based on different SDMs, the first task of the designer or integrator, if possible, is to map all the different data models to one chosen data model before the integration process is initiated. Where the databases have different DBMSs, a target DBMS must be established which can carry out equivalent operations and integrity control, so that the resultant integrated global DBMS and schema would represent all the semantics of all the original databases, their data and all the expected transactions and integrity control. A general framework of database integration is shown in Fig. 2.1.

2.2 Database design tools

Database design tools have been proposed and implemented for either or both of the database design phases. However, the majority of these tools have been concerned with the logical design phase. There is much published research concerning database design tools. There is a considerable diversity of approaches developed and of data models used. Some of these tools are surveyed in Buchanan (1979), Chen (1982), Scheneider & Wasserman (1982), Olle *et al* (1982), Chilson & Kudlac (1983), Navathe (1985) and Avison & Fitzgerald (1989). In this survey, only the research which mentions view integration is discussed, even if it is only looked at as part of their future research. This section does not include papers directly involved in view integration research as these are discussed in detail in the following sections.

The expert systems approach for the development of the schema, has been studied by some researchers. Bouzeghoub & Gardarin (1984 and 1985) propose an expert system which gets its input from natural language statements, and produces a knowledge base for a DBMS. This system, which is called SECSI, handles data abstraction (Smith & Smith 1977), and some integrity constraints. Kersten (1987) propose an expert system called ACME, which also accepts natural language statements as its input, and the conceptual schema is then produced by an interactive dialogue with the designer. Choobineh *et al* (1988) propose an expert system which creates an E-R diagram by analysing a collection of *forms*. Forms are the most widely used formal communication objects in most organizations. It seems natural, therefore, that the collection of an organization's forms be a primary input to the database design process. A form is therefore any structured collection of variables which are appropriately formatted to support data entry or retrieval, Choobineh *et al* (1988). Another example of such tools is presented in Laender (1984).

Some of the tools proposed are based on producing a schema represented in an SDM, from the product of a data analysis or systems analysis phase, which might, for example, be based on Gane & Sarson (1979). Examples of these tools include: DATADICT by Joo *et al* (1984), IRMA by Curtice (1984), ELKA by Gonxalex-Sustaeta (1986), and ADD by Berman (1986).

The majority of all the recent research on the production of database design tools has been based on the ERM. Most of the papers presenting these tools usually present a variation of the original ERM, or an extension to it, and a proposal for an interactive tool. These tools are reported in Chan & Lockousky (1980), Sakai *et al* (1983), Atzeni & Carboni (1983), Batini *et al* (1984), Ferrara & Batini (1984), Meyer & Doughty (1984), Massimo & Batini (1984), Reiner *et al* (1984), Jiang & Chin (1984), Albano & Orsini (1985),

Bracchi *et al* (1985), Hawryszkiewycz (1985), Ferrara (1985a), Roesner (1985), Antonellis & Di Leva (1985), Batini & Di Battista (1988) and Shoval *et al* (1988).

Some researchers have developed their tools to achieve a relational schema directly. The input to such tools can be from a semantic data model or directly in the form of relations supplied by the designer. Examples of these include Bragger *et al* (1984), Bjornerstedt (1984) and Leung & Nijssen (1988).

Research about automating database design is far from mature and it is likely that current and future research will concentrate on the following:

1 The full automation of all phases of database design.
2 The enhancement of semantic data modelling.
3 The use of graphical interfaces to the tools.

2.3 The dependencies view integration approach

In order to create a relational schema free from update anomalies, researchers have studied the dependencies between the data elements of the schema. These dependencies are discussed in Codd (1971 and 1972), Date (1981), Kent (1983) and Ullman (1982). These dependencies include:

1 Functional dependencies.
2 Union functional dependencies.
3 Inclusion dependencies.
4 Exclusion dependencies.
5 Multivalued dependencies.

Using the dependencies between data elements, in particular the functional dependencies, to achieve view integration, is called the *dependencies view integration* approach. Dependencies view integration research is discussed in Bernstein (1976a and 1976b), Vetter (1977), Biskup *et al* (1979), Melkanoff & Zaniolo (1980), Beeri *et al* (1981), Al-Fedeghi & Scheuermann (1981), Rissanan (1982), Casanova & Vidal (1983), Convent (1986) and Biskup & Convent (1986).

Bernstein *et al* (1975), Bernstein (1976a and 1976b) and Beeri *et al* (1979) show how third normal form relations are achieved from first normal form relations. Al-Fedeghi & Scheueremann (1981) studied the use of functional dependencies to integrate a group of relations. Beeri (1981), Zaniolo & Melkanoff (1981), Rissanan (1982) and Jajodia & Springsteel (1983) used the data dependencies to prove that two relational schematas are identical, equivalent or different. Casanova & Vidal (1983) studied the use of inclusion

dependencies to detect subset and superset relations, exclusion dependencies to detect disjoint relations, union functional dependencies to detect synonyms and homonyms, and functional dependencies to achieve the optimization of the GCS. Melkanoff & Zaniolo (1981), Ling (1985a and 1985b) and Makowsky *et al* (1986) have applied the dependencies theory to achieve normalization in the ERM schema. Biskup & Convent (1986) used the functional, inclusion and exclusion dependencies, but give no specific uses for these dependencies in achieving view integration.

As Convent (1986) shows, although attempts have been made to achieve dependencies view integration, more research is needed if this approach to view integration is to be used satisfactorily. Further, since all the research has so far been theoretical, it is not possible to suggest how these approaches will work when implemented. Further, there is no evidence in this literature of these approaches being applied to any organization of a practical size. Dependencies view integration can be criticized for:

1 The difficulty in identifying all the possible dependencies.

2 The time taken in identifying all the possible dependencies.

3 The involvement of the user and the designer in the view modelling and view integration stages becomes complicated, because the mathematical formats of the dependencies theory are difficult to understand.

4 The dependencies between different data elements could interact in the most complex of ways, as Casanova *et al* (1982), Casanova & Vidal (1983) and Chandra & Vardi (1985) show.

5 The dependencies theory is only applicable if the views are modelled using the relational model, or the GCS is to be mapped onto a relational database.

6 The dependencies theory cannot be used for the integration of existing databases, because relational databases should be free from dependencies.

2.4 The object view integration approach

A large organization consists of many views. Each view has one or more objects, where an *object* is defined as the smallest complete element of the model concerned. Therefore, in the relational model, an object could be a relation, and in the ERM, an object could be an E-R. In view modelling, it is likely that common semantics between different views are modelled for each of these views. Common semantics between views can be one or more attributes, one or more entities, or one or more E-Rs. The views v1 to v6 of Fig. 2.2 all have common semantics. The size of the common semantics is determined by the following:

1 The flexibility of the SDM definition.

2 The understanding by the users and the designer of the structure of these semantics.

3 The level of interaction between these users.

4 The view modelling approach.

Although the designer may identify all the views in the organization, it is possible that some of the semantics may be omitted. This loss of semantics can be blamed on the weakness of the view modelling approach. Consider for example, the application of Fig. 2.2, which has been broken down by the designer into views v1, v2, v3, v4, v5 and v6. Although these views completely represent the semantics of their users, their integration would not produce a complete GCS. The omission of some semantics is caused by failing to identify the shaded areas M1, M2, M3, M4 and M5. This loss of semantics cannot be directly identified by the view integrator.

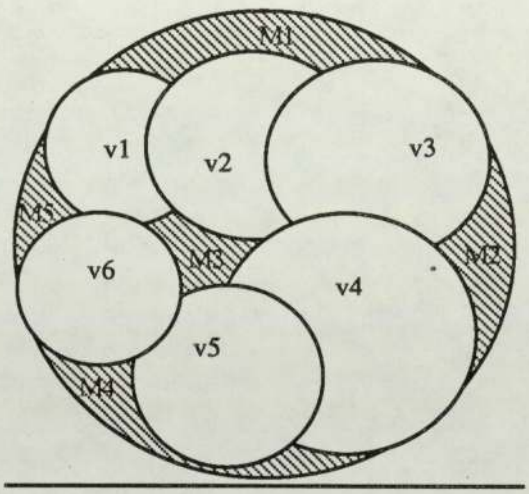


Fig. 2.2 Missing and common semantics in view modelling

No computer system can totally replace the human expert. The database designer has the knowledge, experience and intelligence which cannot in its entirety be included in a view integrator. It is claimed that the average expert knows tens of thousands of rules of thumb, which he can apply in solving the problem at hand. The development of the GCS either by the global approach or by the view modelling and view integration approach, requires the designer to apply some rules of thumb which cannot be included in a view integrator.

There are many types of conflicts which may arise in view integration. These conflicts range from ordinary naming conflicts to completely different structures representing the same semantics. Some of these conflicts can be identified and resolved automatically by the view integrator, whilst other conflicts may not even be identified. Although it would seem feasible for all conflicts to be identified and resolved by the view integrator automatically, this is governed by the following factors:

1 The preciseness of the formal definition of the SDM. A precise and mathematically defined SDM reduces the chance of representing the same semantics differently in view modelling and consequently reduces the number of conflicts. Further, such an SDM allows the view integrator to be designed in a way that it has the maximum built-in conflicts identification and resolutions procedures.

2 The semantic richness of the SDM. The semantic richness of the SDM used to model the views is directly proportional to the amount of semantics it can capture from the application area. A semantically rich SDM can accommodate many types of object and their semantic connections. This allows more room for misrepresentation during view modelling, and makes it very difficult to anticipate all the possible conflicts.

3 The strictness of the view modelling rules. View modelling rules can be made to cover aspects of view modelling such as the naming of objects, the size of the views, the preference of structures, and so on. Strict view modelling rules can be regarded as a hindrance to the designer and must therefore be minimised. Further, the need for strict view modelling rules indicate a weakness on the side of the view integrator.

4 The number of designers involved in view modelling. The more designers involved in view modelling, the more room there is for modelling the same semantics in different ways.

5 The expertise of the designer(s). Extracting knowledge of the application area from the users demands a high level of expertise. This, combined with the need to understand the SDM and using it to model this knowledge, emphasizes the need for skilled designers. Although the view integrator might carry out most of the view integration process, the designer would still be required to resolve some conflicts.

6 Computer software technology. The challenge in view integration is to develop a view integrator which can identify and resolve all conflicts automatically. To do this, the view integrator must learn from the history of conflicts, and consequently enhancing its capabilities. Hopes of expert systems achieving this have not been fulfilled, De Reit (1986).

2.5 The influence of the SDM on view integration

Research in semantic data modelling is extensive and new models, as well as extensions to old models, are reported on a continuous basis. However, there is neither a general

agreement on which is the best data model (Brodie 1984) nor is there presently a data model mapper which can successfully transform all SDMs into a unified whole. Therefore, the development of view integrators is likely to continue to be SDM specific. The ideal SDM has to satisfy many qualities which include semantic richness, dynamic modelling, facilities to specify constraints, implementability, ease of use, and freedom from physical considerations. All these factors directly influence the design of a view integrator. The more comprehensive the SDM, then the more complex the view integrator would be to develop.

The SDMs used in view integration for schema development are shown in Table 2.1. It is noticeable from a study of this table that the original ERM (or variations of it) have been used in most of these methodologies and that the most popular extensions to ERM are in the form of *data abstraction* techniques, especially *generalisation* and *aggregation*, as presented in Smith & Smith (1977).

View integration methodology	Semantic Data Model
1 Raver & Hubbard (1977)	A variation of semantic networks and FDM, Baker (1974).
2 Elmasri & Wiederhold (1979)	The structural Model, Wiederhold & Elmasri (1979).
3 Yao et al (1982)	The Functional Data Model, Shipman (1981).
4 Navathe & Gadgil (1982)	The N-S model, Navathe & Schkolonick (1978).
5 Elmasri & Navathe (1984)	The category concept: an extension of ERM Weeldreyer (1980) and Elmasri *et al* (1985)
6 Batini & Lenzerini (1984)	An extension of ERM, Chen (1976) and Batini & Lenzerini (1984).
7 Navathe et al (1986)	The category concept: an extension of ERM Weeldreyer (1980) and Elmasri *et al* (1985)

Table 2.1 View integration methodologies and their Semantic Data Models

2.6 Sequence of view integration

The number and size of views resulting from the view modelling phase for a large organization depends on the size and type of activities of this organization. These views and their objects must be chosen for integration in a given sequence. In ERM, for

example, each view can consist of one or more E-Rs. The choice of the next view, and consequently the next E-R within the view for integration, may affect the number and type of conflicts which could arise in view integration. The choice of the next E-R of the current view being integrated has not been studied. However, two approaches have been proposed for the choice of the next view for integration. These are binary view integration approach and n-ary view integration approach.

2.6.1 The binary view integration approach

Binary view integration can be defined as the incremental enhancement of the GCS by the integration of one view at a time (see Fig. 2.3). All the view integration methodologies which are based on the object view integration approach are based on binary view integration. With the exception of Navathe & Gadgil (1982), none of the other methodologies discusses how the next view should be chosen for integration. Navathe & Gadgil classify their views into sets prior to integration, depending on the type of match between these views. Matching the views results in sets of identical, equivalent or different views and each of these sets is then integrated separately.

The factors which can be considered relevant to the decision on which a view should be the next for integration are:

1 The size of the view. It is possible to assign higher priorities to larger views. Therefore in ERM, for example, views with more entities, attributes, or E-Rs, may be given a higher priority.

2 The level of the view in the organization. One way of modelling the views is based on the organization hierarchy. Therefore, view integration can be carried out in either a bottom up or top down manner.

The main advantages of the binary view integration approach are:

1 The relative ease of the integration algorithm.

2 The facility to focus attention on the resolution of the conflicts found in the current view being integrated.

3 The resultant schema at the end of every integration phase is complete. This may be required in the case of huge applications which could take months or years to model.

However, the binary view integration approach has the following disadvantages:

1 Since the views are modelled independently and integrated separately, the same conflicts might arise with the integration of each view.

2 If the organization is huge and the views are considered in turn, the view integration process could take a long time.

2.6.2 The n-ary view integration approach

N-ary view integration can be defined as the 'simultaneous' integration of all the views. A possible model of the approach is shown in Fig. 2.4. N-ary view integration appears to have the following advantages:

1 Speed of integration.

2 Once a conflict is encountered, the same resolution could be applied to all the occurrences of this conflict in all the views.

3 The resolution of a conflict could be influenced by the semantics in other views, and hence a better judgement of conflict resolution might be achieved.

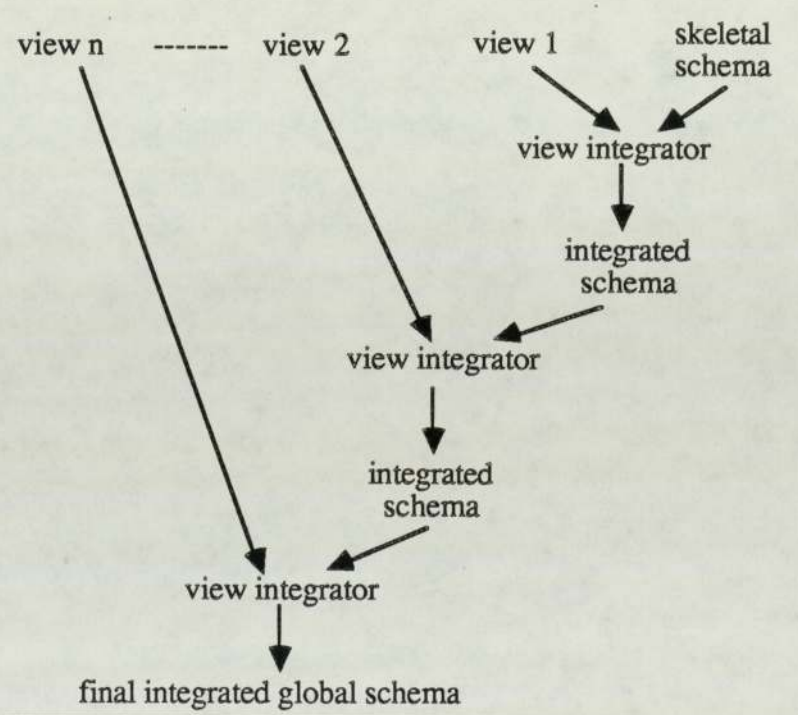


Fig. 2.3 Binary view integration

The expected disadvantages to the approach are:

1 The complexity of the integration algorithm.

2 The partial integration of the GCS cannot be achieved, unless a chosen list of views is integrated separately.

3 The inability to concentrate on one view at a time, requiring the designer to have global knowledge of the application area.

2.7 Components of a view integration methodology

The view integration process is expected to carry out many activities such as view selection, object identification, conflict identification, conflict resolution, GCS creation, and, possibly, schema restructuring and optimisation. The division of these activities into different phases in view integration has been viewed differently in the various view integration methodologies. Navathe and Gadgil (1982) laid the foundations for the definition of view integration research of the object integration type. Whilst the other methodologies are not identical to it, most of them follow similar patterns to Navathe and Gadgil's approach, and hence the schematic diagram showing the components of a view integration methodology shown in Navathe & Gadgil, is presented here (Fig. 2.5).

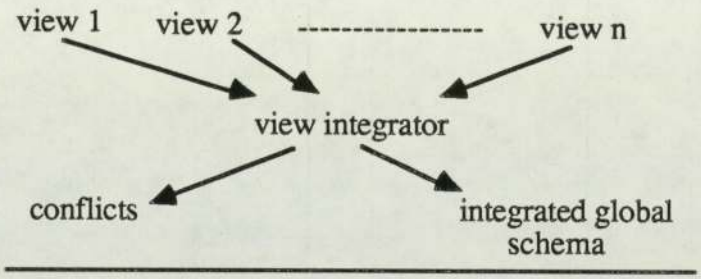


Fig. 2.4 N-ary view integration

2.7.1 The enterprise view

The *enterprise view* (sometimes referred to as the skeletal schema) is defined by Navathe & Gadgil (1982) as the nucleus for the development of the global view, where the enterprise view describes the basic entities and associations of the organization. The enterprise view is usually the top level abstract view of the organization, and presents a basic schema of the organization in the form of a number of major entities and their relationships (assuming ERM as the SDM).

When considering a very large organization for view modelling, it is difficult to decide what the enterprise view should be, as it is not possible to identify a restricted number of entities and relationships as the starting point of the view modelling phase. A natural way of modelling a large organization is by breaking it down into a hierarchical set of views. If this method of view modelling is followed, then the enterprise view would be the top most view of the hierarchical tree. When all the views of the organization are modelled, the contents of the enterprise view would be included in the other views. Therefore, even if the enterprise view is modelled, its integration would add nothing to the contents of the final GCS.

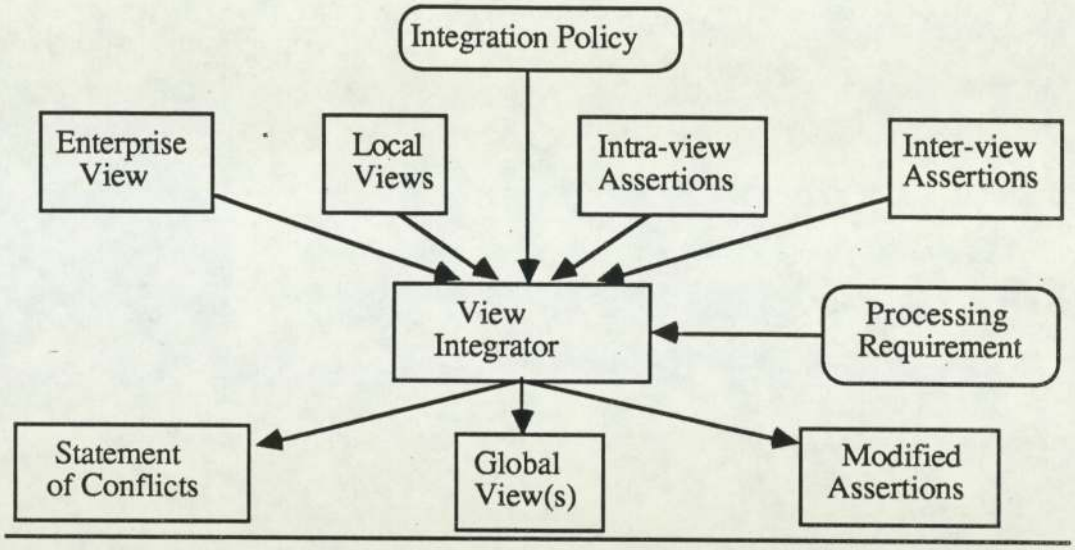


Fig. 2.5 A model for view integration (Navathe & Gadgil, 1982)

The enterprise view is used by Navathe & Gadgil (1982) as the initial GCS with which the views are integrated. Navathe *et al* (1984), Batini & Lenzerini (1984) and Navathe *et al* (1986) all claim to follow the same approach.

2.7.2 Assertions in view integration

It is possible that both modelling assertions and integration assertions are simultaneously required to achieve the correct and complete GCS. Navathe & Gadgil, for example, consider both modelling assertions and integration assertions. An example of modelling assertions to complement the N-S model (Navathe & Schkolnick 1978) used to model views of a hospital as presented in Navathe & Gadgil (1982), is:

'Procedures performed by the service instance called Hospital-trust are always performed free.'

This modelling assertion is represented in their especially developed assertion language as follows:

S Service-Name = Hospital-trust &
< S, P > E PROCEDURE IDENTIFIER &
< P, C, r > E PERFORMED-FREE

where:

S, P, C, r are respectively, instances of entity types SERVICE, PROCEDURE, SCHEDULE and PERSONNEL.

Although certain SDMs are more suitable for particular applications (in the same way that certain programming languages are more suitable for special applications), in general, SDMs should be easy to use and semantically rich. On examination of the N-S model, it clearly fails to meet these two requirements. The model is not easy to use because the

modelling assertions are difficult to identify and model, and it is not semantically rich because it requires many complementary semantics. Developing a view integrator is a complex task, but developing one that is expected to understand complex assertions is even more complex. Since these assertions are modelled for the individual views, their integration would also produce conflicts.

A 'good' view integrator uses the definition of the SDM and the semantics of the views in order to create the GCS. The factors discussed in section 2.4 make the conflicts difficult to anticipate and formalise, and therefore resolve automatically. Therefore either integration assertions or interactive integration (or both) is needed. However, the objective in view integration is to reduce these and thereby increase the level of automation of the view integration process.

An example of integration assertions in Navathe & Gadgil is as follows:

1 view 2 = RSTR [view 1].
2 Preferred view = view 2.

The first integration assertion declares to the view integrator that view 1 and view 2 are RESTRUCTURALLY equivalent. The second integration assertion declares that view 2 is to be 'preferred' in view integration. Therefore, if a conflict took place, the semantics of view 2 would override the semantics of some other view. It is not the intention here to go into the details of this integration assertion language. For the designer to supply these assertions, he has to study the organization, and do some of the activities which would normally be expected from the view integrator. To expect the designer to arrange the views in accordance with their priorities and to decide the type of match between views, is defeating the objective of view integration. Identifying the conflicts, matching the views and proposing resolutions are all tasks that would be expected of the view integrator.

Interactive view integration, on the other hand, allows the view integrator to carry out most of the view integration tasks. However, the designer is expected to intervene in cases of conflicts for which the view integrator has no predefined resolutions or it cannot choose between a number of possible resolutions. Interactive intervention by the designer to resolve a conflict can therefore be regarded as a form of integration assertion. However, the designer is not expected to anticipate these conflicts nor to decide beforehand the situations which would produce these conflicts.

2.7.3 Conflicts in view integration

Conflict identification and resolution is one of the most important functions of a view integrator, yet it has been largely ignored in the literature. This is true of all the

methodologies, whether they operate in a manual or automatic mode. In any case, if the methods of conflict identification and resolution are formalized enough to operate in a manual mode, they can be programmed to operate in an automatic mode. The factors influencing the frequency and type of conflict are discussed in section 2.4.

The quality of the view integrator with regards to conflict identification and resolution, is measured according to the following:

1 The total number of conflicts that it can identify, in relation to the total number of possible conflicts which can occur.
2 The total number of conflicts that it can automatically resolve.
3 The degree of correctness of the resolutions that it automatically carries out.
4 The ability to learn from previous conflicts.

Conflicts in view integration can either be naming conflicts or structural conflicts. These types of conflicts are discussed below.

2.7.3.1 Naming conflicts

Synonym and homonym naming conflicts are difficult to identify by the view integrator. They are normally caused by linguistic misinterpretation, misspelling or are abbreviations. In certain application areas, it is possible that a natural language system which is linked to an on-line English dictionary, may be used to identify these conflicts. However, in a technical or scientific environment, this approach would not prove helpful unless a special data dictionary is created to contain all the technical jargon.

Although all the view integration methodologies define synonyms and homonyms as integration conflicts, none present a method for their identification. Batini & Lenzerini (1984) comment on the problem of identifying synonyms by describing the terms *concept likeness* and *concept unlikeness*, where a 'concept' means an 'object'. The neighbours of a given concept are matched, and, based on the similarity of their 'neighbours', the two concepts are declared to be either alike (concept likeness) or different (concept unlikeness). Neighbours of a given object are all the other objects associated with it. For example, the neighbours of an entity are its attributes and its relationships. Batini & Lenzerini fall short of presenting a method of carrying out the concept likeness and concept unlikeness tests. In any case, if the neighbours of the objects concerned are expected to be identical in order to declare that the two objects have a concept likeness, then it is very unlikely that such a situation would ever be met in a real application. Further, concept unlikeness could be due to the two objects differing on one neighbour, or on all their neighbours, and therefore almost all objects will have concept unlikeness.

2.7.3.2 Structural conflicts

In an SDM, a *structure* can be one or more objects with common semantic connections. In ERM, for example, a structure can be an attribute, an entity, a relationship, an E-R, or a group of E-Rs with common semantic connections. An object is therefore the smallest structure. A structural conflict can occur as follows:

1 The same semantics are modelled in different structures.

2 Different semantics are modelled using the same structure.

If the structures causing the structural conflict are individual objects, then there are two possibilities:

i The same semantics are represented as two different objects. This would be a genuine structural conflict.

ii The two structures suffer from either a synonym or a homonym naming conflict.

If the structures causing the structural conflicts are individual objects, and the conflict is a genuine structural conflict, then only one of the objects can be chosen for inclusion in the GCS. The semantics contained in the object not chosen must be transferred to that object which was chosen. *Object transformation* from one type to another was considered by Batini & Lenzerini (1984). However, only a method of transforming an entity to an attribute (and vice versa) is presented in that paper. Transforming other ERM object types was not considered. Further, they do not give reasons for favouring one object type over another.

Object transformation from one type to another can be formalized and automated for most SDMs. However, in view integration there is difficulty in deciding which object should be transformed, and why. If these objects are already modelled in the GCS, then they would have their own semantic connections (neighbours). The effect of the transformation on the GCS must be considered so that the GCS after the transformation is correct. Whilst the object transformation approach for entities and attributes proposed by Batini & Lenzerini is a contribution towards the resolution of structural conflicts, more research is still needed in this area.

If the structures causing the conflicts are individual objects, and the conflict is a naming conflict, then the resolution is achieved by changing the name of one or both of the objects. Although the view integrator can be designed to identify some of these conflicts, it is not possible to make a firm conclusion that it is a naming conflict and not a genuine structural conflict. In ERM, examples of this kind of conflict can be: entity name existing as a relationship name, a relationship name existing as an attribute name, an entity name

existing as an attribute name, and so on. These kinds of conflicts are referred to here as COT conflicts, regardless whether the conflict between the two structures is a naming conflict or a structural conflict.

COT conflicts are caused by a number of factors:

1 The designer's misunderstanding of the SDM.
2 The flexibility of the SDM in representing the same semantics in different ways.
3 Genuine different views by different users for the same semantics.
4 Genuine naming conflicts.

Where the structures are individual objects, the same semantics modelled in different structures are more identifiable than when different semantics are modelled using the same structure.

Sometimes, structural conflicts can take place between larger structures. Any structure consisting of more than one object, can be regarded as a large structure. Therefore, in ERM for example, any structure consisting of two or more E-Rs with common semantic connections, is a large structure. A structural conflict can exist between two large structures or between one object and a large structure. An example of a structural conflict between larger structures is an E-R structure which is also modelled as two or more E-R structures. Structural conflicts between large structures, can usually be indirectly identified by COT conflicts analysis between their corresponding smaller structures, unless naming conflicts affect all corresponding objects of the two large structures. These structural conflicts are even more difficult to identify, and consequently resolve by the view integrator and, as can be expected, this issue has also been disregarded in the view integration literature.

2.7.4 The Global Conceptual Schema

The main objective of view integration is to achieve a complete and correct GCS through integrating the semantics of the views. The completeness of the GCS largely depends on the following:

1 The quality of the view modelling approach. A good view modelling approach would enable the designer to identify and model all the views of the organization.
2 The semantic richness of the SDM. A semantically rich SDM can be used to represent all the necessary semantics of the application area.

3 The ability by the view integrator to transport all the semantics from the views to the GCS, which is influenced in turn by:.

   a) The level of formal definition of the SDM.

   b) The percentage of the conflicts which can be identified by the view integrator, from all the possible conflicts which could arise in view integration.

   c) The quality of the resolutions to the conflicts identified, either by the view integrator or by the designer.

   d) The level of compromise possible between the views, in case of conflicts.

At the end of the view integration process, the GCS is expected to be free from the following:

1 Redundant objects. These are caused by synonyms and homonyms naming conflicts, and COT conflicts.

2 Redundant structures. These are caused by structural conflicts concerning larger structures, and COT conflicts.

3 Missing semantics. These are caused by one of the following:

   a) Lack of an established view modelling approach.

   b) Inability by the view integrator to transport all the semantics from the views to the GCS.

   c) Lack of schema completeness testing procedures in the view integrator.

4 Inconsistencies. These are mainly caused by the existence of COT conflicts in the GCS.

Raver & Hubbard (1977) do not propose any recommendations regarding the quality of the GCS. Therefore, it is not known how their GCS is formulated. Navathe & Gadgil (1982) do not give a clear decision regarding the format of the GCS. In response to a conflict which occurs between two views, they propose one of the following:

1 Favour the semantics of one view over the semantics of the other view. This preference is predetermined by one of their integration assertions (preferred view = view view name). The danger with this approach is that it is possible that the semantics of the unpreferred view may not exist in any of the other views and is therefore lost.

2 Include both views in the GCS and thus cause redundant semantics in the GCS. However, they do recommend mapping rules so that the data for each of the two views can be obtained from the GCS, without any redundancy. This approach is also followed by Elmasri & Wiederhold (1979). The

problem with this approach is that almost all the views will eventually coexist in the GCS, and thus will require many mapping rules. This can lead to a very complex GCS, which is difficult to test for correctness and completeness. The most critical drawback of this approach is that the GCS would contain information (mapping rules) which is inconsistent with the original definition of the SDM. This is also an indication that the SDM is not rich enough to accommodate the global semantics of the organization.

Yao *et al* (1982) do not discuss the format of the GCS nor how it is affected by the presence of conflicts. The GCS is achieved by the designer, who uses commands such as REMOVE and MERGE to formulate the GCS. The major contribution of this methodology is in providing a specially developed language called TASL to model the dynamics of the organization, and consequently to test the completeness of the GCS. Transactions which cannot be satisfied indicate missing semantics in the GCS and transactions which produce inconsistent results indicate conflicts in the GCS. The processing of these transactions is achieved manually.

Batini & Lenzerini (1984) recommend that only one GCS is achieved as a result of view integration. Therefore, a compromise must be made between conflicting structures, whenever they exist during integration. The resultant GCS must accommodate the semantics from both views. A conflict is resolved either by creating intermediate (cushion) views or favouring the structure of one view over that of another. However, in the absence of COT conflicts analysis, it is difficult to see how the methodology of Batini & Lenzerini would work in such situations.

2.8 Phases and activities in view integration

The activities of a view integrator range from reading the views to the production of a complete and correct GCS. These activities must be arranged and achieved in a predetermined order. Such an order would probably depend on the importance given to each activity in the view integration process. The following factors could influence the arrangement of these activities:

1 Achieving the most correct GCS.
2 Minimising the involvement of the user and the designer.
3 Speeding the view integration process.
4 Producing partially integrated schemas.
5 Identifying the most number of conflicts.
6 Producing a correct GCS at all phases of view integration.

A compromise must eventually be made so as to achieve the best possible output from view integration.

A view integrator must be SDM specific. Therefore, it must interpret the definition of the SDM for which it is built. In the case of the methodologies which also require modelling assertions and integration assertions, the view integrator must also understand these assertions.

A general outline of the activities of a view integrator is as follows:

1 Read and understand the semantics of the views, modelled using a particular SDM.

2 Create the GCS by transferring the semantics of the views to it.

3 Identify all possible conflicts which may exist when integrating the views.

4 Resolve the conflicts for which it has built-in resolutions, and report others to the designer.

5 Accept commands from the designer, relating to the resolution of conflicts - these commands can either be interactive or in the form of integration assertions.

6 Ensure the completeness of the resultant GCS.

Most methodologies present a proposal of achieving 1 and 2 above. Only Navathe *et al* (1984) present a very limited analysis of conflicts' identification and their possible resolutions (3 and 4 above). Point 6 above, which is concerned with the completeness of the resultant GCS, is only considered in Yao *et al* (1982). Finally, since none of the methodologies has been implemented, no judgement can be made regarding point 5. One way round the interactive involvement by the designer in conflicts resolution, is through the modelling of the integration assertions, which act as tailor-made decisions to resolve any anticipated conflicts.

Raver & Hubbard (1977) and Elmasri & Wiederhold (1979), do not propose any phases for view integration. Navathe & Gadgil (1982), divide view integration into three phases: pre-integration, integration and post-integration. The first phase classifies views into sets, depending on the type of match between these views. The pre-integration phase therefore divides views into sets of identical, equivalent or single (different) views. However, unless views are made up of one object each, there is no way that two views can be stated as equivalent. Moreover, finding identical views would be very rare. The next phase in Navathe & Gadgil is to integrate these sets of views. Identical views are represented once in the GCS. Equivalent views are grouped into possible intermediate views, or reported to the designer. The grouping of these equivalent views, would eventually produce many sets of intermediate views, and the process of view integration would ultimately break

down. The final phase is to integrate the intermediate views, but no detailed procedure is given for this.

Yao *et al* (1982) base their approach on the functional model. It divides view integration into three phases. The first is to merge nodes with the same value, the second is to merge nodes which are subsets of other nodes, and the third phase is to remove redundant functions. This third phase also initiates the integration of the corresponding transactions modelled in a language called TASL. These three phases are steps aimed at conflict detection and resolution, and do not provide an outline of the general phases of a view integration methodology.

Batini & Lenzerini (1984) divide their methodology into three phases: the conflict analysis phase, the merging phase, and the schema enrichment and restructuring phase. The conflicts analysis phase is more of a pre-integration phase, whilst the schema enrichment and restructuring phases is more of a post-integration phase. The conflict analysis phase is aimed at identifying the structural conflicts as well as the naming conflicts. The schema enrichment and restructuring phase is aimed at ensuring that the schema is correct and complete. However, though the methodology is complete as far as giving the number of view integration tasks to be achieved in apparently different phases, none of these tasks or phases are discussed in detail.

Navathe *et al* (1984), Elmasri & Navathe (1984) and Navathe *et al* (1986), are all descriptions of different parts and activities of the same methodology. The actual conflicts analysis phase is carried out separately for the object classes (entities) and the E-Rs. This approach attempts to prepare correct definition of entities, so that E-Rs involving these entities can be matched and integrated. This can be seen as a duplication of effort since the entities can directly be integrated as part of the integration of E-Rs.

2.9 Conclusions

This chapter has reviewed the integration methodologies proposed in the literature. It has been shown that view integration can either be achieved using the dependencies integration approach, based on relational theory, or the object integration approach, which uses the semantics of the objects and structures to form the user views. The chapter also described the integration principles and discussed the status of each of the proposed integration methodologies in relation to each of these principles.

Some concluding statements made in some of the most prominent papers regarding integration methodologies are presented below. These statements describe the 'state of the art' as well as show the direction of the integration research.

'The methodology for view integration has to be extended to more general cases, and we are pursuing further research in this area' (Elmasri & Wiederhold 1979).

'A database design system that implements these algorithms for view modelling and integration is currently being developed' (Yao *et al* 1982).

'To our knowledge very little work has been done on the view integration problem' (Navathe & Gadgil 1982).

'From the detailed discussion of object and connector matching it is clear that the task of matching and integrating views represented by objects and connectors is non trivial' (Navathe & Gadgil 1982).

'In the conflict resolution area, considerable human involvement is necessary' (Navathe & Gadgil 1982).

'The whole area of data integration is not yet at a mature stage' ..... 'Research is needed to verify its application [the view integration methodology] to different models' (Batini & Lenzerini 1984).

'Future research will be devoted to take into account the dynamic aspects of the methodology' (Batini & Lenzerini 1984).

'We believe that the main goal of view integration is to aid the designer in identifying possible ways of integration and to help him resolve inconsistencies while working with large real life problems' (Elmasri & Navathe 1984).

'Much work remains to be done in order to develop a comprehensive and usable tool to aid in view integration. The following are just some of the problems to be solved: ..., study the order in which matching, merging and modification may be applied' (Elmasri & Navathe 1984).

'Our overriding philosophy behind view integration is one of arriving at a compromise structure. When presented with alternative views of the same situation, we accept the more general view' (Navathe *et al* 1984).

'Among the areas we wish to address in our future research are: development of methods for specification and representation of intra-view and inter-view assertions, ...' (Navathe *et al* 1984).

'We do not use heuristics because ultimately, the designer decides whether functions are redundant and no simple, reliable measures seem possible' (Mannino & Effelsberg 1984).

'We feel that integration can be accomplished only with interactive design tools' (Navathe & Elmasri 1986).

'Much work remains to be done, however, to develop comprehensive and usable tools to aid view integration. The following have yet to be solved: 1) develop rules to match and integrate two views with more complex interrelationships among object classes and relationship sets. 2) study the order in which matching, integration, and modification may be applied. 3) study the advantages and disadvantages of a binary vs n-ary integration strategy' (Navathe & Elmasri 1986).

'It should be expected that with the growth in size and complexity of conceptual schemata this [integration] will be the object of more research' (De Souza 1986).

'Most of the surveyed methodologies do not provide an algorithmic specification of the integration activities, and they rarely show whether the set of conflicts or the set of transformations considered is complete in some sense' (Batini *et al* 1986).

'..., schema integration is a difficult and complex task' (Batini *et al* 1986).

'...there currently exists no technique attaining integration at the global level, however, we believe that an appropriate extension of the recent studies made in view integration and logical database design for centralized as well as federated databases can contribute much to the solution to this serious problem' (Marinos & Papazoglou 1988).

'The extension of the View Creation System to include view integration is currently under way' (Storey & Goldstein 1988).

Among the issues that have not been researched fully, are therefore:

1 The effect of the SDM on the integration process.
2 The full automation of the integration process.
3 The completeness and convergence of the integration process.
4 The completeness of the integrated schema.
5 The comparison of binary vs n-ary view integration approaches.
6 The development of an n-ary view integration methodology.
7 The complete implementation of a view integrator.
8 The completeness of the analysis of all the possible conflicts, especially COT conflicts.

The next chapter presents a method for matching and integrating E-Rs, entities, relationships and attributes. Chapter 4 presents a method for identifying the synonym naming conflicts and discusses a number of COT conflicts. Chapter 5 presents a view description language which is used to represent the views textually. Chapters 6 and 7 describe the implementations of the binary and the n-ary view integrators. Chapter 8 gives an analysis of the results obtained from integrating a number of views.

CHAPTER 3

E-R INTEGRATION AND CONFLICTS ANALYSIS

3.1 Introduction

This chapter and the following chapters describe a View Integration Methodology (VIM), whereby views modelled in ERM are analysed and integrated to produce a GCS. The emphasis in VIM is placed on the structural semantics of the ERM as well as heuristics to identify and resolve the conflicts which take place during integration. The conflict identification and resolution process is not driven by any form of integration assertion nor does it expect any modelling assertions. The view integrator is therefore expected to identify the conflict, and try to resolve it based on contents of the input views and the definition of ERM. Where a conflict cannot be resolved automatically, the designer is requested to intervene.

This chapter presents a method for integrating E-Rs and gives an analysis of possible conflicts which could arise during the integration of these E-Rs. The conflicts and their resolutions are such that they work in real and general situations, and their methods of resolutions have been successfully implemented in a system called the View Integrator (VI), the prototype implementation of which is described in detail in chapters 6 and 7.

In order to implement the VI in general, and the conflicts identification, analysis and resolution part in particular, a number of factors must be studied. Firstly, an algorithmic approach must be established so that precise rules are used to identify the conflicts. Secondly, as a conflict is identified, a method of resolution must be available which is applicable to the identified conflict. Thirdly, the resolution applied must produce the correct GCS which portrays the requirements of the views being integrated. And finally, the sequence of the conflict analysis must be studied, as in certain situations such sequence has a direct effect on future types of conflicts.

3.2 E-Rs and views in ERM

Representing the semantics of the views using the ERM is achieved by mapping the views to the structure of the ERM. Before a formal definition of the ERM is given (Lien 1980, Ng & Paul 1980, Markowitz 1984 and Chen 1984), let us briefly define the objects which make it. An *entity* is anything which can be distinctly identified in our minds. For example, 'student' and 'department' are entities. An *entity class* is a set of entities of similar structure. For example, student class is the set of all students in a University.

Relationships may exist between entity classes. A *relationship* is a class of objects representing logical associations among entities. For example, 'student studies-in department' is a relationship called 'studies-in' which connects instances of the two entities 'student' and 'department'. The data elements which describe entities and relationships are known as *attributes*. For example, 'student' has the attributes, 'name', 'number', 'course name' and 'year'. The number and type of attributes defining an object are decided by the user in such a way that they should represent domains in the corresponding relation which contain the information needed by the user. The ERM accommodates all the semantics of the organization in the form of a network of E-Rs. Therefore a view or a schema is made up of one or more E-Rs linked in a network.

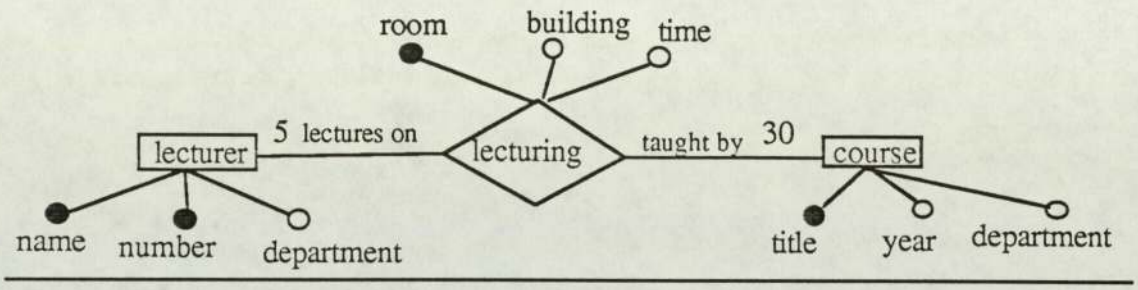


Fig. 3.1 An E-R with roles and attributes

Fig. 3.1 and Fig. 3.2 show two examples of E-Rs. Fig. 3.1 shows a relationship called 'lecturing' between the entities 'lecturer' and 'course'. Both entities have attributes modelled to define them. Some of these are key attributes and the rest are non key attributes. The entity 'lecturer' has the *role* 'lectures on' and the entity 'course' has the role 'taught by'. The role names associated with these entities give more expressive meanings to the role that the entity plays in the relationship. This role does not contradict the relationship name or the entity name, but gives an extra meaning to the entity involvement in the relationship. The modelling of roles is more necessary for the readability of recursive relationships. The *cardinality constraint* of each of the entities is represented by a number or a mnemonic written next to the entity and on the arc connecting the entity to the relationship name. A cardinality constraint places restrictions on the number of instances of one entity class that may be related to an instance of the other entity class involved with it in the same relationship.

Fig. 3.2 is another example of an E-R where a relationship called 'takes' involves entities 'student' and 'course'. This example shows values associated with some of the attributes. Attribute values impose constraints on the attributes with which they are associated. The attribute 'number' of the entity 'student' is restricted to a value ranging from 0001 to 9999. The attribute 'level' of the entity 'course' is restricted to a set of two values 'undergraduate' and 'postgraduate'.

Fig. 3.3 shows an example of a view called 'teaching' containing a number of E-Rs. Chapter 5 discusses in more detail how views are modelled, what sizes they should be and how they are represented in textual form. The example view is used here to show the position of an individual E-R in a view, and show how a group of entities are linked together in a network to form a view of a particular user. See appendix A for more examples of views.

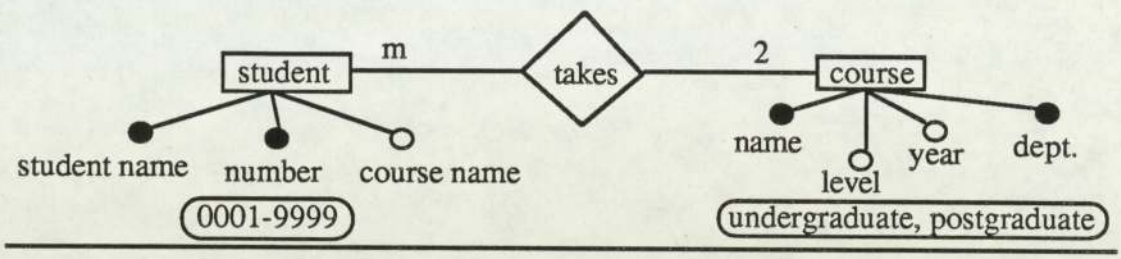


Fig. 3.2 An E-R with attribute values

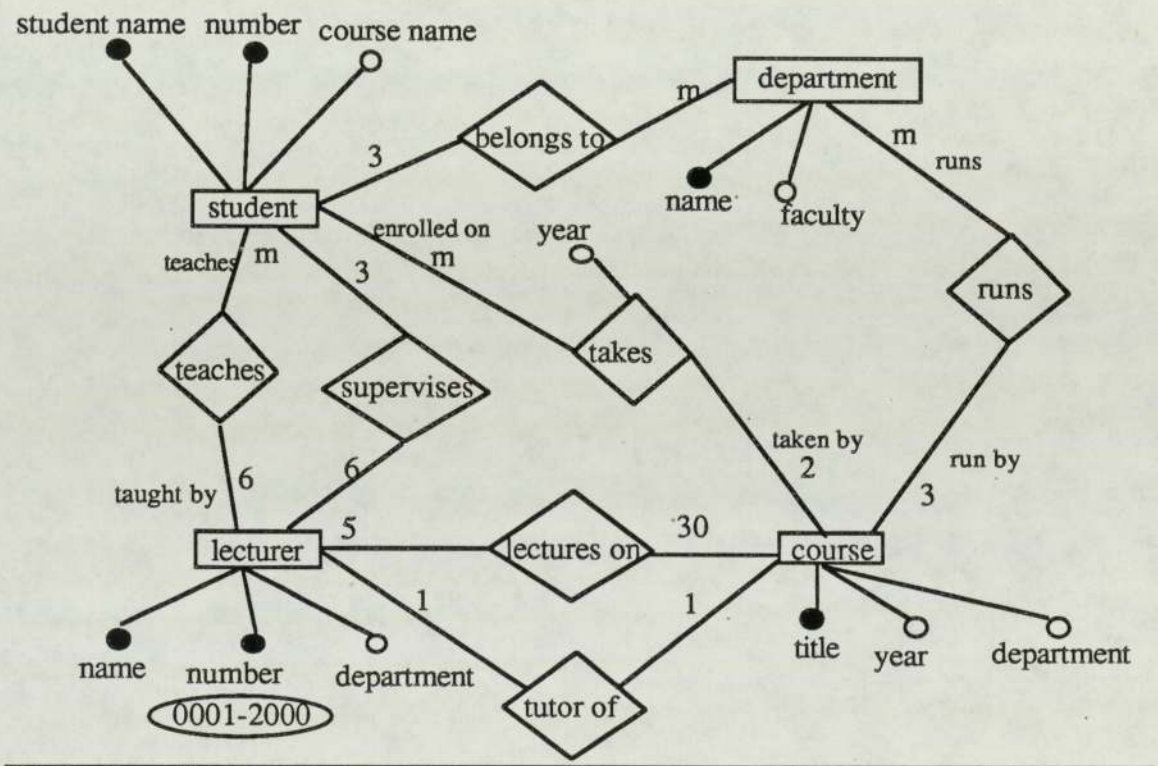


Fig. 3.3 An example of the view teaching

3.3 Integration of E-Rs

This section shows how two E-Rs are matched, have their conflicts (if any) identified and analysed, and then how the two E-Rs are integrated to form a schema made up of the two E-Rs. The decision to use E-Rs as the objects for integration was taken on the basis that integrating E-Rs means that all the other objects in the views are automatically included in

the view integration process. Therefore, if all the E-Rs in the views are integrated, then all the other objects in the views are automatically integrated. There are a number of possible ways to decide on how the sequence of integration should be followed. Chapters 6 and 7 show the sequence in which the views and E-Rs are chosen for integration. This chapter is concerned with the analysis of some of the conflicts which can occur when integrating two E-Rs regardless of the source of these E-Rs. The two E-Rs can be part of the same view or from different views.

Define V as the set of views to be integrated, R is the set of all E-Rs, E is the set of all entities, A is the set of all attributes, L is the set of all roles. Consider the two meta E-Rs in Fig. 3.4 and Fig. 3.5. The E-Rs $R1 \in R$ and $R2 \in R$ could either be from the same view or from different views, but they are most likely to be from different views. Consider that R_1 is from $V_1 \in V$ and R_2 is from $V_2 \in V$. Now consider the following definition of one of the E-Rs (R_1):

$R_1 = \{E_1, R_1, E_2\}$&

$E_1 = \{e_1, e_1A, e_1K, e_1V, e_1NK, e_1NV\}$&

$E_2 = \{e_2, e_2A, e_2K, e_2V, e_2NK, e_2NV\}$&

$R_1 = \{r_1, r_1A, r_1K, r_1V, r_1NK, r_1NV\}$.

such that:

e_1A is the set of attributes of E_1,

s.t. $e_1A = \{e_1\ a_i\},\ i = 0\ ..\ m,$

e_1K is the set of key attributes of E_1,

s.t. $e_1K = \{e_1\ k_j\},\ j = 0\ ..\ J \mid J \leq m,$

e_1V is the set of attributes with values of E_1

s.t. $e_1V = \{e_1\ v_l\},\ l = 0\ ..\ L \mid L \leq m,$

e_1NK is the set of non key attributes of E_1,

e_1NV is the set of attributes without any values of E_1

such that $e_1K \cup e_1NK = e_1V \cup e_1NV = e_1A.$

Also consider that:

r_1A is the set of attributes of R_1,

r_1K is the set of key attributes of R_1,

r_1V is the set of attributes with values of R_1

r_1NK is the set of non key attributes of R_1,

r_1NV is the set of attributes without any values of R_1

such that $r_1K \cup r_1NK = r_1V \cup r_1NV = r_1A.$

Also consider that:

e_2A is the set of attributes of E_2

s.t. $e_2A = \{e_2a_s\},\ s = 0\ ..\ n,$

e_2K is the set of key attributes of E_2

s.t. $e_2K = \{e_2k_t\},\ t = 0\ ..\ T \mid T \leq n,$

e_2V is the set of attributes with values of E_2

s.t. $e_2V = \{e_2v_u\}, \quad u = 0 .. U \mid U \leq n,$

e_2NK is the set of non key attributes of E_2,

e_2NV is the set of attributes without any values of E_2

such that $e_2K \cup e_2NK = e_2V \cup e_2NV = e_2A$.

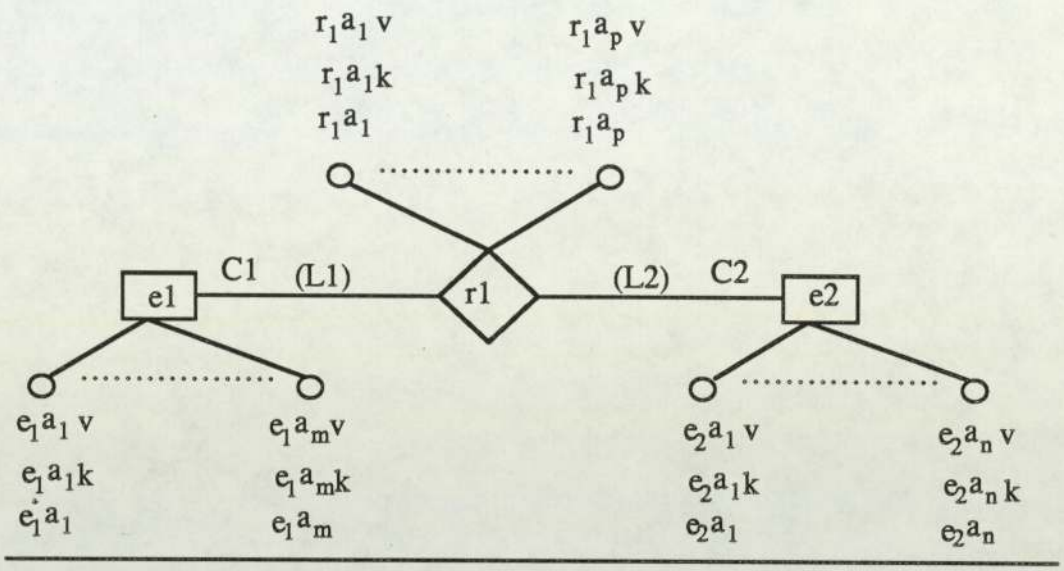


Fig. 3.4 E-R (R_1) of view (V1)

Matching the four entity names and the two relationship names gives 5 pairings of true (T) or false (F), which gives a total of 2^5=32 possibilities. These possibilities are illustrated in Table 3.1. To consider all the possibilities of matching all attributes, attribute values, attribute key statuses, cardinalities and roles of the two E-Rs would give a huge number of pairings. Each extra pairing of T or F considered increases the possibilities by a factor of times 2. To be exact, the total number of possibilities is 2^n, where n is the number of pairings of T or F. For example, to consider each attribute in the two E-Rs, the total number of objects needed to be considered is calculated as:

$$T = 3 \times (m + n + s + t + p + u)$$

where m, n, s, t , p and u are according to the meta definitions of the E-Rs in Fig. 3.3 and Fig. 3.4.

Calculating the possible matchings M between all these objects given by T can be calculated by the formula:

$$M = 2^{(mxs + pxu + nxt + mxt + nxs)}$$

To avoid the inclusion of all the matches in one table, the two E-Rs are matched in the table based on the names of the four entities and the names of the two relationships only.

The matching of the attributes of the entities and the attributes of the relationships is considered in section 3.4.

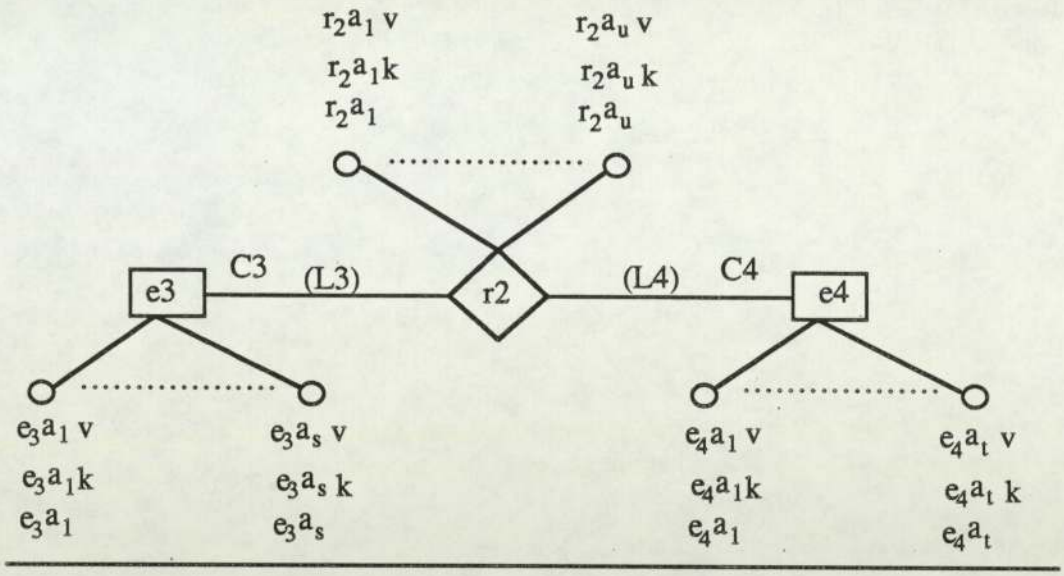


Fig. 3.5 E-R (R_2) of view (V2)

Objects	SITUATIONS
	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
$e_1 = e_3$	T T T T T T T T T T T T T T T T F F F F F F F F F F F F F F
$r_1 = r_2$	T T T T T T T T F F F F F F F F T T T T T T T T F F F F F F F
$e_2 = e_4$	T T T T F F F F T T T T F F F F T T T T F F F F T T T T F F F F
$e_1 = e_4$	T T F F T T F F T T F F T T F F T T F F T T F F T T F F T T F F
$e_2 = e_3$	T F T F T F T F T F T F T F T F T F T F T F T F T F T F T F

Table 3.1 Matching two E-Rs based on entity names and relationship names

Although thirty two possible situations can exist when matching two basic E-Rs, it was found that situations 2, 3, 5, 10, 11, 13, 17 and 25 in table 3.1 are inconsistent and cannot possibly take place in a real situation.

In each of the following types of situation, a diagrammatic illustration is shown to describe the resultant conceptual schema. In these diagrams the abbreviations e indicates an entity name, r indicates a relationship name, C indicates the cardinality of a given entity and L indicates the role of a given entity.

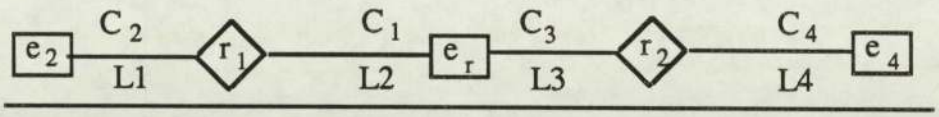
3.3.1 The two E-Rs have one common entity

The integration of two E-Rs satisfying situations 16, 28, 30 and 31 in table 3.1 produces two E-Rs joined together by one common entity. One of the characteristics of these two situations is that they present no cardinalities or role conflicts. Since the four situations are similar, only situation 16 is described below:

Situation 16.

$$e_1 = e_3,\ r_1 \neq r_2,\ e_2 \neq e_4,\ e_1 \neq e_4,\ e_2 \neq e_3$$

Solution:



Such that: $e_r = e_1 = e_3$

Situations 16, 28, 30 and 31 show that the resultant schema is made up of the two E-Rs from the two views, linked in one entity e_r. The roles, cardinalities and the relationships r_1 and r_2, are copied across from the views to the integrated schema exactly as they exist in the views.

Although the common entity e_r matches on name, it may not match on attributes, attribute key statuses, or attribute values. The situation where two or more entities match on names, but not necessarily on their attributes or attribute characteristics, is common when integrating two E-Rs. Therefore, a separate section (section 3.4) shows how two entities which match on names are integrated.

3.3.2 The two E-Rs have one common entity and same relationship name

Situations 8, 20, 22 and 23 are the only four possibilities of this kind of match. These situations introduce a special kind of cardinality matching problem which itself introduces two possible ways of representing each of the four situations. However, the result of integrating these E-Rs is first shown without the cardinality matching considerations. The effect of the cardinality matching is described after these four situations, and therefore the following four resolutions assume equal cardinalities and roles of the common entities and relationship.

Situations 8, 20, 22 and 23 are all based on integrating binary E-Rs. As will be seen from the analysis of these four situations, the results appear to be ternary E-Rs. However, a distinction must be made here between real ternary relationships and two or more binary

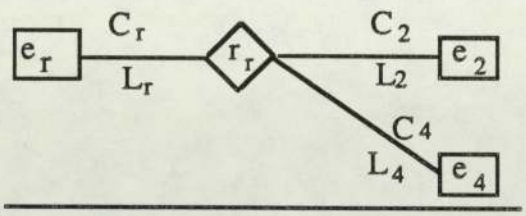
E-Rs which share a common relationship name. Instead of going into detail about the issue of binary vs n-ary E-Rs, let us first show how the E-Rs are integrated based on names analysis only. In section 3.3.2.2 the effect of cardinalities, roles, and the degree (binary or n-ary) of E-Rs is discussed.

Since situations 8, 20, 22 and 23 are similar, only situation 8 is described below:

Situation 8:

$e_1 = e_3,\ r_1 = r_2,\ e_2 \neq e_4,\ e_1 \neq e_4,\ e_2 \neq e_3$

Solution:



Such that: $e_r = e_1 = e_3,\ r_r = r_1 = r_2,\ C_r = C_1 = C_3,\ L_r = L_1 = L_3$

3.3.2.1 Cardinality analysis and integration of E-Rs matching on one entity and relationship name

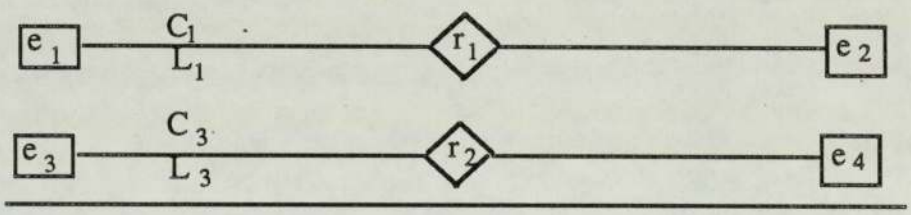


Fig. 3.6 Two E-Rs matching on one side and centre

Consider the four previous situations (8, 20, 22, 23). As illustrated in section 3.3.2, the result of integrating these situations produces two E-Rs with a common entity and relationship name. The problem occurs when the common entity differs on either the cardinalities and the roles or both. Table 3.2 shows four possibilities when matching the cardinalities and roles of such E-Rs. Let us see the effect of each of these four possibilities on the integration of the two E-Rs of Fig. 3.6 and the resultant schema. Since the four situations (8, 20, 22, 23) are similar, only situation 8 will be used to study the effect of the cardinality and role matching as shown in table 3.2.

a) Both entities match on roles and cardinalities:

In this case, the resultant schema is the same as that of situation 8. The analysis of handling the apparently resultant ternary E-R is given in section 3.3.2.2.

Match	Situation
$C_1 = C_3$	Y Y N N
$L_1 = L_3$	Y N Y N

Table 3.2 Cardinality and role matching

b) Entities do not match on roles or cardinalities:

This conflict raises two possibilities:

1 The two relationship names are homonyms

If the designer agrees that the two relationship names are homonyms, then he must supply two names to represent the two relationship names r_1 and r_2. It is more likely that the designer would change only one of the two relationship names. The resultant schema is shown in Fig. 3.7. It contains two E-Rs with the original cardinalities and roles. The two new relationships names are r_{r1} and r_{r2}.

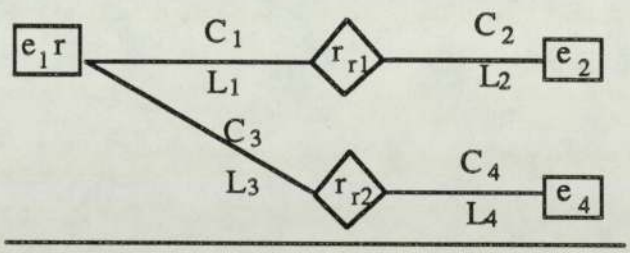


Fig. 3.7 E-R integration - synonym relationship names

Example

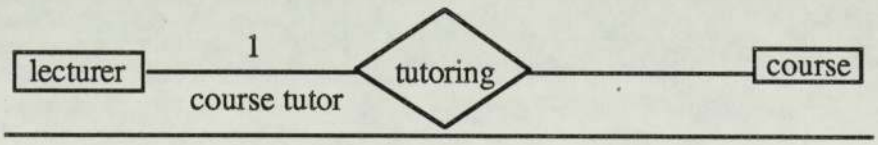


Fig. 3.8 (a) E-R from view 1

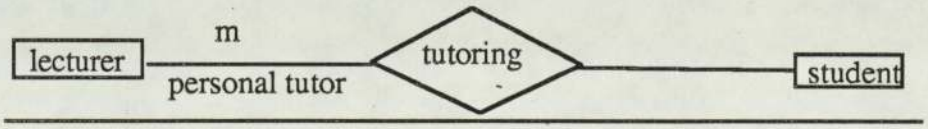


Fig. 3.8 (b) E-R from view 2

The choice made by the designer whether to create two separate E-Rs as shown above or to choose to leave the resultant schema as shown in situation 8, would depend on his understanding of the situation causing the conflict. In the example of Fig. 3.8 above, such a choice has the effect of separating the instances of the two relationship relations in the

database. Should the decision be to keep the format of the resultant schema as in situation 8, then all the instances linking the three entities would be contained in the same relationship relation. A decision as to which of the resultant schema formats is best cannot be given as it depends on the designer's and the user's interpretations of future query and security considerations.

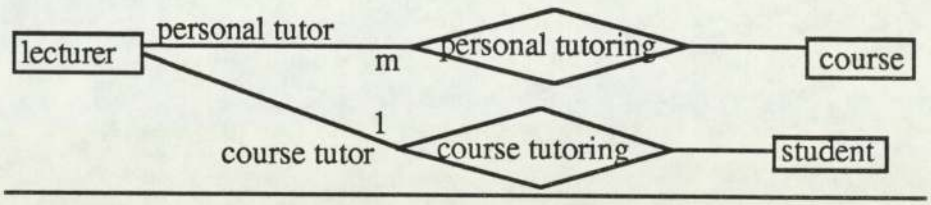


Fig. 3.8 (c) Resultant schema

2 The two roles names are synonyms

In this case, the resultant schema would be the same as that of situation 8 above, except that the designer is requested to decide on the role name L_r and cardinality C_r of the common side. The relationship relation in this case must contain all the key attributes representing the three entities. In any case, possibility 1 above may still apply because of the cardinalities difference.

c) Entities match on cardinalities but do not match on roles:

All the possibilities presented in the situation described in section (b) apply here.

d) Entities match on roles but do not match on cardinalities:

This is an indication that one of the cardinalities of the two entities is incorrect and the designer is requested either to choose one of the cardinalities available or supply a new one. Such a designer decided cardinality is necessary as there is no way to indicate two cardinalities in the same relationship relation, although it effectively represents the instances of two separate binary E-Rs.

3.3.2.2 Analysis of binary and n-ary E-Rs

Binary E-Rs allow direct and easily comprehensible ways of modelling the connection of two objects. However, as shown in situations 8, 20, 22 and 23, the integration of two binary E-Rs sometimes produces what appears to be ternary E-Rs. Although ternary E-Rs (sometimes called three way E-Rs) occasionally appear necessary, they are frequently caused by synonymous names of relationships. A ternary E-R resulting from the integration of two binary E-Rs is most likely not a genuine ternary E-R. The example in Fig. 3.9 ('lecturer teaches student' and 'lecturer teaches course') originated from the integration of two binary E-Rs. However, this does not mean that the resultant E-R is a

ternary E-R. The example schema in Fig. 3.9 could mean that each instance in the relationship relation named 'teaches' must have associate instances from the three entity relations 'lecturer', 'course' and 'student'. Should the schema be a real ternary E-R, then a restriction on the instances of the relationship relation is imposed. For example, there is no way to record the fact that a certain 'student' attends a certain 'course' which is taught by a certain 'lecturer' and 'lecturer' does not teach the 'student' concerned. The semantics of ternary and n-ary E-Rs can become complex (Kent 1978 and Elmasri & Wiederhold 1980). Any ternary or higher order E-R can usually be broken down into one or more binary E-Rs, although this may prove to be a difficult task in certain situations, and especially in E-Rs of a degree more than ternary. The schema in Fig. 3.9 can, for example, be broken down into three binary E-Rs as shown in Fig. 3.10. The cardinalities of the newly created three binary E-Rs must be thought of carefully, so that they impose the same restrictions on the connections of the instances of the three entities.

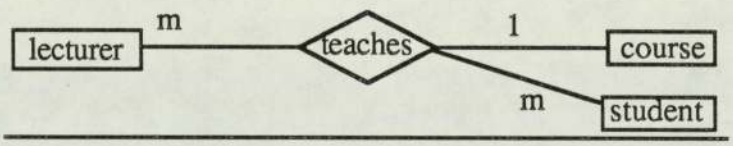


Figure 3.9 An example homonym relationship name

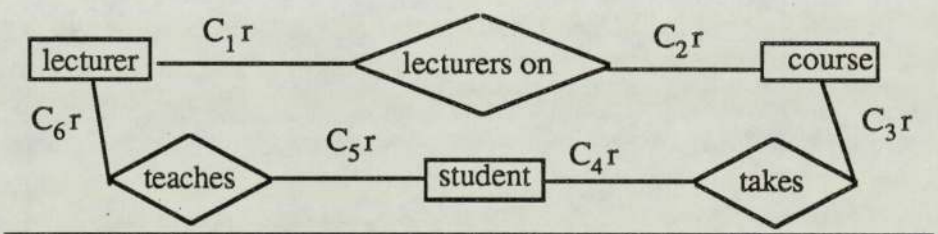


Figure 3.10 Three binary E-Rs instead of one ternary E-R

In other situations, such as the schema shown in Fig. 3.11, it is clear that an instance in the relationship 'attends' does not necessarily mean that 'student' can only attend a conference if 'lecturer' attends (or vice versa). It also does not mean that a conference only takes place if it is attended by a lecturer or a student. However, this appreciation can only be understood by the user and the designer, and it is not possible to make rules or resolutions to decide on such situations. Therefore, instead of the ternary relationship 'attends', one approach is to create two binary E-Rs involving the three entities: 'lecturer', 'student' and 'conference'. However, a naming problem will occur if this approach is followed, since the relationship name 'attends' is representative of the semantic connection between the appropriate entities. Possible relationship names could be 'lecturer conference attendance' and 'student conference attendance'. Clearly these are not desirable names.

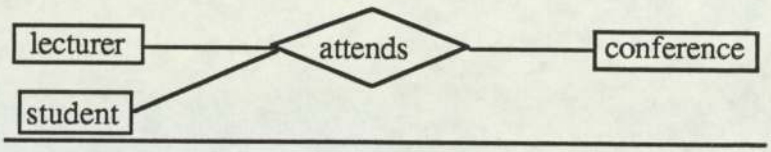


Fig. 3.11 Two E-Rs with a representative homonym relationship name

Another approach is to leave the relationship 'attends' relate the three entities, but it must remain as a binary E-R. In this case, both the cardinality and the role of the common entity must be decided by the designer. The cardinality in this case must be the highest of the two (if they are different). The relationship relation 'attends' in this case must contain null instances in one of its domains which corresponds to one of the entities. Therefore, for each tuple of the relationship relation 'attends', only the domains of the corresponding two entities are instantiated.

Another approach to solving this kind of conflict is to make the entity 'conference' an attribute of both of the entities 'lecturer' and 'student'. In this case the entity 'conference' is either kept in the schema or removed from the schema (if it is not related by any other relationships) and have all its attributes transferred to the entities 'lecturer' and 'student'. This approach may prove to be undesirable if 'conference' has a number of attributes. The constraints imposed by the cardinalities would also be lost. Further, if the relationship 'attends' has attributes, then their transfer to the entities 'lecturer' and 'student' may prove difficult.

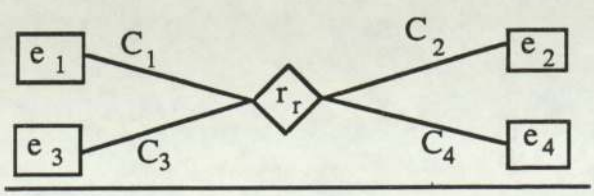
Another approach is to associate a new attribute called 'attender' with entity 'conference' such that 'attender' can have one of two values 'lecturer' or 'attender'. Then the relationship 'attends' may be deleted, and the two entities 'lecturer' and 'student' may also be deleted if they do not participate in any other relationships. This approach also presents problems if the relationship has attributes.

3.3.3 The E-Rs match only on the relationship name

Situation 24:

$e_1 \neq e_3,\ r_1 = r_2,\ e_2 \neq e_4,\ e_1 \neq e_4,\ e_2 \neq e_3$

Solution:



Such that: $r_r = r_1 = r_2$

The integration of binary E-Rs which produce ternary E-Rs was discussed in section 3.3.2.2. The E-R produced by the integration of two or more binary E-Rs cannot usually be identified as ternary or n-ary E-R just because the two E-Rs match on relationship names. Situation 24 here obeys the same principle as that discussed in section 3.3.2.2, and therefore the n-ary E-R above is regarded as two binary E-Rs with a common relationship name. The instances of the two relationships are disjoint and do not influence each other directly. One of the resolutions proposed in 3.3.2.2 is therefore applied to decide on the format of the resultant schema. The most appropriate of these is that the two relationship names are homonyms and consequently the two relationships are given different names. The choice of relationship name may prove to be difficult or inappropriate as in the case of relationship 'attends' of Fig. 3.9.

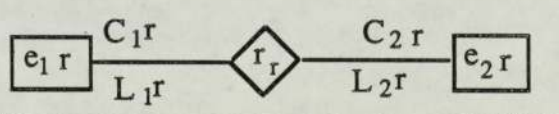
3.3.4 The E-Rs match on all entity names and relationship names

Situations 4 and 21 are of this type, but because they are similar, only situation 4 is shown below:.

Situations 4.

$e_1 = e_3,\ r_1 = r_2,\ e_2 = e_4,\ e_1 \neq e_4,\ e_2 \neq e_3$

Solution:



Such that: $e_1r = e_1 = e_3,\ e_2r = e_2 = e_4,\ r_r = r_1 = r_2$

Regarding the cardinalities and roles, there are four possibilities, as shown in table 3.2. Should both the cardinalities and roles agree, then they are copied directly to the resultant E-R. However, should any of the cardinalities or roles not match, this gives no indication of any synonyms among the entity or relationship names. Instead, it is likely that the roles that do not match are synonyms and the designer must supply the appropriate role names. If the cardinalities do not match, then the higher cardinality values must be chosen so that the E-R can support both views of the E-R.

3.3.5 The E-Rs are recursive

A recursive E-R is one which is defined on the same entity set. This means that instances from the same entity class are related to each other. Examples of such E-Rs are show in Fig. 3.12. Roles are sometimes necessary to indicate the function of an instance in the

entity set to another. It can always be argued that recursive E-Rs can be remodelled into ordinary binary E-Rs, by dividing the entity class into two distinct entity classes according to the role that they play in the E-R. However, this is not always an appropriate approach. And even if it can be arranged, it means that many new types of entities are created, and this may not be acceptable to the user.

A number of situations arise where either one or both of the E-Rs being integrated is recursive. As described in section 3.3.2.2, the integration of two or more binary E-Rs which match on the relationship name does not produce a ternary or n-ary E-R.

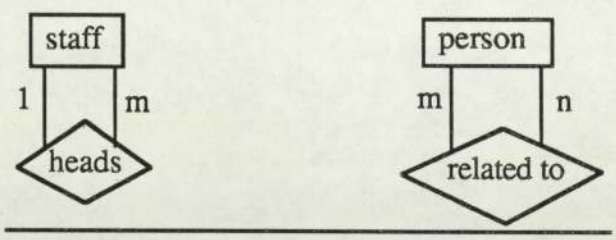
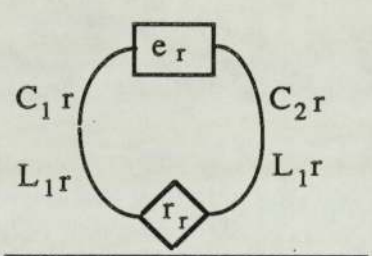


Figure 3.12 Examples of recursive E-Rs

Situation 1:

$e_1 = e_3,\ r_1 = r_2,\ e_2 = e_4,\ e_1 = e_4,\ e_2 = e_3$

Solution:



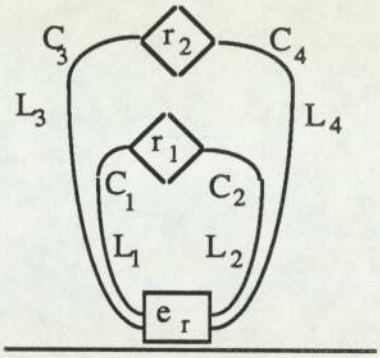
Such that: $e_r = e_1 = e_2 = e_3 = e_4,\quad r_r = r_1 = r_2$

This situation indicates that both E-Rs are recursive and they match on all the entity and relationship names. The handling of the cardinalities and roles is the same as described for situations 4 and 21 in section 3.3.4 above.

Situation 9:

$e_1 = e_3,\ r_1 \neq r_2,\ e_2 = e_4,\ e_1 = e_4,\ e_2 = e_3$

Solution:



Such that: $e_r = e_1 = e_2 = e_3 = e_4$

This situation creates two recursive E-Rs, both of which share the same entity. No conflicts between cardinalities and roles exist as they are transferred exactly as they are in the two E-Rs.

3.3.6 One recursive E-R and one ordinary E-R with same relationship name and one common entity

The integration of a recursive E-R and an ordinary E-R is indicated by the situations 6, 7, 18 and 19 in table 3.1. The integration of the E-Rs for the four possible situations is illustrated below. However, the integration of the cardinality constraints and roles presents a special kind of problem. The four situations show that one entity and the relationship name of the non recursive E-R match the entity and the relationship name of the recursive E-R. The problem here is with which cardinality and role of the recursive E-R should the matching side of the other E-R be compared. Consider the example of Fig. 3.13.

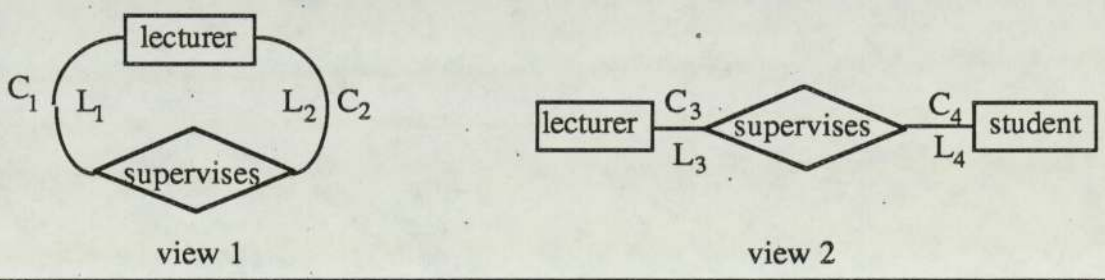


Figure 3.13 Integration of recursive and non recursive E-Rs

Solution:

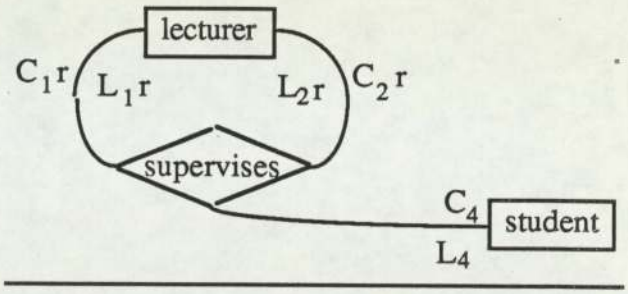


Figure 3.14 A ternary E-R including a recursive E-R

Regarding the cardinalities, the problem is whether to integrate the cardinality C_3 with C_1 or C_2 of the recursive E-R. Should the designer decide to keep the two E-Rs sharing the same name, the resultant E-R remains a binary E-R ('bad' binary E-R, Kent 1978), and any conflicts in the cardinalities and roles must be resolved. If the decision is not to allow 'bad' binary E-Rs, then a new relationship name is given to either or both E-Rs, and hence no conflict occurs. The result is two E-Rs with a common entity (see section 3.3.1). If a 'bad' binary E-R is to be allowed, then, as described in section 3.3.2.2 earlier, the maximum cardinality should always be chosen. In the situation above, the priority is given first to the equality of the roles.

1. $L_3 = L_1$

   solution: C_1r is the maximum of C_1 and C_3

   $L_1r = L_1$, $L_2r = L_2$

2. $L_3 = L_2$

   solution: C_2r is the maximum of C_2 and C_3

   $L_2r = L_2$, $L_1r = L_1$

3. $L_1 \neq L_2 \neq L_3$

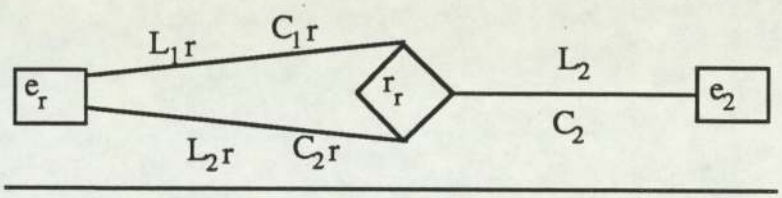
   Designer intervention is requested. However, this is a strong indication that the two relationship names could be changed to their own unique names. In the example above, the two relationship names can be 'student supervision' and 'staff supervision'. All the options for resolution to this kind of conflict are the same as those presented in section 3.3.2.2.

The four situations from table 3.1 are based on the example above, and therefore, the resolutions of the cardinalities and roles are as shown in that example. Since these four situations are similar, only situation 6 is shown below:

Situation 6.

$e_1 = e_3$, $r_1 = r_2$, $e_2 \neq e_4$, $e_1 = e_4$, $e_2 \neq e_3$

Solution:



Such that: $e_r = e_1 = e_3 = e_4$, $r_r = r_1 = r_2$

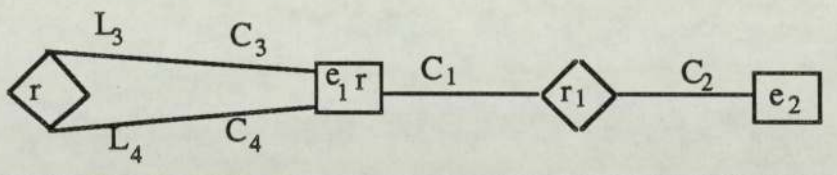
3.3.7 One recursive E-R and one ordinary E-R with one common entity

Situations 27, 26, 15 and 14 are of this kind. Because they are similar, only situation 14 is shown below:

Situation 14.

$e_1 = e_3$, $r_1 \neq r_2$, $e_2 \neq e_4$, $e_1 = e_4$, $e_2 \neq e_3$

Solution:



Such that: $e_{1r} = e_1 = e_3 = e_4$

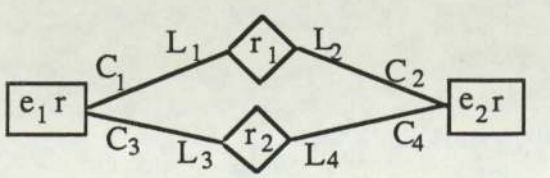
3.3.8 The E-Rs match on entities but different relationship names

Situations 12 and 19 are the only two situations of this type in table 3.1. The two binary E-Rs over the same two entities presents no problems with cardinalities or roles in integration. Because the two situations are similar, only situation 12 is shown below:

Situation 12.

$e_1 = e_3$, $r_1 \neq r_2$, $e_2 = e_4$, $e_1 \neq e_4$, $e_2 \neq e_3$

Solution:



Such that: $e_1r = e_1 = e_3$, $e_2r = e_2 = e_4$

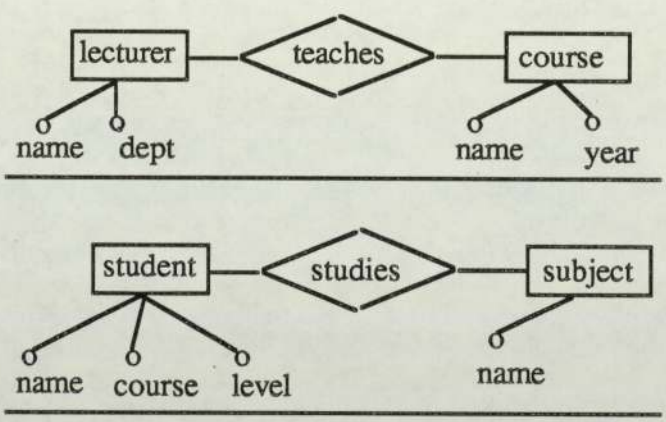
3.3.9 The two E-Rs are different

Situation 32.

$$e_1 \neq e_3,\ r_1 \neq r_2,\ e_2 \neq e_4,\ e_1 \neq e_4,\ e_2 \neq e_3$$

Solution:

This situation means that the two E-Rs have no entity names nor relationship names in common, and therefore the two E-Rs can be classified as different, and added to the schema accordingly. Should the two views considered both consist of a number of E-Rs, then there is a possibility that a match of some kind may be found to one of the two E-Rs. However, it is assumed in this chapter that the two views consist of one E-R each. Two totally different E-Rs are shown below:



3.4 Matching and integrating entities and relationships

The integration of E-Rs was discussed in section 3.3. The matching of any two E-Rs was based entirely on the names of entities and relationships. The cardinalities and roles of entities were analysed as part of the integration of E-Rs. However, the integration of the attributes defining the entities and relationships was not discussed. The reason for not including the attributes in section 3.3 was to avoid repetition of the same analysis and reduce the number of possibilities relating to table 3.1.

This section is concerned with the analysis and integration of the attributes defining entities and relationships. Two entities or two relationships sharing the same name were regarded as the same in section 3.3. Since the two relationships or entities are from different views, they will not normally match exactly on attributes. This section shows how the attributes of one entity or relationship from one view are used to update the attributes of the entity or relationship from the other view. The attributes of the resultant entity or relationship should then contain all the domains of both views. Since the

matching and integration of entities is exactly the same as that of relationships, only the integration of entities is shown here.

The only view integration methodology reviewed in Chapter 2 which gave any analysis to the integration of entities is that by Elmasri and Navathe (1984). The analysis of objects, which are either entity classes or subclasses (based on the Entity-Category-Relationship model described in Weeldreyer 1980 and Elmasri 1985) produces four possibilities:

1 Identical object domains: Both objects match on attributes and a single object class is created in the schema. Any disagreement on key statuses is resolved by the designer. No consideration to the matching of attribute values is given.

2 Contained domains: The attributes of one object are a subset of the attributes of the other object. For these situations, Elmasri & Navathe create one of the objects as a subclass of the other. Whilst this suggestion may be true in certain occasions in a large application, many of the objects may have attributes which are subsets of the attributes of other objects, yet they are totally different objects.

3 Intersection domains: The intersection of the domain of one object with the other object is not equal to nil and neither is it a subset of the other. For these situations, Elmasri & Navathe create a superclass with attributes equal to the union of the other two objects. The two objects are made to be subsets of the newly created superset. Again, in practical applications, such an assumption cannot be automatically made, since many objects which are totally different would fit this assumption, although they are totally disjoint as far as their extension is concerned.

4 Disjoint domains: Objects which have no attributes in common are created in the schema as two different objects. The approach followed in this thesis for disjoint situations is the same as the approach of Elmasri & Navathe, except the assumption here is that the two entities have the same name. Multi-name modelling and analysis (section 5.6) and fuzzy object matching (section 4.2) might prove that some of the disjoint objects are in fact the same.

Fig. 3.15 Meta entities

Consider the two entities $E_1 \in E$ and $E_2 \in E$ shown in Figure 3.15 are to be matched as part of the E-R integration process. The factors to be considered in the integration are the entity names e_1 and e_2 and the sets of attributes of the two entities. Although the definition of the full E-Rs R_1 and R_2

which also included the definition of their entities was presented in section 3.2, a repetition of the definition of the two entities E_1 and E_2 is again presented here for ease of reference.

Consider that E_1 and E_2 respectively, contain the following sets:

1. e_1A & e_2A are the sets of attribute names
2. e_1K & e_2K are the sets of key attributes
3. e_1V & e_2V are the sets of attributes with values (valued attributes)
4. e_1NK & e_2NK are the sets of non key attributes
5. e_1NV & e_2NV are the sets of attributes without values

From these sets the following definitions can be established:

$$e_1A = \{e_1\, a_i\},\quad i = 0 .. m,$$

$$e_1K = \{e_1\, k_j\},\quad j = 0 .. J \mid J \leq m,$$

$$e_1V = \{e_1\, v_l\},\quad l = 0 .. L \mid L \leq m,$$

$$e_1NK = \{e_1\, nk_p\},\quad p = 0 .. P \mid P \leq m,$$

$$e_1NV = \{e_1\, nv_q\},\quad q = 0 .. Q \mid Q \leq m,$$

and

$$e_2A = \{e_2 a_s\},\quad s = 0 .. n,$$

$$e_2K = \{e_2 k_t\},\quad t = 0 .. T \mid T \leq n,$$

$$e_2V = \{e_2 v_u\},\quad u = 0 .. U \mid U \leq n,$$

$$e_2NK = \{e_2 nk_v\},\quad v = 0 .. V \mid V \leq n,$$

$$e_2NV = \{e_2 nv_w\},\quad w = 0 .. W \mid W \leq n.$$

such that $e_1K \cup e_1NK = e_1V \cup e_1NV = e_1A$

and $e_2K \cup e_2NK = e_2V \cup e_2NV = e_2A$

The resultant integrated entity E_r (Fig. 3.16) can be defined as containing the following sets of attributes:

$$e_r A = \{e_r\, a_b\},\quad b = 0 .. B,$$

$$e_r K = \{e_r\, k_c\},\quad c = 0 .. C \mid C \leq B,$$

$$e_r V = \{e_r\, v_d\},\quad d = 0 .. D \mid D \leq B,$$

$$e_r NK = \{e_r\, nk_e\},\quad e = 0 .. E \mid E \leq B,$$

$$e_r NV = \{e_r\, nv_f\},\quad f = 0 .. F \mid F \leq B,$$

such that $e_r K \cup e_r NK = e_r V \cup e_r NV = e_r A$

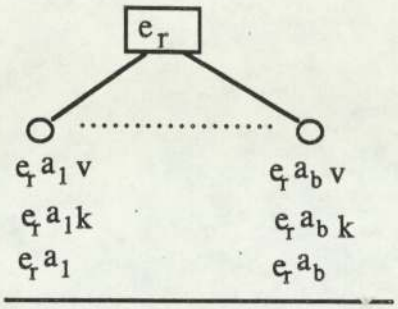


Fig. 3.16 Resultant meta entity

The matching and integration of E_1 and E_2 in order to produce E_r raises many possibilities.

3.4.1 Entities have no common attributes

If two entities share the same name, then they are regarded as representing the same entity class regardless of the difference in the attributes. The attributes defining the same entity name in two different views can suffer from the usual naming conflicts (section 5.5). Further, different users might be interested in different domains of the entity concerned. Consider for example the entity 'book' modelled in two views called 'library' and 'courses'. In the 'library' view the domain represented by the attribute 'book number' suffices, whilst in the 'courses' view, the domains represented by the attributes 'author' and 'title' are needed.

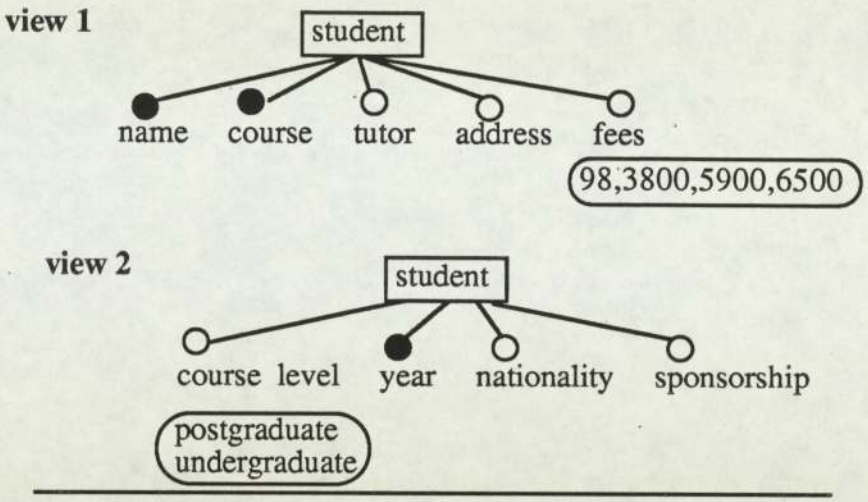
Two entities are disjoint if $e_1A \cap e_2A = \Phi$. The approach followed here to resolve such conflict is to transfer all the attributes from both entities to the resultant entity. Obviously, due to the naming conflicts, it is possible that the same attribute might be represented twice in the same entity. Such a conflict is left to the post integration task of object fuzzy matching (section 4.2). The resultant entity E_r would then contain the following:

$$e_r A \leftarrow e_1A \cup e_2A$$
$$e_r K \leftarrow e_1K \cup e_2K$$
$$e_r V \leftarrow e_1V \cup e_2V$$
$$e_r NK \leftarrow e_1NK \cup e_2NK$$
$$e_r NV \leftarrow e_1NV \cup e_2NV$$

such that: $e_r K \cup e_r NK = e_r A$

and $e_r V \cup e_r NV = e_r A$

Example



The resultant sets of attribute names, key attributes, non key attributes, valued attributes and non valued attributes of the entity 'student' from the two views are shown below:

$e_r A$ = {name, course, tutor, address, fees}

$\cup$ { course level, year, nationality, sponsorship}

$e_r K$ = {name, course} $\cup$ {year}

$e_r NK$ = {tutor, address, fees} $\cup$ { course level, nationality, sponsorship}

$e_r V$ = {fees} $\cup$ {course, level}

$e_r NV$ = {name, course, tutor, address} $\cup$ {year, nationality, sponsorship}

The resultant entity is as shown in Fig. 3.17.

3.4.2 Entities have a complete match on attribute names

Two entities match on all attribute if:

$\forall (e_1 a_i \in m) \exists (e_2 a_j \in n)$

such that $e_1 a_i \in m = e_2 a_j \in n$

Although the two entities match on attribute names, they do not necessarily match on the key status or the values of these attributes. The methods of achieving the resultant set of attribute names $e_r A$, the set of key attributes $e_r K$, and the set of valued attributes $e_r V$ of E_r are described below.

3.4.2.1 Obtaining the resultant set of attributes

Since both entities completely match on attribute names, then the set of attributes e_rA of the integrated entity E_r is made to equal the set of attributes of one of the entities (e_1A for example). Therefore e_r A, is obtained as follows:

$$\forall (e_1a_i \in m) \qquad e_r a_b \in B \leftarrow e_1a_i \in m$$

This means that $e_r A \leftarrow e_1 A$.

3.4.2.2 Obtaining the resultant set of key attributes

This raises three possible situations as described below.

1 Entities match on sets of key attributes

If $\forall (e_1K_{j \in J}) \in e_1A$

$\exists (e_2K_{t \in T}) \in e_2A$

such that:

$$e_1K_{j \in J} = e_2K_{t \in T}.$$

then $e_rK \leftarrow e_1K$

This means that the resultant entity would contain exactly the same set of key attributes which is equal to the set of key attributes of any one of the two entities.

2 Entities have no common key attributes

If $e_1K \cap e_2K = \Phi$

then $e_r K \leftarrow e_1K \cup e_2K$

Since the two entities have no common key attributes, then the resultant entity set of key attributes e_r K must be the union of the two sets of the key attributes e_1K and e_2K of the two entities. This resolution would allow both users to have their key domains in the entity. It is possible that some of the key attributes from the two entities are the same except that they suffer from naming problems.

3 Entities have conflicting key status of attributes

If $e_1K \neq e_2K$

and $\exists (e_1K_{j \in J}) \in e_1A$ and $\exists (e_2K_{t \in T}) \in e_2A$

such that $e_1K_{j \in J} \neq e_2K_{t \in T}$

and either $e_1K_{j \in J} \in e_2A$

or $e_2K_{t \in T} \in e_1A$

then choose either resolution 1 or resolution 2 below.

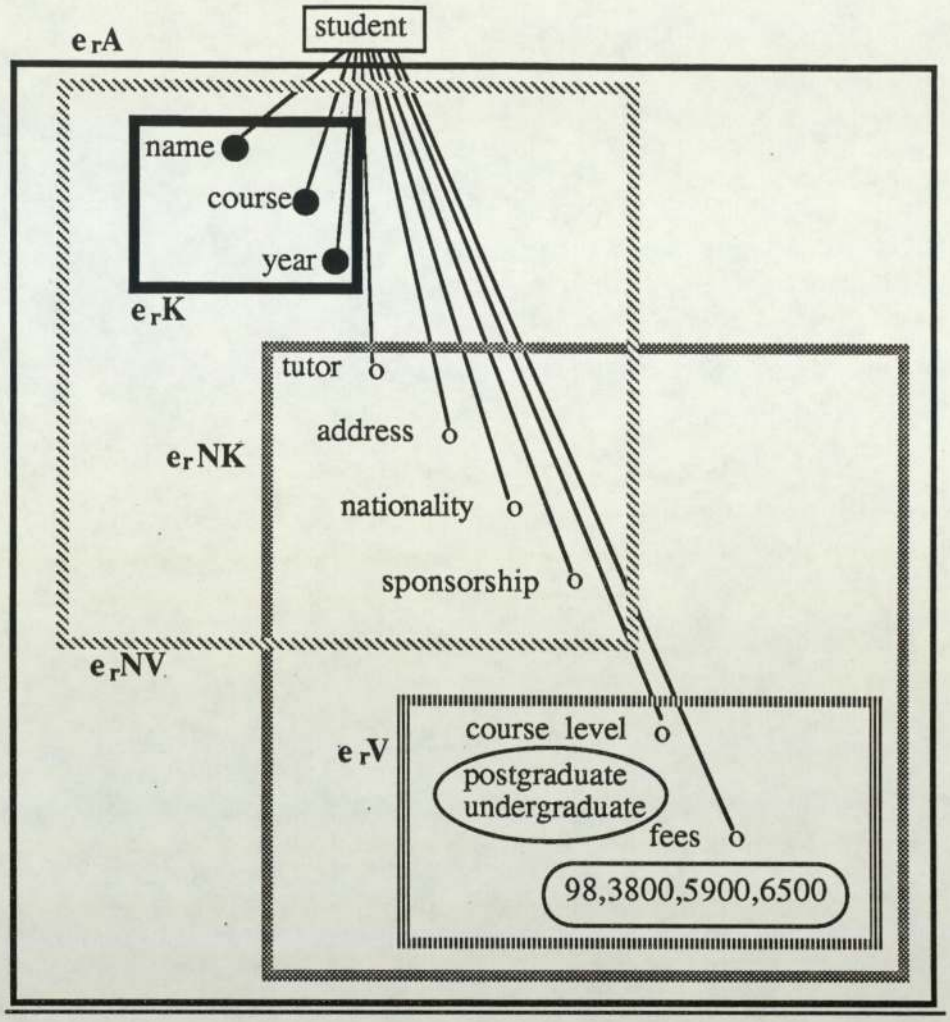


Fig. 3.17 The resultant entity from integrating entity student from views 1 & 2

Solution 1:

$$e_r K = e_1K \cup e_2K$$

This resolution is based on the assumption that any conflict in attribute key status between E_1 and E_2, where the same attribute is declared as a key attribute in one entity and a non key attribute in the other entity, is resolved by assuming that the attribute is a key attribute. This resolution, however, affects the dependencies and normalization process. At the same time, in order to cater for both views of the same entity, the choice of key status to the non key status of the same attribute allows more flexibility.

Solution 2:

The designer is the best judge as to whether the attribute should be a key or a non key attribute. The designer may decide that it is not possible to accommodate both views in the same entity, in which case he may create two separate entities.

3.4.2.3 Obtaining the resultant set of valued attributes

An attribute can have no value, a set of values or a range of values. This means that when matching two attributes for the equality or otherwise of their values, three pairings of true or false equality are produced which gives a maximum of eight possibilities. However, since individual sets are only kept of the valued attributes of entities, this reduces the maximum possibilities to three. Therefore, the two attributes can both be sets of values, or ranges of values, or one is a set of values and the other is a range of values. Another possibility is when one attribute in one entity has a value and the same attribute in the other entity has no value.

Assume that FV is the general function which matches two individual valued attributes from e_1V and e_2V to produce $e_r V$ such that:

$$FV : e_1V \times e_2V \rightarrow e_r V$$

$$FV (e_1v_l \in L, e_2v_u \in U) = e_r v_d \in D$$

The methods of obtaining the resultant set of valued attributes $e_r V$ is as shown below.

a) <u>The two attributes have range values:</u>

Assume that:

e_1VR and e_2VR are the sets of range valued attributes of e_1V and e_2V,

and $\exists (e_1v_l \in L)$ and $\exists (e_2v_u \in U)$

such that:

$$e_1v_l \in L \in e_1A \text{ and } e_2v_u \in U \in e_2A$$

and

$$e_1v_l \in L \in e_1VR \text{ and } e_2v_u \in U \in e_2VR$$

then:

If the two attributes $e_1v_l \in L$ and $e_2v_u \in U$ have the same values, the resultant attribute $e_r v_d \in D$ must have a value equal to the value of one of the attributes. If the two attributes have different values, the designer must have the ultimate decision as to which value is chosen. It is always possible to choose either the bigger range value or the the biggest range calculated from both values. However, this may impose the wrong limits on the instances of the domain represented by the attribute concerned.

b) <u>The two attributes have sets of values:</u>

Assume that:

e_1VS and e_2VS are the attributes holding sets of values of e_1V and e_2V,

and $\exists (e_1v_l \in L)$ and $\exists (e_2v_u \in U)$

such that:

$$e_1v_l \in L \in e_1A \text{ and } e_2v_u \in U \in e_2A$$

and

$e_1v_l \in L \in e_1VS$ and $e_2v_u \in U \in e_2VS$

then $V_r = V_1 \cup V_2$

where: V_1 is the set of values of $e_1v_l \in L$ &

V_2 is the set of values of $e_2v_u \in U$ &

V_r is the set of values of $e_r v_d \in D$.

c) One attribute has a range of values and the other has a set of values:

Assume that:

e_1VS is the set of attributes with sets of values of e_1V and e_2VR is the set of attributes with range valued attributes of e_2V,

and $\exists (e_1v_l \in L)$ and $\exists (e_2v_u \in U)$

such that:

$e_1v_l \in L \in e_1A$ and $e_2v_u \in U \in e_2A$

and

$e_1v_l \in L \in e_1VS$ and $e_2v_u \in U \in e_2VR$

then the value of $e_r v_d \in d$ must be decided by the designer.

d) One attribute has a value and the other has no value:

Both attributes share the same name. However, one of these attributes has a value, whilst the other has no value. In such a situation, the appropriate resolution to this conflict is to assign the value of the valued attribute to the resultant entity attribute under the same attribute name. Therefore:

If $\exists (e_1v_l \in L)$ and $\exists (e_2v_u \in U)$ and $e_1v_l \in L$ is the valued attribute

such that:

$e_1v_l \in L \in e_1A$ and $e_2v_u \in U \in e_2A$

and

$(e_1v_l \in L \in e_1VS$ or $e_1v_l \in L \in e_1VR)$

and $e_2v_u \in U \notin e_2V$

then $e_r v_d \in d \leftarrow e_1v_l \in L$

3.5 Conclusions

This chapter has shown how E-Rs in ERM are represented formally in set theory format. Because the smallest complete object in ERM is the E-R, matching and integrating two or more views can be achieved by matching and integrating their corresponding E-Rs. A method for identifying all the possible situations which can exist when matching the

names of the entities and the names of the relationships of two E-Rs has been presented. To show all the possible situations when matching and integrating two E-Rs, such that these situations include all the characteristics of all the objects in the two E-Rs, would produce many thousands of possibilities. Therefore, the method showing the integration of entities and relationships was presented separately.

The integration approach presented regarded names of objects to be the same only if they were identical. Objects with the synonym naming conflicts were regarded as different, and integrated accordingly.

A number of situations raised some interesting conflicts. Situations where one of the E-Rs is recursive and the two E-Rs have a common relationship name, raised the conflict regarding the matching and integration of the correct 'sides' of the two E-Rs. A method was presented to show that roles can be used to help identify the corresponding sides. However, when either the roles or cardinalities (or both) of the four sides of the two E-Rs differ, then only the designer can resolve the conflict.

The integration of two or more binary E-Rs should produce binary E-Rs. However, in some situations where the two E-Rs match on the relationship names and two entities, the apparent result was a ternary E-R. These situations could be the results of a homonym naming conflict between the relationship names. However, some situations dictate the need for such homonyms (see Fig. 3.11), and hence it was decided to leave the result as a 'bad' binary E-R.

Due to the flexibility of ERM, as well as naming conflicts, it is possible to model objects of different types with the same name. Such conflicts can therefore either be naming conflicts or genuine structural conflicts. In either case, they are referred to here as COT conflicts, and they are discussed in section 4.3.

CHAPTER 4

NAMING AND STRUCTURAL CONFLICTS

4.1 Introduction

The first part of this chapter (section 4.2) describes a method to identify synonymous ERM objects. The result of carrying out this object fuzzy matching method on two objects is an SLF value which ranges between 0 and 1 and shows the level of similarity between the matched objects. This section describes the object fuzzy matching method for each of the ERM objects and concludes that only fuzzy matching of entities produces relevant results.

The second part of this chapter (section 4.3) presents the definitions and suggested resolutions for a number of COT conflicts.

4.2 Objects fuzzy matching approach

Fuzzy logic, fuzzy sets and possibility theory have all been used to study the level of match between information sets. In real life situations, there is a great deal of common sense knowledge and assumed knowledge which, when modelled, looses its preciseness either because of the weakness of the model used or because of the misunderstanding by the designer in representing this knowledge. Fuzzy set research became an established area of research in the 1960s (Zadeh 1965 and Wang & Chang 1980). Fuzzy set theory is used to analyse information in sets based on predetermined evaluation and matching techniques, and the process is called the *Fuzzy Set Analysis*. Fuzzy set analysis of two or more sets of information usually produces a numeric value ranging from 0 to 1. A total difference between the two sets is indicated by a result of 0, whilst a total match is indicated by a result of 1. This numeric value is called here the *similarity level factor* (or *SLF*). SLF should give a realistic evaluation of the level of similarity between sets.

Naming conflicts affect all types of objects in ERM. Entities, for example, can be modelled in different views to have the same name, but have different attributes and are related by different relationships. Because of the name similarity, VI integrates these entities as one (see section 3.4). However, if these entities were modelled with a synonym naming conflict, then VI would integrate them with the GCS as different entities.

In order to identify synonyms, VI must depend on the structural semantics of objects. To achieve this, VI compares all objects of the same type and finds all the *fuzzy situations*. Fuzzy situations occur where the value of the SLF indicates a borderline match. VI bases its object fuzzy matching on the neighbours and characteristics of the objects concerned. For example, object fuzzy matching of entities can be based on their attributes, cardinalities and relationships. Object fuzzy matching of relationships is based on their entities and attributes. Object fuzzy matching of attributes is based on their owner objects, which are either entities or relationships, as well as their key statuses and values. Only immediate neighbours are used in VI to find the SLFs. Considering the non-immediate neighbours could distort the value of the SLFs, and would require complex methods of weight assignments.

The object fuzzy matching of objects is not concerned with the linguistic analysis of their names, an area in natural language known as morphology. Should the application be a technical or scientific one, then the names are likely to be symbolic, and morphology is not relevant.

4.2.1 Fuzzy matching of entities

An entity is defined by one or more attributes, related by one or more relationships, constrained by cardinalities and further defined by roles. To study the combined effect of attributes, relationships, roles and cardinalities on the evaluation of the SLF between two entities, weights must be assigned to each of these characteristics.

Consider the two entities e_1 and e_2 of Fig. 4.1, then consider the following:

e_1 A = the set of attributes of e_1.
e_2 A = the set of attributes of e_2 .
e_1 K = the set of key attributes of e_1.
e_2 K = the set of key attributes of e_2.
e_1 V = the set of valued attributes of e_1.
e_2 V = the set of valued attributes of e_2 .
e_1 R = the set of relationships involving e_1.
e_2 R = the set of relationships involving e_2.

Assume that P is the SLF, and that P is a numeric value from 0 to 1. P is calculated on the basis of four values achieved from matching the sets e_1A and e_2A, e_1K and e_2K, e_1V and e_2V, and e_1R and e_2R, such that:

Pa = the SLF of e_1 A and e_2 A.
Pk = the SLF of e_1 K and e_2 K.
Pv = the SLF of e_1 V and e_2 V.
Pr = the SLF of e_1 R and e_2 R.
Then P = Pa + Pk + Pv + Pr

And Pa, Pk, Pv and Pr are calculated as shown below:

$Pa = \Sigma (e_1\,A \cap e_2\,A) / \Sigma (e_1\,A \cup e_2\,A)$.

$Pk = \Sigma (e_1\,K \cap e_2\,K) / \Sigma (e_1\,K \cup e_2\,K)$.

$Pv = \Sigma (e_1\,V \cap e_2\,V) / \Sigma (e_1\,V \cup e_2\,V)$.

$Pr = \Sigma (e_1\,R \cap e_2\,R) / \Sigma (e_1\,R \cup e_2\,R)$.

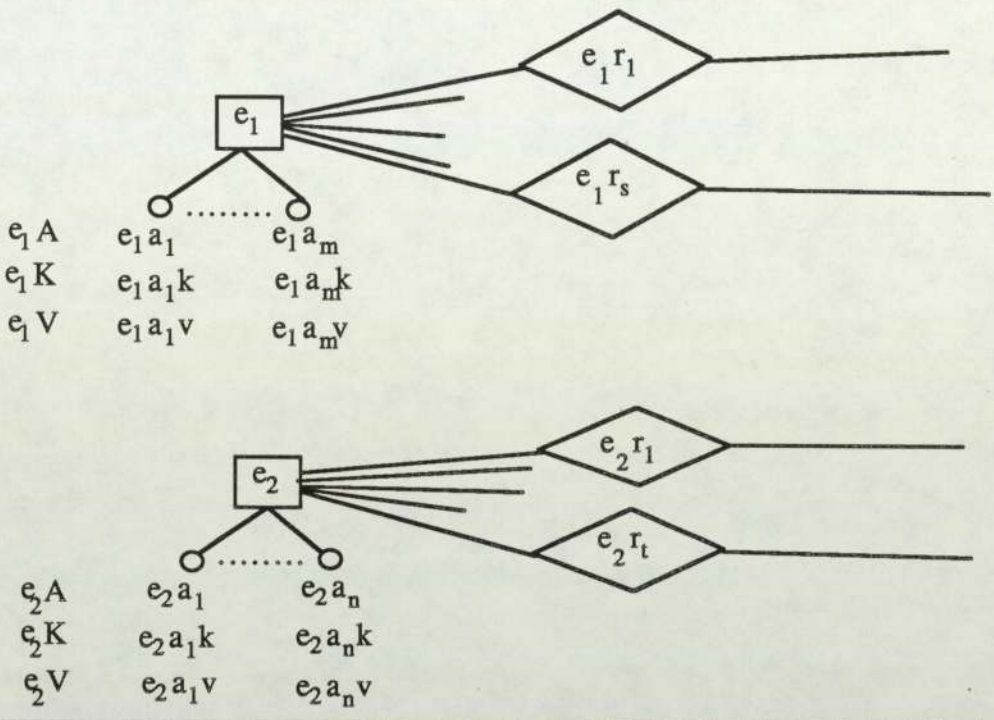


Fig. 4.1 Neighbours of entities

These equations have been formulated in such a way that they give an accurate indication of the level of similarity between the sets of neighbours concerned. Each of the above equations will yield a value of 1 if there is total match, and 0 if no common neighbours and their characteristics exist between the two entities. All other situations will yield a value between 0 and 1. Let us illustrate both a total match situation and a total mismatch situation. Rather than calculate each of the SLFs Pa, Pk, Pv and Pr, only Pa will be calculated to illustrate the approach.

Example 1: total match:

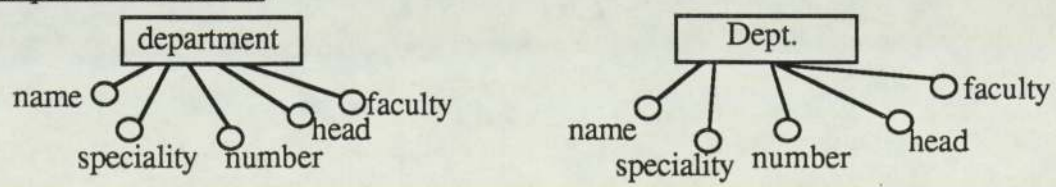


Fig. 4.2 Entities with synonym naming conflict and matching attributes

$e_1\,A$ = {name, speciality, number, head, faculty}

$e_2\,A$ = {name, speciality, number, head, faculty}

Therefore: $e_1 A \cap e_2 A = e_1 A$

and

$e_1 A \cup e_2 A = e_1 A$

Then:

$Pa = e_1 A / e_1 A = 1$

or:

$\Sigma(e_1 A \cap e_2 A) = 5$

$\Sigma(e_1 A \cup e_2 A) = 5$

and therefore:

$Pa\ (e_1 A, e_2 A) = 5/5 = 1.$

The number 1 for Pa indicates a total match on the attributes of the two entities 'department' and 'Dept.'. Notice that the names were not used in the calculation of Pa. Although the two entities have different names, the SLF Pa establishes that the two entities considered for fuzzy matching are the same, regardless of the type of naming conflict between them.

Example 2: Total difference:

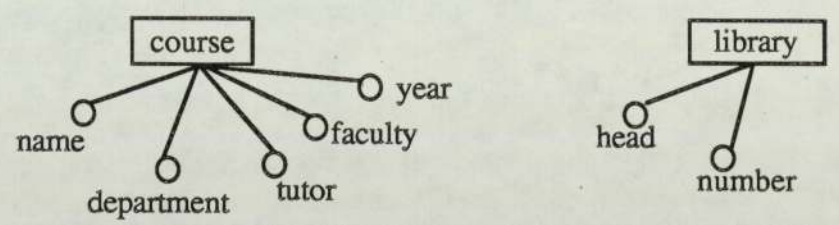


Fig. 4.3 Entities with totally different attributes

$e_1 A =$ {name, department, tutor, faculty, year}

$e_2 A =$ {head, number}

Therefore: $e_1 A \cap e_2 A = \Phi$

and $e_1 A \cup e_2 A =$ {name, department, tutor, faculty, year, head, number}

This means that: $\Sigma(e_1 A \cap e_2 A) = 0$

$\Sigma(e_1 A \cup e_2 A) = 7$

and therefore: $Pa = 0 / 7 = 0.$

The SLF Pa achieved by object fuzzy matching the two entities 'course' and 'library', based on the names of their attributes only, is 0. This indicates that the two entities have no attributes in common, and this is a strong indication that they are different entities.

The two examples above were used to illustrate the effect of object fuzzy matching on entities based on their sets of attribute names only. Two extreme cases of object fuzzy matching showing either a total match or total difference were shown. However, to rely on only the sets of attribute names to achieve the final SLF is not realistic, and does not

show the real SLF between entities. All the neighbours of the entities must contribute with different weights in calculating the total SLF. Each of the SLFs Pa, Pk, Pv and Pr, is assigned a weight. Since the importance of each of the neighbours is dependent on the SDM definition and our understanding of it, the weights must be decided manually. Assume that the weights associated with Pa, Pk, Pv and Pr are W_1, W_2, W_3 and W_4 respectively. Then the total SLF P is calculated as:

$$P = W_1 \times Pa + W_2 \times Pk + W_3 \times Pv + W_4 \times Pr \quad \ldots\ldots\ldots (1)$$

The method illustrated by equation (1) above will be called the *weigh and add* method. W_1 to W_4, when applied to Pa, Pk, Pv and Pr should always yield a value for P ranging from 0 to 1. The difficulty is in deciding on the values of these associated weights so that each of Pa, Pk, Pv and Pr plays the exact representative role it should do in achieving P. Should the wrong weight be given to any of the 4 sub SLFs, P would not be the representative total SLF. Since P must be in the range of 0 to 1, and each of the 4 sub SLFs used to achieve P be in the range of 0 to 1, then the total value of the weights W_1, W_2, W_3 and W_4 must be ≤ 1. Therefore, assuming that the four sub SLFs play equal roles in achieving P, the weights of W_1, W_2, W_3 and W_4 must all be = 0.25.

The weigh and add method illustrated by equation (1) above, works in such a way that the sub SLFs complement each other in a cooperative way to make the total SLF. Another method is to ensure that a low SLF reduces (weighs down) the total SLF. This can be achieved by the *weigh and multiply* method in accordance with equation (2) below:

$$P = (Pa)^{W1} \times (Pk)^{W2} \times (Pv)^{W3} \times (Pr)^{W4} \quad \ldots\ldots\ldots (2)$$

Both the weigh and add method and the weigh and multiply method have been implemented in order to compare their results.

Example: Calculation of the total similarity factor of entities.

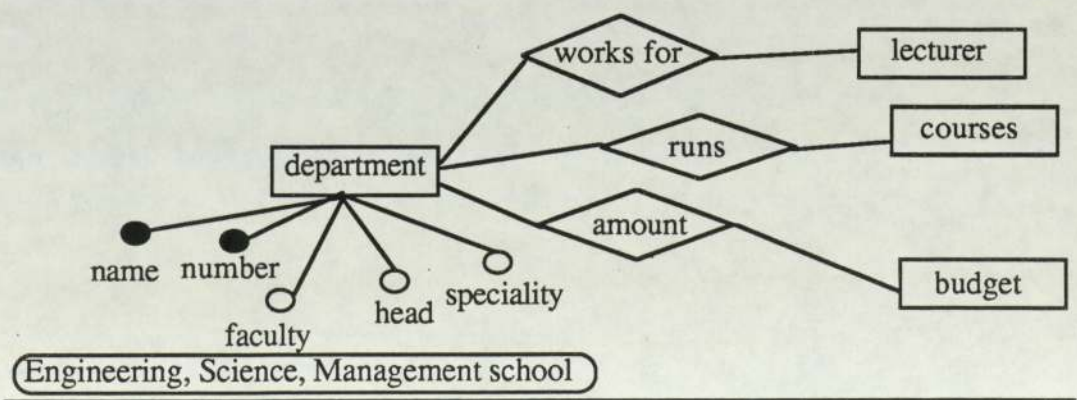


Fig 4.4 An example entity with neighbours

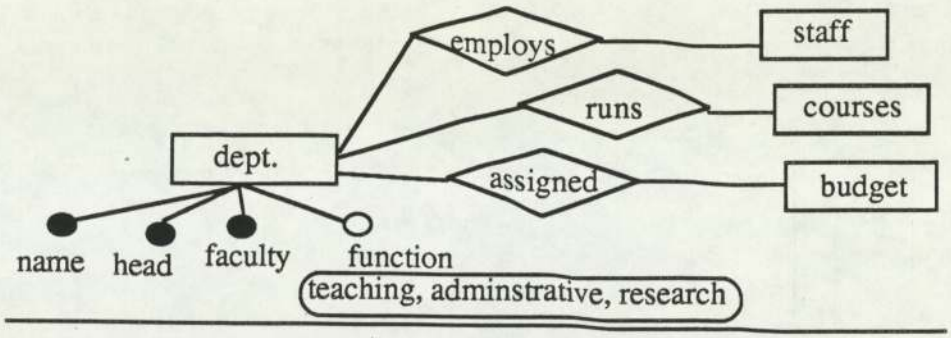


Fig. 4.5 An example entity with neighbours

Consider the two entities 'department' and 'dept.' in Fig. 4.4 and Fig. 4.5.

$Pa = \Sigma(e_1 A \cap e_2 A) / \Sigma(e_1 A \cup e_2 A)$
$= 3/6$
$= 0.5$

$Pk = \Sigma(e_1 K \cap e_2 K) / \Sigma(e_1 K \cup e_2 K)$
$= 1/3$
$= 0.333$

$Pv = \Sigma(e_1 V \cap e_2 V) / \Sigma(e_1 V \cup e_2 V)$
$= 0/2$
$= 0$

(Note the assumption is made here that attributes which have values automatically match on their values. This is not realistic, and a more detailed description of matching values is shown in section 4.2.2)

$Pr = \Sigma(e_1 R \cap e_2 R) / \Sigma(e_1 R \cup e_2 R)$
$= 1/5$
$= 0.2$

As shown earlier, the total SLF is calculated using equation (1) as:

$$P = W_1 \times Pa + W_2 \times Pk + W_3 \times Pv + W_4 \times Pr$$

Therefore:

$$P = W_1 \times 0.5 + W_2 \times 0.33 + W_3 \times 0 + W_4 \times 0.2$$

Assume that W_1=0.5, W_2=0.1, W_3=0.1, W_4=0.3
such that $W_1+W_2+W_3+W_4=1.0$
Then:

$P = 0.5x0.5 + 0.1x0.33 + 0.1x0 + 0.3x0.2$
$= 0.284$

The same weights can be used in the weigh and multiply method according to equation (2).

Although the two entities 'department' and 'dept.' are meant to be the same object, the object fuzzy matching yielded a total SLF of 0.268. The example was chosen by assuming that different designers modelled the corresponding views of the two entities.

The weights associated with the different neighbours were chosen based on our understanding of ERM. However, as in any fuzzy matching approach, the resultant SLF cannot be used to make a 'confident' decision regarding the type of match.

After modelling sixteen views from the Department of Computer Science, it was observed that valued attributes form a very small percentage (about 10%) of the total number of attributes. Consequently, the inclusion of values in the calculation of the total SLF of entities would distort the results. Therefore, the two methods for calculating the total SLF as shown in equations (1) and (2) above were modified to equation (3) and (4) below:

The weigh and add method:

$$P = W_1 \times Pa + W_2 \times Pk + W_3 \times Pr \quad \ldots\ldots\ldots (3)$$

The weigh and multiply method:

$$P = (Pa)^{W1} \times (Pk)^{W2} \times (Pr)^{W3} \quad \ldots\ldots\ldots\ldots (4)$$

The above methods were programmed into VI and applied to real life views (see appendix A), and the weights of the three sub SLFs were varied to find the best way of distributing them in order to achieve the most representative total SLF P.

It can be assumed that any SLF above the average of all the SLFs is an indication of a possible similarity between the objects concerned. But the total SLF is not only influenced by the sub SLFs of the corresponding neighbours, it is also influenced by the number of objects in the neighbourhood. For example, if each entity is related by only one relationship and these relationships are different, then the corresponding SLF is 0.

4.2.2 Object fuzzy matching of attributes

An attribute belongs to either an entity or a relationship, it has a key status and may have a value. Fuzzy matching of attributes can be carried out for the attributes of the same object (*intra object attributes*) or for attributes of different objects (*inter object attributes*).

4.2.2.1 Object fuzzy matching of inter object attributes

Fuzzy matching of inter object attributes, will mostly yield meaningless results. Since the objects to which the two attributes being fuzzy matched belong are different, these objects cannot be considered as a contributing factor in the calculation of the SLF. The key status and the values of the two attributes are therefore the only two characteristics which may carry any significant weight on the total SLF. Regarding the key status, it is possible for

two attributes which are different to have the same key status. Should the key status be given any weight to play in evaluating the SLF, it would be contrary to the real situation.

The value of the two attributes is possibly the best indication of any similarity between them. This is because it is unlikely that two different attributes have the same values, whether these are range or set values. This is illustrated in the two examples shown below:

Example 1:

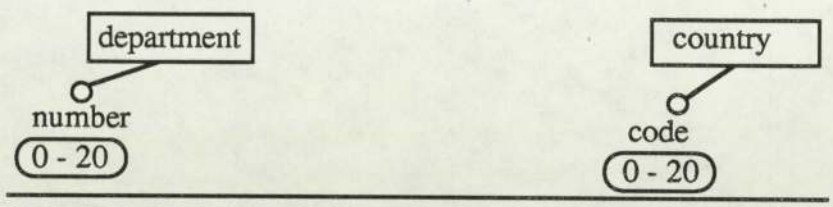


Fig. 4.6 Different entities with synonymous attributes which have identical values

Example 2:

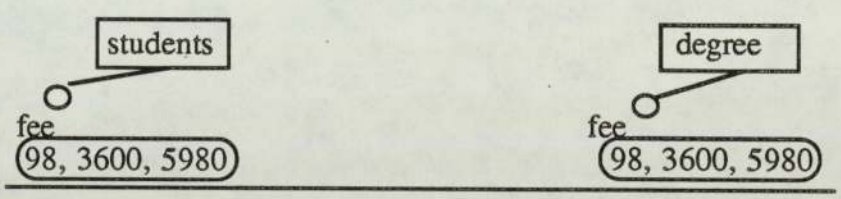


Fig. 4.7 Different entities with the same attribute which have identical values

However, the only use of calculating the SLF of inter object attributes is to help identify some possible objects which the concerned attributes define, as shown in the two examples above. The same attribute may belong to different objects, yet it may have the same name, key status and value, and therefore, inter object attribute fuzzy matching is meaningless.

4.2.2.2 Object fuzzy matching of intra object attributes

When integrating two entities from different views, their attributes are used to complement each other so that the resultant entity contains the attributes from the two entities, with the conflicts resolved (see section 3.4). When integrating entities, their attributes are accumulated in such a way that any two attributes which do not match on name are regarded as different. The two examples of Fig. 4.8 and Fig. 4.9 illustrate this problem. The attributes 'number' and 'code', and 'fees' and 'fee' are meant to be the same, but because of naming conflicts they are integrated as two different attributes in each of the two examples.

Example 1:

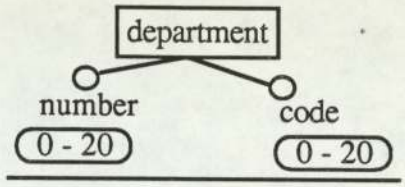


Fig 4.8 Synonymous attributes of the same entity with equal range values.

Example 2:

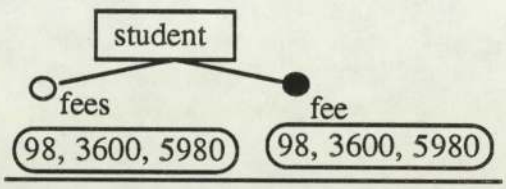


Fig. 4.9 Synonymous attributes of the same entity with equal sets of values.

An attribute can either have a set of values or a range of values. Assume that Av is the SLF of two attributes a_1 and a_2, and Ak is the SLF of the key status of these two attributes. Also assume that T is the total SLF of Av and Ak. Assuming that a match on parts of the values of a_1 and a_2 is regarded as a match, Av can either yield a total match or total difference. Av therefore, can either be equal to 0 or 1. Ak always yields a value of 0 or 1. Assume that W_1 is the weight Av plays in T, and W_2 is the weight Ak plays in T, then:

$$T = W_1 \times Av + W_2 \times Ak$$

Now assume that W_1 is put to equal 0.8 and W_2 is put to equal 0.2. Then T can have the values 0, 0.2, 0.8 or 1. Example 1 of Fig. 4.8 above would achieve:

$$T = 0.8 \times 1 + 0.2 \times 1 = 1$$

This indicates a total match.

Example 2 of Fig. 4.9 would achieve:

$$T = 0.8 \times 1 + 0.2 \times 0 = 0.8$$

Which indicates a match on values only.

The above assumption regarding W_1 and W_2 and the method of calculation, indicates that any SLF < 0.8 means a match on the key status only, and therefore must be disregarded. However, for two attributes of the same object to have the same value is regarded as a

strong indication of similarity between the two attributes, and hence a high weight. Therefore the values are given a weight=1.

4.2.3 Object fuzzy matching of relationships and E-Rs

Object fuzzy matching of relationships includes the neighbours of these relationships, which are the relationship attributes (if any) and the entities related by these relationships. Since relationship attributes are not commonly modelled, the final result will depend on the entities. However, the same two entities can be related by more than one relationship, and thus the object fuzzy matching will yield very similar results for all the relationships relating the same two entities.

Example:

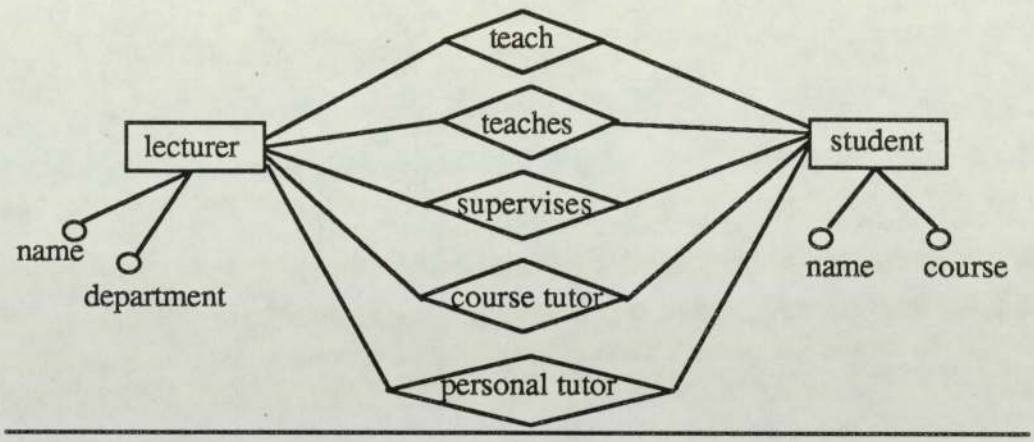


Fig. 4.10 Examples of relationship neighbours

Consider the example schema in Fig. 4.10, which is the result of integrating a number of views. By studying these relationships it can be established that only three are valid. The VI regards the two relationships 'teach' and 'teaches' as being different, and hence they are included in the schema as such. The same applies for 'course tutor' and 'personal tutor', assuming here that they are meant to be the same. The object fuzzy matching of these five relationships will yield the same result. Therefore, the application of object fuzzy matching for relationships involving the same two entities is not useful in finding synonyms. The same relationships with the synonym naming conflict and involving the same two entities will remain as undetectable conflicts.

Now let us examine object fuzzy matching of relationships involving one common entity. Consider the situation in Fig. 4.11. Since e_1 is common to both r_1 and r_2, it cannot be considered in the evaluation of the total SLF, as it will contribute the same amount to the SLFs of both relationships. Regarding e_2 and e_3, if they are similar, this would have been

established from the object fuzzy matching of entities. Since they are different, the SLF between them is most likely to be low. Therefore, object fuzzy matching of relationships with one common entity would not normally give a good indication of their similarity level.

Object fuzzy matching of relationships with totally different entities such as those shown in Fig. 4.12 would produce SLFs totally dominated by the object fuzzy matching of their entities. Since the entities are different, the total SLF is once again not representative.

Object fuzzy matching of E-Rs suffers from the same drawback as the object fuzzy matching of relationships.

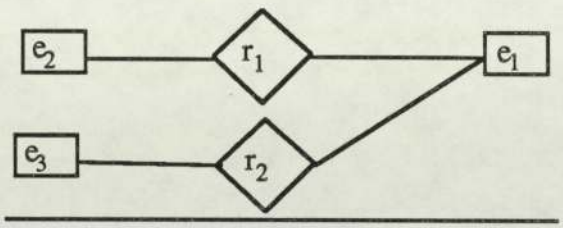


Fig. 4.11 E-Rs with a common entity

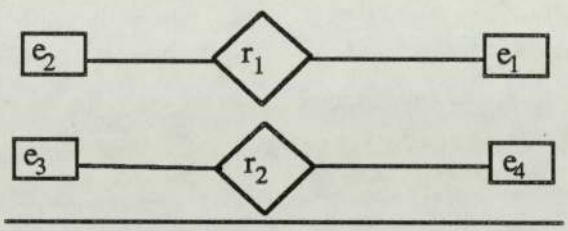


Fig. 4.12 Totally different E-Rs

4.3 COT conflicts

Although ERM objects should in theory have unique definitions (Ng *et al* 1980, Lien 1980 and Markowitz 1984), in practice it is possible for different designers to model the same semantics as two different objects and in different structures. In theory each ERM object is supposed to accommodate certain types of semantics. In practice, semantics can be modelled in a variety of ways, and these objects are used almost interchangeably. For example, attributes can be modelled as entities, entities can be modelled as attributes, attributes can be modelled as relationships, relationships can be modelled as attributes, and so on.

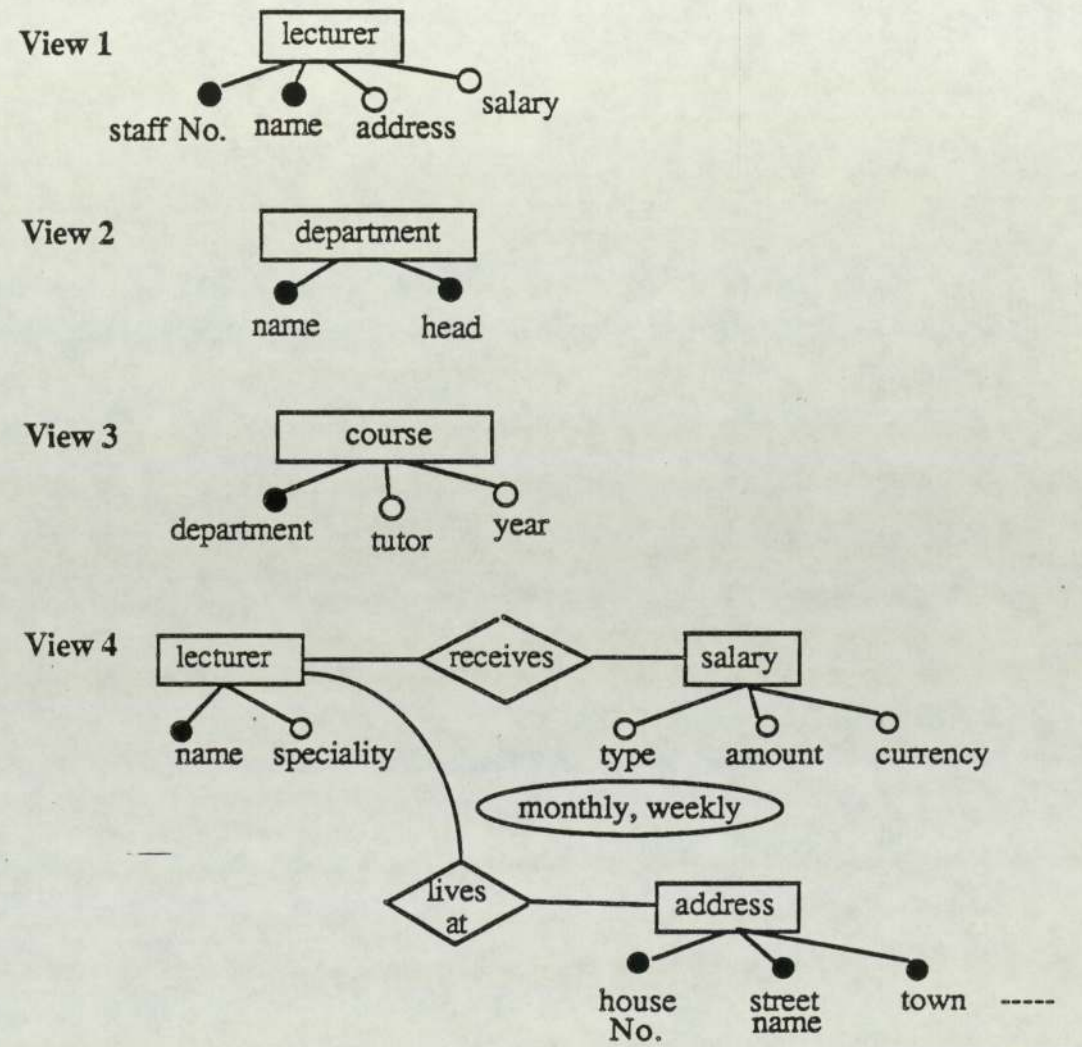


Fig. 4.13 Examples of views modelled with COT conflicts

Consider the example of the four different structures shown in Fig. 4.13, modelled in different views. These structures illustrate a sample of the possible COT conflicts which can be identified in view integration. The entity 'department' in view 2 is modelled as an attribute in view 3. View 1 consists of the entity 'lecturer' and its associated attributes, these include the attributes 'address' and 'salary', and both of these attributes, are

modelled in view 4 as two separate entities. These entities are related to the entity 'lecturer'. In view 1, the attribute 'salary' would be needed to represent the amount of salary domain of the entity 'lecturer'. In view 4, which is possibly part of the 'pay-roll' or 'personnel' view, the 'salary' and 'address' form individual entities related by relationships.

By identifying the COT conflicts, it is possible to allow either the designer or the VI to determine which of these objects can be left in the GCS, and which must be unified or deleted. For example, an attribute existing as an entity COT conflict can be accommodated and would cause no problems in both data representation and queries, provided it is only the name which is the same and not a duplication of semantics. However, an entity existing as a relationship COT conflict cannot be allowed to coexist in the GCS, because both the relationship and the entity would exist as relations under the same name.

4.3.1 COT conflicts vs transformation of objects

The transformation of ERM objects from one type to another is possible, for the same reasons that it is possible to model the same semantics using different object types and structures. The transformation of objects from one type to another is not the same as COT conflict analysis. The latter was not considered in the view integration literature reviewed in chapter 2.

Since the two object types exist in the GCS, the difficulty lies in deciding:

1 Which object type is to override the other?

2 What semantics from the overridden object type are to be transferred to the other object type?

3 How can we be sure that the resultant structure or object contains the correct and complete semantics from both views?

Transformation is not needed for COT conflicts, as both object types exist in the GCS, but under the same name. In COT conflicts, which are not caused by naming conflicts, one of the objects must be eliminated and the other kept in the GCS. Further, the semantics from the eliminated object must be transported to the target object. Object type transformation is therefore not usually a necessary part of view integration, but when it is needed, it can only be requested by the designer.

4.3.2 Attribute-value as E-R / same E-R

Conflict definition

An *attribute-value as E-R same E-R* COT conflict occurs when there is an attribute a_1 of an entity e_1, such that a_1 is also modelled as a relationship r, and a_1 has a set of one or more values $V=\{v_x \in X\}$, such that one of these values is also modelled as an entity e_2, and e_1 and e_2 are related by the relationship r.

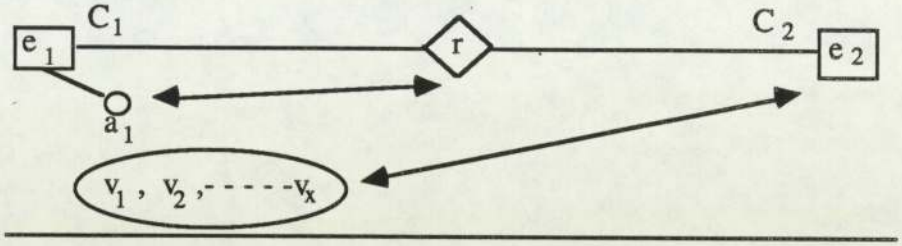


Fig. 4.14 Attribute-value as E-R / same E-R conflict

This conflict is caused when one designer models the semantics as a valued attribute of a given entity, whilst the other models the same semantics as an E-R. Whilst both structures are valid in their own view, once integrated, the two views result in an E-R with both the attribute-value and the E-R coexisting in the same E-R structure. This conflict is not caused by a naming conflict.

The coexistence of both structures in the GCS causes duplication and inconsistency. Therefore, the VI must either unify both representations into one, or choose one of them so that the resultant structure contains the correct and complete semantics. However, there is difficulty in deciding which of the two structures is more semantically expressive and correct. Semantic expressiveness is a measure of the final number of domains in the resultant structure. The E-R structure would contain more domains, as well representing a semantic connection between the two entities, and therefore the E-R structure is preferred. The attribute-value structure represents the value constraint on the instances of the attribute domain, but this is indirectly represented in the E-R structure in the form of the number of entities. The steps to achieve this resolution are given below:

1 Remove from attribute a_1 the value $v_{x \in X}$ such that $v_{x \in X} = e_2$.

2 If the set of values V of a_1 is empty ($V=\Phi$), then remove the attribute a_1 from the entity e_1, provided there is another key attribute in e_1. If e_1 has no key attributes once a_1 is removed, then inform the designer.

3 If after the removal of $v_{x \in X}$ such that $v_{x \in X}=e_2$, V is still not empty, this could mean any of the following:

   a) The rest of the values are synonymous with the entities already related by r, including e_1.

b) The rest of the values must be created as entities, involved with e_1 by r.

c) The remaining values are genuine attribute values, and therefore are kept with their attribute, as part of the definition of the entity e_1.

Consider the two views in Fig. 4.15. In view 1, the entity 'course' has the attribute 'course level' and the latter has the values 'postgraduate' and 'undergraduate'. In view 2, the entity 'course' is modelled as part of an E-R structure with a relationship called 'course level'. The latter also involves two other entities called 'postgraduate' and 'undergraduate'. As a result of integrating the two views, the resultant E-R shown in Fig. 4.16 is produced. This E-R contains the attribute-value as an E-R COT conflict, where the attribute 'course level' of the entity 'course' also exists as a relationship of the same name, and the attribute values 'postgraduate' and 'undergraduate' also exist as entities related by the relationship 'course level'. As discussed earlier, in COT conflicts of this kind, the E-R structure is preferred to the attribute-value structure. Therefore, since the entity 'course' has another key attribute, the attribute 'course level' and its associated values are removed from the entity, and the final E-R is as shown in Fig. 4.17.

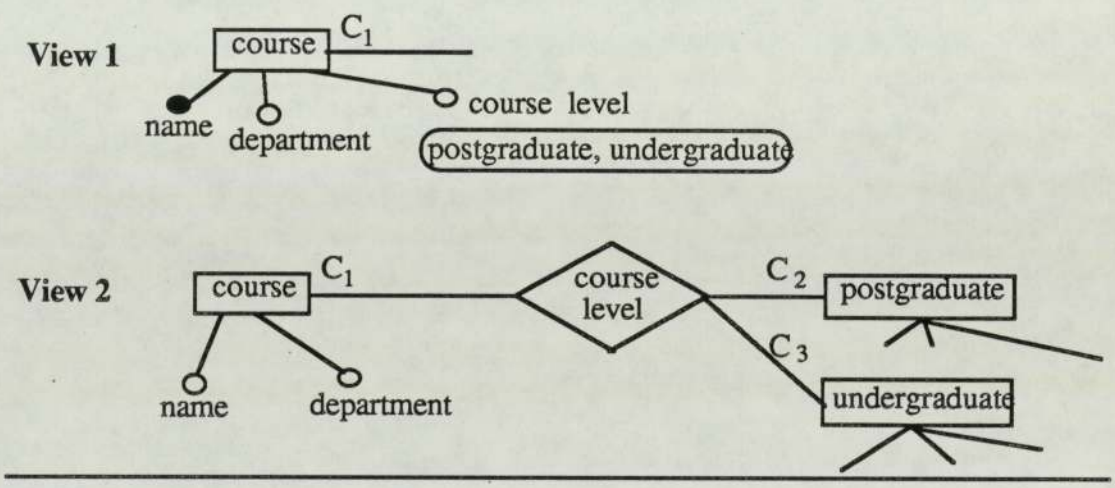


Fig. 4.15 Modelling the same semantics in attribute-value structure and E-R structure

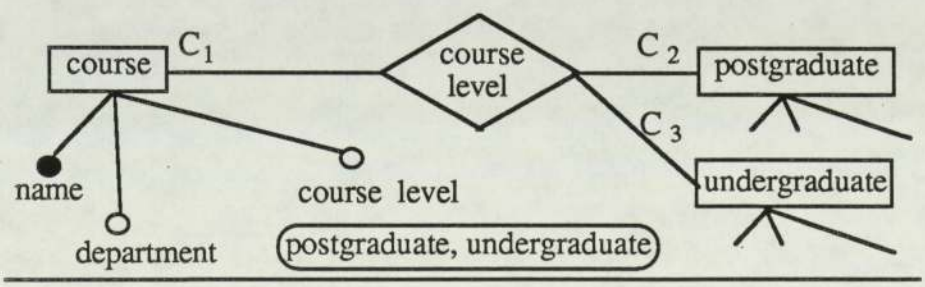


Fig. 4.16 An example of attribute-value as E-R conflict

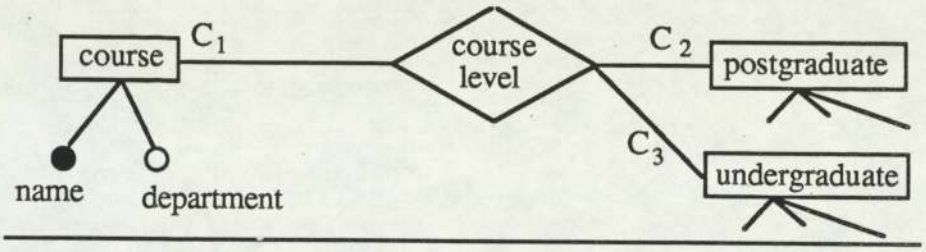


Fig. 4.17 Schema after removing attribute-value as E-R conflict

Now, assume that the set of values of this attribute is still not empty. This situation is caused by one of two possibilities:

a) The remaining values are synonymous to the values which exist as entities.

Based on the example of Fig. 4.15, consider the situation shown in Fig. 4.18. Due to the abbreviation by one designer in one view, the VI accumulated the values of the attribute 'course level' as shown (see section 3.4). In the COT conflict analysis, the VI would have identified the attribute-value as E-R COT conflicts of the two values 'postgraduate' and 'undergraduate' as shown earlier. However, the attribute 'course level' cannot be deleted as it still contains the values 'postgrad.' and 'undergrad.'. The VI can recommend that the values are synonymous with the other values which were deleted, and already exist as entities. However, this is only a possibility and not necessarily a fact. If the designer is satisfied that they are synonyms, then the values are deleted, and if not then case (b) below is considered.

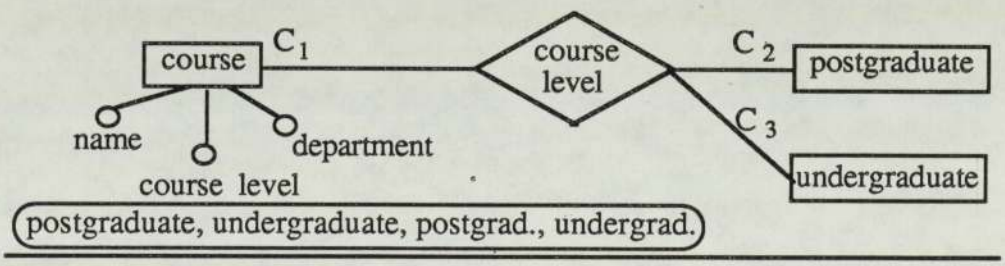


Fig. 4.18 Synonymous values in attribute-value as E-R

b) The remaining values of the attribute can be created as entities.

Assume that the resultant E-R is as shown in Fig. 4.19. Although the values 'postgraduate' and 'undergraduate' of the attribute 'course level' were deleted in accordance with point (a) above, the attribute still has the value 'research', which does not exist as an entity. This remaining value could possibly be created as a new entity related by the relationship 'course level', in the same way as the other two entities. However, although the VI would identify the conflict and recommend this resolution, the final

decision is again left to the designer. If the designer chooses to create the value 'research' as a new entity, then the attribute 'course level' would be removed from the relationship, and the final relationship is as shown in Fig. 4.20. In this case, the attributes and the cardinality of the newly created entity would be similar to the other two entities, but in any case would have to be supplied by the designer.

c) The attribute and its remaining values are kept as part of the entity definition.

If the situation in both points (a) and (b) above are not satisfied, then the attribute must be kept with its remaining values as part of the definition of the entity e_1. Although it is possible to change the name of the attribute, the database is not affected by not changing it. The attribute 'course level' would define one of the domains of the entity relation e_1 and would not conflict with the name of the relationship 'course level'. However, confusion may be caused to the user, if unfamiliar with this structure. If the user wants to add a new course level to the attribute 'course level', such an addition cannot be accommodated. Therefore, special effort must be made to avoid the existence of this type of naming conflict.

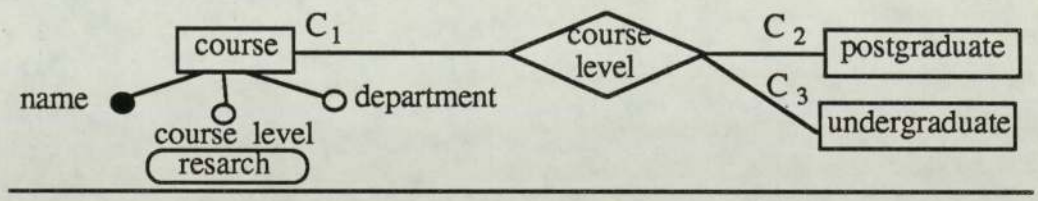


Fig. 4.19 Values which could be made into entities

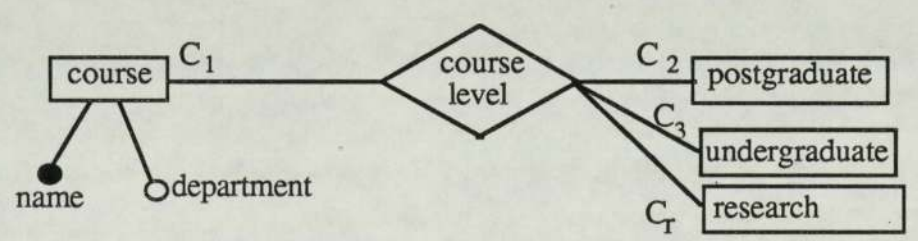


Fig. 4.20 Schema after transforming values into entities

4.3.3 Attribute-value as E-R / different E-Rs

Conflict definition

An *attribute-value as E-R /different E-Rs* COT conflict occurs when there is an attribute a_1 of an entity e_1, such that a_1 is also modelled as a relationship r_2, and a_1 has a set of one or more values $V=\{v_x \in X\}$, such that one of these values is also modelled as an entity e_3, but e_1 and e_3 are not related by the relationship r_2.

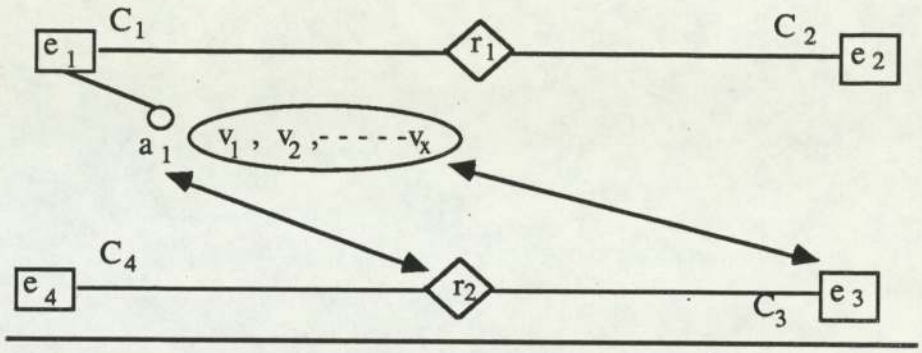


Fig. 4.21 Attribute-value as E-R / different E-Rs

Fig. 4.21 shows a general situation of this type of conflict, and Fig. 4.22 shows an example of two E-Rs in the GCS with the attribute-value as E-R/ different E-Rs COT conflict. The existence of this kind of conflict could mean either or both of the following:

1 e_1 and e_4 could be synonymous.

2 e_1 should be related to e_3 by r_2.

The E-Rs in Fig. 4.22 are part of a GCS, and were chosen to be representative of this type of conflict. The first possibility, according to point 1 above, is that the entities 'course' and 'courses' are synonymous. The VI is not able to detect this type of naming conflict, and therefore the two entities are included in the GCS as different. In this particular example the VI makes the recommendation to the designer that these entities are synonymous. If this is accepted, then the VI can integrate the two entities as one. The names 'course' and 'courses' are both associated with the same entity (see section 5.6), and the attributes of the two entities are used to complement each other to define the new entity. This discovery of synonymous entities is a reward of the identification of this COT conflict.

The identification of the synonymous entities above does not alleviate the attribute-value as E-R/different E-Rs COT conflict. According to point 2 above, e_1 should be related to e_3 by r_2. This can be achieved in the same way as that described in section 4.3.2 above, where the E-R structure is preferred to the attribute-value structure. Therefore, if the two entities 'course' and 'courses' were declared synonymous in the GCS, the resultant GCS is as shown in Fig. 4.23. If the two entities 'course' and 'courses' were not declared synonymous, then the resultant GCS is as shown in Fig. 4.24. The cardinality of the entity 'courses' related by the relationship 'course level' must be determined by the designer.

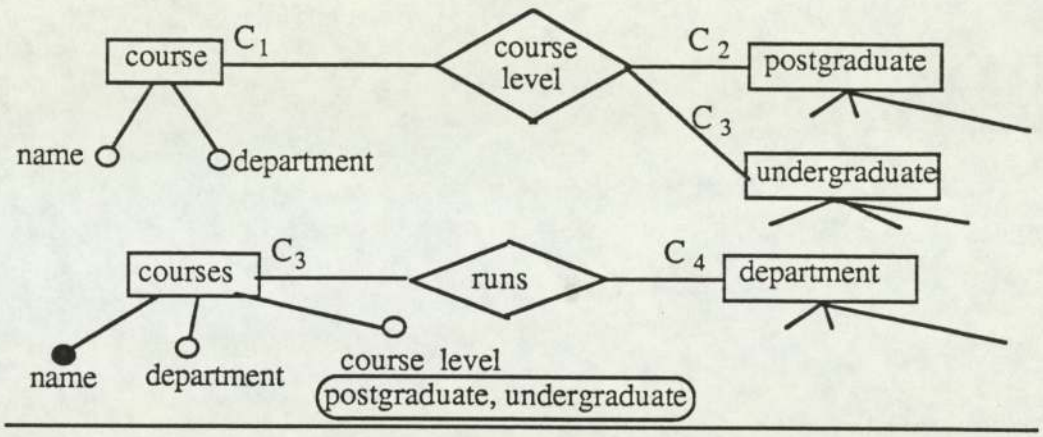


Fig. 4.22 An example of attribute-value as E-R / different E-Rs

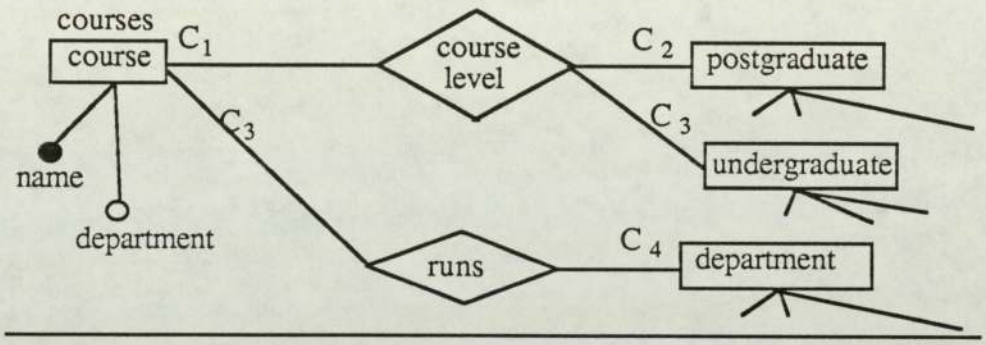


Fig. 4.23 Identifying synonymous entities after analysis of attribute-value as E-R / different E-Rs COT conflict

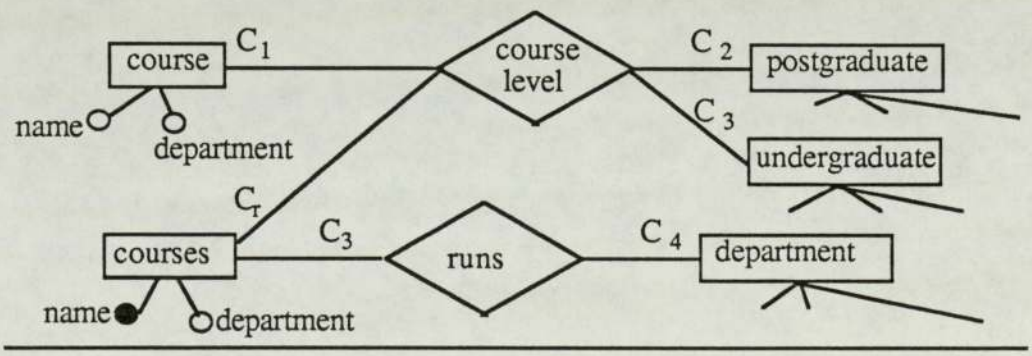


Fig. 4.24 Schema after attribute-value as E-R / different E-Rs

4.3.4 Valued attribute as relationship / same E-R

Conflict definition

A *valued attribute as relationship / same E-R* COT conflict occurs when there is an attribute a_1 of an entity e_1, and a_1 has a set of values $V=\{v_1, v, ..., v_x\}$ such that a_1 is also modelled as a

relationship r, and e_1 is related by the relationship r, but the values of the attribute does not equal an entity.

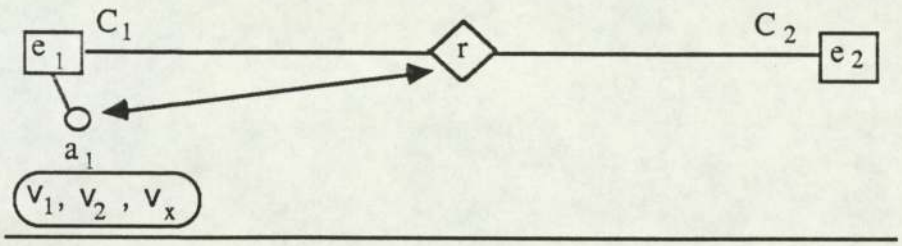


Fig 4.25 Valued attribute as relationship / same E-R

Since the value(s) of the attribute do not exist as entities, the value as entity COT conflict (see section 4.3.10) is ruled out. Further, the attribute has value(s), and this rules out the attribute as relationship COT conflict (see sections 4.3.6 and 4.3.7). The assumption here is that the value(s) are not numeric. The order of identification and resolution of this conflict in relation to the attribute-value as E-R, value as entity and attribute as relationship COT conflicts is important. The correct sequence of their identification and analysis is:

1 Attribute-value as E-R
2 Value as entity
3 Valued attribute as relationship
4 Attribute as relationship

The resolution(s) to the valued attribute as relationship/ same E-R is similar to the attribute-value as E-R /same E-R conflict. They are both caused as a result of modelling (and consequently integrating) the same semantics either as a valued attribute structure or as an E-R structure. Further, the same applies in both type of conflicts in that only one of the structures can be left in the schema. Further, the E-R structure is preferred to the attribute-value structure and the valued attribute structure.

The most likely resolution to the valued attribute as relationship/ same E-R conflict is that one or more of the values are synonyms to e_2. If this is the case, then the three step resolution described in section 4.3.2 to resolve the attribute-value as E-R apply here.

4.3.5 Valued attribute as relationship / different E-R

Conflict definition

A *valued attribute as relationship / different E-Rs* COT conflict occurs when there is a valued attribute a_1 of an entity e_1, and a_1 has a set of values $V=\{v_1, v, ..., v_x\}$ such that a_1 is also modelled as a relationship r_2, and e_1 is not related by the relationship r, but the values of the attribute does not equal an entity.

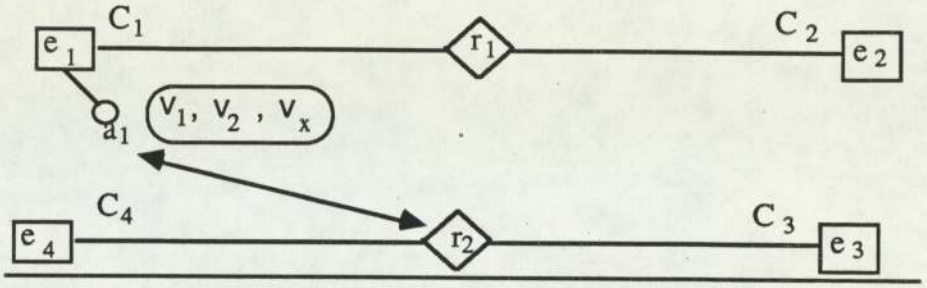


Fig 4.26 General valued attribute as relationship / same E-R

The order in which this conflict must be analysed is the same as shown in section 4.3.4. The possible resolutions to this conflict are similar to those described for the attribute-value as E-R /different E-Rs conflict. In this particular conflict, e_1 could be synonymous to either e_4 or e_3. However, it is also possible that one or more of the the values are synonymous to either e_4 or e_3. If this is the case, then the E-R structure is favoured over the valued attribute structure.

4.3.6 Attribute as relationship / same E-R

Conflict definition

An *attribute as relationship /same E-R* COT conflict occurs when there is an attribute a_1 of an entity e_1 , such that a_1 is also modelled as a relationship r, and e_1 is related by the relationship r.

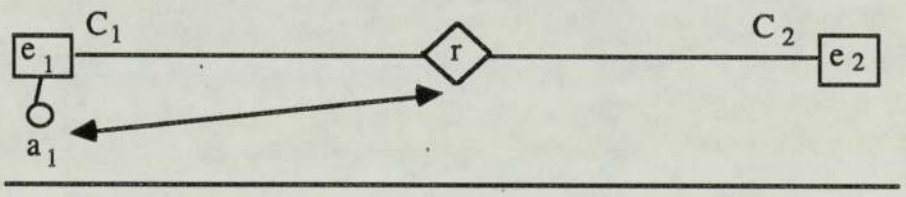


Fig. 4.27 General attribute as relationship / same E-R

This conflict could be caused either by integrating two or more occurrences of the entity e_1 from different views or it could be modelled as such in the same view. In modelling the semantics of the entity e_1, the designer might have decided that a domain represented by the attribute a_1 is sufficient, whilst in another view, more knowledge was identified to represent this domain in the form of an E-R. The existence of this conflict could mean any of the following:

1 The domain represented by the attribute a_1 represents duplicate semantics, and therefore its instances would be directly or indirectly embedded in the instances of the domains of the entity e_2 which is related by the relationship r.

2 The conflict is a genuine naming conflict, and therefore, one of the names needs to be changed.

From experience, it was found point 1 above is normally true. If the designer accepts that this is the case, then VI deletes the attribute a_1 . In the example GCS shown in Fig. 4.28, the attribute 'course level' exists as a relationship of the same name. The domain of this attribute would be represented indirectly by the entity 'postgraduate' and its attributes. Therefore, the attribute 'course level' is removed from the entity 'course'.

If the designer decides that this is a naming conflict and not a structural conflict, then the designer may wish to change either or both names, and the structure of the E-R is maintained. However, it is not necessary to change the names of either the relationship or the attribute, as the common name between the attribute and the relationship would cause no problems when querying the database.

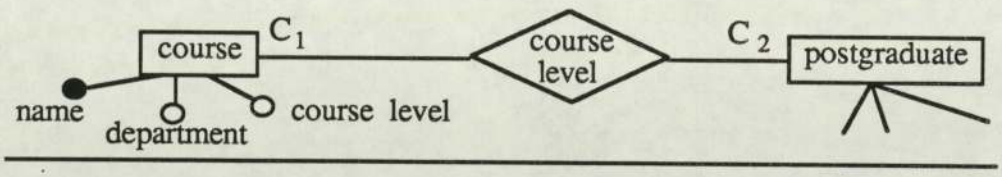


Fig. 4.28 An example of attribute as relationship / same E-R

4.3.7 Attribute as relationship / different E-Rs

Conflict definition

An *attribute as relationship/different E-Rs* COT conflict occurs when there is an attribute a_1 of an entity e_1 , such that a_1 is also modelled as a relationship r_2 , but e_1 is not related by the relationship r_2 .

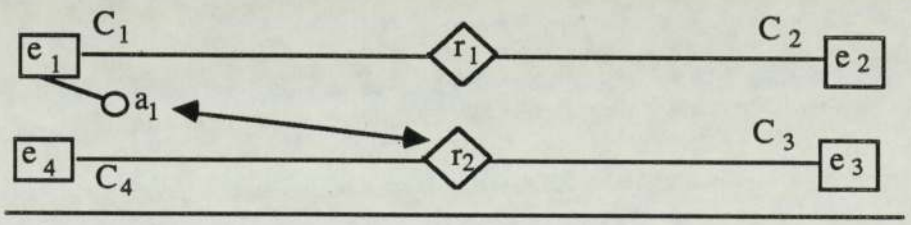


Fig. 4.29 General attribute as relationship / different E-Rs

Fig. 4.29 shows a general situation of this type of conflict, and Fig. 4.30 shows an example of two E-Rs in the GCS with the attribute as relationship/ different E-Rs COT conflict. The existence of this kind of conflict could mean any of the following:

1 e_1 and e_4 or e_1 and e_3 could be synonymous.

2 e_1 could be related to e_3 or to e_4 by r_2 .

3 Either the name of the attribute or the name of the relationship should be changed.

The E-Rs in Fig. 4.30 were especially chosen to be representative of this kind of conflict. The first possibility according to point 1 above is that the entities 'course' and 'courses' are synonymous. VI cannot detect this kind of naming conflict, and therefore the two entities are included in the GCS as different. In this particular example VIM makes the recommendation that these entities are synonymous. If the designer accepts the existence of this naming conflict, VI can then integrate the two entities. The names 'course' and 'courses' are both associated with the same entity, and the attributes of the two entities are used to complement each other to define the new entity (see section 3.4).

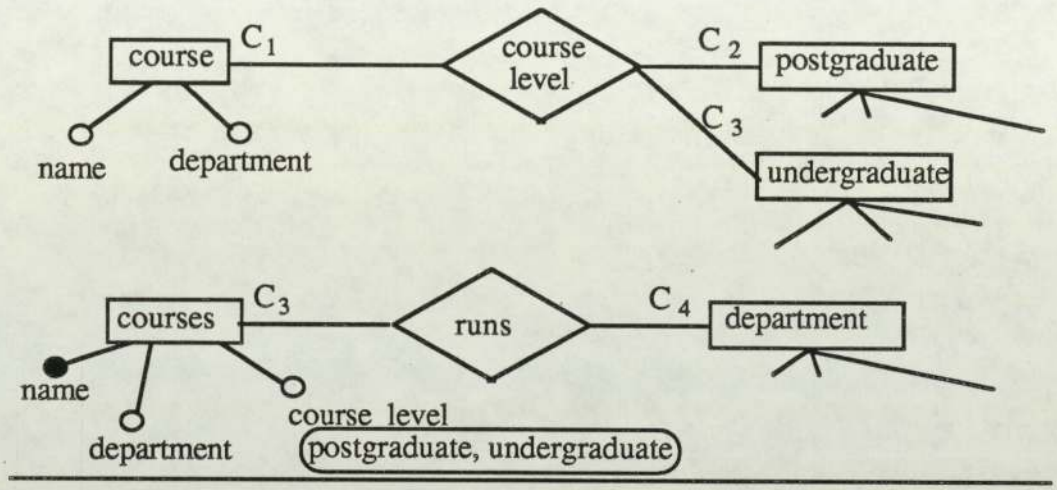


Fig. 4.30 An example of attribute as relationship / different E-Rs

The identification of the synonymous entities above does not alleviate the attribute as relationship/different E-Rs COT conflict. According to point 2 above, e_1 should be related to either e_3 or e_4 by r_2. This can be achieved in the same way as in section 4.3.3 above, and therefore the E-R structure is favoured over the attribute structure. The description of the full process was given in section 4.3.3. Therefore, if the two entities 'course' and 'courses' were declared synonymous in the GCS, the resultant GCS is as shown in Fig. 4.23. If the two entities 'course' and 'courses' were not declared synonymous, then the resultant GCS is as shown in Fig. 4.24. The cardinality of the entity 'courses' related by the relationship 'course level' must be determined by the designer.

If neither of points 1 or 2 are acceptable to the designer, then the VI makes the recommendation that either the name of the relationship or the name of the attribute should be changed. Again it is not vital to change either of the names.

4.3.8 Entity as attribute / own entity

Conflict definition

An *entity as attribute/own entity* COT conflict occurs when there is an attribute a_1 which is also modelled as the entity e_1 such that a_1 defines e_1 .

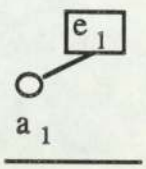


Fig. 4.31 Entity as attribute / own entity

An illustration of this conflict is shown in Fig. 4.31. This is not a common conflict, because a mistake like this is not usually made when view modelling. However, should such a conflict be encountered, the cause is most likely a naming conflict. If this is accepted as a naming conflict, then either or both of the names could be changed.

4.3.9 Entity as attribute / foreign entity

a) Entities have a common relationship

Conflict definition

An *entity as attribute/foreign entity* with common relationship COT conflict occurs when there is an attribute a_1 of an entity e_1 , such that a_1 is also modelled as an entity e_2 , and e_1 is involved with e_2 by a relationship r.

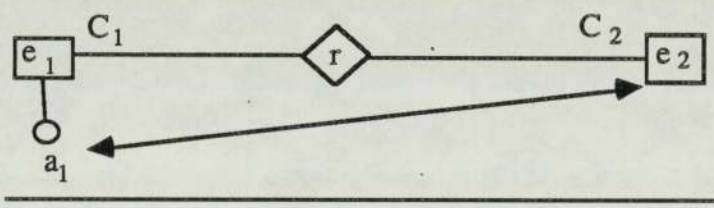


Fig. 4.32 Entity as attribute / foreign entity and a common relationship

An illustration of this conflict is shown in Fig. 4.32. This conflict is most likely to be caused by integrating objects from different views, where the designer in one view considers that a domain represented by an attribute of an entity is sufficient for the semantics needed at that view, whilst in another view an entity with its attributes is deemed necessary. However, it is also possible that this could be a naming conflict and is modelled as such in the same view.

Therefore, the existence of this conflict could mean:

1 There is a genuine naming conflict between a_1 and e_2.

2 There is a structural conflict, and the attribute a_1 represents a duplication of semantics in the E-R (e_1 r e_2)

In Fig. 4.33, the attribute 'course' of entity 'student' is also modelled as an entity 'course', and the entity 'student' is involved with the entity 'course' by the relationship 'enrolled on'. The domain of the attribute 'course' is repeated in the entity 'course' represented by the attribute 'course name'. This causes problems when the instances of the domain of either the attribute 'course' or the attribute 'course name' are updated. If this recommendation (point 2 above) is accepted, then the attribute 'course' is deleted from the entity 'student'. If this is a key attribute, then the designer must be informed.

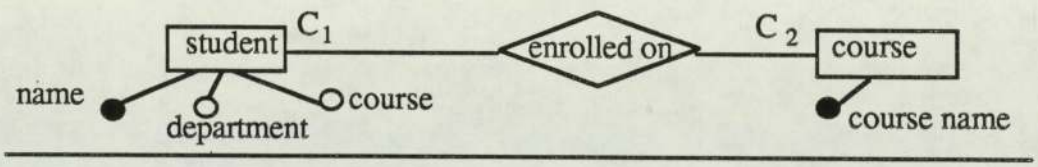


Fig. 4.33 An example of entity as attribute / foreign entity and a common relationship

If the conflict is a genuine naming conflict, then either the name of the attribute or that of the entity could be changed. This is however not necessary.

b) Entities have no common relationship

Conflict definition

An *entity as attribute/foreign entity* with no common relationship COT conflict occurs when there is an attribute a_1 of an entity e_1, such that a_1 is also modelled as an entity e_3, and that e_1 is not involved with e_3 by any relationship.

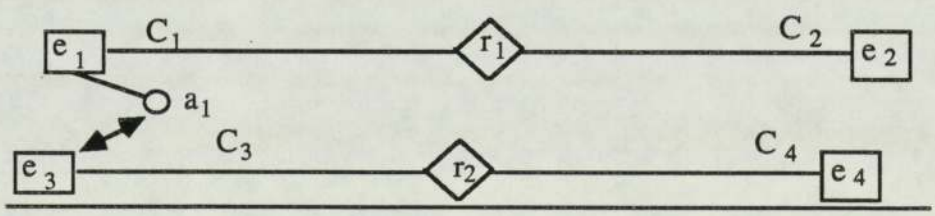


Fig. 4.34 Entity as attribute / foreign entity and no common relationship

An illustration of this conflict is shown in Fig. 4.34, and an example is shown in Fig. 4.35. This conflict is most likely to be caused when the designer does not model all the possible relationship(s) between the entities concerned. The main reason for the missing relationships is that in view modelling, certain boundaries between views may not have been identified for modelling. Therefore, although it is possible that the attribute a_1 and the entity e_2 have a naming conflict, it is most likely that this COT conflict is a structural

one, where a relationship between e_1 and e_3 needs to be declared. In the absence of a relationship between e_1 and e_3, the attribute a_1 plays the role of a reference attribute in e_1 for e_3.

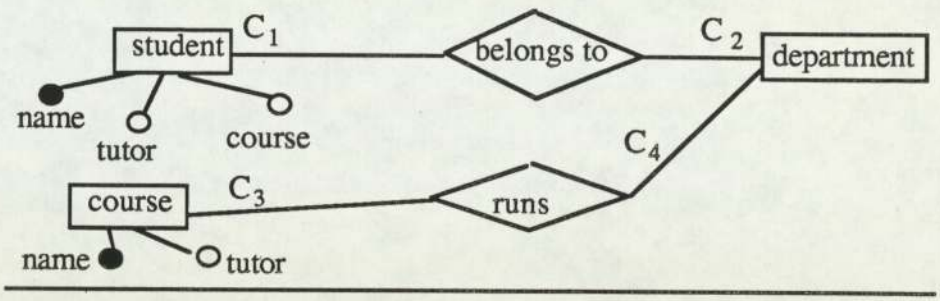


Fig. 4.35 An example of entity as attribute / foreign entity and no common relationship

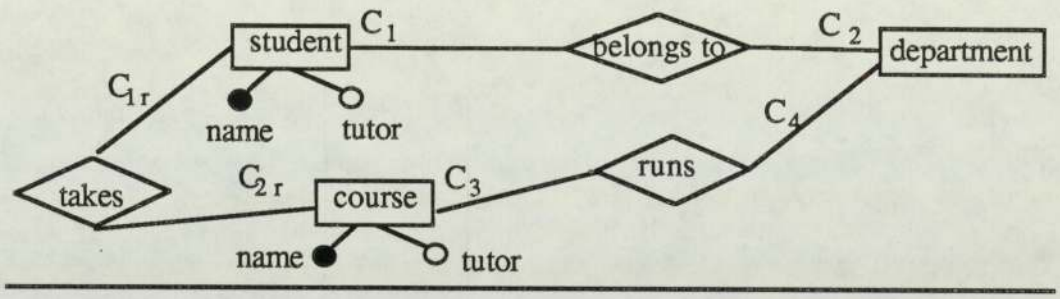


Fig. 4.36 Creating a new E-R as a result of the entity as attribute / foreign entity conflict

In the example of Fig. 4.35, the attribute 'course' of the entity 'student' is also modelled as an entity 'course', and the entity 'student' is not involved with the entity 'course' by any relationship. The most likely recommendation, based on the existence of this conflict, is that a relationship between the entities 'course' and 'student' should be created. If such a recommendation is acceptable to the designer, then a relationship 'takes' may be created as shown in Fig. 4.36, and the attribute 'course' could be deleted from entity 'student'. If this is not acceptable, then there is a possibility that either the attribute 'course' or the entity 'course' are wrongly named.

4.3.10 Value as entity / own entity

Conflict definition

A *value as entity/own entity* COT conflict occurs when there is an attribute a_1 of an entity e_1, such that a_1 has a set of values V, and there is a value in v_x such that V matches e_1 on name.

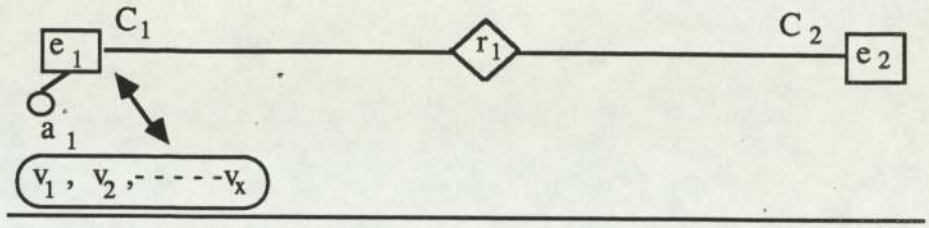


Fig. 4.37 Value as entity / own entity

An illustration of this conflict is shown in Fig. 4.37. This is a most unlikely conflict, and if it ever takes place, it is likely to be a naming conflict and not a structural conflict. If it is accepted as a naming conflict, then either or both of the names have to be changed. The coexistence of the value of the attribute as the instances of the domain represented by the attribute a_1 in the entity relation e_1 is structurally acceptable.

4.3.11 Value as entity / foreign entity

a) Entities have a common relationship

Conflict definition

A *value as entity/foreign entity* with common relationship COT conflict occurs when there is an attribute a_1 of an entity e_1, and a_1 has a set of values V, such that one of the values of V is also modelled as an entity e_2, such that e_1 is involved with e_2 by a relationship r.

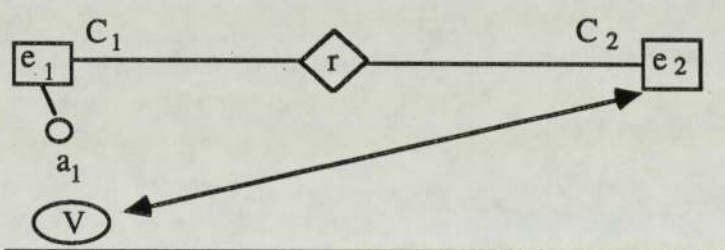


Fig. 4.38 Value as entity / foreign entity and a common relationship

An illustration of this conflict is shown in Fig. 4.38, and an example is shown in Fig. 4.39. This conflict is most likely caused by integrating objects from different views, where the designer in one view considers that a domain represented by an attribute of an entity is sufficient for the semantics needed at that view, whilst in another view an entity with its attributes is deemed necessary. Added to that is a naming conflict which exists between the attribute a_1 and the the relationship r. In Fig. 4.39, the attribute 'course level' has the values 'postgraduate' and 'undergraduate'. At the same time the entity 'course' is involved with the entities 'postgraduate' and 'undergraduate' by the relationship 'level'. As illustrated by this example, it can be noticed that this conflict is similar to the attribute-value as E-R COT conflict. The only difference is that a naming conflict exists between

the attribute 'course level' and the relationship 'level'. If the designer accepts this recommendation, then a similar procedure to that of the attribute-value as E-R conflict is followed (see section 4.3.2), and the attribute and its values are handled in the same way. Since according to this recommendation the attribute is removed from the entity, it is possible that the designer may wish to either change the name of the relationship.

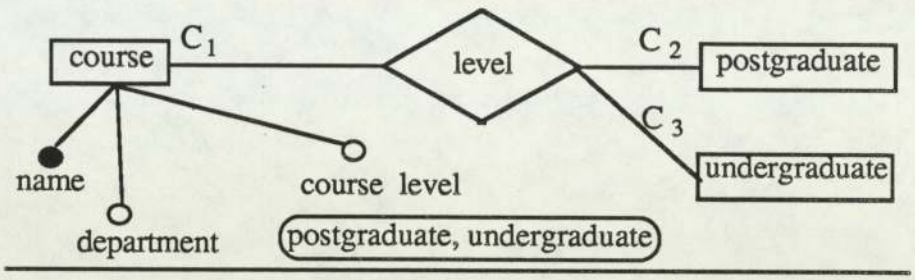


Fig. 4.39 An example of value as entity / foreign entity and a common relationship

b) Entities have no common relationship

Conflict definition

A *value as entity/foreign entity* with no common relationship COT conflict occurs when there is an attribute a_1 of an entity e_1, and a_1 has a set of values $\{v_1, v_2, ..., v_x\}$, such that a_1 is also modelled as an entity e_4, and e_1 is not involved with e_4 by any relationship.

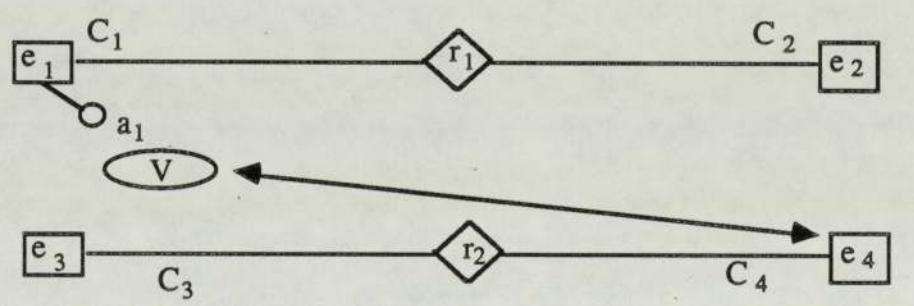


Fig. 4.40 Value as entity / foreign entity and no common relationship

This resolution of this COT conflict could be in one of the following ways:

1 e_1 is synonymous to e_4 and must therefore be integrated.

2 A relationship need to be created to relate e_1 to e_4. The name of this relationship coûld be the name of the attribute a_1.

An illustration of this conflict is shown in Fig. 4.40, and an example is shown in Fig. 4.41. This conflict can take many forms, but the example shown in Fig. 4.41 is the most common and representative. A value associated with an attribute is a constraint on the instances of the domain represented by this attribute, and therefore the existence of such a value as a foreign entity is not necessarily a structural conflict. The existence of the value 'Ph.D' in the set of values of the attribute 'name' of entity 'degree' as an entity raises the

possibility that a relationship needs to be created to relate entity 'degree' and entity 'Ph.D'. If such a recommendation is acceptable to the designer, then it is most likely that the other values, which are 'Diploma', 'M.Sc.', and 'B.Sc.', could also either already exist as entities, or need to be created as entities. If these remaining values already exist as entities, then a relationship needs to be created to relate them with the entity 'degree', and the name of this relationship for all four values (newly created entities) is most likely to be the name of the attribute, which in this case is 'name'. If any of these values already exist as an entity, and is related to the entity 'degree' by a relationship, then the name of this relationship could be given to all the newly created entities. The example relationship of Fig. 4.42 might already be part of the GCS, and if so, the relationship name 'type' is the name to be given to the relationship relating all the newly created entities.

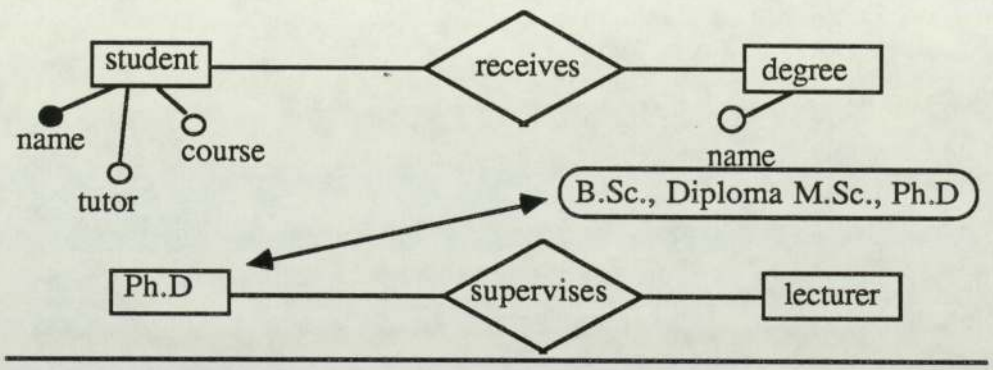


Fig. 4.41 An example value as entity / foreign entity and no common relationship

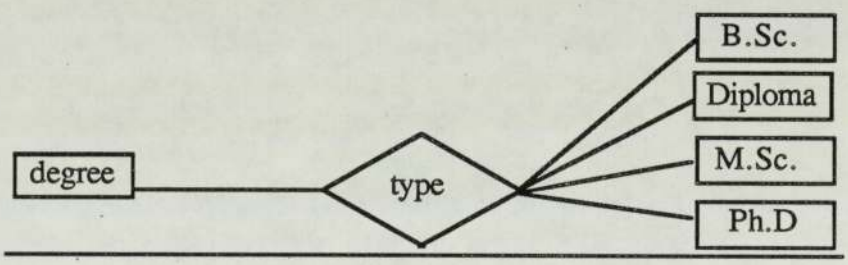


Fig. 4.42 Creating new E-Rs as a result of value as entity / foreign entity conflict

4.3.12 Entity as relationship/ own E-R

Conflict definition

An *entity as relationship / own E-R* COT conflict occurs when there is an entity e_1 related by a relationship r, such that e_1 matches r on name.

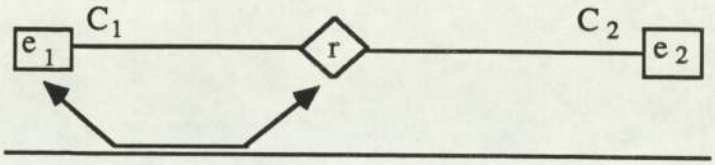


Fig. 4.43 Entity as relationship/ own E-R

This is a very unlikely COT conflict. It is not caused by integration, and if it takes place, it is most likely a naming conflict between e_1 and r. The only recommendation which could be made by VI is to request the designer to change the name of the entity or the relationship or both. This naming conflict must be resolved by the designer, for if it is left in the GCS, two relations of the same name would coexist in the database.

4.3.13 Entity as relationship / foreign E-R

a) common entity

Conflict definition

An *entity as relationship/foreign E-R* with common entity COT conflict occurs when there is a relationship r_1 of an E-R (e_1 r_1 e_2), such that r_1 is also modelled as an entity e_3, and e_3 belongs to the E-R (e_2 r_2 e_3), and e_3 is not related by r_1.

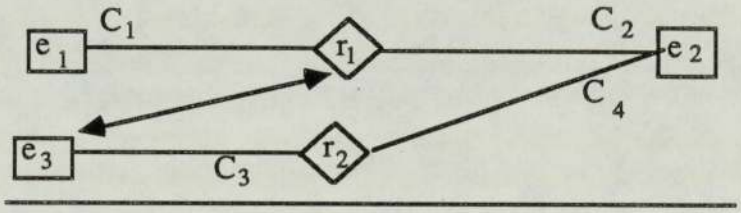


Fig. 4.44 Entity as relationship / foreign E-R

The general situation of this conflict is shown in Fig. 4.44, and two examples are shown in Fig. 4.45 and Fig. 4.46. This is a complex COT conflict, and its existence usually indicates that the relationships r_1 and r_2 are synonymous. In the example of Fig. 4.45 the relationships 'course tutor' and 'activity' are possibly synonymous, and in the example of Fig. 4.46, the relationships 'lives at' and 'residence' are also possibly synonymous. In either case, this is only a recommendation which could be made by VI.

Regarding the entities e_1, e_2, and e_3 of the general situation shown in Fig. 4.44, there are two possibilities:

1 e_1 and e_3 are synonymous.

2 e_3 could be made an attribute of e_1 or e_2 .

The above 2 recommendations are made by VI to the designer. If the recommendation in point 1 is acceptable, then both entities are integrated by VI into one entity. This recommendation is true in example 2 of Fig. 4.46, where the entities 'residence' and 'address' are integrated into one entity. The resultant GCS is as shown in Fig. 4.48.

Example 1:

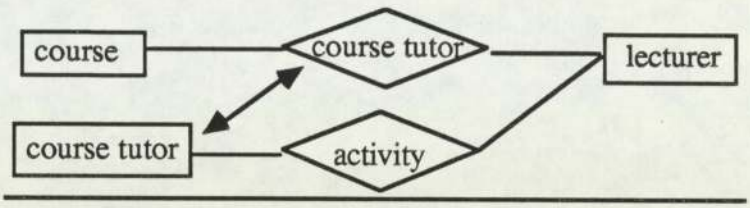


Fig. 4.45 An example entity as relationship/ foreign E-R

Example 2:

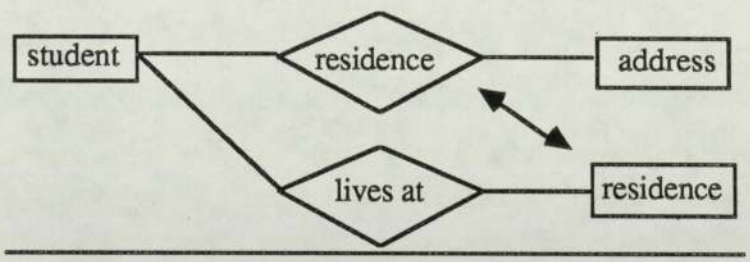


Fig. 4.46 An example entity as relationship/ foreign E-R

In the example of Fig. 4.45, the recommendation in point 2 seems more appropriate. The semantics modelled by the E-R ('lecturer' 'activity' 'course tutor') is indirectly represented in the E-R ('lecturer' 'course tutor' 'course'). The deletion of the E-R ('lecturer' 'activity' 'course tutor') would not cause any loss of semantics. The deletion of this E-R may necessitate the transfer of some of the attributes of one entity to the other. In this case, the attributes of the entity 'course tutor' might be transferred to the entity 'lecturer'.

Example 1:

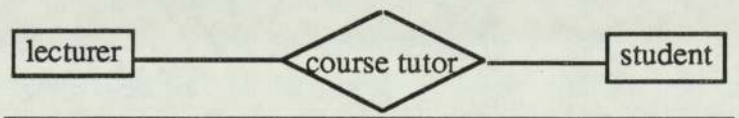


Fig. 4.47 An example E-R resulting from entity as relationship conflict

Example 2:

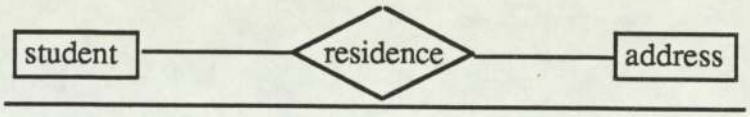


Fig. 4.48 An example E-R resulting from entity as relationship conflict

b) totally different E-Rs

Conflict definition

An *entity as relationship/foreign E-R* with no common entity COT conflict occurs when there is a relationship r_1 of an E-R (e_1 r_1 e_2), such that r_1 is also modelled as an entity e_3, and e_3 belongs to the E-R (e_3 r_2 e_4), and e_3 is not related by r_1.

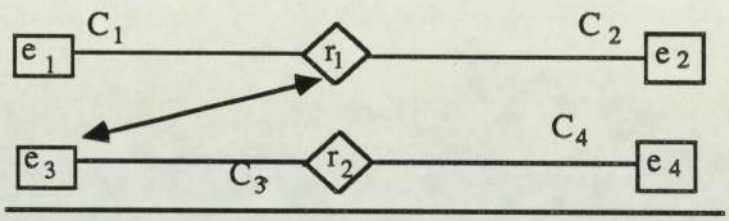


Fig. 4.49 Entity as relationship/ totally different E-Rs

Example 1:

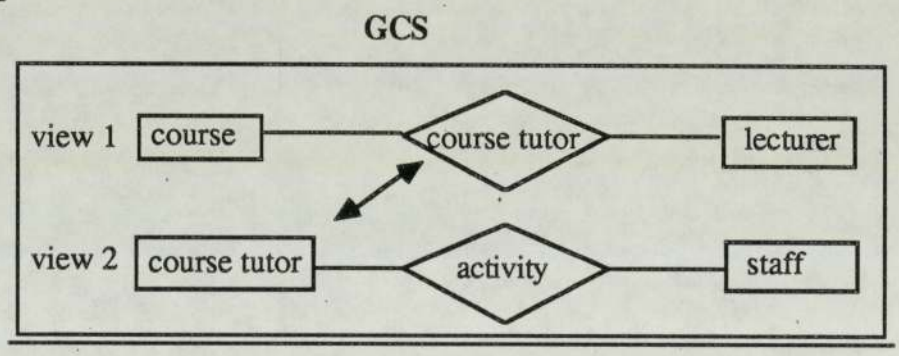


Fig. 4.50 An example entity as relationship / totally different E-Rs

Example 2:

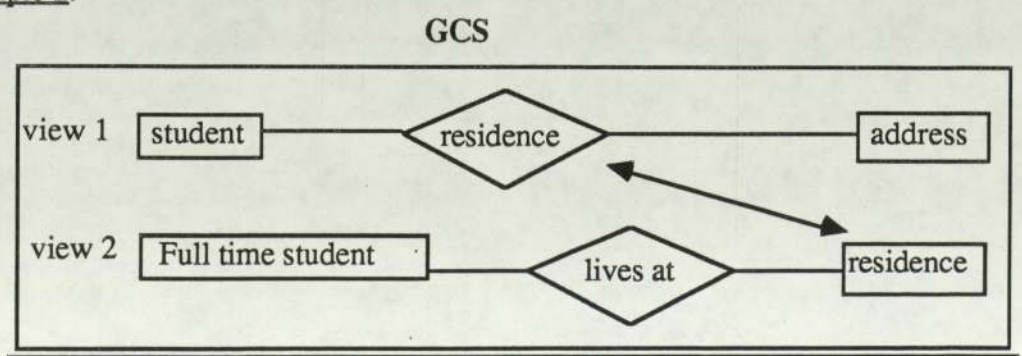


Fig. 4.51 An example entity as relationship / totally different E-Rs

The general situation of this conflict is shown in Fig. 4.49, and two examples are shown in Fig. 4.50 and Fig. 4.51. This conflict is very similar to the conflict discussed in section 4.3.12 (a) above. It is a complex COT conflict, and its existence usually indicates that the relationships r_1 and r_2 are synonymous. Although, not many examples of this kind of conflict have been encountered, it tends to indicate that the two relationships are also synonymous. In the example of Fig. 4.50 the relationships 'course tutor' and 'activity' are possibly synonymous, and in the example of Fig. 4.51, the relationships 'lives at' and 'residence' are also possibly synonymous. The corresponding entities could also be synonymous. Should such a recommendation be acceptable to the designer, then the corresponding entities are integrated accordingly. The resultant GCSs for the two examples are shown in Figs. 4.52 and 4.53. The designer may either rearrange some of the semantics concerning the attributes, or add new attributes to compensate for the unification of the two E-Rs. For example, the values 'student' and 'full time student' could be created as part of a new valued attribute in the resultant entity.

Example 1:

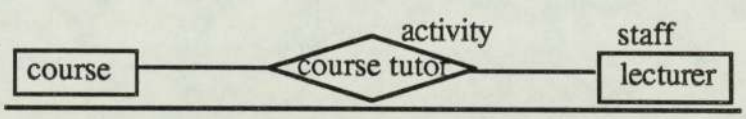


Fig. 4.52 An example of resultant E-R as a result of E-R conflict

Example 2:

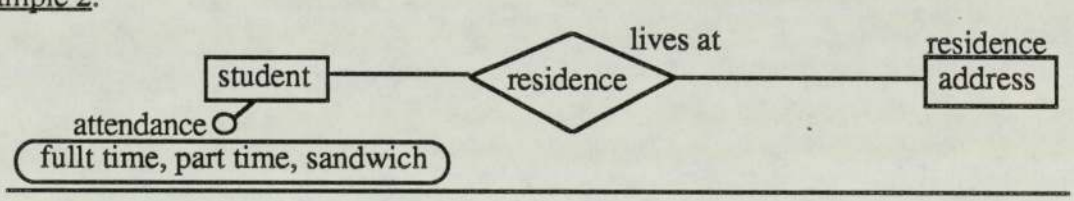


Fig. 4.53 An example of resultant E-R as a result of E-R conflict

4.4.14 Value as attribute

Since a value can be modelled as an entity, and an attribute can be modelled as an entity or as a relationship, it would seem that a *value as attribute* COT conflict can also exist. However, it was found that such a conflict cannot normally take place, and it was difficult to invent a situation which would provide an example. What is more important, is that although VI can detect all the occurrences of this conflict, no recommendations to help the designer can be made. If this conflict occurs, it would take one of the following formats:

1 Value as attribute/ same entity. The general situation is shown in Fig. 4.54.

2 Value as attribute/ different entities, common relationship. The general situation is shown in Fig. 4.55.

3 Value as attribute/ different entities, no common relationship. The general situation is shown in Fig. 4.56.

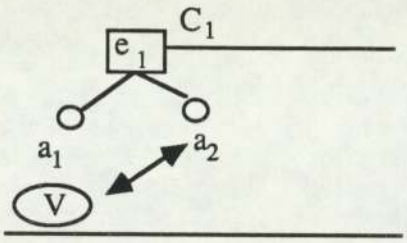


Fig. 4.54 Value as attribute same entity

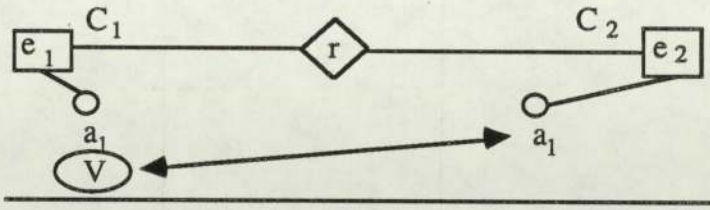


Fig. 4.55 Value as attribute/ foreign entity and a common relationship

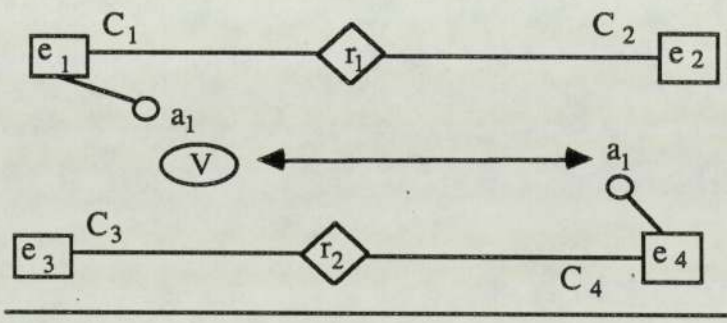


Fig. 4.56 Value as attribute/ foreign entity and no common relationship

4.4.15 Value as relationship

Since a value can be modelled as an entity, and an attribute can be modelled as an entity or as a relationship, it would seem that a *value as relationship* COT conflict can also exist. However, it was found that this conflict cannot normally take place. Although VI can detect all the occurrences of this conflict, no recommendations to help the designer can be made. If this conflict occurs, it would take one of the following formats:

1 Value as relationship/ same E-R.

2 Value as relationship/ different E-Rs.

4.5 Conclusion

Section 4.2 presented an approach to identify objects of the same type suffering from the synonym naming conflict. These synonyms cannot normally be identified by VI, but with the introduction of the object fuzzy matching approach, a SLF is produced as a result of matching all the neighbours of the two objects concerned. Whilst the value of the SLF is representative of the level of similarity between objects in most situations, this value can be disturbed by the number of neighbours of the objects concerned.

One of the biggest difficulties in object fuzzy matching is in deciding the value of the SLF which can be regarded as a clear cut match or difference between objects. Another difficulty is in assigning weights to each of the neighbours which contribute to the calculation of the SLF. Further, in the same application, most entities share attributes, and therefore, most of the SLFs are on average the same. This contributes to the difficulty of deciding the clear cut SLF. Therefore, object fuzzy matching should be used to help the designer identify many of the synonymous objects of the same type, but some of these could still remain undetected. Further, it was concluded in section 4.2.3 that object fuzzy matching of attributes, relationships and E-Rs normally yields meaningless SLFs.

Section 4.3 described the COT conflicts which can exist in view integration. Although it was possible to give relatively 'confident' resolutions or recommendations to some of these conflicts, the problem is whether these 'confident' resolutions can be guaranteed to work for all the situations of the same type. In the situations where the decision cannot be prescribed beforehand, the responsibility for the resolution is handed to the designer. The diversity and the number of possible special cases of the same type of conflict are the reason for the difficulty of prescribing resolutions for such conflicts.

The analysis of the COT conflicts presented in this chapter did not include the situations where the same object conflicts with a number of other objects simultaneously. These situations can be termed *multiple COT conflicts*. An example of these conflicts occurs where, at the same time, an attribute can exist as an entity, as a relationship, or as a value. In this chapter it is considered that the resolution of the individual COT conflicts in the prescribed order should eventually produce the same result. However, this needs further investigation and support.

CHAPTER 5

VIEW MODELLING

5.1 Introduction

This chapter shows how views which are modelled in ERM, are transformed by the designer from their ERM pictorial representation to a specially developed language called the VDL. The latter forms the input to the VI. A discussion about the naming conflicts at the view modelling phase is presented, and it is shown that by using multiple names for objects, some of the otherwise unsolvable conflicts are resolved.

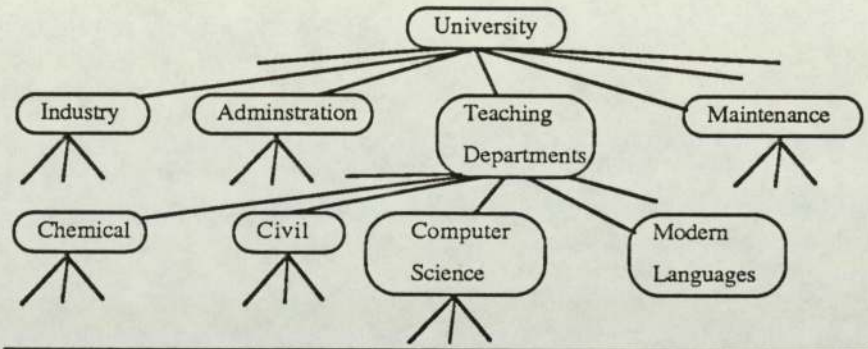


Fig. 5.1 a Tree view modelling approach for the University of Aston

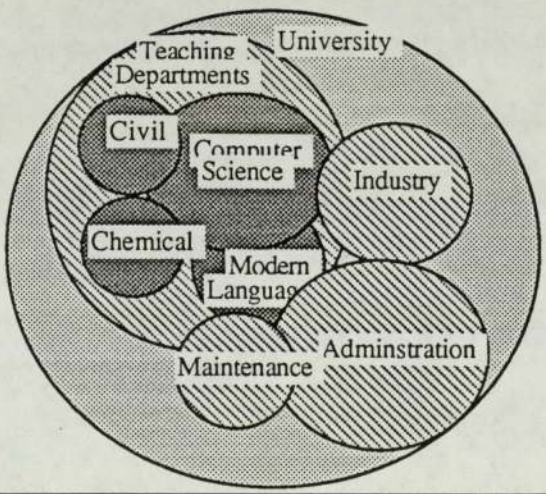


Fig. 5.1 b Set view modelling approach for the University of Aston

Fig. 5.1 View modelling for the University of Aston

5.2 Identifying the views

It is not the intention here to describe the traditional systems analysis or data analysis approaches to designing systems or databases (Gane & Sarson 1979 and Howe 1983).

Instead, an approach is outlined which illustrates how a large organization can be broken down into views. The result of this is either a *view modelling tree* (Fig. 5.1 a) or a set of views (Fig. 5.1 b). The tree view modelling approach results in a view modelling tree which shows the hierarchy of all the views in the organization. The set view modelling approach shows the sets and subsets of the views within the organization. The discussion of the best view modelling approach is beyond the scope of this thesis.

Fig. 5.1 a shows part of the view modelling tree for the University of Aston. At the very top level is the view 'University'. The 'University' view has a number of immediate subviews, examples of which are 'Industry', 'Administration', 'Teaching Departments' and 'Maintenance'. The subview 'Teaching departments' is broken down into further subviews, for example 'Chemical', 'Civil', 'Modern Languages' and 'Computer Science'. Fig. 5.1 b shows the sets and subsets of the views in the organization. The Computer Science set for example has the subsets of 'Chemical', 'Civil', 'Modern Languages' and 'Computer Science'.

The result of the view modelling approach would depend on the designer's skill and experience, and on the structure of the organization. Some of the views or subviews in an organization could be similar. The problem which would be encountered by the designer, after identifying all the views of the organization, is whether to model all the similar subviews or to model just one on the basis that all the others are the same. An answer to this problem is not easy to give and the ultimate decision rests with the designer. If he feels that the subviews are exactly the same, then one is enough. On the other hand, it can be argued that if there is a slight difference between them it is better to model them all and let the VI delete all duplication.

5.3 The View Description Language

The pictorial representation of an SDM serves as a good communication tool which enables the designer to model the organization, and at the same time it enables the user to understand how his organization is represented in the database. The VI described in this thesis integrates views to create the GCS. However, since the VI has been written to accept textual rather than pictorial input, a special language which represents the pictorial form of ERM was especially developed. This language is called the VDL.

Using the VDL, the designer can represent all the semantics from the pictorial form to the textual form. The general format of the VDL is shown in Fig. 5.2. The keywords are shown in capital letters. The VDL statements start with the keyword VDL, and end with the keyword END VDL. The VDL statements for the subviews of a given view, must be

written within the VDL statements indicating the start and end of the VDL statements of this view. An E-R is indicated by the RELATIONSHIP keyword, and its details, followed by all the entity definitions of the entities it relates. The degree of the relationship (binary, ternary or n-ary) can be declared using the DEGREE keyword. The number of entities following the relationship VDL statement should be equal to its degree declaration. Three of the E-Rs of the pictorial representation of the view 'courses', in Fig. 5.5, are represented in their VDL equivalent form in Fig. 5.4.

5.4 Transforming views from ERM pictorial format to VDL format

A binary E-R usually involves two entities. However, some situations arising from the integration of two or more binary E-Rs which have a common relationship name, can produce a binary E-R which involves more than two entities ('bad' binary E-Rs). Sometimes these 'bad' E-Rs are modelled in the same view. These situations were described in section 3.3.2.2. However, as the VDL statements are written at the modelling stage, all E-Rs are considered to be binary. As described earlier, the only reason for this is that the VI was developed to handle binary E-Rs only. Conveniently, an entity can be related by any number of E-Rs. Therefore, when transforming views from pictorial to the VDL formats, one E-R at a time is transformed into the VDL format until all the E-Rs are transformed. The only unavoidable drawback to this approach is that the same definition for each entity is repeated in the VDL the same number of times the entity is related by relationships, with the exception of the role name (if given) and the cardinality. A pseudo algorithm which shows the steps to transform a view from its pictorial formats to its equivalent the VDL format, is given in Fig. 5.3. The same algorithm can be used by a graphical interface which could be built into VI to achieve the same effect.

```
VDL    application name.
VIEW   view name.
[SUBVIEW  subview name].
RELATIONSHIP  relationship name [other names]  [DEGREE degree level].
                ENTITY entity name [other names]
                                        [ROLE role name]
                                        CARDINALITY cardinality value.
                        ATTRIBUTE       attribute name [other names]
                                        key status
                                        [VALUE {SET, RANGE} values].
                        ATTRIBUTE       attribute name [other names]
                ..........
                ENTITY entity name.
                ..................
RELATIONSHIP  relationship name [other names]  [DEGREE degree level].
                ENTITY entity name [other names]
                                        [ROLE role name]
                                        CARDINALITY cardinality value.
                        ATTRIBUTE       attribute name [other names]
                                        key status
                                        [VALUE {SET, RANGE} values].
                        ..........
                ENTITY entity name.
                .........
RELATIONSHIP  relationship name [other names]  [DEGREE degree level].
.............
[END SUBVIEW  subview name].
END VIEW view name.
END VDL application name.
```

Fig. 5.2 The template format of the View Description Language

1 Choose the next view.

2 If the view is a subview then declare by writing the subview VDL statement, otherwise declare by the view declaration statement.

3 Choose the next entity.

4 Represent all the E-Rs involving this entity in their VDL form, according to Fig. 5.2.

5 Delete this entity from the view.

6 Delete all the other entities, which are only related by relationships with the deleted entity from the current view.

7 Repeat steps 3 - 6 until there are no more undeleted entities.

9 Declare the end of the current view (or subview) by the end view (or end subview) statement.

Fig. 5.3 Algorithm for transforming views from ERM to VDL

5.5 Naming conflicts caused at view modelling

Kent (1978) gives a detailed analysis of the different methods of naming objects and the difficulties and conflicts that these names can cause. His main argument is that the names we give to objects are influenced by our view of these objects, not what the objects represent. In view modelling this is more apparent as different users view the same objects differently. They also differ in the context that they use them. It was assumed in chapter 3 that any two objects of the same type, but differing on names, are different. Whether the view integration approach is applied for the integration of existing databases or for schema development, naming conflicts are always a serious problem causing duplication and inconsistencies in the resultant GCS. Common naming conflicts are homonyms and synonyms, but other naming conflicts also exist. These can be due to the abbreviations of certain names, using a mixture of singular and plural names or using what might be called functional names, that is names explaining the function of the object for that particular view. The different types of naming conflicts are described below.

Plural and singular names: In the two E-Rs in Fig. 5.6 and Fig. 5.7 the entity 'courses' in view 1 is a plural of the entity 'course' in view 2.

Synonyms: The entity 'lecturers' in view 1 of Fig. 5.6 is a synonym of the entity 'staff' in view 2 of Fig. 5.7. (Synonyms are not used in a strict linguistic sense. In a chemistry application for example, 'H2O' and 'water' could be regarded as synonymous).

```
VIEW teaching.
SUBVIEW     courses.

RELATIONSHIP belongs to DEGREE binary.
        ENTITY student CARDINALITY 3.
                ATTRIBUTE    student name key.
                ATTRIBUTE    number key.
                ATTRIBUTE    course name notkey.
        ENTITY department CARDINALITY many.
                ATTRIBUTE    name key.
                ATTRIBUTE    faculty notkey.

RELATIONSHIP lectures on DEGREE binary.
        ENTITY lecturer CARDINALITY 5.
                ATTRIBUTE    name key.
                ATTRIBUTE    number key VALUE RANGE 0001-2000.
                ATTRIBUTE    department notkey.
        ENTITY course CARDINALITY 30.
                ATTRIBUTE    title key.
                ATTRIBUTE    level notkey VALUE SET postgraduate, undergraduate.
                ATTRIBUTE    year notkey.
                ATTRIBUTE    department key.

RELATIONSHIP runs DEGREE binary.
        ENTITY department CARDINALITY many ROLE runs.
                ATTRIBUTE    name key.
                ATTRIBUTE    faculty notkey.
        ENTITY course CARDINALITY 3 ROLE run by.
                ATTRIBUTE    title key.
                ATTRIBUTE    level notkey VALUE SET postgraduate, undergraduate.
                ATTRIBUTE    year notkey.
                ATTRIBUTE    department key.
END SUBVIEW courses.
END VIEW teaching.
```

Fig. 5.4 VDL representation of part of the view 'courses

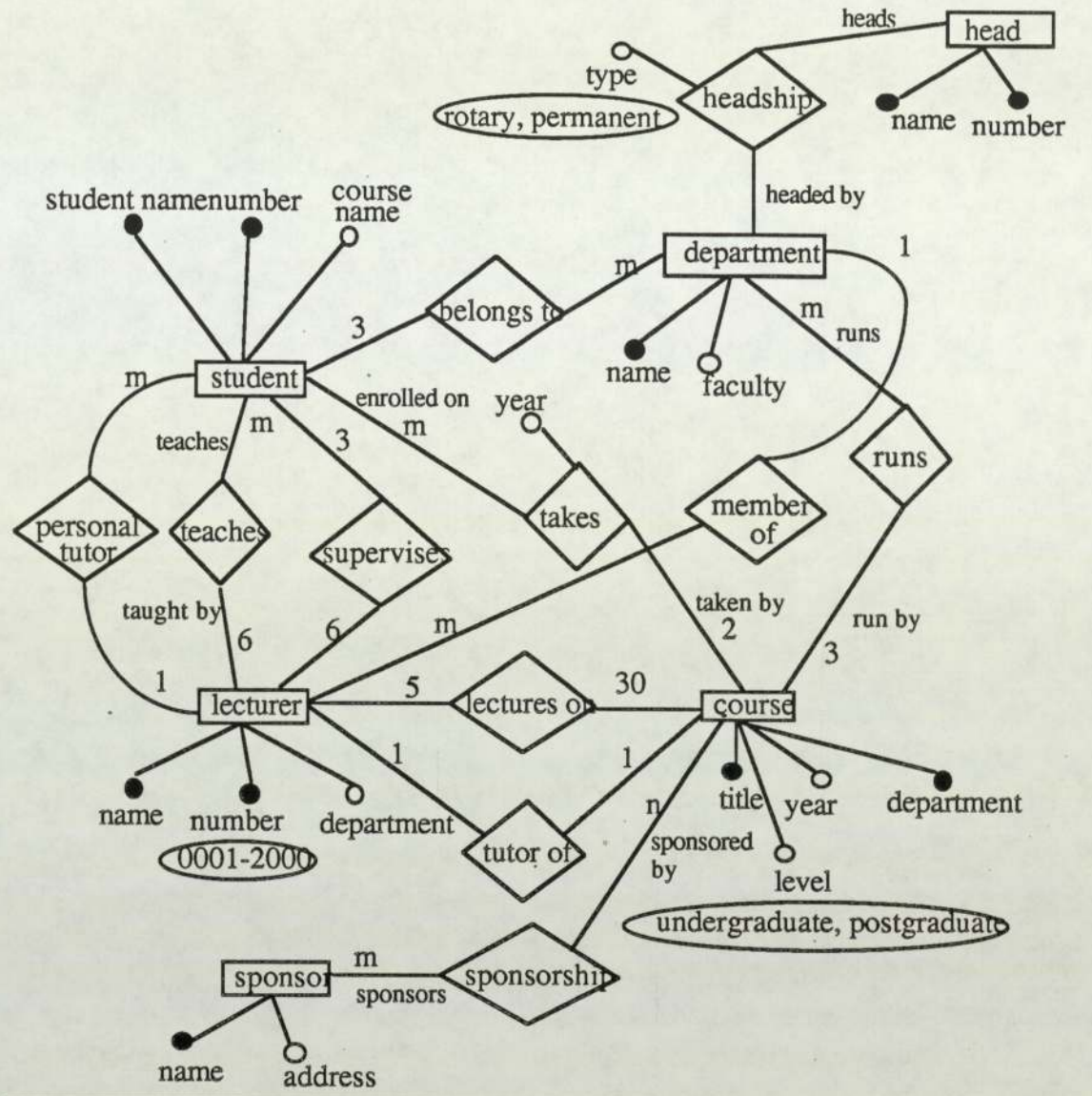


Fig. 5.5 A sample of the 'courses' view

Homonyms: This kind of naming conflict often takes place when naming objects of different types, for example COT conflicts.

Abbreviations: These are a form of synonym naming conflicts. Because of familiarity with the application, designers and users might abbreviate some names and these may not obey any standards that exist. The attribute 'department' of entity 'courses' in view 1 is abbreviated to 'dept.' for the entity 'course' in view 2.

Misspelling and random errors: This is a random synonym naming conflict where the designer writes an object name incorrectly. The attribute 'building' of relationship 'teaching', is misspelt for relationship 'teaches' in view 2.

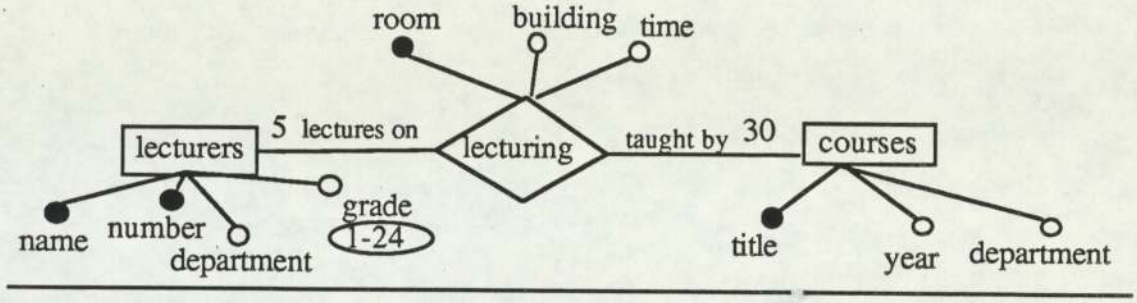


Fig. 5.6 View 1

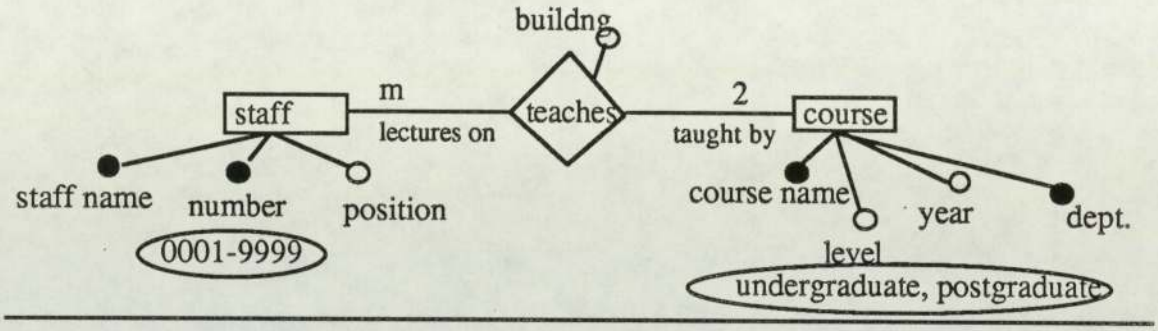


Fig. 5.7 View 2

5.6 Multiple naming of objects

The two E-Rs in Fig. 5.6 and Fig. 5.7 were modelled to represent the same semantics, but because they are modelled separately in different views, they suffer from naming conflicts. With the exception of identifying some of the objects meant to be the same in the object fuzzy matching process described in section 4.2, the majority of these naming conflicts would not be detected. When integrating the two E-Rs in Figs. 5.6 and 5.7, they will be regarded as two separate E-Rs, and they will coexist in the GCS.

One way to avoid this problem is to allow the designer to model all the possible names that an object can have at the view modelling stage. Whilst it is neither practical nor possible to always give all the names for each and every object, it is helpful to give the most appropriate name alternatives wherever possible. These names should be what the designer or user thinks that the same object might be named elsewhere. The hope is that, at integration, the intersection of object names would produce a non empty set which would indicate a synonym naming conflict. Fig. 5.8 shows an E-R with multi-naming of objects. The main name for the object is written in the usual place, as in ordinary ERM modelling. The other names are written adjacent to the object concerned. The entity 'staff' for example has another possible name 'lecturer'. The relationship 'teaches' could be given the other names, 'teaching' and 'lectures on'.

Multi-naming of objects does not impose much burden on the designer, but it can cause added complexity in the view diagram with too many names. However, this is not seen as a severe drawback. In database integration, multiple naming of objects could be presented to the VI as integration assertions.

Although the multiple naming of objects is a helping factor in the integration process, it is also recommended to keep these names to help the user when querying the database. At the final tuning and restructuring of the integrated schema, some of these names may be removed.

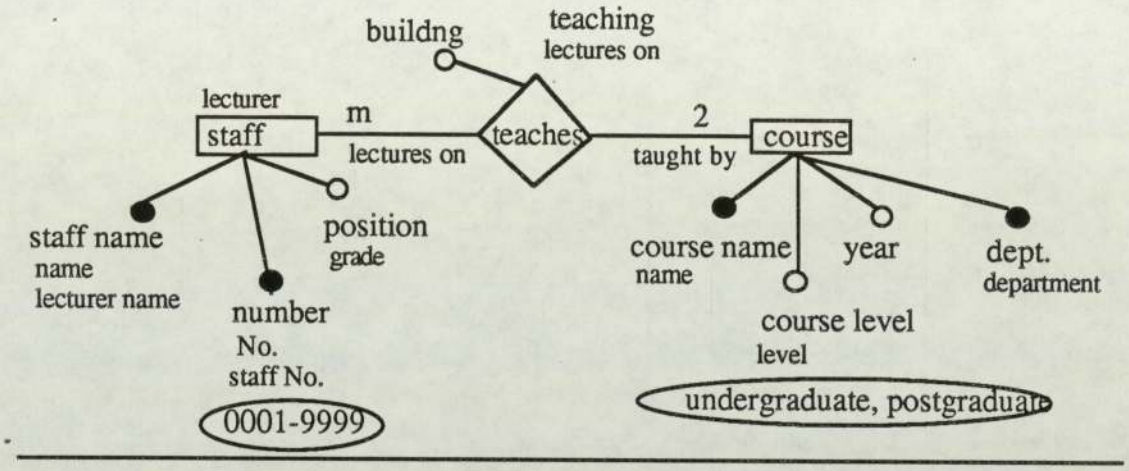


Fig. 5.8 A relationship with multi-naming of objects

5.7 Conclusions

This chapter presented an outline of a method for breaking down a large organization into all its views and subviews, a process which results in a hierarchical view modelling tree. The designer can then proceed to model each of the views and subviews in the tree into ERM. Because there is not a graphical interface, which can read the pictorial form of view, the VDL was developed. The VDL can be used to represent the semantics of the ERM views in textual form. The VI can then read and integrate the views.

Chapters 3 and 4 demonstrated that certain naming conflicts cannot be detected by the VI. Therefore, a method was presented to model each object with all its possible names, which could be used detect synonymous names. These names are written adjacent to the usual object name.

The contribution of this chapter is in providing a way of representing the same ERM semantics textually using VDL. The VDL may eventually be used by the expert designer to model the views directly in textual form.

CHAPTER 6

BINARY VIEW INTEGRATOR

6.1 Introduction

This chapter describes the implementation of the BVI, which has been written in Quintus Prolog on a SUN workstation.

This chapter describes the internal representation chosen for the views and the GCS, where both the views and the GCS can simultaneously exists in the same Prolog database. The internal representation chosen for the views is described in sections 6.2 and that of the GCS is described in section 6.3. This chapter also describes the implementation of the object fuzzy matching and seven of the fourteen COT conflicts.

The BVI starts the view integration process with an empty GCS and integrates the views with it one at a time. An E-R from the current view is integrated at a time with the GCS and the GCS is updated with the semantics of this E-R depending on the type of match it has with all the E-Rs of the GCS.

The BVI is processed in three phases. These are the pre-integration phase, the view integration phase and the post-integration phase. The pre-integration phase reads the views in VDL form and represents them in a relational internal from. The view integration phase chooses the views and their E-Rs and integrates an E-R at a time with the GCS. The post-integration phase identifies and resolves all the COT conflicts. Object fuzzy matching is processed as a post-integration phase. The program listing of the BVI is in appendix J.

6.2 Representing views in BVI

Section 3.2 showed the definition of E-Rs and views. Section 3.3 showed the general meta representation of an E-R (see Fig. 3.4). This section shows the data structures which define the views internally in BVI. This internal representation of the views within the BVI must define and relate all the objects in the views in exactly the same way as they are defined and related pictorially: views and subviews must be linked according to the view modelling tree shown in Fig. 5.1, attributes must be linked to their corresponding entities and relationships, entities must be linked to their corresponding relationships, and so on.

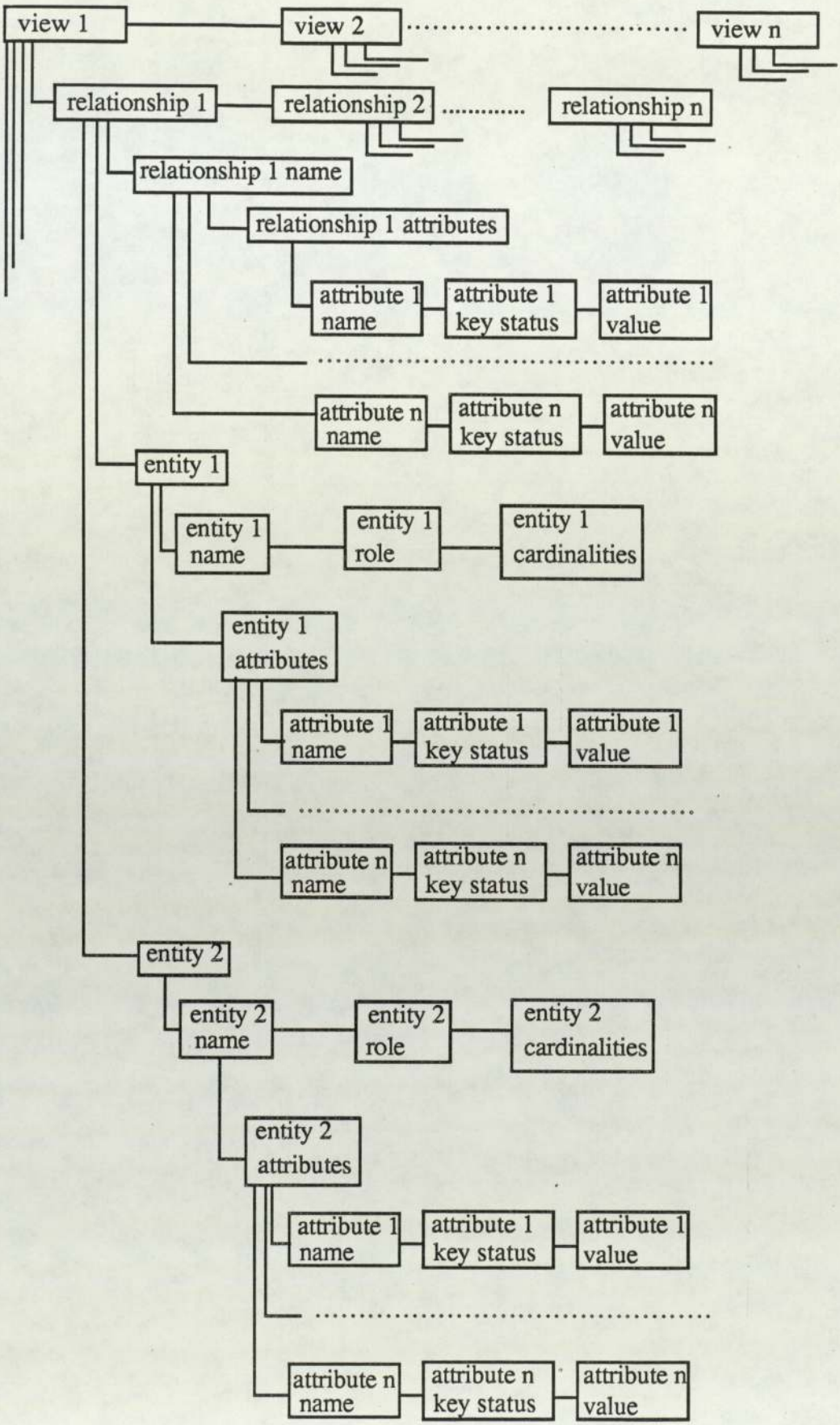


Fig. 6.1 ERM views represented in a linked list structure

One way of representing the ERM views is a nested linked list, as shown in Fig. 6.1. This nested linked list shows:

1 There are a number of views.
2 Each view contains a number of E-Rs.
3 A relationship may have attributes.
4 A relationship relates two or more entities.
5 Each entity has a name, a role and cardinalities.
6 Each entity contains a number of attributes.
7 Each attribute has a name, a key status and may be a value.

At the early stages of the BVI implementation, ERM views were represented as individual lists: lists representing each of the E-Rs, lists representing each of the entities, and so on. It was found that two E-Rs of the same view which match on the names of their entities and the names of their relationships could not be represented uniquely and therefore the BVI cannot distinguish which objects belong to which of these E-Rs.

In order to avoid the problems caused by the individual list structure, the nested linked list structure in Fig. 6.1 was chosen next, and a prototype of BVI was implemented accordingly. However, two factors were found not to be in favour of the nested linked list data structure representation. The first is that should the decision be taken to develop BVI to include some ERM extensions, such as data abstraction, then the nested linked list could be too complex to comprehend when debugging the program. Further, although the prototype BVI was fairly fast when used to integrate a very small set of views, it slowed down drastically when the number of views was increased. The second is that the language chosen to implement BVI is Prolog, and Prolog is especially adapted to relational database structures for which it has the equivalent of a built-in DBMS. Based on these two factors, the relational data structure shown in Fig. 6.2 was derived to represent the views internally in BVI. These Prolog predicates form the definitions of internal Prolog database relations which, once populated with the semantics of the views, would contain the domains of these views.

Although it is possible to define the whole data structure by one compound and complex relation, such an approach is tedious when passing parameters between the BVI predicates, and complex when debugging the program. Therefore, the structure was broken down into a number of smaller and less complex relations. Between them, these relations can contain all the semantics contained in the views.

Before describing the relations structure in Fig. 6.2, it is necessary to describe some of the variables shown in some of these relations.

Object priorities

The variables allocated to hold priority values are: View_prio for views, Subview_prio for subviews, Rel_prio for relationships, Ent_prio for entities and Att_prio for attributes. These priorities can either be modelled by the designer as integration assertions, or calculated by BVI. However, currently the BVI does not deal with priorities.

Object occurrence numbers

Occasionally two E-Rs from the same view match on the entity names and relationship names. Although this is not a common feature in view modelling, it can occur, and therefore one of two actions had to be made to cater for it. One action is to include, as part of the pre-integration tasks of BVI, a procedure to identify similar E-Rs before they are represented in the internal relations. The second action is to associate a unique number with each of the E-Rs and their corresponding entities. This number, which is called the *occurrence number*, is assigned automatically to both entities and relationships by BVI in sequence. The first E-R of the first view read is given the occurrence number 1, and this is incremented by 1 for each new E-R. Each time a new view is started, the relationship occurrence number is started from 1 again. Entities are assigned occurrence numbers in the same way as relationships. No occurrence numbers are needed for attributes, since they are uniquely identified by their names, and the name and occurrence number of the objects they belong to.

```
view(V_name, View_prio).
subview(V_name, Subview_name, Subview_prio).
att(V_name, Att_name).
ent(V_name, ent(Ent_name, Ent_occ, Ent_prio)).
ent_att(V_name,
        ent(Ent_name, Ent_occ),
        att_det(Att_name, Vals, K_stat, Att_prio)).
rel(V_name, rel(Rel_name, Rel_occ, Rel_prio)).
rel_att(V_name,
        rel(Rel_name, Rel_occ),
        att_det(Att_name, Vals, K_stat, Att_prio)).
ent_rel(V_name,
        rel(Rel_names, Rel_occ),
        ent(Ent_name, Ent_occ, Cards, Role),
        ent(Ent_name, Ent_occ, Cards, Role)).
```

Fig. 6.2 Representation of ERM views in a relational form

Now let us describe briefly the format of the relations in Fig. 6.2:

The **view** relation

This relation defines the domain of all the view names without duplications, indicated by the variable V_name. The variable View_prio indicating the view priority, exists for the reason described above.

The **subview** relation

Based on the view modelling tree approach, a view can consist of one or more subviews, and subviews can further be broken down into smaller subviews. The depth of the view modelling tree depends on the organization structure, for example, the view modelling tree in Fig. 5.1 has a depth of 5 levels. The subview relation associates each view with its immediate subviews. The view name is indicated by the variable V_name, and the subview is indicated by the variable subview_name.

From the view modelling tree in Fig. 5.1, the following subview relations would be created:

```
subview('Computer Science', teaching, _).

subview(teaching, attendance, _).
```

where _ represents the uninstantiated priority value of the view or the subview.

The **att** relation

This relation associates attributes with the view in which the object defined by the attribute is modelled. The V_name variable indicates the name of the view, and the attribute is indicated by the variable Att_name.

This relation is not compulsory as far as the semantics of the views is concerned. It is included for more efficient and speedy queries by BVI.

The **ent** relation

The ent relation relates entities to the views in which they are modelled. An entity is indicated by the variable Ent_name, and the occurrence number of the entity is indicated by the variable Ent_occ.

This relation, like the att relation, is not compulsory, but is used for more efficient and speedy queries by BVI.

The **ent-att** relation

This relation links each entity to its attributes, for all the occurrences of the entity in all the views. In order to ensure the correct link, both the entity name and its occurrence number are given. Since it is possible that the same entity, and some or all of its attributes can be modelled in more than one view, it is necessary to indicate the view name in which this entity attribute connection occurs. As mentioned above, the ent and att relations are not regarded as part of the compulsory data structure. Therefore, the view name must be included in the ent_att relation, to ensure the uniqueness of the entity attribute connection. The details of each attribute of the entity is indicated by another relation called att_det which is defined within the ent_att

relation. The att_det relation consists of the domain indicating the attribute name, the attribute value(s) (if any), and the attribute key status. The attribute value(s) is indicated by the variable Vals, which can either contain a list of values or a range of values. The K_stat variable always contains either the word 'key' or 'notkey', to indicate that the attribute is a key attribute or otherwise.

The **rel** relation

The rel relation links the relationships to the view in which they are modelled. The relationship name is indicated by the variable Rel_name, and the relationship occurrence number is indicated by the variable Rel_occ. Again, like the att and ent relations, the rel relation is not a compulsory part of the data structure.

The **rel-att** relation

The rel_att relation links the attributes to the relationships they define. Since the same relationship can belong to more than one view, the view name is included in this relation to uniquely identify each occurrence of this type of connection. The inner relation att_det serves the same purpose as the att_det relation which is defined within the ent_att relation above.

The **ent-rel** relation

This relation links the relationship name to its entities. Each entity involved in the relationship is indicated by a relation called ent defined within the ent_rel relation. There are as many ent relations within the ent_rel relation as the degree of the relationship, for a ternary E-R for example, there are three ent relations. The ent relation shows the entity name, the entity occurrence number, the entity cardinality and the role of the entity. The entity name is indicated by the variable Ent_name. The entity occurrence is indicated by the variable Ent_occ. The cardinality of the entity is indicated by the variable Cards, and this can contain a numeric value, or it can contain the mnemonic 'many'. The relationship involving the entities is represented by the relation rel. The rel relation defines the domains representing the relationship name, and the relationship occurrence number. And finally, the view name must be represented in the ent_rel relation in order to uniquely link E-Rs to their correct views. The declaration of the view name in the ent_rel relation is <u>not</u> a duplication of its domain in the two relations rel and rel_att, because firstly the rel relation is not part of the compulsory data structure, and secondly the rel_att relation only exists if the relationship has attributes.

Although BVI handles binary E-Rs only, the ent_rel relation can include as many ent relations as an n-ary E-R might involve.

<u>Example</u>

Figs. 5.4 and 5.5 show the VDL and pictorial formats of the view 'courses' respectively. The internal representation of one of these E-Rs (the E-R 'student belongs to department') is shown in Fig. 6.3. All the positions where the priority value is supposed to be given, an underscore is used to represent an empty priority value.

Since the E-R 'student belongs to department' has no attributes defining it (excluding the attributes defining the entities to which it relates), no rel_att relations are shown in Fig. 6.3. The two entities related by this binary E-R have not been given any role name, and therefore the variable Role of the ent_rel relation is uninstantiated for both entities 'student' and 'department' in the view 'courses'. Since there is only one relationship of the name 'belongs to' in the view 'courses', only one entity of the name 'student' and only one entity of the name 'department', they all have the occurrence number 1 in the corresponding relations. The attributes defining the two entities of the E-R have no values, and hence the corresponding variables in the ent_att relations are uninstantiated.

In section 5.6 a method of modelling multiple names for objects was described. Some naming conflicts cannot be identified by BVI, and therefore it is necessary that wherever possible, all the possible names which can be given to an object, must be given at the view modelling stage. A number of ways were considered to represent these multiple names internally in the relations, and it was found that the best and easiest way of achieving this is to include all the multiple names of any object in all the relations where the name of the object is represented. The example in Fig. 6.3 does not show any object with multiple names. However, the names of objects are represented in lists (indicated by square brackets), and should the object have multiple names, they are included in the list and separated by commas. Thus, there are no restrictions on the number of different names which can be given to an object.

The first thing to notice about the relations defining the GCS in Fig. 6.4, is that no occurrence numbers are represented. The reason for the removal of the corresponding variables for the occurrence number of objects from the relations, is that BVI ensures that no two entities or relationships of the same name are allowed in the GCS. Also, no relations representing views or subviews are included as part of the definition of the GCS. Since the GCS is based on ERM, the relations defining its internal data structure representation are similar to the relations defining the views. Therefore, only brief descriptions are given below for the GCS relations of Fig. 6.4. The relations names have been slightly modified from their corresponding relations which define the views, and this is to allow both definitions to coexist in the same Prolog database simultaneously. The general concept for the change is that at the end of each relation name defining the GCS, the _s symbol is added to indicate that this is a schema and not a view relation.

6.3 Representing the GCS in the BVI

The GCS is produced by the BVI as a result of integrating all the views. The BVI updates the GCS by the addition, deletion or changing of objects and their connections. The details of how the GCS is formed by the BVI is described in a later section. The structure

of the internal representation of the GCS is similar to the structure of the internal representation of the views.

```
view(teaching, _).
subview(teaching, courses, _).

att(courses, ['student name']).
att(courses, [number]).
att(courses, [name]).
att(courses, [faculty]).

ent(courses, ent([student], 1, _).
ent(courses, ent([department], 1, _).

ent_att(courses,
        ent([student], 1),
        att_det(['student name'], _, key, _)).
ent_att(courses,
        ent([student], 1),
        att_det([number], _, key, _)).
ent_att(courses,
        ent([student], 1),
        att_det(['course name'], _, notkey, _)).
ent_att(courses,
        ent([department], 1),
        att_det([name], _, key, _)).
ent_att(courses,
        ent([department], 1),
        att_det([faculty], _, notkey, _)).

rel(courses, rel(['belongs to'], 1, _)).

ent_rel(courses,
        rel(['belongs to'], 1),
        ent([student], 1, 3, _),
        ent([department], 1, many, _)).
```

Fig. 6.3 A sample of a view in relational form

The **att-names-s** relation

This relation gives the domain of all the attributes in the GCS.

The **rel-names-s** relation

This relation gives the domain of all the occurrences of all the relationship names in the GCS.

att_names_s(Att_names).

rel_names_s(Rel_name, Rel_prio).

ent_names_s(Ent_name, Ent_prio).

ent_att_names(Ent_name, Att_name).

rel_att_names_s(Rel_name, Att_name).

ent_att_det_s(Ent_name,

att(Att_name, Vals, K_stat, Att_prio)).

rel_att_det_s(Rel_name,

att(Att_name, Vals, K_stat, Att_prio)).

ent_rel_s(rel(Rel_name),

ent(Ent_name, Cards, Role),

ent(Ent_name, Cards, Role)).

Fig. 6.4 Representation of the GCS in relational form

The **ent_names_s** relation

This relation gives the domain of all the occurrences of all the entities in the GCS.

The **ent-att-names** relation

This relation links entity names to their corresponding attribute names.

The **rel-att-names** relation

This relation links relationship names to their corresponding attribute names.

The **ent-att-det-s** relation

This relation links each entity with its corresponding attribute names and attribute details. The attributes details are indicated by the relation att which is within the ent_att_det_s relation. The variables Vals and K_stat represent the attribute values and key status in the same way as in the relation ent_att of the relations describing the views.

The **rel-att-det-s** relation

This relation is similar to the ent_att_det_s relation, except that this links the relationships with their attributes and attributes details.

The **ent-rel-s** relation

This is similar to the ent_rel relation used in the view definition section. The ent_rel_s relation has three or more other relations within it, and these are ent and rel. The ent relations define the entities involved in the relationship, whose name is given by the relation rel. Ternary and n-ary E-Rs can be defined using the ent_rel_s relation by the inclusion of as many ent relations in the ent_rel_s relation as the degree of the E-R.

Example

In order to show the internal representation of the GCS using the relations described above, the E-R 'student belongs to department' is assumed to have been created by BVI

as part of GCS, as a result of view integration. This E-R is part of the view 'courses' which is shown in Fig. 5.5. The GCS relations for this E-R is shown in Fig. 6.5.

6.4 BVI implementation

6.4.1 The BVI algorithm

Fig. 6.6 shows the steps taken by the designer, before the views are fed to the VI. These views are passed to BVI, which in turn integrates them one at a time with the latest GCS, until all these views are integrated to form the final GCS. BVI chooses the next view, and integrates all its E-Rs with the latest GCS available. The conflicts which occur as a result of the integration of the E-Rs of the current view and the GCS E-Rs are reported to the designer if they cannot be resolved automatically by BVI. As shown in Fig. 6.7 the designer can query the contents of the view and the contents of the GCS before making a decision on the resolution of a conflict. The designer can make direct changes to the GCS

```
att_names_s('student name').
att_names_s(number).
att_names_s('course name').
att_names_s(name).
att_names_s(faculty).

rel_names_s('belongs to', _).

ent_names_s(student, _).

ent_att_names(student, 'student name').
ent_att_names(student, number).
ent_att_names(student, 'course name').
ent_att_names(department, name).
ent_att_names(department, faculty).

ent_att_det_s(student, att('student name', _, key, _)).
ent_att_det_s(student, att(number, _, key, _).
ent_att_det_s(student, att('student name', _, key, _)).
ent_att_det_s(student, att('course name', _, notkey, _)).
ent_att_det_s(department, att(name, _, key, _).
ent_att_det_s(department, att(faculty, _, notkey, _)).

ent_rel_s(ent1(student, 3, _),
        rel('belong to'),
        ent2(department, many, _)).
```

Fig. 6.5 An example GCS E-R in relational form

in order to resolve a conflict; these changes can be in the form of deletions, additions or changes to any of the objects and objects connections. Only queries can be made to the views, as changes to their contents are unnecessary and could have unforeseen effects.

When all the objects of the current view have been integrated, the GCS would consist of its previous contents, along with the contents of the view just integrated. The BVI then considers the next view for integration and subjects it to the same integration process. This process is continued until all the views have been successfully integrated. The full outline of the BVI algorithm is shown in Fig. 6.9.

As can be seen from Fig. 6.7, when BVI considers the very first view, there is no GCS to be matched with it. This problem can be resolved either by modelling an enterprise view (sometimes called the skeletal schema) or assuming that a GCS is available, but it is empty. The use of an enterprise view in integration was described in section 2.7.1. However, it was found that it is not necessary for BVI to start with an enterprise view. Instead, the first view is integrated with an empty GCS. At the end of this process, the GCS would contain the semantics of the view, but represented in the GCS format.

6.4.2 Choice of next view for integration

In binary view integration, the choice of the next view may have an effect on the number and type of conflicts which may arise during integration. Further, if views are chosen in a bottom up tree-like manner (according to the view modelling tree), then the GCS is always complete for a particular part of the organization. Assuming that the views in the view modelling tree of Fig. 5.1 are being integrated in a bottom up manner, then it is possible, for example, to produce the GCS for the 'Computer Science' view only. Then, in theory at least, the GCS would be complete for this department, and it can be mapped to a DBMS. The top down view integration would ultimately be the same as the bottom up integration; this is because, for all non terminal views (views which have subviews), all the semantics are contained in the subviews. Therefore, any top down integration must consider the terminal views in order to produce the parent view, and thus the integration process ends up being similar to the bottom up integration.

Another approach is to choose the views randomly. This means that the next view to be integrated can be from any part of the view modelling tree; so long as it has not been considered previously. Ultimately, the final GCS should not be any different to that produced by integrating the views in a bottom up tree-like manner. However, it is not possible using this approach to produce completed GCSs for particular sections (views) of the organization. Further, the type and total number of conflicts may be different to that encountered in the bottom up tree-like approach. One disadvantage which can be associated with this type of binary view integration is that the user whose view is being

integrated could be any of the users represented in the view modelling tree. This may cause problems to the designer. In the bottom up tree-like-manner, the designer would see the gradual incrementation of the GCS, and he will slowly develop familiarity with it. This will not occur when following a random approach. Further, random binary view integration might cause some views to exist as a separate set of E-Rs in the GCS. Consider, for example, that the views 'rooms', 'conferences' and 'exams' have been integrated to form the GCS. Then assume that the view 'Industry' is randomly chosen as the next view for integration. It is highly unlikely that the current GCS and the view 'Industry' would share any entities or relationships.

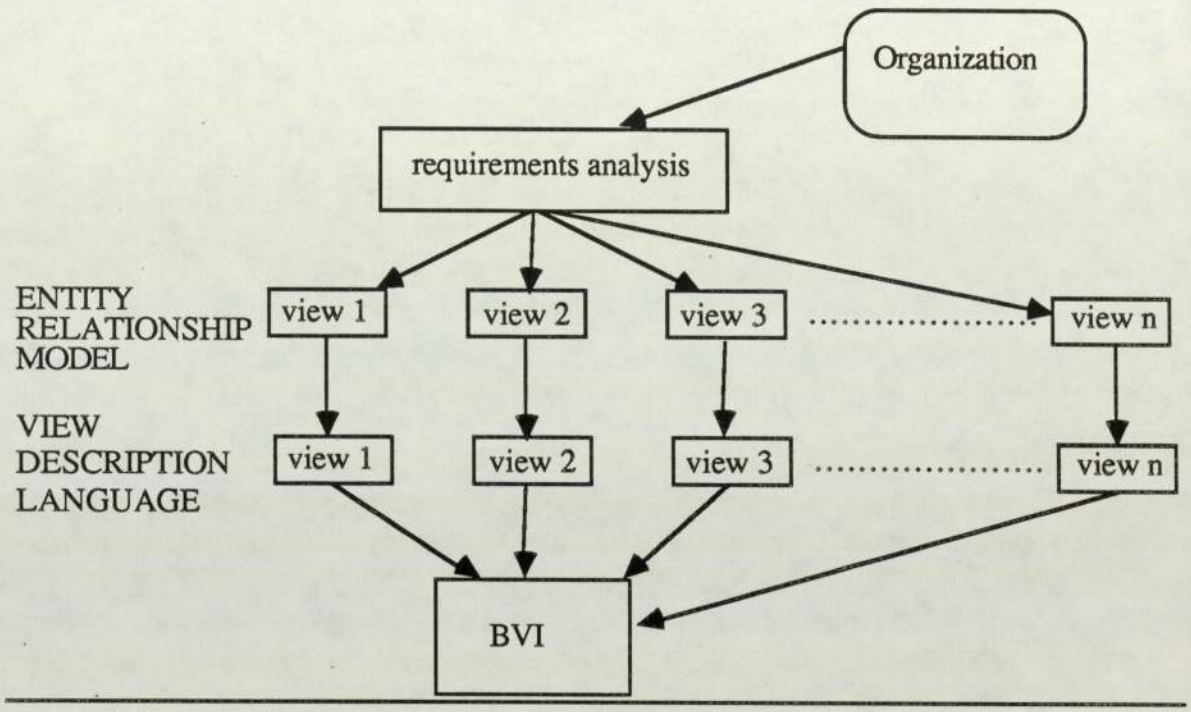


Fig. 6.6 A framework of view modelling and view integration

6.4.3 Choice of next E-R

Once a view is chosen for integration as described above in section 6.4.2, and illustrated in Fig. 6.7, then BVI must decide which E-R within the current view should be integrated. In binary view integration, the next E-R to be integrated from the current view can either be chosen randomly, or based on some predetermined or calculated priority. Since all the E-Rs of a given view are all related and belong to the same user, the random choice has no significant effect on either the number or type of conflicts resulting from integration. Therefore, BVI was designed to randomly choose E-Rs from the current view.

6.4.4 Overview of BVI structure

Before going through the details of how views and E-Rs are integrated to form the GCS, the general layout of BVI will be described. BVI is made up of a number of modules, the top most module of which is called 'Binary VIM'. The 'Binary VIM' module plays two roles: the first role is the implementation of the algorithm which chooses views and E-Rs for integration as well as carry out all the other integration tasks, and the second is that it is the centre of activities required for all the other modules to run. Fig. 6.9 shows an outline of the binary view integration algorithm upon which BVI was based.

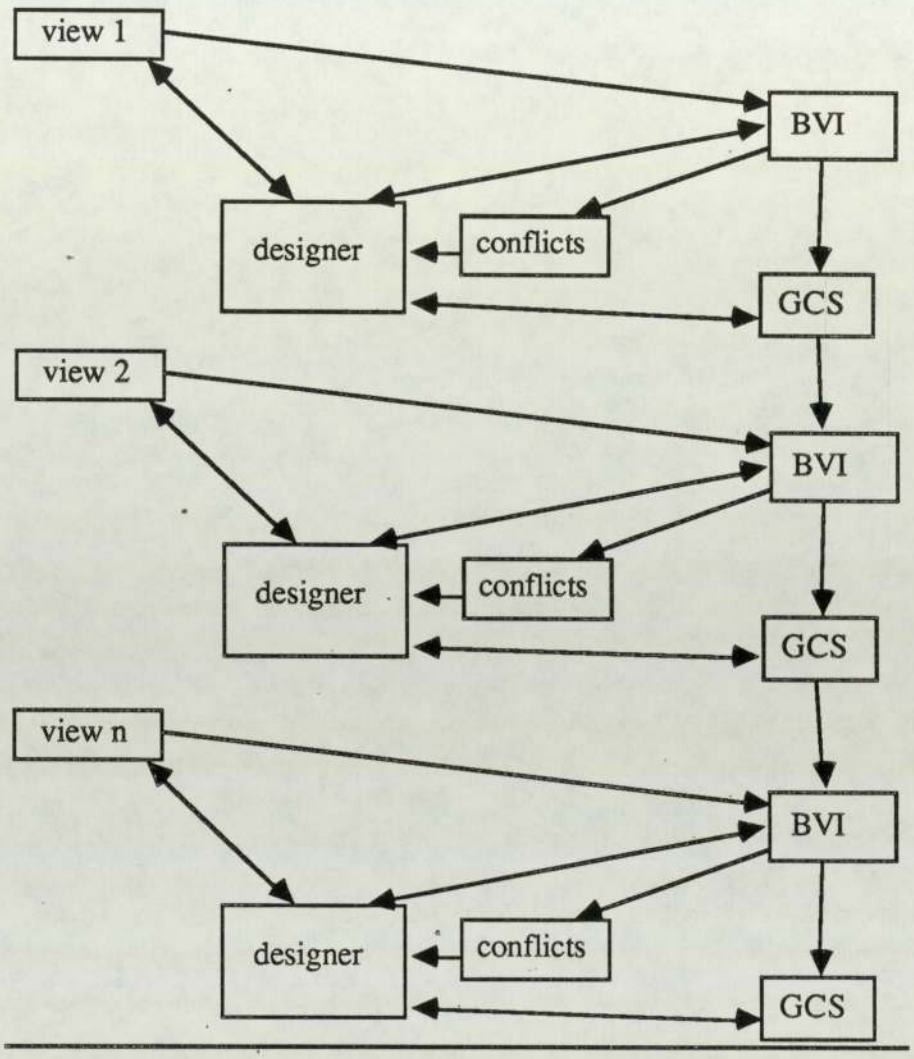


Fig. 6.7 An outline of binary view integration

The first function of the 'Binary VIM' module is to read in all the definitions of the views which are written in VDL. The definitions of these views must be according to the VDL syntax, as shown in Fig. 5.2. Should the definitions of the views differ from the VDL syntax, then the designer is called in to correct the syntax. Each syntactically correct view is transformed to its equivalent internal data structure representation in the form of the Prolog relations defined in section 6.2. This process is achieved by the module 'VDL to internal relations'. The reading in of all the views, and their transformation to their equivalent internal relations is a pre-integration activity.

At first, the GCS is empty. BVI chooses the next view, either randomly or in a bottom up tree-like manner as described earlier. Once a view is chosen for integration, BVI randomly chooses the next E-R to be integrated from this view, and calls the module 'relationship matcher' to handle all the activities needed to achieve the integration of this E-R. The integration of E-Rs is based on the integration situations shown in chapter 3.

The module 'relationship matcher' calls a number of its submodules to establish the kind of match that exists between the current E-R of the current view and all the E-Rs of the GCS. The current E-R of the current view being integrated will be referred to as the *current view E-R*. The GCS is updated by the integration of the current view E-R, so that it does not have any duplicate objects. This means that the GCS contains only one occurrence of each entity and relationship. Therefore, the module 'relationship matcher' finds only one possible match in the GCS for the current view E-R. In order to avoid programming all the possible situations shown in table 3.1, and discussed in chapter 3, BVI was designed so that E-R matching between the current view E-R and the GCS E-Rs falls into one of two categories. The GCS E-Rs will either have a *matching E-R* to the current view E-R or all the GCS E-Rs will be *different* E-Rs to the current view E-R. Both matching E-Rs and different E-Rs categories, and the way they are handled by the module 'relationship matcher' and all its submodules, are explained in section 6.4.5.

Once the module 'relationship matcher' and all its submodules establish the type of match between the current view E-R and the GCS E-Rs, it calls the module 'merger' to update the GCS with these new semantics acquired from the current view E-R. The module 'merger' adds the semantics from the current view E-R to the GCS and ensures that no duplicate semantics are added to the GCS. Duplication in the GCS does not affect the attributes, and, as will be seen later, it is possible for the same attribute to be associated with different objects (entities and relationships). Duplication of attributes defining the same object is not permitted.

The module 'merger' is not responsible for exposing any inconsistencies which might be caused to the schema, in the form of synonyms. Further, the module 'merger' is not responsible for identifying and resolving COT conflicts. The implementation of COT conflicts is described in section 6.4.6. The object fuzzy matching method developed to identify synonyms is achieved by the module 'objects fuzzy matcher'. This module is described in section 6.5.

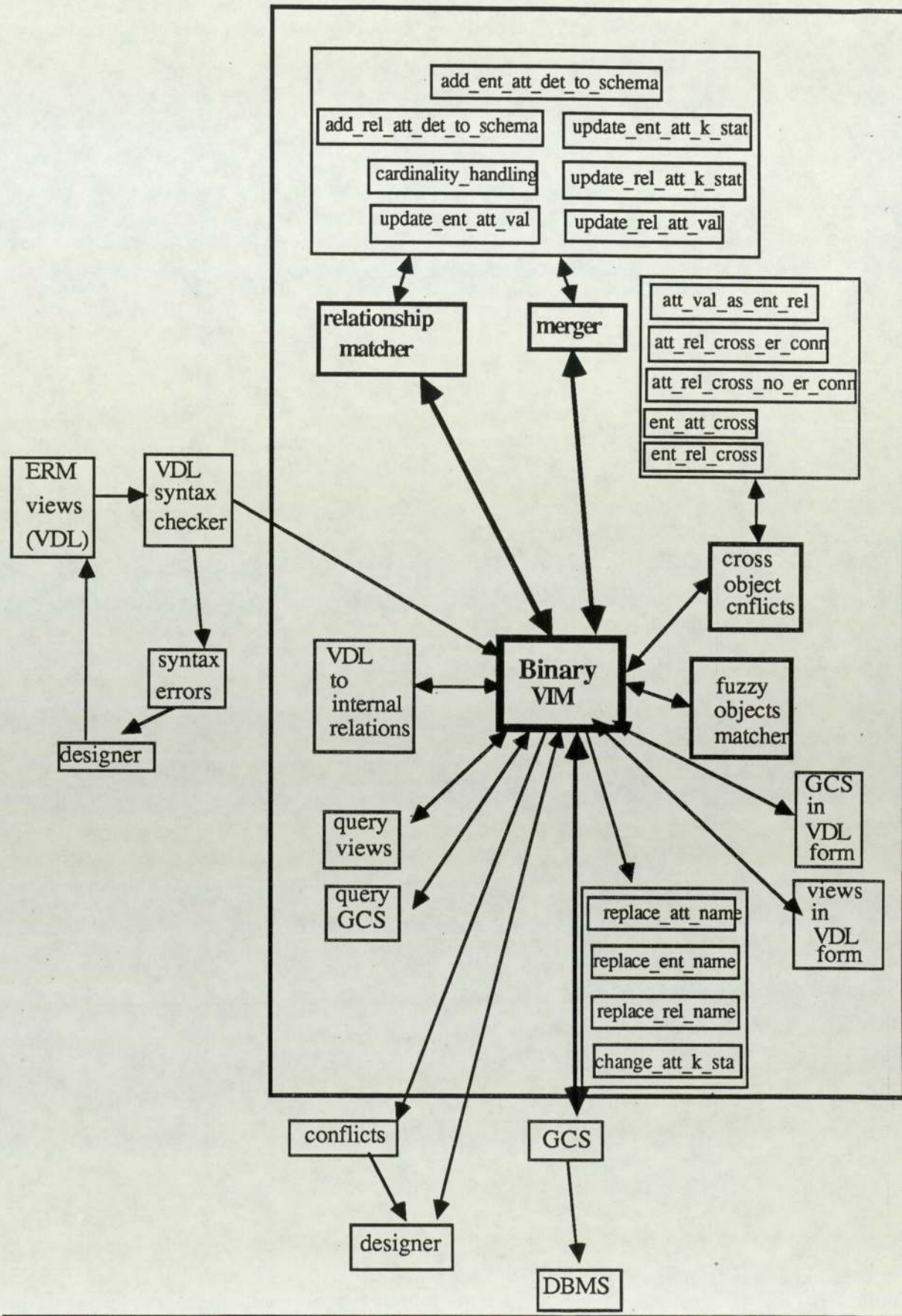


Fig. 6.8 Overall structure of BVI

6.4.5 Matching and integrating E-Rs in BVI

Table 3.1 showed thirty two situations which can arise when matching two E-Rs. Twenty four out of these thirty two situations were valid. These ranged from total match to total difference on entity names and relationship names. The attributes which define the entities and relationships were not included as part of the conditions for match in the table. The main reason for this is that the inclusion of more conditions would create a great number of possibilities.

BVI in general, and the two modules '`relationship matcher`' and '`merger`' and their submodules in particular, were designed to match E-Rs, so that the match always yields one of two major possibilities. As mentioned in section 6.4.4., the two E-Rs can either be matching E-Rs or different E-Rs. BVI decides that two E-Rs are of the matching E-Rs category if they are *identical*, *nearly identical*, *closely similar* or *similar*. Any two E-Rs that achieve any of these matches, causes BVI to initiate certain updating actions to the attributes of the corresponding GCS E-Rs (see the BVI algorithm in Fig. 6.9).

For BVI to decide that the current view E-R is a different E-R to all the GCS E-Rs, the former must have one or more objects different to the latter, where objects here does not include attributes. Based on this, two E-Rs are classified by BVI as different E-Rs if they differ on two entity names, two entity names and the relationship names, four entity names, four entity names and the relationship names or the relationship names. The way in which BVI handles the matching of different E-Rs and update the GCS is described in section 6.4.5.5.

6.4.5.1 Identical E-Rs

For the current view E-R to be identical to an E-R from the GCS, the two E-Rs must match on everything, that is:

1 The entity names and relationship names.
2 The entity and relationship attributes.
  2.1 The attribute names.
  2.2 The attributes key statuses.
  2.2 The attribute values.
3 The cardinalities.
4 The roles (if given).

Since the two E-Rs are identical, all the semantics that are contained in the current view E-R, are also contained in the GCS E-R. Therefore, the semantics of the GCS are not updated by any of the semantics of the current view E-R.

1 **Pre-integration.**

1.1 Read the views in VDL and check their syntax.

1.2 Represent the views in the Prolog relations.

2 **View integration.**

2.1 Choose the next view for integration (either at random or bottom up).

2.2 Choose the next E-R for integration at random, and call it the current view E-R.

2.3 Match the current view E-R with all the GCS E-Rs.

2.3.1 If an *identical* GCS E-R is found, then do not change the GCS.

2.3.2 If a *nearly identical E-R* is found, then update the GCS relationship with the attributes of the relationship of the current view E-R.

2.3.3 If a *closely similar E-R* is found, then update the GCS E-R with the attributes of the entities and relationship of the current view relationship.

2.3.4 If a *similar E-R* is found, then update the GCS E-R with the attributes of the entities and relationship of the current view E-R, and update the cardinalities and roles.

2.3.5 If a *different E-R* is found, then update the GCS E-R with any object or connection that is not already part of the GCS.

2.3.6 If there are E-Rs in the current view not yet considered, then go to 2.2.

2.3.7 If there are any more views not yet considered, then go to step 2.1.

3 **Post-integration.**

3.1 COT conflicts.

3.2 Fuzzy matching of objects.

Fig. 6.9 The binary view integration algorithm and stages

It is very rare for a current view E-R to be identical to a GCS E-R because the GCS E-Rs and their corresponding objects are continually updated with semantics from the views E-Rs and it is very rare for two E-Rs to be modelled independently in an identical way (assuming that the corresponding GCS E-R has not been updated). However, identical E-R situations could still take place, even though it is a rarity, and BVI has to accommodate such situations.

Assume that the E-R in Fig. 3.4 defines the current view E-R and the E-R in Fig. 3.5 defines the GCS E-R. Rather than describe the modules and predicates involved in establishing that the two E-Rs are identical, the path followed through these modules and predicates is briefly outlined (see appendix J). The predicate 'match_view_rel_with_schema_rels' of the module 'Binary VIM' calls the predicate 'match_rels' of module 'relationship_matcher', and passes it the full 'rel' relation defining the current view E-R, and the 'ent_rel_s' relation defining the GCS E-R. The 'ent_rel_s' relation is passed to indicate the GCS E-R instead of the 'rel_names_s' relation because the GCS relationship name can be used in two different binary E-Rs (see section 3.3.2.2). Therefore, since the GCS relationships are not associated with occurrence numbers, the only way to be sure that the correct E-R is being considered for integration is to pass the 'ent_rel_s' relation. Therefore, these two relations have enough information to uniquely identify the two E-Rs, and enable the other predicates and modules to retrieve any of their details. The 'match_rels' predicate in turn calls the 'identical_rels' predicate, where the latter initiates the testing to establish that all the objects of the two E-Rs are identical.

The 'identical_rels' predicate calls the predicates 'match_ent_cards', 'ident_rel_atts', and the 'ident_rel_ents'. The 'match_ent_cards' matches the corresponding cardinalities of the entities. The 'ident_rel_atts' establishes that the attributes of the two relationships (if given) are identical. This means that the attributes of the current view E-R, and the attributes of the GCS relationship match exactly on all of the attribute names, key statuses and values. The 'ident_rel_ents' predicate sets out to establish that the entities of the two E-Rs are identical. This predicate ends up calling the predicate 'ident-ents' which first of all matches the corresponding entity names, and once the corresponding entities of the two E-Rs are established, it initiates the testing of their attributes, by calling the appropriate predicates. These latter predicates ensure that for each attribute in one of the entities of the current view E-R, there is a corresponding attribute from the other entity of the GCS E-R, such that the two attributes match on name, value and key status.

Once the two E-Rs are established to be identical, the 'match_rels' predicate is satisfied, and control is returned to the 'Binary VIM' module to extract another E-R from the current view, or if the current view has no more E-Rs, start on a new view.

6.4.5.2 Nearly identical E-Rs

For the current view E-R to be regarded by BVI to be nearly identical to one of the GCS E-Rs, the two E-Rs must match on the following:

1 The entity names and relationship names.

2 The attributes of entities.

  2.1 The attribute names.

  2.2 The attribute key statuses.

  2.2 The attribute values.

3 The cardinalities.

4 The roles (if given).

The difference between identical E-Rs and nearly identical E-Rs, is that the latter do not match on relationship attributes. This does not mean that the relationship attributes are necessarily totally different, but that they could differ on one or more attributes or on one or more of the characteristics of their attributes. The aim of establishing such a test is to update the GCS relationship attributes with the attributes of the current view relationship, so that the GCS relationship ends up with the semantics of both sets of attributes, with the conflicts resolved.

All the predicates used to ensure that the current view E-R and the GCS E-R are identical, are called in exactly the same way as described in the identical E-R section above, with the exception of the predicate `'ident_rel_atts'`. The module `'relationship_matcher'` initiates the predicate `'near_identical_rels'` after it has proved that the two E-Rs are not identical. Once the predicate 'near_identical_rels' is satisfied, the predicate `'update_near_ident_rel'` of module `'merger'` is called, which in turn calls the predicate `'add_rel_att_det_to_schema'` of a module with the same name.

The predicate `'add_rel_att_det_to_schema'` has the task of identifying all the differences between the two sets of attributes of the two relationships, and updating the schema accordingly. Section 3.4. showed how two entities which match on names are matched and integrated, so that the resultant entity contains all the semantics from both entities, with the conflicts resolved. Exactly the same approach is used by BVI to update the GCS relationship attributes from the current view relationship attributes. The outline of the procedure is as follows:

1 Each attribute of the current view relationship, which is not in the GCS relationship, is added to the GCS relationship, with all its characteristics.

2 Each attribute in the current view relationship which matches an attribute from the GCS relationship on all its characteristics, is disregarded.

3 Each attribute from the current view relationship, which matches an attribute from the GCS relationship on name, but differs from it on key status or value or both, requires the designer to decide on the final key status. The updating of the value is done automatically in most cases (see section 3.4.2.3)

The modules 'update_rel_att_k_stat' and 'update_rel_att_val' are called to match the key statuses and values of each of the attributes which match on name. For non matching key statuses and some cases of non matching values, their final status is decided by the designer. This decided value or key status is kept in a history of conflicts relation, and used to help resolve all future key status or value conflicts involving this attribute.

6.4.5.3 Closely similar E-Rs

Once the current view E-R and the GCS E-Rs are established not to be identical or nearly identical, the 'relationship_matcher' module initiates the closely similar test. The two E-Rs can be closely similar if they match on the following:

1 The entity names and relationship names.

2 The cardinalities.

3 The roles (if given).

This means that the two E-Rs mismatch on the entity attributes and the relationship attributes. Again, the difference in the attributes between the four entities and the relationships can range from a difference on one of the characteristics of the attributes, to total difference on all attributes. The updating of the relationship attributes is exactly the same as described above in the nearly identical E-Rs section. The updating of the GCS entity attributes is achieved in exactly the same way as the relationship attributes update. The updating of the GCS entity attributes is carried out by the modules 'add_ent_att_det_to_schema', which in turn calls the modules 'update_ent_att_k_stat' and 'update_ent_att_val'. These modules ensure that the GCS entities are updated with all the semantics of the attributes of the matching current view entities, with the conflicts resolved.

6.4.5.4 Similar E-Rs

The similar E-R test is carried out by the module 'relationship_matcher', after it has proved that the current view E-R has no identical, nearly identical or closely similar E-R in the GCS. For BVI to decide that the current view E-R is similar to a GCS E-R, the two E-Rs need only match on the entity and relationship names. They can therefore differ on the entity attributes, the relationship attributes, the cardinalities and the roles (if given). The updating of the GCS relationship attributes and entity attributes were discussed in the

nearly identical and closely similar sections above. If either the cardinalities or roles do not match for their corresponding entities in the two E-Rs, then the designer must be requested to make the decision of issuing a new cardinality or role.

6.4.5.5 Different E-Rs

The current view E-R is classified by BVI as a different E-R to all the GCS E-Rs, if the current view E-Rs differs with all the GCS E-Rs on any of the following:

1 Two entity names.

2 Two entity names and two relationship names.

3 Four entity names.

4 Four entity names and two relationship names.

5 Two relationship names.

As discussed in section 3.3.2.2, the problems of naming conflicts causing two separate binary E-Rs to appear as one ternary or n-ary E-R after integration, cannot be directly resolved. Although some suggestions for the resolution of these situations were given in that section, none of these resolutions can be guaranteed to always be the correct one. Therefore, for testing purposes only, the decision was made to regard two binary E-Rs matching on relationship names, but differing on two or more entities, as different E-Rs. This assumption has to be considered carefully when the final GCS is to be mapped to a DBMS, as the attributes of the relationship relation would contain the key attributes of all the entities involved in the relationship. Of course, it is always possible to make BVI request the designer to supply two separate names for the two relationships, thus ensuring that bad binary E-Rs do not exist in the GCS. However, as demonstrated in section 3.3.2.2 (see Fig. 3.11), sometimes relationship names are such that they are most representative of the semantic connection of the three or more entities. The choice of relationship names in particular, and object names in general is significant when querying the database, especially by the casual user.

If an E-R relates more than two entities, then such an E-R is not binary. The mapping of the GCS to a DBMS necessitates that a decision must be taken at the view integration phase, between the choice of one bad binary E-R with a more representative name, and two separate binary E-Rs with less representative names. If two E-Rs are modelled as binary E-Rs in different views, then their integration should produce two binary E-Rs. However, since the argument between the two choices is not decisive as to which one should be chosen, BVI was designed to accept bad binary E-Rs in the GCS. BVI can however be adapted to do otherwise.

If any of the 5 conditions above is satisfied, the predicate 'assert_rel' of the module 'merger' is called. This predicate updates the GCS E-R with any of the semantics from the current view E-R which is not available in the GCS E-R. This is achieved by updating the entities in the GCS E-R with the attributes from the current view E-R, updating the corresponding E-R attributes, and adding to the GCS any entity or relationship which is involved in the current view E-R, but not declared as part of the GCS. To do this, the 'assert_rel' predicate calls a number of modules and consequently their predicates. Any conflicts between any objects or their characteristics are dealt with as in the matching E-Rs situations.

6.4.6 COT conflicts processing

The identification and analysis of the COT conflicts by BVI could be carried out at different stages:

a) COT conflicts before the integration of each object.

Identifying and analysing these conflicts before the integration of each object means that should a COT conflict occur between the current object of the current view E-R and any object in the GCS, BVI must resolve this conflict before the integration process is continued. This approach imposes on BVI the burden of searching through all the objects of the GCS to check for COT conflicts each time a new object is considered for integration. The only advantage to this approach is that the GCS is always free from COT conflicts.

b) COT conflicts after the integration of all objects.

BVI continually updates the GCS with the contents of the views. After integration, the total number of objects in the GCS is less than the total number of objects in the views. Therefore, to carry out the COT conflicts identification and analysis after integration would require BVI to search through the GCS only once for each type of conflict. Further, the existence of one of these conflicts during integration has no effect on the final state of the GCS, as long as such a conflict is identified after integration. Since the majority of these conflicts cannot be resolved automatically by BVI, it is better to leave them until after the GCS is completed, so that the designer is able to use the semantics of the GCS and get the necessary statistics regarding the conflict concerned. Therefore, the COT conflicts identification and analysis was implemented in BVI as a post-integration process (see Fig. 6.9).

Fourteen COT conflicts were identified and their method(s) of resolutions discussed in section 4.3. Seven of the fourteen algorithms are implemented for demonstration purposes, and their outline is presented here. The others can be implemented in similar ways.

1 Attribute-value as E-R COT conflict.

This COT conflict is identified by BVI through the module 'att_val_as_ent_rel' based on the algorithm in Fig. 6.10.

2 Attribute as relationship COT conflict

a) Same E-R.

The module which checks all the occurrences of this COT conflict in the GCS is 'att_rel_cross_er_conn', where the abbreviation er_conn refers to the involvement of the entity in the E-R concerned. The outline of the algorithm upon which this module is designed is shown in Fig. 6.11.

b) Different E-Rs.

The module checks all the occurrences of this COT conflict in the GCS is 'att_rel_cross_no_er_conn', where the abbreviation no_er_conn refers to the fact that there is no involvement of the entity in the relationship concerned. The outline of the algorithm upon which this module is designed is shown in Fig. 6.12. This module must be called after the module 'att_val_as_ent_rel', so that the attribute concerned is not removed from the entity, although it might contain a value.

1 Choose the next entity of GCS.
2 Choose the next attribute of the chosen entity.
3 Match the chosen attribute and its value(s) with all the relationship names of the GCS, and the entities they relate.
  3.1 If the attribute name matches the relationship name, and a value of the attribute matches any of the entities involved in this relationship, then the conflict 'attribute-values as E-R' exists, therefore do:
    3.1.1 If the attribute is the only key attribute of the entity, then inform the designer, and suggest the resolution of removing the attribute from the entity.
    3.1.2 If the attribute is not the only key attribute of the entity, then inform the designer, and remove the attribute from the entity (if it has no more values).
  3.2 If the attribute has more values which match one of the entities involved in the relationship, then go to step 3.1.
  3.3 If although the attribute existed as a relationship, and one or more of its values existed as entities involved in the relationship, yet one or more of the values of the attribute do not exist as entities, then, suggest to the designer:
    1 The remaining value(s) are synonyms to the other values (entities) already considered.
    or
    2 The remaining value(s) must be created as entities, involved in the same relationship.
4 If the entity has more attributes, go to step 2.
5 If the GCS has more entities, go to step 1.
6 End.

Fig. 6.10 Attribute-value as E-R / same E-R algorithm

1 Choose the next entity of GCS.
2 Choose the next attribute of the chosen entity.
3 Match the attribute with all the relationship names.
  3.1 For any relationship that matches the attribute on name such that the entity of the attribute is involved in this relationship, do
    3.1.1 If the attribute is a key attribute, then
      3.1.1.1 If the entity has another key attribute and the attribute has no value, then remove the attribute from its entity.
      3.1.1.2 If the attribute is the only key attribute, then leave the decision to the designer.
  3.2 If the attribute is not a key attribute, then remove the attribute from the entity.
4 If the entity has more attributes, go to step 2.
5 If the GCS has more entities, go to step 1.
6 End.

Fig. 6.11 Attribute as relationship /same E-R algorithm

1 Choose the next entity of GCS.
2 Choose the next attribute of the chosen entity.
3 Match the attribute with all the relationship names.
  3.1 For any relationship that matches the attribute on name, and the relationship does not involve the entity to which the attribute belongs, suggest to the user one of the following:
    1 Remove the attribute from the entity.
    2 Change attribute name.
    3 Change relationship name.
    4 Leave both attribute and relationship as they are.
4 If the entity has more attributes, go to step 2.
5 If the GCS has more entities, go to step 1.
6 End.

Fig. 6.12 Attribute as relationship /different E-Rs algorithm

3 Entity as attribute COT conflict

The module '`ent_att_cross`' was implemented as part of BVI to achieve the analysis of this conflict, and an outline of the algorithm used for the module is shown in Fig. 6.13.

4) Entity as relationship COT conflict

This COT conflict is identified by the module '`ent-rel-cross`'. The outline of the algorithm upon which this module was implemented is shown in Fig. 6.14.

6.4.7 Object fuzzy matching in BVI

Object fuzzy matching of objects can be carried out at pre-integration phase, during integration or at post-integration. To implement a VI to carry out the object fuzzy matching of objects prior to their integration, would mean that the designer must

ultimately make the decision on the matching of each and every object, or that VI can be given the precise levels of match above which it can decide that the two objects are either the same or different. As will be described in chapter 8, the precise levels of match cannot be issued, and therefore, the designer must decide on each match. The total number of ways of uniquely matching two objects from n objects is n combinations 2 or n C 2, where: n C 2 = n! / (n-2)! 2! = n(n-1) / 2. Assume that we have a number of views containing between them a entities, b attributes and c relationships. This creates a total number of matches to be checked by the designer of

$$a(a-1) / 2 + b(b-1) / 2 + c(c-1) / 2$$

1 Choose the next entity of GCS.
2 Choose the next attribute of the chosen entity.
3 Match the attribute with all the entity names.
  3.1 If the GCS has an entity which matches the attribute, then:
    3.1.1 If the entity is the owner of the attribute, then inform the designer, and allow him one of the following actions:
      1) Delete the attribute.
      2) Change the attribute name.
      3) Change the name of the entity.
      4) Accept the conflict.
    3.1.2 If the entity is not the owner of the attribute (foreign entity), then
      3.1.2.1 If the owner entity and the foreign entity are involved in a relationship, then suggest to the designer that it is likely that the attribute is not needed as part of the owner entity, and allow him one of the following actions:
        1) Delete the attribute.
        2) Change the attribute name.
        3) Change the name of the foreign entity.
        4) Accept the conflict.
      3.1.2.2 The owner entity and the foreign entity are not involved in a relationship, then suggest to the designer that may be a relationship needs to be created between the owner entity and the foreign entity, and allow him one of the following actions:
        1) Delete the attribute.
        2) Change the attribute name.
        3) Change the name of the foreign entity.
        4) Accept the conflict.
        5) Create a relationship between own and foreign entities.
4 If the entity has more attributes, go to step 2.
5 If the GCS has more entities, go to step 1.
6 End.

Fig. 6.13 Entity as attribute algorithm

1 Choose the next E-R of GCS.

2 Match the relationship with the next entity name from the GCS.

2.1 If the GCS has an entity which matches the relationship on name, then:

2.1.1 If the entity is involved in the relationship (own ent-rel, then inform the designer, and allow him one of the following actions:

1) Change the entity name.

2) Change the relationship name.

3) Accept the conflict.

2.1.2 If the entity is not involved in the relationship (foreign entity-rel), then inform the designer, and allow him one of the following actions:

1) Change the entity name.

2) Change the relationship name.

3) Create a new relationship between the entity of the relationship and the entity concerned.

4) Accept the conflict.

2.1.3 If the owner entity and the foreign entity are involved in a relationship, then suggest the designer that may be a relationship need to be created between the owner entity and the foreign entity, and allow him one of the following actions:

1) Delete the attribute.

2) Change the attribute name.

3) Change the name of the foreign entity.

4) Accept the conflict.

5) Create a relationship between own and foreign entities.

3 If the GCS has more entities, go to step 2.

4 If the GCS has more relationships, go to step 1.

Fig. 6.14 Entity as relationship algorithm

For example, if a = 50, b = 200 and c = 100, the total number of matches is 26075. It is obvious from this that a large size application of the numbers above would be too tedious for the designer to go through. Another drawback of doing the fuzzy matching of objects prior to integration is that the objects are not in their updated status. For example, entities are not yet updated with their attributes from other entities of the same name, which exist in other views. The application of object fuzzy matching of objects during integration suffers from the same drawbacks as the pre-integration phase.

Carrying out the object fuzzy matching of objects as a post-integration phase benefits from the final state of the integrated schema. The integrated schema contains all the updated entities and relationships with all the attributes of all their other existences in all the views. It also has all the COT conflicts identified and resolved.

The module '`object fuzzy matcher`' calculates all the SLFs for all the objects of the same type, and the value obtained for this factor can be used by either the designer or by BVI to decide if two objects of the same type are synonyms. The SLF can range from 0 to 1, where 0 indicates total difference, and 1 indicates a total match. The unsolved problem in BVI (see chapter 8) is that it is not possible to give a precise value where the SLF indicates that the two objects concerned are either the same or different. The SLF

values are influenced by the the number and type of neighbours, and the weight given to each type of neighbour. BVI produces tables of SLF values for the objects concerned to the designer, and the latter makes the decision. The SLF values are classified into low, medium, and high categories.

Assuming that BVI decided that two objects are the same, then the two objects concerned are declared in the GCS as such. This is achieved by assuming that the two names are multiple names of the same object. Regarding the other characteristics of the two objects, they are combined with each other as in view integration.

6.5 Conclusions

The approach presented in chapter 3 to integrate E-Rs, was implemented under a view integration system called the BVI. This reads the views and transforms them from VDL form to an internal relational form for which Prolog has the equivalent of built-in DBMS. The GCS internal representation was based on that of the views, but with a few necessary changes which enabled both representations to coexist in BVI.

The relational data structure of the GCS can be populated with the extension of the database for which the GCS was developed. The transformation of the views and the GCS from the ERM structure to a relational structure, can be regarded as a mapping technique of ERM to a relational database system. BVI does not include any module to create the relationship relations by assigning them the key attributes of the entities they involve. It is interesting to notice that should the normalization procedure be implemented as part of BVI, then the GCS relations can be regarded as a dynamic database themselves. Relations can be added, deleted or modified - activities which are part of the normalization process.

The view integration part of BVI is based on the E-R integration approach presented in chapter 3. However, the approach in chapter 3 was slightly modified during the implementation of BVI, and instead of the thirty two different situations of table 3.1, and all the thousands of possibilities once the characteristics of the E-R objects are considered, the matching and integration of E-Rs was divided into five categories. This was devised to reduce the size of BVI, and for the testing of the effect of view modelling and view integration on the repetition of E-Rs. However, the process of implementing these five types in this prototype, can be successfully achieved in a commercial system.

In an expert system environment, the internal relational representation of the views and the GCS can be regarded as the definition of the knowledge base and the Prolog predicates as the inference engine. An expert system approach might differ slightly from the approach followed in implementing BVI but an expert system would not necessarily achieve better results. The reason behind this is that the most difficult part of view

integration is understanding the conflicts and their possible methods of resolutions. These conflicts and their possible resolutions are already interpreted in BVI.

CHAPTER 7

N-ARY VIEW INTEGRATOR

7.1 Introduction

N-ary view integration is another approach to view integration, which has as its main objective the simultaneous integration of all the views to form the GCS. From the literature reviewed in chapter 2, Elmasri & Navathe (1984) is the only paper which presents an approach to n-ary view integration, by showing how a group of entities can be simultaneously integrated. These entities are represented in the schema either as subsets or supersets of each other. As long as the intersection of the attribute sets of the entities concerned is not nil, then the subset/superset conclusion is made. By studying this approach, it was found that in the same application almost all entities are subsets of or supersets to other entities. The paper does not give any algorithm showing how this proposed simultaneous integration of entities is achieved nor how the general integration of the views is achieved. Further, the paper does not present any solution to the problem of n-ary integration of both relationships and E-Rs.

The simultaneous integration of views requires the simultaneous integration of all the E-Rs and their corresponding objects, which is not directly possible. Therefore, the following questions remain unanswered in the literature: does n-ary view integration mean the simultaneous integration of all the views, or the simultaneous integration of E-Rs, or the simultaneous integration of the objects forming the E-Rs? The decision was taken here to consider n-ary view integration as the simultaneous partial integration of views, where partial means an E-R, an entity, a relationship or an attribute from all the views. In this way the n-ary view integrator considers all the views, but simultaneously uses only part of their semantics in view integration. This chapter presents three new methods of achieving n-ary view integration, these are:

1) The entity n-ary view integration.
2) The relationship n-ary view integration.
3) The mass n-ary view integration.

These three types of n-ary view integration differ from binary view integration in that they consider all the views simultaneously. However, they are similar to binary view integration in that they integrate one E-R at a time. Further, all the pre-integration and post-integration phases of these n-ary view integration approaches are processed in the same way as in binary view integration.

7.2 Entity n-ary view integration

An *entity view* of a given entity is defined here as all the E-Rs in which this entity is par of. Assume the following:

V is the set of all views, such that $V=\{v_1, v_2, ..., v_m\}$,

E is the set of entities in V, and $\exists (e_1 \in E)$,

R is the set of E-Rs which involve e_1.

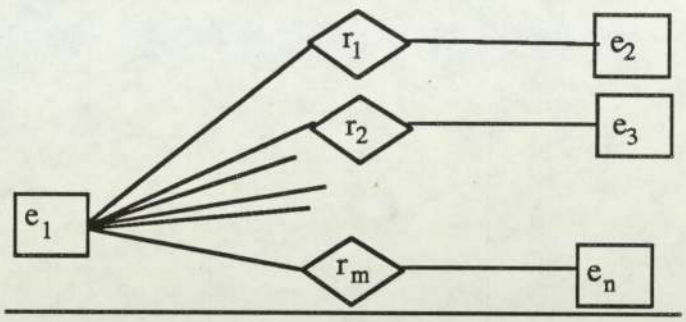


Fig. 7.1 An entity view

Then, the entity view of e_1 is R. Fig. 7.1 shows the entity view of the entity e_1. Assume that the n-ary view integrator to achieve the entity n-ary view integration is called ENVI, and ENVI is based on the algorithm shown in Fig. 7.6. After the pre-integration phase is completed, the view integration phase is started by combining all the entities of the views in one set E, such that there is only one occurrence for each entity in E, regardless of the number of views in which this entity is modelled. This entity set is called the *entity pool.* ENVI then chooses the first entity from the entity pool E, and integrates all the E-Rs in which this entity is involved with the GCS E-Rs, regardless of the view in which these E-Rs are modelled. The entity view of the current entity being considered by ENVI is referred to as the *current entity view.* The integration of the current entity view with the GCS has the eventual effect of the n-ary integration of the entity concerned, because all the occurrences of this entity in all the views are considered together. Therefore, the resultant entity would contain all the attributes from all its occurrences in all the views. This is the same as repeating the process of integrating two entities (shown in section 3.4). Assume that the entities $e_j, e_k, ..., e_m$, are modelled in views $v_a, ..., v_b, v_g$ respectively, such that the entities $v_a e_j$, $v_b e_k$, ..., $v_g e_m$, all match on name. Each time a new occurrence of the same entity is considered, regardless of the view in which it is modelled, the new occurrence of this entity is integrated with the corresponding GCS entity, as shown in Fig. 7.2. The effect of this approach is that these entities are integrated in a binary manner. However, unlike binary view integration, all the occurrences of this entity in all the views are integrated before another entity is considered.

The final entity view of the entities in the GCS is the integration of the entity view of each of these entities in its corresponding view. ENVI considers in turn each occurrence of

each entity in its corresponding view, and integrates its entity view in that view with the GCS.

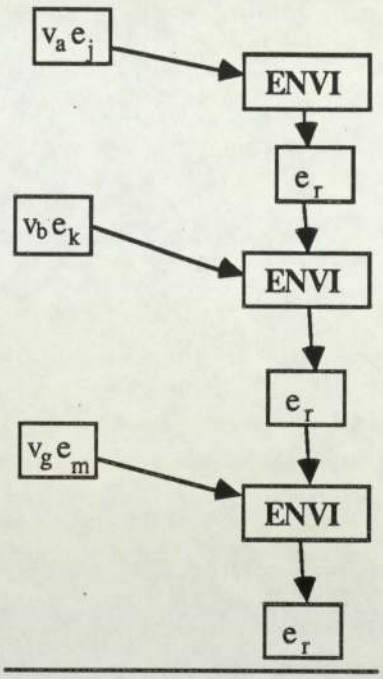


Fig. 7.2 Integration of entities in ENVI

The reasons for the choice of E-R integration in binary view integration, as described in chapter 3, remain the same for n-ary view integration. Therefore, the entity n-ary view integration would effectively be the integration of an E-R at a time with the GCS. However, the main difference between binary view integration and entity n-ary view integration is that in the latter, the current view E-R could be from any view. The current view E-R of the current entity view is integrated with the GCS in exactly the same way as in binary view integration, and therefore, it would either have a matching E-R, or it would be a different E-R to all the GCS E-Rs. The current view E-R of the current entity view is then integrated with the GCS depending on the type of match it has with the GCS E-Rs (see section 6.4.5).

Once the entity view of e_1 of Fig. 7.1 is fully integrated, e_1 becomes e_r of Fig. 7.2. The entity view of e_1 as shown in Fig. 7.1, consists of the E-Rs $\{e_1\, r_1\, e_2\}$, $\{e_1\, r_2\, e_3\}$, ..., $\{e_1\, r_m\, e_n\}$. Once the first entity view is successfully integrated in the GCS, the next entity view is considered, and the latter is achieved in exactly the same way as the first. However, the entity to be considered next, and the effect this choice would have on the entity n-ary view integration, must be considered. One approach is to consider the next entity in the entity pool E. Another approach is to consider the entity view of one of the entities which are already part of last entity view shown in Fig. 7.1. If the second approach is considered, the entity view of the entities e_2, e_3,....., e_m is considered.

However, it was found that the choice of the next entity view has no effect whatsoever on view integration and, therefore, ENVI was designed to choose the next entity from the entity pool E. Assuming that the next entity in E is e_2, the resultant GCS would then consist of the two entity views of e_1 and e_2, as shown in Fig. 7.3. The problem with the entity n-ary view integration, is that duplicate integration of some E-Rs is possible. For example, consider the situation of Fig. 7.3. The entity view of e_2 includes the E-R $\{e_1\ r_1\ e_2\}$. However, this E-R has already been integrated, and is already part of the GCS. Therefore, its integration as part of the entity view of e_2 produces an identical E-R situation (see section 6.4.5.1). However, due to the way that the views are represented internally using relations (see section 6.2), it is an easy task to program NVI to keep a history of such E-Rs to avoid their duplicate integration. In any case, apart from the unnecessary time spent by NVI in duplicate integration of these E-Rs, such integration does not have any effect on the final state of the GCS.

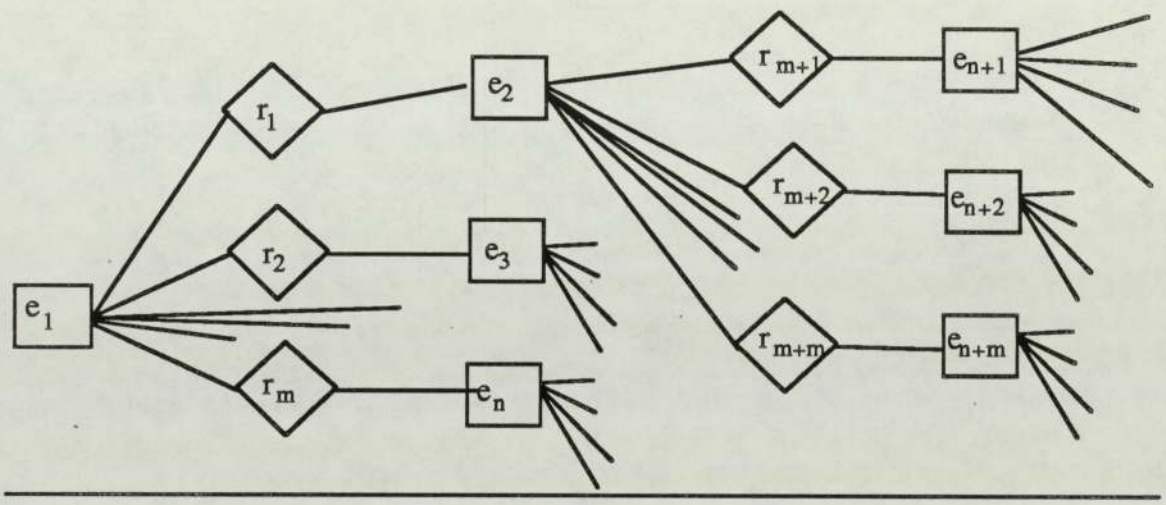


Fig. 7.3 GCS consisting of two entity views

An example of entity n-ary view integration

Consider the three sample entity views of the entity 'student' as shown in Fig. 7.4. The entity n-ary view integration of these three entity views results in the GCS shown in Fig. 7.5. The most obvious advantage of this approach is that the designer may notice any duplicate E-Rs involved in the entity 'student'. He may also notice any synonyms and homonyms. The main reason for such possible observations is that ENVI produces the entity view individually from the rest of the GCS after each entity view is integrated. Of course, should the designer wish to change the GCS, he can issue the appropriate commands.

In binary view integration it is difficult for the designer to notice any duplicate E-Rs and entities caused by naming conflicts, because they are modelled in different views. BVI would recognize such duplicate E-Rs and entities, provided their corresponding entities or

relationships do not suffer from naming conflicts. Object fuzzy matching would contribute to the identification of synonyms, but only after view integration is completed. The integration of the three entity views of Fig. 7.4, results in the GCS entity view shown in Fig. 7.5. The designer may notice some of the conflicts present in the GCS, and may consequently decide to issue the relevant commands to ENVI so as to change the GCS in order to resolve these conflicts. For example, the relationships 'registered on' and 'from' are synonyms, and, therefore, can be declared as the same relationship. The same applies to the relationships 'belongs to' and 'registered in', as well as the entities 'department' and 'dept.'.

Object fuzzy matching (see section 4.2), and COT conflicts analysis (see section 4.3), have the same effect when applied to n-ary view integration or binary view integration. In entity n-ary view integration, object fuzzy matching can be applied to the current entity after each entity view is integrated. The only difference between applying object fuzzy matching after each entity view and applying it at post integration, is that in the first case the designer might make a better decision in regards to the SLFs produced. However, since during integration some entities would simultaneously exist in both the GCS and the views, a decision has to be made as to whether the current entity should be matched with the entities in the views or with the entities in the GCS. Should object fuzzy matching be applied after each entity view, then it is regarded here that the current entity should be object fuzzy matched with the GCS entities only, in order to benefit from the updated status of the entities.

Object fuzzy matching and COT conflicts analysis in entity n-ary view integration are handled in exactly the same way as in binary view integration. Therefore, once all the E-Rs from all the views are integrated, object fuzzy matching and COT conflicts analysis can be applied. Exactly the same reasons given for not doing object fuzzy matching and COT conflicts analysis as a pre-integration task or during integration in binary view integration apply here.

7.3 Relationship n-ary view integration

A *relationship view* of a particular relationship is the set of the E-Rs related by this relationship. Assume the following:

V is the set of views, such that $V=\{v_1, v_2, ..., v_m\}$,

R is the set of relationships in V, and $\exists (r \in R)$,

R_1 is the set of E-Rs from all the views which involve r.

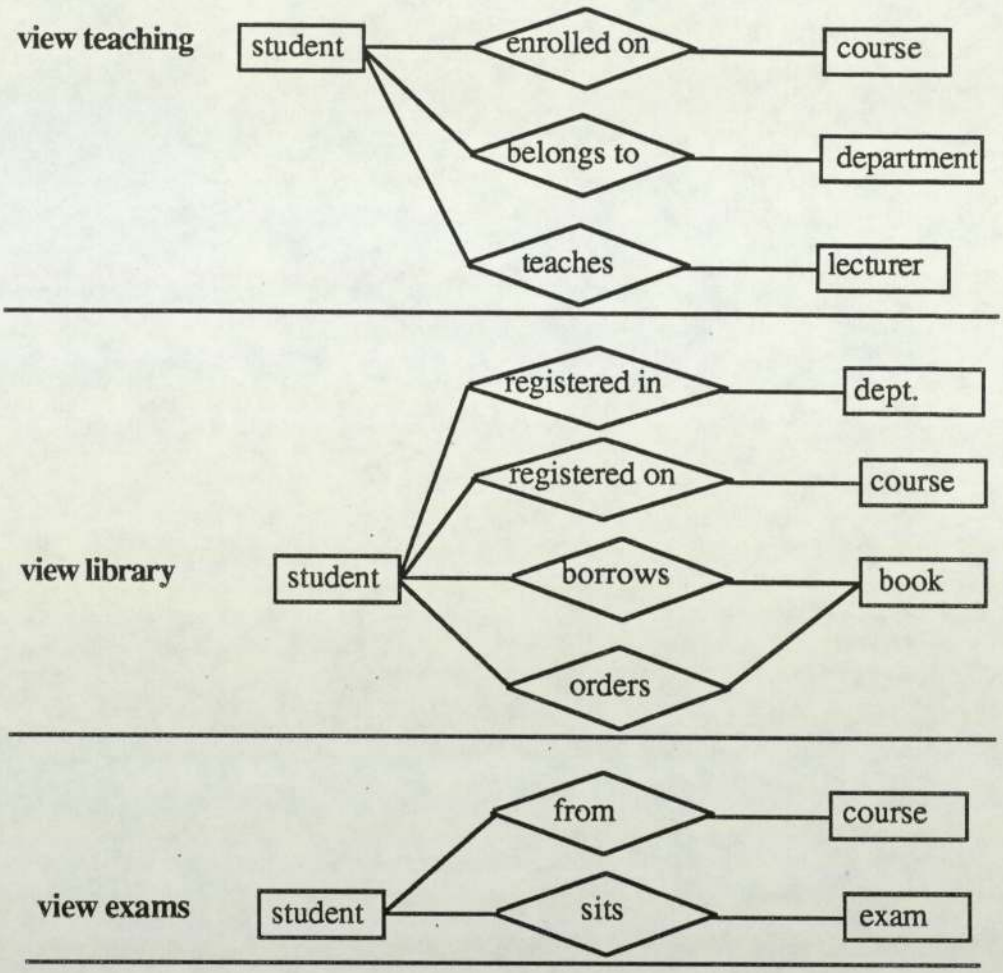


Fig. 7.4 Examples of entity views from different views

Then, the relationship view of r is R_1. Fig. 7.7 shows examples of relationship views of the relationship 'works for' from different views. However, since these are binary E-Rs, the relationship view of 'works for' from one view is one binary E-R. The relationship view of 'works for' from all the views is a set of E-Rs, which could possibly be similar. In effect, the relationship n-ary view integration is the integration of identical, near identical, closely similar or similar E-Rs (see section 6.4.5) from all the views simultaneously. This gives the designer the opportunity of studying all matching E-Rs, and, therefore, he is likely to notice any entities which are synonyms or homonyms involved in these E-Rs. In the GCS relationship view of Fig. 7.8, the entity 'staff' could be synonymous to entities 'technician', 'secretary' and 'lecturer'. Further, the entity 'dept.' and 'department' are synonyms. Synonymous entities can be integrated as one entity (see section 3.4).

Relationship n-ary view integration is executed by RNVI, and it is based on the algorithm shown in Fig. 7.10. The relationships from all the views are combined in one set called the relationship pool R, such that there is only one representation for each relationship. RNVI then chooses the first relationship from R, and integrates all the E-Rs in which this

relationship is involved with the GCS. The relationship view of the current relationship being considered by RNVI is called the *current relationship view*, which is normally a set of identical, near identical, closely similar or similar E-Rs. The integration of the current relationship view with the GCS has the effect of the n-ary integration of the relationship concerned, because all the occurrences of the relationship in all the views are considered simultaneously. Therefore, the resultant relationship would contain all the attributes from all its occurrences in all the views, which is the same as repeating the process of integrating two relationships, as shown in section 3.4. Assume that a relationship r is modelled in the views v_a, v_b,...., v_g, then, the relationship view of r in the GCS is the integration of the relationship view of each of these relationships from its corresponding views. As shown in Fig. 7.9, RNVI considers each occurrence of each relationship in its corresponding view, and integrates its relationship view in that view, with the GCS occurrence of the same relationship.

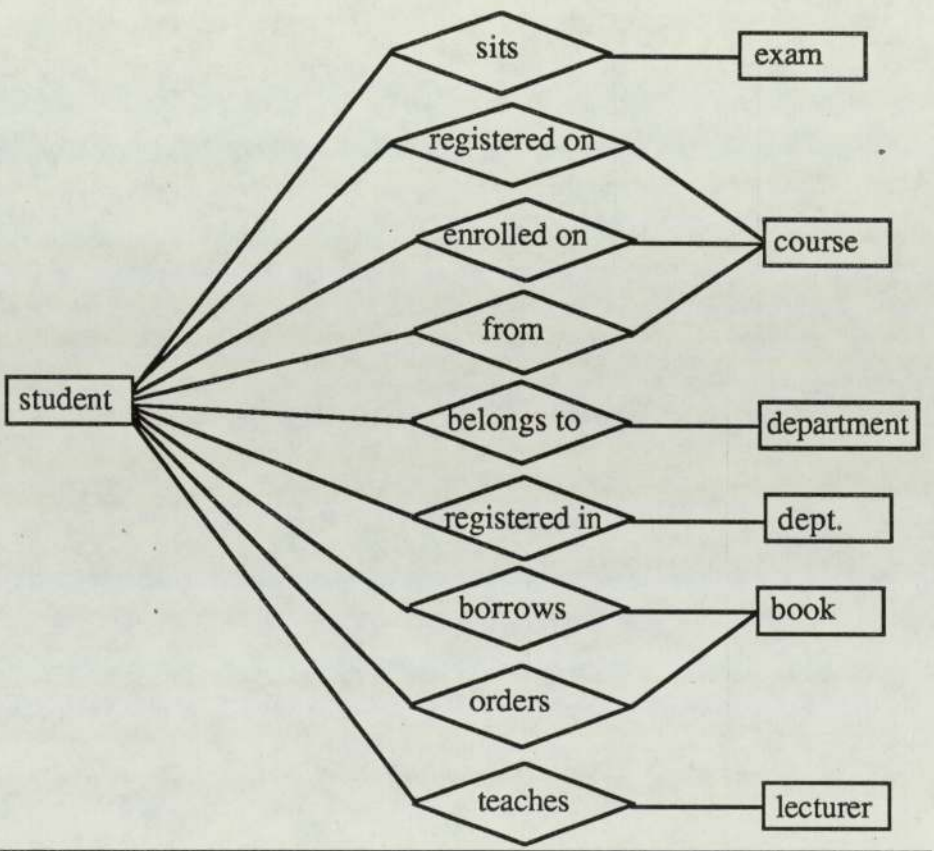


Fig. 7.5 Entity view of entity student after integration by ENVI

Since only binary E-Rs are modelled, the relationship view would contain only binary E-Rs. Occasionally, there might be the bad binary E-R (see section 3.3.2.2), which might involve more than two entities. Therefore, the relationship n-ary view integration should encounter E-Rs with matching (similar) entities. Although these entities could be related

by other relationships, the relationship n-ary view integration integrates all their occurrences in the current relationship view.

1 **Pre-integration.**
- 1.1 Read the views in VDL and check their syntax.
- 1.2 Represent the views in the Prolog relations.

2 **View integration.**
- 2.1 Identify all the occurrences of each entity in all the views, and represent all the occurrences of the same entity once in the entity pool E.
- 2.2 Choose the next entity E_1 from the entity pool E, and this becomes the entity view.
- 2.3 Choose the next E-R which involves E_1 from any view, and this becomes the current view E-R.
- 2.4 Match and integrate the current view E-R with all the GCS E-Rs.
  - 2.4.1 If an *identical* GCS E-R is found, then do not change the GCS.
  - 2.4.2 If a *nearly identical* E-R is found, then update the GCS E-R with the attributes of the relationship of the current view E-R.
  - 2.4.3 If a *closely similar* E-R is found, then update the GCS E-R with the attributes of the entities and relationship of the current view E-R.
  - 2.4.4 If a *similar* E-R is found, then update the GCS E-R with the attributes of the entities and relationship of the current view E-R, and update the cardinalities and roles.
  - 2.4.5 If a *different* E-R is found, then update the GCS E-R with any object or connection that is not already part of the GCS.
- 2.5 If there are E-Rs in the current entity view not yet considered, then go to 2.3.
- 2.6 If there are any more entities in the entity pool E not yet considered, then go to step 2.2.

3 **Post-integration.**
- 3.1 COT conflicts.
- 3.2 Object fuzzy matching.

Fig. 7.6 Entity n-ary view integration algorithm

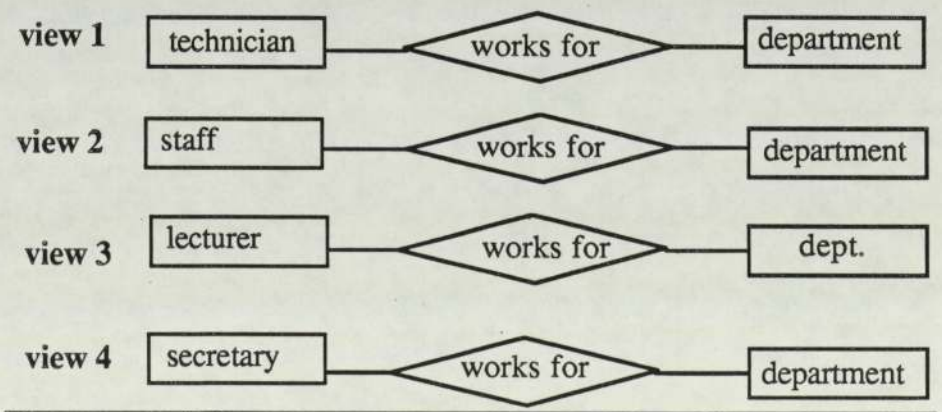


Fig. 7.7 Examples of relationship views

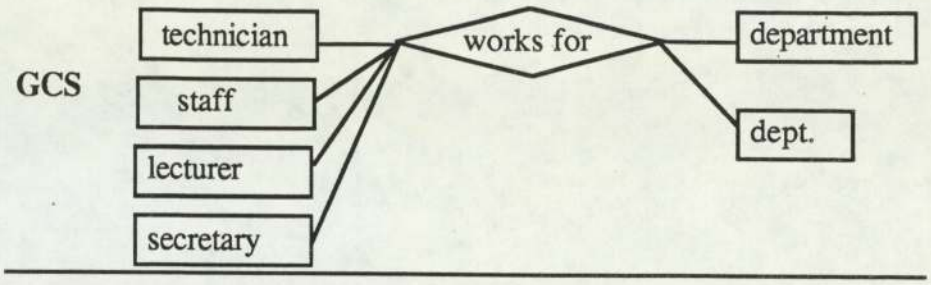


Fig. 7.8 A relationship view of relationship 'works for'

The reasons for the choice of E-R integration in binary view integration remains the same for relationship n-ary view integration. Therefore, the relationship n-ary view integration would effectively be the integration of an E-R at a time with the GCS. However, the main difference between binary view integration and relationship n-ary view integration is that in the latter, the current view E-R could be from any view. The current view E-R of the current relationship view is integrated with the GCS in exactly the same way as in BVI. Therefore, the current view E-R of the current relationship view would either have a matching E-R in the GCS, or it will be a different E-R to all the GCS E-Rs. Depending on the resultant type of match, the current view E-R of the current relationship view is integrated, as shown in section 6.4.5. Once the first relationship view is successfully integrated with the GCS, the next relationship view is considered, and the latter is integrated in exactly the same way as the first.

7.4 Mass n-ary view integration

The difference between the views and the GCS as far as the internal representation in NVI is concerned, is the format of the relations in which they are represented (see sections 6.2 and 6.3). Therefore, if the view relations are transformed to the GCS relations, then the the GCS can be considered complete, although it will contain conflicts. However, since all the semantics from all the views are resident in the GCS, all that is needed is to identify and resolve all the conflicts in the GCS, and the integration process would be complete. The process of transforming the views from their internal relations format to the GCS internal relations format, and then identifying and resolving the conflicts in the GCS, is called here the *mass n-ary view integration.*

The main advantage of the mass n-ary view integration, is that the GCS at any of its stages during integration can be considered as one view which could have been modelled by the designer instead of modelling a number of views. Therefore, mass n-ary view integration can be used if the designer wishes to model the whole organization as one view, and thereby avoid the view modelling approach presented in chapter 5. In this research, MNVI for mass view integration was not implemented, but its algorithm was

designed (see Fig. 7.11). The main disadvantage of the mass n-ary view integration is that since neither the view name nor the occurrence number are associated with the objects, it may not be possible to refer to the correct user should a conflict occur.

Within mass n-ary view integration, it is also possible to use the approaches of entity view or relationship view integration. The only difference between applying these kinds of integration to the GCS instead of applying them to the views, is that no reference to views is maintained. The algorithm in Fig. 7.11 shows mass n-ary view integration, based on entity view integration. The algorithm of steps 2.1.1 to 2.1.5, shows how the transformations of the views relations format to the GCS relations format is carried out. The entity pool E is then established, before entity view integration is started.

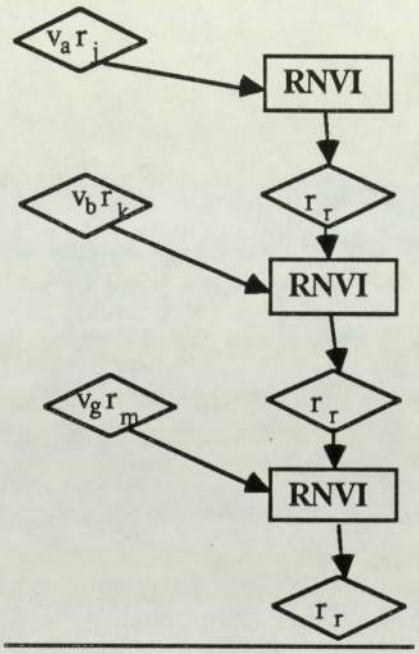


Fig. 7.9 Integration of relationships in RNVI

7.5 Overview of entity view NVI

NVI is the view integrator to process both the entity view and relationship n-ary view integration. NVI is designed in such a way that it can access all the semantics of the views simultaneously. The correct semantics from the views are added to the GCS, and the conflicts which cannot be resolved automatically by NVI are reported to the designer. As in BVI, the designer can correct these conflicts by changing the semantics of the GCS, but the designer may not change the semantics of the views. The designer can query the semantics of both the views and the GCS. Fig. 7.12 shows the layout of NVI. The overall structure of NVI showing the main modules is the same as that of BVI (see Fig. 6.8).

1 **Pre-integration.**
- 1.1 Read the views in VDL and check their syntax.
- 1.2 Represent the views in the Prolog relations.

2 **View integration.**
- 2.1 Identify all the occurrences of each relationship in all the views, and represent all the occurrences of the same relationship once in the relationship pool R.
- 2.2 Choose the next relationship R_1 from R, and this becomes the current entity-relationship view.
- 2.3 Choose the next E-R which involves R_1, this becomes the current view E-R.
- 2.4 Match and integrate the current view E-R with all the GCS E-Rs.
  - 2.4.1 If an *identical* GCS E-R is found, then do not change the GCS.
  - 2.4.2 If a *nearly identical* E-R is found, then update the GCS relationship with the attributes of the relationship of the current view E-R.
  - 2.4.3 If a *closely similar* E-R is found, then update the GCS E-R with the attributes of the entities and relationship of the current view E-R.
  - 2.4.4 If a *similar* E-R is found, then update the GCS E-R with the attributes of the entities and relationship of the current view E-R, and update the cardinalities and roles.
  - 2.4.5 If a *different* E-R is found, then update the GCS relationship with any object or connection that is not already part of the GCS.
- 2.5 If there are E-Rs in the current E-R view not yet considered, then go to 2.3.
- 2.6 If there are any more relationships in the relationship pool R not yet considered, then go to step 2.2.

3 **Post-integration.**
- 3.1 COT conflicts.
- 3.2 Object fuzzy matching.

Fig. 7.10 Relationship n-ary view integration algorithm

7.6 Conclusions

This chapter introduced three new approaches to n-ary view integration, these are: entity view, relationship view and mass view. An algorithm was designed for each of these approaches. The algorithms for the entity and relationship n-ary view integration were implemented.

The view modelling tree of Fig. 5.1 cannot be used in the three types of n-ary view integration, and therefore, there is no way of achieving a partially completed GCS. Therefore, whilst it is possible to have one or more complete entity views or relationship views, the final GCS can only be regarded as complete once all the entity views or relationship views are integrated successfully.

1 **Pre-integration.**
- 1.1 Read the views in VDL and check their syntax.
- 1.2 Represent the views in the Prolog relations.

2 **View integration.**
- 2.1 Transform the views relations format to their GCS equivalent relations format.
  - 2.1.1 Choose the next view.
  - 2.1.2 Choose the next E-R from the current view.
  - 2.1.3 Transform the cu rent view E-R to its equivalent GCS relations format.
  - 2.1.4 If the current view has more E-Rs, go to step 2.1.2.
  - 2.1.5 If there are more views, go to step 2.1.1.
- 2.2 Identify all the occurrences of each entity in the GCS, and represent all the occurrences of the same entity once in the entity pool E.
- 2.3 Choose the next entity E_1 from E, and E_1 becomes the current entity view.
- 2.4 Choose the next E-R which involves E_1, this becomes the current view E-R.
- 2.5 Match and integrate the current view E-R with all the GCS E-Rs.
  - 2.5.1 If an *identical* GCS E-R is found, then do not change the GCS.
  - 2.5.2 If a *nearly identical* E-R is found, then update the GCS E-R with the attributes of the current view E-R.
  - 2.5.3 If a *closely similar* E-R is found, then update the GCS relationship with the attributes of the relationship of the entities and relationship of the current view E-R.
  - 2.5.4 If a *similar* E-R is found, then update the GCS E-R with the attributes of the entities and relationship of the current view E-R, and update the cardinalities and roles.
  - 2.5.5 If a *different* E-R is found, then update the GCS E-R with any object or connection that is not already part of the GCS.
- 2.6 If there are E-Rs in the current entity view not yet considered, then go to 2.4.
- 2.7 If there are any more entities in the entity pool E not yet considered, then go to step 2.3.

3 **Post-integration.**
- 3.1 COT conflicts.
- 3.2 Object fuzzy matching.

Fig. 7.11 Mass n-ary view integration algorithm (entity view)

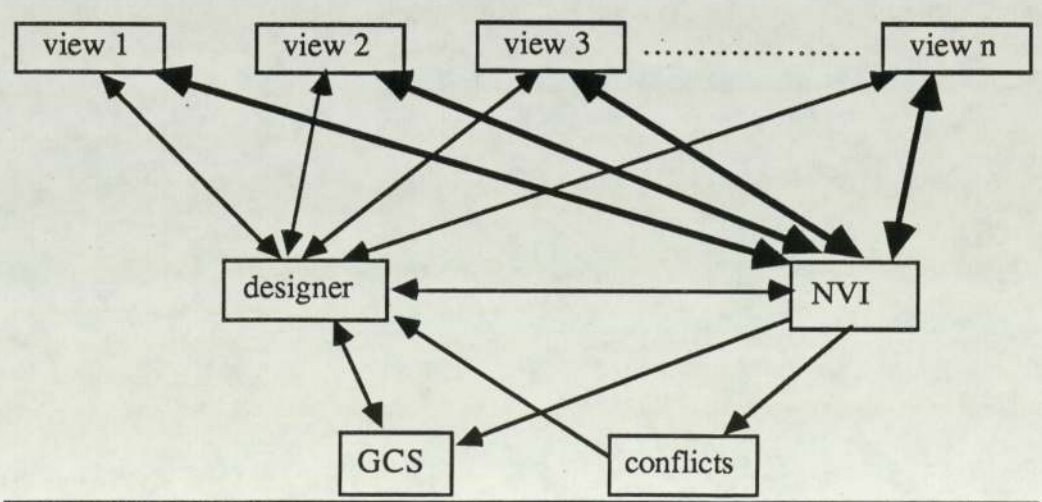


Fig. 7.12 An outline of n-ary view integration

Object fuzzy matching and COT conflicts analysis can be applied after each entity view or after each relationship view or as a post integration task. Further, both can be applied against the GCS, against the views or against both the GCS and the views simultaneously. It was concluded however, that in order to avoid confusion, the best benefit is achieved by applying both against the GCS only. Whether these are applied after each entity view or relationship view or they are applied as post integration, does not affect the designer or to the GCS, as ultimately the same effect is achieved.

The concept of entity view could be used as a view modelling approach, which could be called *entity view modelling*. Entity view modelling can be achieved by first identifying all the entities in the organization, then the designer sets out to identify the entity view of each of these entities separately. These entity views can then be integrated. Entity view modelling could have one advantage over ordinary view modelling in that the designer does not have to consider a particular user view, where the latter might have a number of entities and E-Rs.

Whilst the n-ary view integration approaches do not give the immediate impression that all the views are integrated simultaneously, the eventual effect is that of n-ary view integration. Due to the method developed for COT conflicts and all the reasons given for its use as a post integration task, n-ary view integration does not seem to benefit from the collective semantics of the views. This means that there is no need to refer to the views for certain semantics or statistics to help the designer resolve a conflict, since the same effect is achieved in post integration.

The approach followed to achieve n-ary view integration presented in this chapter showed that it does not have to be complex. This reduction in complexity was achieved by the entity view and the relationship view approaches.

CHAPTER 8

TESTING AND RESULTS

8.1 Introduction

> 'A design methodology can be thought of as a collective set of tools and techniques employed within an organizational framework that can be applied consistently to successive database structure development projects' (Teory & Fry 1982).

Teory & Fry (1982) suggest the following as indications of a good database design methodology:

1. Produce database structures within a reasonable amount of time and with a reasonable amount of effort.
2. General and flexible to be used by experts as well as the user.
3. It should be reproducible so that two persons (programs) applying the methodology to the same problem will produce the same or approximately the same results.

The View Integration Methodology (VIM) presented in this thesis is embodied mainly in BVI and the two types of NVI (ENVI and RNVI). VIM reads the views and produces the GCS faster than a human designer, and hence satisfying point 1. VIM takes approximately 8 minutes to integrate the sixteen views in appendix A. VIM is flexible in that it can be used by experts and experienced users, and therefore satisfying part of point 2. With the exception of some of the conflicts, VIM carries out the view integration process automatically. In case of conflicts, VIM provides the designer with the appropriate messages and allows him to interact and change the contents of the GCS. Further, VIM allows the designer to query the contents of the GCS and the views prior to resolving a conflict. VIM is general enough in that it can integrate views from any application area, though it is restricted to ERM. Regarding point 3, the same designer using VIM will result in producing the same GCS each time it is given the same set of views. Since VIM satisfies most of these suggestions, and it consists of a series of steps, it can be seen as a methodology as defined above.

The objectives of this research have been to study the view integration approach for the development of very large conceptual schemas. Among the other objectives was to identify all the conflicts which may occur during view integration and propose methods for their resolution. To test the feasibility of these objectives, prototypes of BVI and NVI were implemented. This chapter presents a demonstration and test runs integrating a

simple case study of sixteen views. Each type of conflict which took place during testing is reported, and its method of resolution as presented in chapters 3 and 4 is analysed.

8.2 Modelling the application views: case study

The method of measuring the size of an organization in terms of its data is not known. However, it is generally accepted that an organization with more than fifty entities results in a complex and large conceptual schema. Examples of very large organizations which normally consist of more than fifty entities are universities, large industrial firms, government departments and hospitals. Modelling organizations of this size can take between several months and several years, and this cannot be achieved during this research project.

It was decided to model the department of computer science at Aston University on the basis that it contained enough views to test VIM. However, it was seen that to model all the semantics of this Department would constitute building a very large conceptual schema. Therefore, only a sample of sixteen views were modelled (see appendices A and B).

The amount of semantics to be modelled depends on the user's requirements. Consider for example the number of attributes which can be modelled for the entity 'student'. These could include his last college, last schools, subjects studied, name of teachers, results obtained in each subject, interests, behaviour reports, and so on. It is clear that both the university and the department of computer science are not interested in all these attributes. The same applies for relationships between entities. It is possible to model many relationships between some entities, but only those relationships which are relevant to the organization should be modelled. Regarding the entities and relationships, most of the necessary ones were modelled. However, in the case of attributes, only the absolute minimum were modelled for their corresponding entities and relationships in each view. The reasons for this were:

1. Once all the views of the organization are modelled, then all the attributes would consequently have been modelled.
2. In each view only the relevant attributes to this view are modelled. The view integrator would accumulate the attributes in their corresponding entities.
3. In some cases the number of attributes for a particular entity is too large to model for the sake of testing only. An example of this kind of entity is 'application form'. It contains almost 100 attributes.

The view modelling tree presented in Fig. 5.1 was used to model the computer science department. The resultant view modelling tree is shown in Fig. 8.1. This is a one level

tree, and therefore no deep hierarchy was achieved. In these cases, it is possible that the division of the views is achieved in an arbitrary manner.

The only parent view is the 'Computer Science' view. If all the subviews of this view are modelled, then the parent view should not contain any E-Rs of its own. It is assumed here that all the views are modelled completely. Table 8.1 shows statistics of the occurrence of objects in the sixteen views.

Occasionally the designer models entities without attributes. Examples of these situations are shown in Table 8.2. This is either unintended or, if only one designer is involved, is done deliberately, since the designer knows that the same entity is modelled in another view. However, this is not advisable, since in certain cases, there is only one occurrence of the entity in all the views.

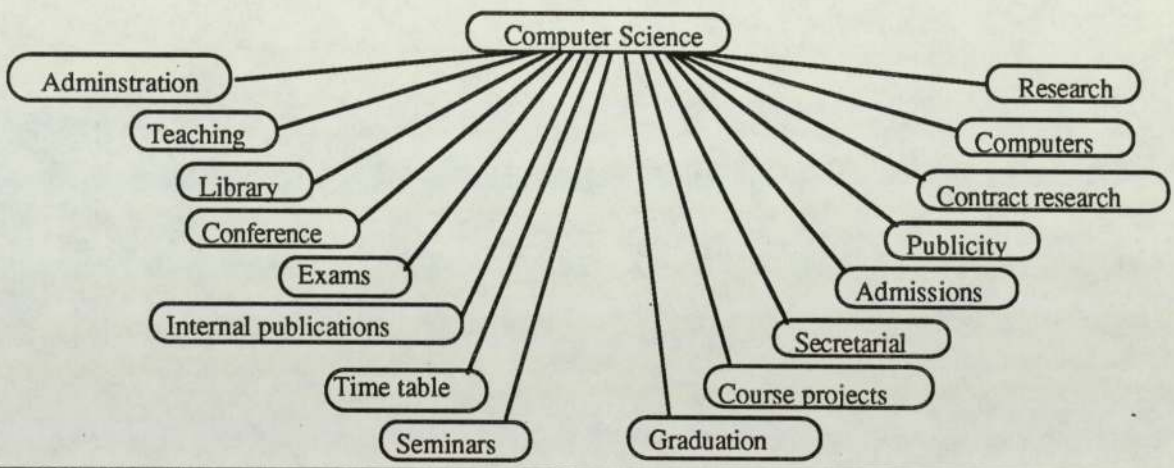


Fig. 8.1 A view modelling tree of the computer science department

8.3 Choice of views and E-Rs for integration

It was concluded in 6.4.2 that there is little difference between choosing the views for integration in a hierarchical way (bottom up or top down) or in a random way. The number and type of conflicts may vary slightly, but the resultant GCS should be the same. To make BVI choose the views in a hierarchical manner, a simple algorithm to traverse through the view modelling tree is needed. The choice of the next E-R was discussed in section 6.4.3, and it was also concluded that these are chosen randomly. The views and their E-Rs are integrated in the sequence they are input and represented internally (see section 6.2). BVI reports the views and their corresponding E-Rs being integrated to inform the designer of the level reached in the integration process. The full list of the sequence of integration of the views and their E-Rs is in appendix F. A sample of this list is shown below:

```
Integrating view: adminstration
     integrating entity-relationship:    department, heads, head
     integrating entity-relationship:    department,employs,
     secretaries
     integrating entity-relationship:    department, work in, staff
     integrating entity-relationship:    staff, live in, room

Integrating view: teaching
     integrating entity-relationship:    course, course, student
     integrating entity-relationship:    lecturer, teaches, subject
     integrating entity-relationship:    student, use, equipment
     integrating entity-relationship:    subject, part of, course
     integrating entity-relationship:    technician, repairs, equipment
     integrating entity-relationship:    lecturer, teaches, student
     integrating entity-relationship:    department, runs, course
```

Matching E-Rs results in one of five types of matches: identical, near identical, closely similar, similar or different (see section 6.4.5). After integrating the sixty six E-Rs of the sixteen views, the following results were obtained:

```
Identical=0, Near identical=0, Closely similar=1, Similar=0,
Different=65.
```

Although the views share some entities and relationships, they differ on all but one E-R. The E-R which was reported to be closely-similar is:

```
lecturer teaches subject
```

This E-R was modelled in view 'teaching' and view 'exams'. No justification can be given to the frequency of occurrence of the matches between E-Rs. If more than one designer are involved in the view modelling phase, then each might include the most possible E-Rs in each view to ensure completeness. In the case of these two closely similar E-R, it is more appropriately modelled in the 'teaching' view than in the 'exams view'. In any case apart from the extra view modelling effort by the designer, the occurrence of these matches is automatically handled by VI.

8.4 Conflicts encountered

8.4.1 Key status conflicts

The conflicts where VI needs the designer to intervene are:

1. Key attribute status conflicts.
2. Attribute value conflicts.
3. Cardinality conflicts.

The first key status conflict was reported by VI as shown below:

```
CONFLICT between view and schema entity attribute key statuses:

VIEW name is          :  admissions
ENTITY name is        :  [course]
ATTRIBUTE  with conflict is:  [department]
schema entity has key status   :     key
view entity has key status     :     notkey

Please do one of the following :
    1. Enter a new key status
    2. Query the views or schema
|: key.
```

	Number of					
Views	Relationships	Entities	E-R	Entity attributes	Relationship attributes	Valued attributes
Adminstration	4	5	5	13	0	0
Teaching	10	10	10	23	0	1
Library	2	3	2	10	0	0
Conference	3	4	5	11	1	0
Exams	3	4	4	7	0	0
Research	4	5	4	13	1	1
Computers	4	7	5	13	0	1
Contract research	6	4	6	12	1	3
Publicity	1	2	1	6	0	1
Admissions	5	5	5	14	4	2
Internal publications	1	3	2	8	0	2
Secretarial	1	2	1	4	1	1
Time table	7	7	8	12	0	2
Course projects	4	4	4	9	0	1
Seminars	1	2	1	11	0	2
Graduation	3	3	3	8	0	1
TOTALS	59	70	66	174	10	18

Table 8.1 Occurrence of objects in the views

In view 'teaching' the attribute 'department' is a key attribute. In view 'admissions', the attribute 'department' of entity 'course' modelled is a notkey attribute. Since view 'teaching' is integrated with the GCS before view 'admissions' (see appendix F), the attribute 'department' is included in the GCS as a key attribute, and hence the key status conflict. Almost always the designer would choose the key status over the notkey status. However, since this cannot be guaranteed, it was decided to request the designer's intervention. This response from the designer is stored by VI and used to automatically resolve all future conflicts of this type relating to this attribute.

view	enties
Teaching	Ph.D, teaching
Library	lecturer
Exams	external examiner
Computers	student, computer officer
Admissions	admissions tutor
Internal publications	research student
Course projects	lecturer

Table 8.2 Entities modelled without attributes

Occasionally the designer is not able to respond with a decision before querying the contents of the views or the GCS, and therefore, a facility was included in VI to allow for this. In the key status conflict above, the designer may choose option 2, and hence entering the query mode. In response to the command 'q' or 'query' the designer can interchange between the two menus shown below. He may continue to query the views and the GCS until he can make the appropriate decision. When he quits twice consecutively, VI returns to display the conflict menu. These menus can be extended to include more queries. A number of predicates are already included in the modules 'query_schema' and 'query_views' to produce the necessary queries. However, these have not yet been added to the menus. The presentation of the query menus in this section was regarded as appropriate since a real test run is being demonstrated. In response to the command 'q' or 'query' given by the designer from the conflict menu, the following query menu is displayed:

```
You have requested to query the semantics of either
the views or the schema
Please request one of the following:
 1   Query the views
 2   Query the schema
 n   To stop querying
|: 1.
```

Query Menu A

In this case the designer requested to look at the semantics of the views. Therefore, VI displays the following views query menu for which all the queries have been implemented:

```
1     List view names
2     Check if a certain view name exists
3     List all the relationship names in a certain view
4     List entities of a certain relationship in all views
5     List entities of certain relationship in certain view
6     Show cardinalities of entities in a relationship
7     List the attributes in a given entity
8     List entities with a given attribute
9     List all relationship names in all views
10    List key attributes of a given entity in a given view
n     To end the querying of the views
|: 1.
```

Query Menu B

At the end of each query the same menu is redisplayed for possible further queries. To end the queries provided by the current menu, the designer must type 'n', which ends the current menu and returns to Query Menu A above. If the designer requests option 2 from Query Menu A, then Query Menu C is displayed, which allows the designer to query the semantics of the schema as follows:

```
1     List all relationship names in the schema
2     List entities involved in a given relationship
3     List all E-Rs without their attributes
4     List all E-Rs in the schema
5     List all attributes in the schema
6     List attributes of a given entity
7     List key attributes of a given entity
n     To end the querying of the schema
|: 2.
```

Query Menu C

The menu is redisplayed after each schema query until n is entered. The designer can alternate between the three menus until all the necessary queries are made.

The only other attribute key status conflict which took place during integration is as reported by VI below:

```
   CONFLICT between view and schema entity attribute key status

VIEW name is          :     course projects
ENTITY name is        :     [lecturer]
ATTRIBUTE  with conflict is:  [name]
schema entity has key status :  key
view entity has key status :  notkey

Please do one of the following :
    1. Enter a new key status
    2. Query the views or schema
|: key.
```

8.4.2 Cardinality conflicts

VI reported the integration of the 'graduation' view as follows:

```
Integrating view:  graduation
     integrating entity-relationship:   student,attends,graduation
```

During the integration of this E-R, a cardinality conflict is reported by VI as shown below:

```
CONFLICT between view and schema relationship entity cardinalities

VIEW name is              :  graduation
RELATIONSHIP name is      :  [attends]
ENTITY name is            :  [student]
schema entity has card.        :  [many]
view entity has ent card.      :  [1]

Please do one of the following :
 1.    Enter a new entity cardinality
 2.    The "two" relationship names are homonyms
 3.    Query the views or schema
|: [many].
```

The entity 'student' is modelled in the view 'conference' as being involved the relationship 'attends' with a cardinality = many. The entity 'student' is also modelled in the view 'graduation' as being involved in the relationship 'attends', but this time with a cardinality = 1. Cardinality conflicts of this type were discussed in section 3.3.2.1, where example situations of this type of conflict were presented and resolutions were recommended. The resolution recommended for this situation in that section is that the two relationship names from the two views are homonyms. If, as recommended in section 3.3.2.1, the two relationship names are homonyms, they can consequently be changed as in Fig. 8.2. However, if this method of naming relationships is followed in view modelling, then most relationship names would not be as they are currently modelled in appendix A. Since this is the only situation of this conflict encountered in the integration of the views of appendix A, it is not possible to make a firm decision about the resolution. Experience is needed through using VI to integrate views from a much larger application. The 'two' relationship names 'attends' represent the exact semantic connection in both views. If this is acceptable, then the relationship 'attends' is kept in the GCS as a 'bad' binary relationship (see section 3.3.2.2), and one cardinality out of the two cardinalities for entity 'student' is chosen. As in this case, the higher cardinality is almost always chosen.

8.4.3 Attribute values conflicts

Four conflicts may occur when matching the values of two attributes (section 3.4.2.3). Two of these conflicts are resolved automatically, whilst the other two need the designer's intervention. Table 8.1 shows that the total number of attributes in the sixteen views is one hundred and seventy four, and only eighteen of these are valued attributes. However, none of the 2 conflicts which need the designer intervention occurred during the integration of these views.

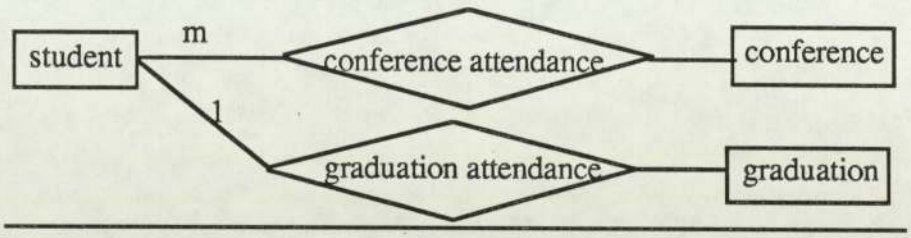


Fig. 8.2 Example

8.4.4 Conclusions

After all of the sixteen views are integrated, the GCS should now contain all the semantics of these views. The GCS at this stage (before COT conflicts analysis and object fuzzy matching of entities) is shown in appendix C (VDL form) and appendix D (ERM form). Table 8.3 shows the statistics of the number of different types of objects in the GCS.

If a particular E-R is modelled in different views and its integration results in a 'bad' binary relationship (see section 3.3.2.2), and the relationship names are considered to be homonyms, and one of these relationships has attributes, then the resultant relationship attributes causes concern. Consider the example E-Rs:

1 'student' 'registered on' 'computer' in view 'computers'.

2 'accepted candidate' 'registered on' 'course' in view 'admissions'.

The relationship in case 2 above has the attribute 'date of enrollment'. After integration, the resultant relationship 'registered on' has this attribute. The question is to which E-R does this attribute belong? This problem and the cardinality conflicts in section 8.4.2 are two reasons why 'bad' E-Rs should not be allowed. The integration of the sixteen views which resulted in 'bad' binary E-Rs were accepted as such in order to allow more representative names for relationships (see Fig. 3.11). However, since this approach causes some possible structurally unacceptable situations, it is appreciated that VI should be modified to reject 'bad' binary E-Rs.

Object	Total
Relationships	52
Entities	49
Entity-relationship	63
Entity attributes	155
Relationship attributes	9
Valued attributes	15

Table 8.3 Statistics of GCS objects

The GCS may contain homonyms, synonyms and COT conflicts. The solution of COT conflicts is processed as a post-integration task. Identifying synonyms is part of the object fuzzy object matching process, and this follows the COT conflict analysis phase (see Fig. 6.9). Identifying homonyms is not part of VI, and we foresee no future method to identify them automatically.

8.5 COT conflicts

Fourteen COT conflicts were identified and analysed in section 4.3. These constitute all the possible conflicts of this type, which may take place when integrating views modelled in ERM. The identification, analysis and resolutions for seven of these conflicts have been implemented in VI. These conflicts are:

1 Attribute value as E-R /same E-R
2 Attribute as relationship / same E-R
3 Attribute as relationship / different E-Rs
4 Attribute as entity / own entity
5 Attribute as entity / foreign entity
6 Entity as relationship / own relationship
7 Entity as relationship / foreign relationship

Brief outlines of the algorithms for these seven COT conflicts were shown in section 6.4.6. The GCS after the view integration phase is shown in appendices C (ERM pictorial form) and appendix D (VDL form). This GCS contains many COT conflicts. Some of these conflicts are resolved automatically and the resolution for the others is requested by VI from the designer. However, even those resolved automatically are reported to the designer by VI for bookkeeping purposes. The full run of the COT conflicts and their resolutions is shown in appendix F. A sample of these COT conflicts is shown below:

```
                ATTRIBUTE AS RELATIONSHIP COT CONFLICT

The ATTRIBUTE  activity  which belonged to ENTITY  department
existed as the RELATIONSHIP  activity
and  the RELATIONSHIP activity  involves the ENTITY
department

SOLUTION:    the ATTRIBUTE activity was removed from ENTITY
                   department by the view integrator.
```

The removal of the attribute 'activity' from the GCS is correct and agrees with the analysis presented for the attribute as relationship/ same E-R (see section 4.3.4). If the entity 'department' was not involved with the entity 'teaching', then this attribute should stay and its domain may eventually contain values such as 'teaching', 'research', and 'services'.

The next conflict to be reported by VI is the attribute as relationship / same E-R COT conflict, as shown below:

```
                ATTRIBUTE AS RELATIONSHIP COT CONFLICT

The ATTRIBUTE  course  of ENTITY  students
exists as a relationship of the same name
AND the RELATIONSHIP course does NOT involve the ENTITY
students

The solution to such a conflict cannot be given by VI.
Therefore, Please choose one of the following solutions:
 1.    Remove attribute from ENTITY students
 2.    Change the ATTRIBUTE name
 3.    Change the RELATIONSHIP name
 4.    Leave attribute and relationship names as they are

ENTER your order by typing the appropriate number
|: 2.
```

The designer chose option 2. Therefore, VI presents him with the message:

```
Enter the new attribute name:  'course name'.
```

The designer decided to change the name of the attribute from 'course' to 'course name'. This attribute name is consequently changed by VI for entity 'students' only.

The next conflict encountered is the *attribute as relationship/ different E-Rs*, as follows:

```
ATTRIBUTE AS RELATIONSHIP COT CONFLICT

The ATTRIBUTE location   of ENTITY conference
exists as a RELATIONSHIP of the same name
AND the RELATIONSHIP location  does NOT involve the ENTITY
conference

The solution to such a conflict cannot be given by VI
Therefore, Please choose one of the following :

 1.  Remove ATTRIBUTE  location from ENTITY  conference
 2.  Change attribute name
 3.  Change relationship name
 4.  Leave attribute and relationship names as they are

ENTER your order by typing the appropriate number
|: 4.
```

The relationship 'location' in the GCS involves the entities 'computer' and 'room'. The existence of the attribute 'location' in entity 'conference' has no effect on the validity of the GCS, and therefore, the designer responded by choosing option 4. However, VI allows the designer three other options, as shown above. These constitute all the possible resolutions to this type of conflict.

Seven more attribute as relationship COT conflicts existed in the GCS as reported by VI. These are shown in appendix F.

The next COT conflict to take place is the attribute-value as E-R as reported below:

```
            ATTRIBUTE-VALUE AS E-R CROSS OBJECT CONFLICT

The ATTRIBUTE part of   of ENTITY subject has a VALUE course
and exists as an E-R

The VALUE course  was removed from ATTRIBUTE part of.

Because the ATTRIBUTE part of  has no other values,
it was removed from ENTITY subject.
```

The next COT conflict to take place is the entity as attribute as reported below:

```
                    ENTITY ATTRIBUTE COT DEFINITION

The ATTRIBUTE department  of ENTITY  course
exists as the  ENTITY department
BUT:
       The ENTITY course  of ATTRIBUTE department
       is connected to ENTITY  department
       by relationship(s):

CHOOSE any of the following:
    1. DELETE the ATTRIBUTE department
    2. CHANGE the name of the ATTRIBUTE  department
    3. CHANGE the name of the ENTITY  department
    4. CHANGE the name of the ENTITY  course
    5. Accept the situation
    6. CREATE a relationship between ENTITIES course and
           department
|: 5.
```

As can be seen from the GCS the entity 'course' is related to the entity 'department' by the relationship 'runs'. The designer chose to accept the situation, because the existence of this conflict does not affect the validity of the GCS. Further, the attribute 'department' is regarded as a necessary part of the entity 'course'. The two other most appropriate resolutions to be chosen are represented by options 1 and 2.

Four other entity as attribute COT conflicts were reported by VI (see appendix F). Five of the six conflicts of this type are of the entity as attribute/ foreign entity type (see section 4.3.6). The only conclusion which can be drawn from this is that all the necessary E-Rs were modelled. This could either be because of the familiarity of the application area or because of the relatively small size of the organization modelled.

The next COT conflict to take place is the entity as relationship/ own relationship, as reported below:

```
              ENTITY RELATIONSHIP CROSS TYPE DEFINITION

The ENTITY course  exists as RELATIONSHIP  course
AND
the ENTITY course  is involved in RELATIONSHIP  course

This could mean any of the following:
    1. The ENTITY is wrongly named
    2. The RELATIONSHIP is wrongly named

CHOOSE any of the following:
    1. CHANGE the name of the ENTITY course
    2. CHANGE the name of the RELATIONSHIP course
|: 2.
```

The choice of any of the options would depend on what the designer thinks of the names of the two objects concerned. However, this conflict must be resolved by choosing one of the two options above. Assuming a relational DBMS as the target database, both the relationship and the entity would be represented as individual relations of the same name. This is not acceptable.

In a very large GCS COT conflicts would be very difficult to find manually, and therefore eventually all these must be implemented. Samples of the non implemented conflicts are shown below and a discussion of their resolutions is presented. The resolutions of the manually identified COT conflicts have not been applied to the internal representation of the GCS. Therefore, their resolutions are not included in the object fuzzy matching phase.

The following are the COT conflicts which have been identified manually:

Value as an entity:

It was suggested in sections 4.3.10 and 4.3.11 that this conflict must be identified and resolved before the valued attribute as relationship conflict (see sections 4.3.5 and 4.3.7). As can be seen from the GCS, the entity 'researcher' has the attribute 'type' with the values 'student' and 'staff'. However, the attribute 'type' also exists as the relationship 'type'. But, the relationship 'type' does not relate the entity 'researcher' to either of the entities 'student' or 'staff'. Therefore, applying the valued attribute as a relationship would not achieve the required resolution.

This particular value as an entity/ foreign entity conflict is described in section 4.3.11. The values 'student and 'staff' both exist as entities. However, the entity 'researcher' is not related to either of the entities 'student' or staff. This is similar to the example situation shown in Fig. 4.41. The resolution to this conflict could also be similar to that shown in Fig. 4.42. Therefore, a relationship called 'researcher type' could be created between the entity 'researcher' and the entities 'student' and 'staff'. The cardinalities have to be supplied by the designer.

Another conflict of this kind is the attribute 'type' of the entity 'machines'. All the values of this attribute exist as entities. The resolution to this conflict is achieved in a similar way to the previous conflict. In this case, the suggestion that the entity 'machines' and the entities 'micro', 'mini' and 'main frame' are synonymous appears to be more valid. Further, in this particular situation, the valued attribute as relationship conflict resolution could also be applied. However, the value as entity conflict is processed first and this would eliminate the valued attribute as relationship conflict. There is a certain amount of overlap between these two conflicts and therefore it is possible that in the future they could be analysed as the same type of COT conflict with a number of variations. Further,

the advancement of ERM to include abstraction capabilities, especially those of generalisation and categorization (see Weeldryer 1980, Elmasri *et al* 1985, Furtado 1986 and Elmasri & Navathe 1989) would reduce the existence of this type of conflict.

Value as attribute

Although it was suggested in section 4.3.14 that such a conflict does not normally occur, one occurrence of this conflict exists in the GCS. Further, it was suggested in section 4.3.14 that this conflict can take one of three formats. The entity 'staff' has the valued attribute 'type', and 'type' has the value 'lecturer'. The entity 'subject' has the attribute 'lecturer'. The occurrence of this conflict depends on the resolutions taken to resolve some other types of conflicts, namely, the entity as attribute and value as entity COT conflicts. The entity as attribute conflict was identified by VI for the attribute 'lecturer' of entity 'subject' and entity 'lecturer' (see appendix F). One of the options presented to the designer to resolve this conflict was to change the attribute name, possibly to 'lecturer name'. If this resolution is decided, this value as attribute conflict would not take place. Further, the attribute 'type' of entity 'staff' had 'lecturer' as one of its values (see above). This caused a value as entity conflict. If the resolution adopted to resolve this conflict had been to create E-Rs (see section 4.3.11) then the conflict value as attribute reported here would not have taken place.

The last COT conflict described in section 4.3 is the value as relationship COT conflict. The GCS is free from this conflict.

The COT conflicts analysis phase discovered a number of these conflicts in the GCS. With the exception of two of these conflicts, all the other conflicts had to be resolved by the designer. Regarding the seven conflicts which have been implemented, VI presents their possible resolutions to the designer who has to choose from the list of resolutions given. In all the conflicts of this kind that were encountered in the GCS, the menus of resolutions were acceptable to the designer. Regarding conflicts that were not implemented, the analysis and suggestions for their resolutions as presented in section 4.3 were also generally acceptable to the designer.

8.6 Object fuzzy matching of entities

The GCS contains both synonyms and homonyms. The homonyms in the GCS cannot be identified by the VI, but in any case, they have been identified manually and shown in Table 8.5. This task is tedious if the GCS is much larger.

Regarding the relationships, if the policy of handling 'bad' binary E-Rs as a result of integrating two E-Rs with a common relationship name is changed, then these homonyms

will be identified by the VI during integration. The homonym naming conflicts and the cardinality conflicts discussed in sections 3.3.2.1, 3.3.2.2 and 8.4.2, are good reasons for not allowing 'bad' binary E-Rs.

Cross Object Type Conflict	Frequence
Attribute-value as an entity-relationship	
a) Same entity-relationship	1
b) Different entity-relationships	0
Attribute as a relationship	
a) Same entity-relationship	1
b) Different entity-relationships	8
Valued attribute as a relationship	
a) Same entity-relationship	0
b) Different entity-relationships	5
Entity as an attribute	
a) Own entity	0
b) Foreign entity	
I) Entities have a common relationship	5
II) Entities have no common relationship	1
Value as an entity	
a) Own entity	0
b) Foreign entity	
I) Entities have a common relationship	0
II) Entities have no common relationship	6
Entity as a relationship	
a) Own entity-relationship	1
b) Foreign entity-relationship	
I) Common entity	0
II) Totally different entity-relationships	0
Value as an attribute	2
Value as a relationship	0

Table 8.4 Frequency of COT conflicts

Object type	Object name	First view	Second view
Relationship	gives	time table	contract research
Relationship	attends	conference	graduation
Relationship	teaches	teaching	teaching
Relationship	part of	teaching	time table
Relationship	registered on	computers	admissions
Entity	project	contract research	course projects

Table 8.5 Homonyms in the GCS

Synonyms are more common in the GCS. The synonyms that were identified manually are:

A **Entities:**

1 staff, lecturer, technician, computer officer, admissions tutor, secretary, secretaries.

2 accepted candidate, students, student and research student.

3 subject and subjects.

4 equipment, computer, machines.

5 Ph.D and research student.

6 Ph.D and thesis.

7 external examiner and examiner.

8 room and class room.

B **Relationships.**

1 live in and location.

2 employs and works in.

3 course and registered on.

4 viva, examination and marks.

5 part of and consists of.

6 owns and owner.

7 teaches and taught by.

8 use, uses and registered on.

C **Attributes:**

1 type and level of entity degree.

2 course name, course and name of entities course, subject and student.

3 staff No., staff number and number of entities head, secretaries, computer officer and staff.

4 dept., Dept. and department of entities secretaries, student, lecturer, technician and supervisor.

5 course tutor and tutor of entities course and students.

6 student name and name of entities thesis, report and student.

7 model number and model of entities micro, mini and main frame.

8 building and building name of entities room and class room.

A method of identifying synonyms was introduced in section 4.2. Object fuzzy matching of attributes, relationship and E-Rs were discussed respectively in sections 4.22 and 4.23 and it was argued that object fuzzy matching of these objects would normally produce inapplicable results. Two methods (the weigh and add method and the weigh and multiply method) for the possible identification of synonymous entities automatically were described in section 4.2.1. These two methods are used to calculate the SLFs between all

entities in the GCS. It is not possible to predetermine the entities for which the SLF should be calculated.

The SLFs for entities are calculated based on their attributes, their attribute key statuses, and the relationship which relate these entities. The attribute values were excluded from the SLF calculations because their numbers are very limited in the GCS (see table 8.3). Further, only the immediate neighbours were used in the calculation of the SLFs. The involvement of the non immediate neighbours in the calculation of the SLFs would require complex weighting systems, and in any case, their contribution in the final value of the SLFs would be negligible.

The weigh and add method calculates the SLF as follows:

$$P = W1 \times Pa + W2 \times Pk + W3 \times Pr$$

where Pa, Pk and Pr are respectively the SLF values for the attributes, key statuses and relationships of the entity, and W1, W2 & W3 are their corresponding weights.

The modules '`objects fuzzy matcher`' calculated the SLFs as shown in appendix G. The three weights W1, W2 and W3 were varied six times in order to evaluate the effect each of the neighbours has on the total SLFs, and to find the best weights to be used in order to produce the most representative SLF values. The weights chosen are shown in Table 8.6. The weights chosen for SLF1, SLF2 and SLF3 are effectively ways of testing the level of similarity based respectively on the attributes, key statuses and relationships only. The weights chosen for SLF4, SLF5 and SLF6 test the effect on the total SLF values by associating varying weights to the attributes, key statuses and relationships. The weights chosen for the calculation of SLF4 give equal importance to attributes and relationships, and disregards the importance played by the key statuses. The weights chosen for SLF5 put most importance on the attributes and least importance on the key statuses. The weights chosen for the calculation of SLF6 give more importance to relationships, and least importance to key statuses.

Since there is no exact theoretical approach to associating these weights, they were chosen based on our understanding of ERM and the experience gained in modelling and integrating the views shown in appendix A. In any case, the weights chosen give the widest possible range of significant weight assignments. View modelling is influenced by the number of designers involved and their level of familiarity with the application area. These factors influence the completeness of modelling the entity attributes in the individual views. The entity 'subjects' was modelled with one attribute on the basis that the same entity is modelled in another view with more attributes, which would ultimately be integrated by the VI. However, the existence of the synonym naming conflict between

the entities 'subject' and 'subjects' causes serious distortion in the SLF values. Further, since the two entities are modelled in different views, the entity view (see section 7.2) of entity 'subjects' is different to that of entity 'subject'. This causes a *chain effect*. The chain effect takes place between two entities when two or more of the following is satisfied:

1 They suffer from a synonym naming conflict.
2 They are modelled in different views.
3 They have different entity views in each of the views they are modelled.
4 They are modelled with incomplete attributes.
5 Some or all of their attributes suffer from synonym naming conflicts.
6 They have different key attributes (this could be due to point 5).

The entities 'subject' and 'subjects' suffer from the effect of 1, 2, 3, 4, 5 and 6. This is obviously an extreme situation. However, any of the above six factors contributes in distorting the SLF values. The entities identified manually as synonyms in section A above all suffer from one or more of the above six factors. No method of weight distribution could eliminate the chain effect.

	Weights chosen					
	SLF1	SLF2	SLF3	SLF4	SLF5	SLF6
Pa	1	0	0	0.5	0.7	0.2
Pk	0	1	0	0	0.1	0.1
Pr	0	0	1	0.5	0.2	0.7

Table 8.6 Weights chosen for the calculation of SLFs

The results obtained by the six variations of weights for the weigh and add method are shown in appendix G. A section of these results are shown below:

		Wpa=1 Wpk=0 Wpr=0	Wpa=0 Wpk=1 Wpr=0	Wpa=0 Wpk=0 Wpr=1	Wpa=.5 Wpk=0 Wpr=.5	Wpa=.7 Wpk=.1 Wpr=.2	Wpa=.2 Wpk=.1 Wpr=.7
Entity 1	Entity 2	SLF1	SLF2	SLF3	SLF4	SLF5	SLF6
department	head	H:0.25	M:0.33	M:0.14	H:0.19	H:0.24	H:0.18
department	secretaries	M:0.2	M:0.25	M:0.14	H:0.17	M:0.19	H:0.16
department	staff	H:0.33	H:0.669	M:0.079	H:0.2	H:0.31	H:0.19
department	room	L:0.0	L:0.0	L:0.0	L:0.0	L:0.0	L:0.0
department	course	M:0.14	M:0.2	M:0.089	M:0.11	M:0.14	M:0.11
department	student	M:0.14	L:0.0	L:0.0	M:0.069	M:0.1	L:0.03
department	lecturer	M:0.17	M:0.17	L:0.0	M:0.079	M:0.14	M:0.05
department	subject	M:0.2	M:0.2	L:0.0	M:0.1	M:0.16	M:0.059
department	equipment	M:0.2	M:0.25	M:0.11	H:0.15	M:0.19	H:0.14
department	technician	M:0.2	M:0.25	L:0.0	M:0.1	M:0.16	M:0.059

Wpa, Wpk and Wpr are the weights associated to the attributes, key attributes and relationships respectively. Although the values obtained are in the range 0 to 1, the number of possibilities are infinite. Therefore, it is not possible to give precise SLF values which indicate a similarity between the entities concerned. A number of statistical techniques such as means, distributions and averages could be used. The method of evaluating averages was applied here. Each of the six SLF values obtained for any two entities is classified as a Low (L), Medium (M) or High (H). These categories are shown in the sample above and in appendix G as L, M or H. For example the SLF value H:0.24 indicates a high similarity SLF value equal to 0.24. To achieve these three categories, the following averages were calculated:

Global average = Total of SLFs / Number of SLFs

Higher average =

(Total of SLFs > global average) / Number of SLFs above global average

Therefore, the low, medium and high categories are classified as follows:

Low: Any SLF value < global average.

Medium: Any SLF value ≥ global average and < higher average.

High: Any SLF value ≥ higher average.

After matching all the entities in the GCS shown in appendices E, the following global and higher averages in Table 8.7 were obtained.

	SLF1	SLF2	SLF3	SLF4	SLF5	SLF6
Global average	0.077	0.133	0.022	0.049	0.072	0.044
Higher average	0.213	0.457	0.273	0.125	0.191	0.123

Table 8.7 Averages of *weigh and add* fuzzy matching method

In theory, based on the averages obtained and the categories devised, the following should hold:

1 Low SLF values indicate a low probability of the entities being synonyms.
2 Medium SLF values indicate a medium probability of the entities being synonyms.
3 High SLF values indicate a high probability of the entities being synonyms.

However, as will be discussed in section 8.6.1, this is not always true.

The second method for calculating the SLF values is the weigh and multiply method shown below:

$$P = (Pa)^{Wpa} \times (Pk)^{Wpk} \times (Pr)^{Wpr}$$

The weigh and multiply method works in a way that small sub SLF values weigh down the total SLF. The results obtained for the weigh and multiply method are shown in appendix H. A sample of these is shown below:

		Wpa=1 Wpk=0 Wpr=0	Wpa=0 Wpk=1 Wpr=0	Wpa=0 Wpk=0 Wpr=1	Wpa=.5 Wpk=0 Wpr=.5	Wpa=.7 Wpk=.1 Wpr=.2	Wpa=.2 Wpk=.1 Wpr=.7
Entity 1	Entity 2	SLF1	SLF2	SLF3	SLF4	SLF5	SLF6
department	head	H:0.25	M:0.33	M:0.14	M:0.19	M:0.23	M:0.17
department	secretaries	M:0.2	M:0.25	M:0.14	M:0.17	M:0.19	M:0.16
department	staff	H:0.33	H:0.669	M:0.079	M:0.16	H:0.27	M:0.13
department	room	L:0.0	L:0.0	L:0.0	L:0.0	L:0.0	L:0.0
department	course	M:0.14	M:0.2	M:0.089	M:0.11	M:0.13	M:0.11
department	student	M:0.14	L:0.0	L:0.0	L:0.0	L:0.0	L:0.0
department	lecturer	M:0.17	M:0.17	L:0.0	L:0.0	L:0.0	L:0.0
department	subject	M:0.2	M:0.2	L:0.0	L:0.0	L:0.0	L:0.0
department	equipment	M:0.2	M:0.25	M:0.11	M:0.15	M:0.18	M:0.13
department	technician	M:0.2	M:0.25	L:0.0	L:0.0	L:0.0	L:0.0

The low, medium and high categories described earlier for the weigh and add method were calculated in the same way for the weigh and multiply method. The averages obtained for the weigh and multiply method are shown in table 8.8.

	SLF1	SLF2	SLF3	SLF4	SLF5	SLF6
Global average	0.077	0.133	0.022	0.0087	0.0067	0.007
Higher average	0.213	0.457	0.273	0.204	0.256	0.274

Table 8.8 Averages for weight and multiply object fuzzy matching method

8.6.1 Analysis of results and comparison of the two methods

The results obtained by applying the two methods to the forty nine GCS entities are presented in appendices G and H. The number of unique entities identified manually as synonyms is twenty five (see A above). This means that 51% of the entities in the GCS suffer from possible synonym naming conflicts. Without the chain effect described above, only the SLF values of the category high should be considered seriously. However, because of the chain effect, some low and medium SLF values must also be considered. The ultimate aim of the analysis of the SLF values obtained is to automate the process of identifying the synonyms, and thus remove a major problem in view integration.

Columns SLF1, SLF2 and SLF3 of appendices G and H are the results obtained by respectively using only the attributes, key attributes and the relationships. The results

obtained by using the the weights SLF1, SLF2 and SLF3 of table 8.6 are the same for the two methods. In the case of the weigh and multiply method, the results for SLF1, SLF2 and SLF3 were achieved by modifying the weigh and multiply equation in such a way that it reflects the use of one neighbour. Using it otherwise would produce zero results in these columns.

Using only the attributes to obtain SLFs

There are eight sets of synonymous entities identified in A above. Let us consider the SLFs obtained for some of these sets using the attributes only.

secretaries	secretary	H:0.25
staff	secretaries	M:0.14
staff	lecturer	H:0.29
staff	technician	M:0.14
staff	computer officer	M:0.14
staff	secretary	M:0.17
lecturer	secretaries	M:0.17
lecturer	technician	M:0.17
lecturer	secretary	M:0.2
student	students	M:0.13
student	accepted candidate	L:0.0
student	research student	M:0.2
subject	subjects	L:0.0

Regarding all the synonymous entities relating to the different types of staff in the department, the only high category obtained is for 'staff' and 'lecturer'. Medium categories were obtained for all the others. Object fuzzy matching of 'secretary' and 'secretaries' resulted in a high SLF. Object fuzzy matching of 'subject' and 'subjects' however resulted in a low category. By studying these results in particular and the results of all the synonymous entities listed in A above in general, it can be concluded that using only attributes as a basis for this technique does not always produce reliable and representative results. Column SLF1 has too many cases of high categories for the designer to consider. Because of these many extra cases of high categories, it is not reliable to make the VI take the decision about synonyms. The problem is made worse by the fact that there are many situations in the SLF1 column where the categories are incorrect. These incorrect classifications of categories are caused by the chain effect.

Using only the key attributes to obtain SLFs

Using only the key attributes results in column SLF2 of appendix G. Because it is possible that in the same application many different entities share key attributes, these SLFs do not give a good representation of the level of similarity between entities. This explains the hundreds of high and medium categories in SLF2. Therefore, key attributes

should not be used by themselves to obtain the total SLFs nor should they play major roles in obtaining the total SLFs.

Using only the relationships to obtain SLFs

Using only the relationships results in column SLF3 of appendix G. It is noticeable that most of these SLFs are of the 'ow category, and that the SLFs of the high category are very limited. Further, all the SLFs in this column which are > 0 indicate that the two entities are related by one or more relationships. The entities which are not related result in SLFs equal to 0. The conclusions which can be drawn from this are:

1 Using only the relationships does not produce representative SLFs of the real level of similarity between the entities.

2 The relationships should not play major role in the calculation of the total SLFs.

Using all the neighbours and their characteristics

The results obtained by using all the neighbours and their characteristics in achieving the total SLFs are in columns SLF4, SLF5 and SLF6 of appendices G and H. These three columns of SLF values are based on the weights chosen in columns SLF4, SLF5 and SLF6 of table 8.6. Based on the conclusions made earlier regarding the use of each type of neighbour separately, the most representative weighting system is where the attributes are given the highest weight and the key statuses are give the least weight.

The results obtained vary a great deal between the two methods. The weigh and add method (appendix G) resulted in a great number of high and medium categories. The weigh and multiply method (appendix H), however, resulted mostly in low categories. Further, all the low categories have SLF values equal to 0. Each of the two methods has a number of drawbacks. The weigh and add method has the following drawbacks:

1 There are many high categories.

2 There are many categories in the wrong place. These are caused by the number of neighbours and the chain effect.

In the case of the weigh and multiply method its only drawback is that when any sub SLF is equal to 0 the total SLF becomes equal to 0. This is not always desirable.

Let us now consider the results obtained using the two methods for some of the synonymous entities of A above, and compare these results. The abbreviations W&A is used to indicate the weigh and add method, and the abbreviation W&M is used to indicate the weigh and multiply method.

head (W&A)	staff	M:0.17	H:0.5	L:0.0	M:0.08	M:0.17	M:0.079
head (W&M)	staff	M:0.17	H:0.5	L:0.0	L:0.0	L:0.0	L:0.0

Both entities have a medium match on attributes, a high match on key statuses and low match on relationships. The weigh and add method achieved medium matches for the three weighting systems, whilst the weigh and multiply method achieved 0 for the same weighting systems. Therefore, the weigh and add methods results are more representative in this case.

secretaries (W&A)	secretary	H:0.25	H:0.5	L:0.0	H:0.13	H:0.22	M:0.1
secretaries (W&M)	secretary	H:0.25	H:0.5	L:0.0	L:0.0	L:0.0	L:0.0

The entities 'secretaries' and 'secretary' are synonymous, and therefore the results obtained using the weigh and add method are more representative.

staff (W&A)	lecturer	H:0.29	M:0.2	L:0.0	H:0.14	H:0.22	M:0.079
staff (W&M)	lecturer	H:0.29	M:0.2	L:0.0	L:0.0	L:0.0	L:0.0

The entities 'staff' and 'lecturer' are synonymous, and therefore the results obtained by using the weigh and add method are more representative.

student (W&A)	students	M:0.13	L:0.0	L:0.0	M:0.059	M:0.089	L:0.03
student (W&M)	students	M:0.13	L:0.0	L:0.0	L:0.0	L:0.0	L:0.0

The entities 'student' and 'students' are synonymous and therefore the results obtained using the weigh and add method are better.

In almost all the results of columns SLF4, SLF5 and SLF6 the results obtained by the weigh and add method are more representative. The main advantage of the weigh and multiply method is that it produces a limited number of high and medium categories, whilst all the low categories are straight zeros. Therefore, should these high and medium categories be representative, there are fewer situations to be considered manually. However, this method has the drawback mentioned above, and as seen from the examples above, its results are mostly incorrect. One way to avoid the effect of the sub SLFs which are equal to 0 making the total SLF equal to 0, is to change these sub SLFs from 0 to 1. This approach would have the same effect as that of assigning a zero weight to any of the neighbours in the weigh and add method.

The six different weighting systems used produced more representative results in SLF1 and SLF5.

8.7 N-ary and binary view integration

8.7.1 Entity n-ary view integration

The first entity chosen by ENVI for entity n-ary view integration is 'department' from view 'adminstration'. The choice of this entity is made randomly from the entity pool (see Fig. 7.6). The entity 'department' is related by seven relationships from the sixteen views (see appendix A). These relationships are 'owns', 'employs', 'heads', 'work in', 'runs',

'activity' and 'owner'. The final 'department' entity view is created by integrating the 'department' entity view from all the views in which it is modelled. Fig. 8.3 shows the final status of the 'department' entity view. Once the integration of the 'department' entity view is completed, the entity 'department' is then considered complete. The only possible changes to this entity may come from the COT conflicts analysis phase or from the object fuzzy matching phase. The eventual effect of the entity n-ary view integration on the entity 'department' is n-ary (see Fig. 7.2). The seven other entities of the current entity view could be changed as more of their occurrences are integrated.

As explained in section 7.2, one of the advantages of entity n-ary view integration is the possibility of manually identifying synonymous entities and relationships. In the 'department' entity view, possible synonymous relationships are 'work in' and 'employs', and possible synonymous entities are 'staff' and 'secretaries'. Due to the small size of the schema consisting of one entity view, the designer may find synonyms easier to detect. Entities detected to be synonymous can be integrated by VI as shown in section 3.4.

8.7.2 Relationship n-ary view integration

The first relationship chosen by RNVI is 'employs'. The choice of relationships for integration is made randomly from the relationship pool (see Fig. 7.10). The integration of this relationship view results in the E-R shown in Fig. 8.4.

Relationship n-ary view integration has the effect of randomly integrating E-Rs with the same relationship name. At the end of integrating a relationship view, the relationship would have all its attributes (see Fig. 7.9).

The same relationship can be modelled in such a way that it may relate more than two entities in the same view or it may relate different entities in different views. An example of this kind of relationship is 'type'. The relationship 'type' is modelled in a number of views and its integration results in a non binary E-R as shown in the GCS (see appendix D). As discussed in section 3.3.2.2, the integration of two or more binary E-Rs should result in a binary E-R, and any other result is regarded as a 'bad' binary E-R. The E-Rs related by the relationship 'type' are 'bad' E-Rs. These E-Rs are often caused by the homonym naming conflict, though occasionally the homonym naming conflict appears necessary (see Fig. 3.11).

The other examples of 'bad' binary E-Rs in the GCS are related by the relationships 'gives', 'produces' and 'teaches'. Extending ERM to include the data abstraction techniques of generalization and specialization (Smith & Smith 1977) would reduce the

existence of the 'bad' binary E-Rs in the GCS. In the case of the relationship 'type' for example, the entities 'micro, 'mini, and 'main frame' can all be modelled as subtypes of the entity 'computer'. However, using the basic ERM, these three entities can either be regarded as synonymous to the entity 'computer', or they can be made into an attribute-value structure of the entity 'computer'.

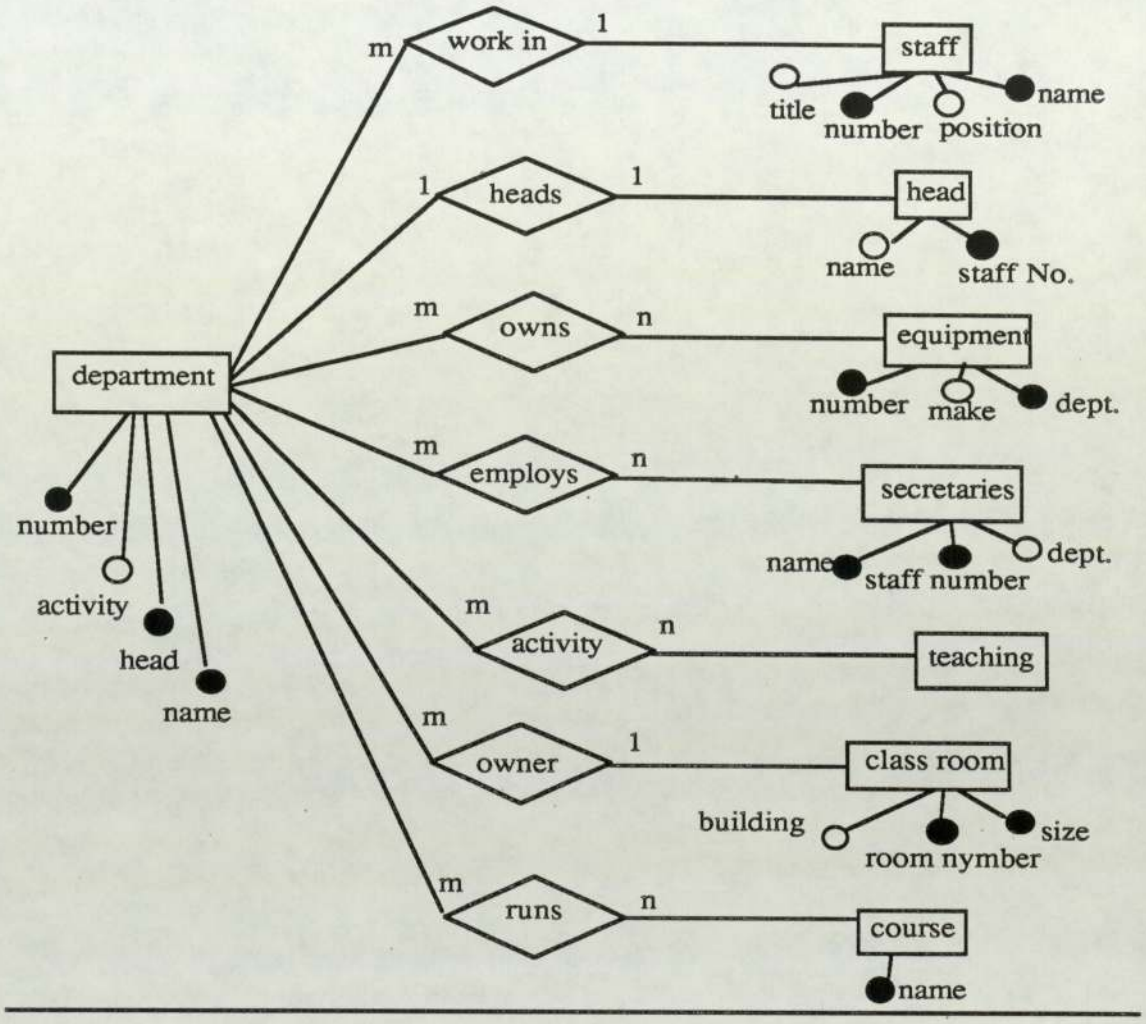


Fig. 8.3 'department' entity view from the GCS

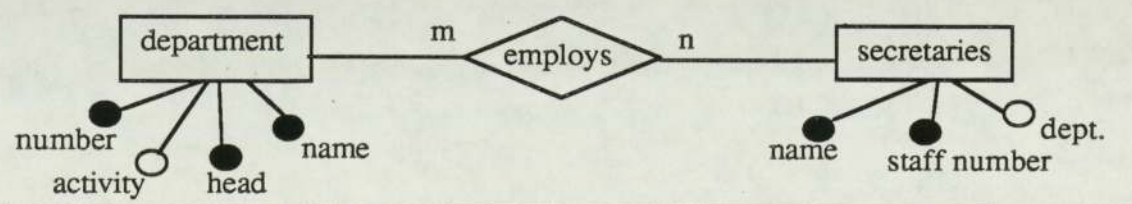


Fig. 8.4 'employs' relationship view from the GCS

8.7.3 Comparing view integration approaches

Both BVI and NVI are divided into pre-integration, view integration and post-integration phases. The post-integration phase consists of the COT conflicts analysis phase and the object fuzzy matching phase. The removal of the COT conflicts analysis phase from the main body of view integration phase reduces the complexity of the n-ary view integration algorithm. Further, the resultant GCS is not affected by the division of these phases. The delay of COT conflicts analysis makes the designer intervention more beneficial because he can refer to an already integrated GCS before making a decision. In any case the following conclusions can be made:

1 Unlike n-ary view integration where the next E-R could be from any view, binary view integration allows the designer to concentrate on one view at a time. In binary view integration the E-Rs from one view at a time are integrated before another view is considered. Therefore, any conflicts which may take place can be resolved with the help of the corresponding user which is significant when integrating the views from a very large organization.

2 Relationship n-ary view integration achieved by RNVI helps the designer in identifying homonym relationships and some synonym entities which are part of the current relationship view. The relationship n-ary view integration has the effect of integrating a set of matching E-Rs (see section 6.4.5) at a time from all the views. This approach exposes any 'bad' E-Rs and homonym relationship names such as 'type', 'teaches' and 'gives' in the GCS (see appendix D).

3 Entity n-ary view integration by ENVI helps in identifying synonym relationships and entities which are part of the current entity view. The integration of a particular entity view means the integration of all the E-Rs in which this entity is involved (see Figs. 7.5 and 7.6). Since ENVI displays each entity view after its integration is completed, the designer may observe the naming conflicts in the relationship and entity names which form this particular entity view.

4 NVI is a variation of BVI. The basic object used in the binary and the n-ary view integration process is the E-R. Therefore, the difference between these different types of view integration approaches is in the choice of the next E-R and not in the view integration principle.

5 The speed of integration is similar (including time for the designer intervention to resolve conflicts). The time taken by the VI to integrate the views is the time it takes to integrate all the E-Rs from all the views plus the time taken by the designer to resolve conflicts. Since in both the binary and n-ary view integration approaches one E-R at a time is integrated, and the total number of E-Rs integrated is the same, then the speed of integration is similar.

6 The number and type of conflicts is the same for the three types of integrators. Since only the sequence of integrating the E-Rs which is different but the number and semantics of the E-Rs being integrated is the same, the same type of conflicts would take place.

7 True n-ary view integration means the simultaneous integration of all the objects in all the views and using the global semantics of these views to resolve conflicts. Therefore the entity and relationship n-ary view integration, in association with the COT conflicts analysis, have made some contribution towards achieving true n-ary view integration.

8 Processing the COT conflicts analysis separately from view integration reduced the complexity of the integration algorithms, especially in the case of the n-ary view integration algorithms. If COT conflicts analysis are processed as part of view integration, then confusions would occur with regards to the source of the objects to be matched and compared. Should the current object be compared with the GCS objects or with the view objects and which is to be given preference.

9 The resultant GCS is the same for all three types of view integrators. The three types of view integrators integrate the same E-Rs in the same way; they differ only on the sequence in which they integrate these E-Rs. Further, COT conflicts analysis and object fuzzy matching are processed as post-integration phases and thus not influencing the view integration process. Therefore, since the resultant GCS is created from the same set of E-Rs and subjected to the same post-integration processes, it is the same regardless of the type of VI used to create it.

CHAPTER 9

CONCLUSIONS AND FURTHER RESEARCH

9.1 Conclusions

This thesis described VIM for the integration of views modelled in ERM. The system to achieve VIM is VI. Two types of VI were developed. BVI integrates the views in a binary manner, whilst NVI integrates the views in an n-ary manner. Two types of NVI were implemented, the ENVI, and the RNVI. A specially developed language called VDL is used to map the ERM pictorial representation of the views to VI. The result of integrating the views is a GCS which should represent the global semantics of the organization.

This section presents conclusions about view integration research in general and about VIM in particular.

The objectives of this view integration research were to:

1 Identify all possible conflicts which may occur when integrating ERM views.
2 Design unique and correct resolutions to the identified conflicts.
3 VI can transport all the semantics from the views to the GCS.

Conflicts are either caused by the misinterpretation of semantics at view modelling or by genuine different views of the same data by different users. The identification of both types of these conflicts requires VI to understand respectively the syntactic and semantic structures of ERM and the views. To accommodate all the different user views of the same data in the same GCS, a compromise must be made. Failing to reach a compromise means the need to create intermediate views to act as interfaces between the GCS and the individual user views. The misinterpretations of semantics at view modelling results in either naming or structural conflicts which must be exposed by the VI.

Chapter 3 showed how the semantics of ERM views are eventually integrated through the integration of their individual E-Rs. Therefore the individual integration of E-Rs of the views with those of the GCS ensures that all the semantics are transported from the views to the GCS. This integration of one E-R at a time with the GCS identifies conflicts such as cardinality conflicts and attribute conflicts. In implementing this E-R integration approach, VI classifies the matches between the view E-Rs and the GCS E-Rs in one of five categories: identical, near identical, closely similar, similar or different. The GCS is updated with the semantics of the view E-R depending on the type of match.

COT conflicts are either caused by naming conflicts or by genuine structural conflicts. The resolutions to some of these conflicts could be predetermined and automated by VI.

However, other conflicts could be identified but precise resolutions cannot be predetermined. For each of these conflicts, a number of possible resolutions could be used. The difficulty was in choosing one of these proposed resolutions. Further, although the predetermined resolutions were effective and correct in most situations, it was not possible to prove that they are effective and correct in all conflict situations.

Conflicts in view integration can either be naming conflicts or structural conflicts. Naming conflicts can either be synonyms or homonyms. No method could be devised to identify objects of the same type suffering from homonym naming conflicts. However, homonym naming conflicts which occur between objects of different types are identified as COT conflicts (section 4.3). COT conflicts can either be caused by genuine naming conflicts or by genuine structural conflicts. Some of the COT conflicts which are caused by genuine naming conflicts must be resolved, whilst others may reside in the GCS without causing any problems. For example, an entity and a relationship sharing the same name is an unacceptable COT naming conflict but an attribute sharing the same name as an entity is an acceptable COT naming conflict.

COT conflicts caused by genuine structural conflicts must be resolved. These conflicts are mostly caused by the flexibility in ERM which makes it possible to model the same semantics using different ERM objects or structures. For example, an attribute structure can be modelled as an entity structure or as an E-R structure. Fourteen COT conflicts were identified and discussed in section 4.3. Some of these conflicts are identified and resolved automatically by VI, whilst the resolution of others, though they can be identified by VI, must be decided by the designer. Suggestions for possible resolutions were given for all the COT conflicts, however, only seven were implemented. The integration of the sixteen views resulted in a number of these conflicts (see section 8.5 and appendix F). The suggested resolutions presented in section 4.3 proved adequate and sufficient in all these conflicts. To enhance and increase the automation of these COT conflicts, experience is needed by modelling and integrating views from large organizations. Modelling a very large application would create a diversity of COT conflicts which will contribute to better and more accurate recommendations. Further, COT conflicts will always be inherent in ERM modelling.

The semantics of genuine n-ary E-Rs can be very complex and their integration is not currently handled by VI. However, sometimes homonym relationship names cause the integration of two or more binary E-Rs to result in an n-ary E-R (see section 3.3.2.2). Occasionally, such homonym relationship names appear representative and to choose another name for either or both would result in undesirable names. Currently VI allows 'bad' E-Rs to coexist in the GCS, but these 'bad' E-Rs must be corrected before the GCS is mapped to a DBMS.

The object fuzzy matching approach presented in section 4.2 to identify synonyms was tested in a practical view integration session. Sixteen views of the department of 'Computer Science' (appendix A) were integrated to produce a GCS (appendix D). The neighbours of each of the GCS entities were matched with all the other GCS entities to find the level of similarity between these entities. It was discovered that the resultant SLFs achieved were not always representative. The chain effect (see section 8.6) and the number of neighbours of the entities being matched contributed to the inaccuracy of the SLF values.

The two methods (the weigh and add method and the weigh and multiply method) were tested. The results obtained by using the weigh and add method are shown in appendix G, and those obtained by using the weigh and multiply method are shown in appendix H. Six different weights were used in order to study the effect each type of neighbour have on the resultant SLFs. The first three weights were used to obtain SLFs based on using one type of neighbour. The second three weights were applied to find the level of importance each type of neighbour should play in order to evaluate the most representative SLFs. The choice of the weights was based on experimentation rather than any mathematical formula. The six chosen weights represent all the variations necessary to test the effect of the different neighbours on the object fuzzy matching results. Although other variations of these weights could be tested, the new weights would not produce better results.

SLF values were divided into three categories of low, medium and high. It was discovered that the placement of these SLF categories were occasionally incorrect. Further, no resolution to this problem could be identified. The results obtained by using the two object fuzzy matching methods varied a great deal. Although the weigh and multiply method produced less high and medium categories, these were mostly not representative, and occasionally misplaced. Although its categories were sometimes incorrect, the weigh and add method on the whole produced better classification of categories. The SLF values and consequently the incorrect SLF categories are caused by the chain effect rather than the object fuzzy matching approach. Admittedly, better statistical techniques could contribute to better classification of some of the categories, however, unless the chain effect is removed, any object fuzzy matching method coupled with any statistical analysis of the SLF values would produce incorrect SLF categories.

The work reported in this thesis benefited from the view integration and some of the database integration research already reported in the literature. However, this is the first complete view integration methodology for the development of the conceptual schema to be implemented. VI was simplified by separating the COT conflicts analysis from the view integration phase and by using the E-R as the basic object for integration.

VI and its binary (BVI) and n-ary (ENVI & RNVI) work for practical applications and would consequently integrate any set of views modelled using ERM regardless of their size or application area. However, due to the nature of semantics (Abrial 1974, Kent 1978 and Kent 1981) and the flexibility of ERM, VI will always require the designer's intervention to resolve certain types of conflicts. Although VI has the the resolutions to these conflicts, there are situations where it cannot choose from among a number of built-in resolutions. Therefore, the level of automation of any view integrator will always be related to the preciseness, flexibility and semantic richness of the SDM used.

VI has some of the characteristics of an expert system. The knowledge base is defined by the relational representation of the views and the GCS as described in sections 6.2 and 6.3. However, although an expert system may provide a modified environment for view integration, it would not enhance the capabilities of VI because the difficulty in view integration is in identifying conflicts and giving precise resolutions to these conflicts.

No computer system, expert or otherwise, is able to totally replace the human expert. The reason for this is that the common sense and intelligence normally attributed to the human expert cannot all be written as rules, yet common sense, knowledge and intelligence are rules which in theory can be programmed. To record all the knowledge known to the designer or the user is a costly exercise. Consider the example E-R 'student attends course'. To record this fact is simple, but to record all an academic can infer from such fact would run into many thousands of possibilities. Therefore due to the absence of all the expertise and knowledge mastered by the designer, the view integrator is unable to resolve some conflicts.

VI does not solve all the logical database design problems, an area of research which will continue to be investigated. However, it has contributed to view integration by providing the following:

1 It is application independent.
2 It imposes no restrictions on the size or number of the views.
3 It does not require any form of integration assertions. Instead, it uses the semantics of the views and the definition of the ERM to carry out view integration.
4 It contains the first complete analysis of COT conflicts.
5 Though the object fuzzy matching approach needs further investigation, it represents the first effort to identify synonyms based on their ERM neighbours.

9.2 Further research

Including data abstraction and other semantic data modelling concepts in view integration

The most common data abstraction techniques are generalization and aggregation, first proposed by Smith & Smith (1977). As well as generalization and aggregation, other data abstraction techniques have been recommended for ERM. Further, numerous non data abstraction extensions to ERM have been reported in the literature. VI could be extended to accept and integrate views with some of these extensions. Therefore, this research will have to combine all the unique extensions to ERM and enhance VI to integrate ERM views with these extensions.

Modelling and integrating transactions

A complete conceptual schema must cover both the statics and dynamics of the organization. VI deals only with the modelling and integration of the statics of the organization. Some attempts have been made to model the dynamics of the organization and use it to test the completeness of the conceptual schema (Yao *et al* 1982 and Batini *et al* 1985b). Currently, VI considers the GCS complete on the basis that all the semantics of the organization are included in the views, and that all the views are modelled completely. This research requires the development of a transaction specification language for ERM and the enhancement of VI to process these meta transactions against the GCS as a post-integration phase.

Creating ERM views directly from requirements analysis

Prior to modelling the views in ERM, the designer is expected to carry out the requirements analysis stage. The requirement analysis are achieved by interviewing users, studying forms and activities, identifying the objects and so on. A pre-integration phase could be included in VI to read these statements and produce ERM views directly. Storey & Goldstein (1988) reported the first attempt of this kind. However, their interactive approach could be enhanced by producing ERM views from English sentences based on Chen (1983). Although Chen shows some of the translation rules from English to ERM, more research is needed to enhance these rules to work for more general cases.

Global view integrator

A global view integrator is one which can integrate views modelled using any data model. One approach to developing a global view integrator is by developing a data model mapper which can translate the semantics of views modelled using any SDM to a common SDM. A better approach would be to develop a view integrator which accepts the

definition of the SDM used to model the views together with the definitions of the conflicts and use this knowledge to carry out the view integration process.

General enhancements of VI

VI reads and integrates the ERM views and produces the GCS, analyses COT conflicts and provides querying facilities for the views and the GCS. However, to promote VI from a prototype to a complete tool, the following enhancements would be desirable:

1 Providing a graphical interface to display the contents of the views and the GCS: Attempts have already been reported to achieve this (Gilberg 1985, Tamassia *et al* 1983, Batini *et al* 1985a and Tamassia 1985). These algorithms could be implemented in VI.

2 The query facility provided by VI to query the semantics of the views and the GCS is only elementary. Therefore, more query menus are needed to help the designer investigate the contents of the views and the GCS in order to resolve conflicts.

3 Currently VI deals with commands such as delete an attribute, change an attribute name, change an entity name, and so on. Since most conflicts require the designer to change the semantics of the GCS, many more interactive commands can be developed. Examples of such commands are create a new entity, create a new E-R, delete an E-R, change the cardinality of an entity, and so on.

4 The attributes of the entities and the relationships are accumulated by integrating all their occurrences from all the views. The resultant entities and relationships remain unnormalized. A possible extension to VI is to include a post integration phase of normalization. Some attempts have been reported about the normalization of ERM schema (Chung *et al* 1983, Shan & Shixuan 1984, Ling 1985a, Ling 1985b, Makowsky *et al* 1986 and Dawson & Parker 1988), and therefore such extension may draw from such research.

5 The object fuzzy matching approach presented in this thesis contributed to the identification of synonymous entities. However both the weigh and add method, and the weigh and multiply method have drawbacks. Therefore, the modification of either or both of these methods to identify synonymous entities, attributes or relationships is an open problem. This could possibly be achieved by finding ways of reducing the chain effect and by applying better statistical techniques to analyse the results produced by the two methods.

6 The COT conflict analysis discussed in this thesis did not consider when one object is to be eliminated and its semantics are to be transferred to the other object. COT conflicts analysis reduced the need for object transformation (Batini & Lenzerini 1984) but occasionally the resolution of COT conflicts require some transformations; this needs to be investigated further.

7 VI analyses COT conflicts between two objects at a time. A future enhancement of this approach is to consider multiple COT conflicts at a time so as to learn from their cumulative semantics.

8 Integration of n-ary E-Rs.

Assertions : These are instructions given by the designer, either to complement the semantics of the SDM or to direct the view integrator to carry out the view integration process.

Binary view integration : This is an approach to view integration where one view at a time is integrated with the GCS.

Chain effect : This is the accumulative sequence of conflicts causing the same semantics to progressively be represented differently.

conceptual schema : The conceptual schema is a description of the part of the organization which is to be represented by the data in the database. See also global conceptual schema.

Cross object type conflicts (COT conflicts) : A cross object type conflict is a type of integration conflicts. A cross object type conflict occurs when the same semantics is modelled using different ERM objects.

Current view E-R : This is the current E-R of the current view being integrated.

Data model : This is a mechanism for specifying the structure of a database and the operations that may be performed on the data in that database

Database integration : This is the process of integrating the schemas of existing databases, distributed or otherwise.

Database integrator : This is the program to achieve *database integration.*

Dependencies integration approach : This is a view integration approach based on identifying the dependencies between the data elements and using these dependencies to carry out the view integration process.

Designer : This is the person who designs the conceptual schema and the physical schema. This person could be the Data Base Administrator (DBA).

Enterprise view : This is the top level abstract view of the organization, and presents a basic schema of the organization in the form of a number of major entities and their relationships.

Entity view : The entity view of a particular entity are all the E-Rs in which this entity is involved.

Form : This is any structured collection of variables which are appropriately formatted to support data entry or retrieval.

Global Conceptual Schema (GCS) : This is the overall schema for the whole of the organization.

Global view integrator : This is a view integrator which supposedly can integrate views modelled in different data models.

Homonym : This is a type of *naming conflict* which occurs in view integration when different objects of the same type are modelled with the same name.

Integration assertions : These are the instructions modelled by the designer to direct the activities of the view integrator, and may be to present solutions for the anticipated conflicts.

Integration conflicts : Conflicts occur during view integration as a result of the same semantics being modelled differently in different views. Integration conflicts can either be *naming conflicts* or *structural conflicts.*

Inter-view assertions : These are assertions pertaining to multiple views.

Intra-view assertions : These are assertions pertaining to a single view.

Modelling assertions : These are the assertions modelled by the designer to complement the semantics of the views. A special language is normally used for this purpose.

N-ary view integration : This is an approach to view integration where all the views are considered for integration simultaneously.

Naming conflicts : A naming conflict is a type of integration conflicts. Naming conflicts occur when two objects of the same type are either *homonyms* or *synonyms*. Naming conflicts also occur between objects of different types.

Object : In ERM (Chen 1976) an object is an attribute, an entity, a relationship or an E-R.

Object fuzzy matching (OFM) : This is process of finding the level of similarity between ERM objects based on the similarity of their neighbours and neighbours characteristics. The result of object fuzzy matching is a value called the *Similarity Level Factor (SLF)*.

Object integration approach : This is a view integration approach based on identifying similarities and differences between the objects of the views before integrating them. The views are modelled using an SDM.

Object neighbours : The neighbours of an SDM object are all the other objects with which it has semantic connections. For example, the neighbours of an entity are its attributes and the relationships in which it is involved.

Object occurrence number : This is a number associated with each object in the views in order to uniquely identify it.

Object transformation : This is the process of transforming an object from one SDM type to another SDM type. It is applied to resolve *structural conflicts* in view integration.

Occurrence number : This is a number associated with entities and relationships to uniquely identify them in their view internal relational structure representation.

Relationship view : The relationship view of a particular relationship are all the E-Rs which are related by this relationship.

Semantic Data Model (SDM) : These are advanced types of data models which are used to logically structure the information in a database in a manner that captures more of the meaning of the data than conventional data models.

Similarity Level Factor (SLF) : This is a value which is normally in the range of 0 to 1, produced by the object *fuzzy matching process* to indicate the level of similarity between ERM objects.

Structural conflicts : A structural conflict is a type of integration conflicts. Structural conflicts occur when the same semantics are modelled differently in different views. The most common structural conflicts are the *cross object type conflicts*.

Structure : In ERM (Chen 1976) a structure is two or more objects modelled in accordance with the definition of ERM.

Synonym : This is a *naming conflict* which occurs in view integration when the same objects of the same type are modelled using different names.

Valued attribute : Any attribute which has a value is a valued attribute.

View : This is a particular section of the organization which represents the data (static and dynamic) for a particular user.

View integration : This is the process of integrating the views to form the Global Conceptual Schema (GCS).

View integrator : This is the program or the person who performs view integration.

View modelling : This is the process of breaking a large organization into its views and then modelling each view using a chosen SDM.

ABRIAL (1974), "Data Semantics", *Data Base Management*, North-Holland, Amsterdam, pp. 1-59.

AL-FEDAGHI S. & SCHEVERMANN P. (1981), "Mapping Considerations in the Design of Schemas for the Relational Model", *IEEE Trans. Software Eng.*, SE-**7** (1), Jan., pp. 99-111.

ALBANO A. & ORSINI R. (1985), "Dialogo: An Interactive Environment for Conceptual Design in Galileo", in: CERI S. (Ed.) (1985).

ALBANO A., CARDELLI L. & ORSINI R. (1985a), "GALILEO: A Strongly-Typed, Interactive Conceptual Language", *ACM Trans. on Database Systems*, **10** (2), June, pp. 230-260.

ALBANO A., DE ANTONELLIS V. & DI LEVA A. (Eds.) (1985b), *Computer-Aided Database Design - the DATAID project*, North-Holland.

ANTONELLIS V. & DI LEVA A. (1985), "DATAID-1: A Database Design Methodology", *Information Systems*, **10** (2), pp. 181-195.

ATZENI P. & CARBONI E. (1983), "INCOD (A System for Interactive Conceptual Design) Revisited after the Implementation of a Prototype", in: Davis *et al* (Eds.) (1983) .

ATZENI P., BATINI C., CARBONI E., DE ANTONELLIS V., LENZERINI M., VILLANELLI F. & ZONTA B. (1985), "INCOD-DTE: A System for Interactive Conceptual Design of Data Transactions and Events", in CERI S. (Ed.) (1985).

AVISON D.E. & FITZGERALD G. (1989), *Information Systems Development: Techniques and Tools*, Information systems series, Blackwell Scientific.

BAKER C.T. (1974), " Inherent structures in data", *Technical report, TR 21.545*, IBM, Kingston, NY.

BALBO G., DEMO B.G, Di LEVA A. & GIOLITO P. (1984), "Dynamic analysis in database design", *Proc. IEEE Int. Conf. on data Engg.*, Los Ang., pp. 238-243.

BATINI C. & Di BATTISTA G. (1988), "A methodology for conceptual documentation and maintenance", *Inform. Systems*, **13** (3), pp. 297-318.

BATINI C. & LENZERINI M. (1983), "A Conceptual Foundation for View Integration", *Proc. of the IFIP Working Conference 8.1*, Budapest, Hungary, May, pp. 109-139.

BATINI C. & LENZERINI M. (1983), "A Methodology for Data Schema Integration in the Entity-Relationship Model - Extended Abstract", in: Davis C.A. *et al* (Eds.) (1983), *Entity-Relationship Approach*, North-Holland.

BATINI C. & LENZERINI M. (1984), "A Methodology for Data Schema Integration in the Entity Relationship Model", *IEEE Trans. on Software Engg.* , **10** (6), Nov., pp. 650-664.

BATINI C. (Ed.) (1988), *Proc.7th Int. Conf. on Entity-Relationship Approach*, Roma, Nov 16-18.

BATINI C., DE ANTONNELLIS V. & DI LEVA A. (1984), "Database Design Activities Within the DATAID Project", *Database Engg.* , **7** (4).

BATINI C., FURLANI L. & NARDELLI E. (1985a), "What is a good diagram", in: Ferrara (Ed.) (1985).

BATINI C., LENZERINI M. & MOSCARINI M. (1985b), "Views Integration", in: Ceri S. (Eds.) (1985), pp. 57-84.

BATINI C., LENZERINI M. & NAVATHE S.B. (1986), "A Comparative Analysis of Methodologies for Database Schema Interpretation", *ACM Computing Surveys*, **18** (4), Dec., pp. 333-365.

BATINI C., LENZERINI M. & SANTUCCI G. (1982), "A Computer-Aided Methodology for Conceptual Database Design", *Information Systems*, **7**(3), pp. 265-280.

BATINI C., NARDELLI E., TALAMO M. & TAMASSIA R., (1985c), "GINCOD: A Graphical Tool for Conceptual Design of Data Base Applications", In: Albano A. *et al* (1985b) (Eds.).

BEERI C., MENDELZON A.O., SAGIV Y. & ULLMAN J.D. (1981), "Equivalence of Relational Database Schemes", *SIAM J. of COMPUT.*, **10** (2), May, pp. 352-370.

BERMAN S. (1986), "A semantic Data Model as the Bases for an Automated Database Design Tool", *Inform. Systems,* **11**(2), pp. 149-165.

BERNSTEIN P.A. (1976a), "Synthesizing Third Normal Forms from Functional Dependencies", *ACM Trans. on Database Systems*, **1** (4), pp. 277-298.

BERNSTEIN P.A. (1976b), "Segment synthesis in logical database design", *IBM J. Res. and Develop.*, **20 (4),** July.

BERNSTEIN P.A., SWENSONS J.R. & TSICHRITZIS D.C. (1975), "A unified approach to functional dependencies and relations", *Proc. ACM 1975 Conf.*, San Jose, Cal., pp. 237-245.

BERRA P.B. & MITKAS P.A. (1988), "An initial design of a very large knowledge base architecture", *Information Technology for Organizational Systems*, H.J. Bullring *et al* (Eds.), North Holland, Elsevier Science Publ.

BILLER H. & NEUHOLD E.J. (1978), " Semantics of Data Bases: The Semantics of Data Models", *Information Systems*, **3**, pp. 11-30.

BISKUP J. & CONVENT B. (1986), "A Formal View Integration Method", *ACM SIGMOD Int. Conf. on Management of Data*, pp. 398-407.

BISKUP J., DAYAL U. & BERNSTEIN P.A. (1979), "Synthesizing Independent Database Schemes", *ACM SIGMOD Int. Conf. on Management of Data*, pp. 143-151.

BJORNERSTEDT A. & HULTEN C. (1984),"RED1: A Data Base Design Tool for the relational model of data", *IEEE Database Engg.*, **7**(4), pp. 34-39.

BOUZEGHOUB M. & GARDARIN G. (1984), "Design of an expert system for database design", in Gardarin & Glenbe (Eds.) (1984).

BOUZEGHOUB M. & GARDARIN G. (1985), "Database Design Tools: An Expert System Approach", *Proc. of the Very Large Data Bases*, Stockholm, pp. 82-95.

BRACCHI G., CERI S. & PELAGATI G. (1985), "A Set of Integrated Tools for the Conceptual Design of Database Schemas and Transactions", in: CERI S. (Ed.) (1985).

BRAGGER R.P., DUDLER A., REBSAMEN J. & ZEHNDER C.A. (1984), "Gambit: An Interactive Design Tool for Data Structures, Integrity Constraints and Transactions", *Int. Conf. on Data Engg.* , Los. Ang., CA, USA, 24-27 Apr., pp. 399-407

BRANCO R.R. & YADAV S.B. (1985), "A methodology for modelling dynamics of structures of an organization", *Inform. Systems*, **10** (3), pp. 299-315.

BREITBART Y., OLSON P.L. & THOMPSON G.R. (1986), "Database Integration in a Distributed Heterogeneous Database System", in: *Proc. Second Int. Conf. on Data Engg.* , Los Angeles, CA, Feb. 5-7, pp. 301-310.

BRODIE M.L (1981), "On modelling behavioural semantics of data", *Proc. seventh Int. Conf. on Very Large Data Bases*, Cannes, France, Sep.

BRODIE M.L. (1984), "On the Development of Data Models", in: Brodie *et al* (Eds.) (1984).

BRODIE M.L., MYLOPOVLOS J. & SCHMIDT J.W. (Eds.) (1984), *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, Springer Verlag.

BUBENKO Jr J.A. (1979), "On the Role of Understanding Data Models in Conceptual Schema Design", *Proc. of the Fifth Int. Conf. on Very Large Data Bases*, Rio de Janeiro, Brazil, Oct. 3-5, pp. 129-139.

BUCHMAN A.P. & DALE A.G. (1979), "Evaluation Criteria for Logical Design Methodologies", *CAD* **11**, May.

CASANOVA M.A. & VIDAL V.M.P. (1983), "Towards a Sound View Integration Methodology", *ACM-SIGACT-SIGMOD Symposium on Principles of Database Systems*, pp. 36-47.

CASANOVA M.A. (1982), "A Theory of Data Dependencies over Relational Expressions", *Proc. ACM SIGMOD-SIGACT Conf. on Principles of Database Systems.*

CASANOVA M.A., FAGIN R. & PAPADIMITRIOU C. H. (1982), "Inclusion Dependencies and their Interaction with Functional Dependencies", *1st ACM-SIGACT Symposium on Principles of Database Systems*, Los Ang, March 29-31, pp. 171-176.

CERI S. (Ed.) (1985), *Methodology and Tools for Database Design*, North-Holland.

CHAN E.P.F. & LOCHOUSKY F.H. (1980), "A Graphical Data Base Design Aid using the Entity-Relationship Model", in: Chen (Ed.).

CHANDRA A.K. & VARDI M.Y. (1985), "The Implication Problem for Functional and Inclusion Dependencies is Undecidable", *SIAM Journal of Computing*, **14** (3), pp. 671-677.

CHEN P.P. (1976), "The Entity Relationship Model: Towards a Unified View of Data", *Trans. on Database Systems*, **1**(1), pp. 9-36.

CHEN P.P. (1977), "The Entity-Relationship Approach to Logical Data Base Design", *Q.E.D. Monograph Series*, Wellesley, Massachusetts.

CHEN P.P. (Ed.) (1980), "Entity Relationship Approach to Systems Analysis and Design", *Proc. of the First Int. Conf. on Entity-Relationship Approach*, North-Holland.

CHEN P.P. (Ed.) (1981), *Proc. of the Second Int. Conf. on Entity-Relationship Approach*, ER Institute, Sangus, CA, USA: North-Holland.

CHEN P.P. (1982), "Survey of State of the Art Logical Database Design Tools (Final Report)", *California University*, Los Angeles, Graduate School of Management, Apr.

CHEN P. P. (1983), "English Sentence Structure and Entity-Relationship Diagrams", *Information Sciences*, **29**, pp. 127-149.

CHEN P.P. (1984), "An algebra for a directional binary Entity-Relationship model", *Int. Conf. on data engg.*, LA, CA, USA, 24-27 Apr.

CHEN P.P. (1985), "Database Design Based on Entity and Relationship", in: Yao *et al* (Eds.) (1985), pp. 174-210.

CHILSON D.W. & KUDLAC E. (1983), "Database Design: A Survey of Logical and Physical Design Techniques", *Data Base*, **15** (1), pp.11-19.

CHOOBINEH J., MANNINO M., NUNAMAKER J.F. & KONSYSKI B.R. (1988), "An expert database Design System based on Analysis of Forms", *IEEE Trans. Softw. Engg.*, **14** (2).

CHUNG I., NAKAMURA F. & CHEN P.P. (1983), " A Decomposition of relations using the Entity-Relationship Approach", in: Chen P.P. (Ed.) (1983).

CODD E.F. (1971), "Normalized Database Structure: a brief tutorial", *ACM SIG-FIDET Workshop on Data Description, Access, and Control*, San Diago, California.

CODD E.F. (1972), "Further Normalization of Database Relational Model", in: Rustin R. (Ed.), *Database Systems(Courant Computer Science Symposia 6)*, Prentice Hall.

CONVENT B. (1986), "Unsolvable Problems Related to The View Integration Approach", *Int. Conf. on Database Theory*, Roma, Italy, Sep. 8-10, pp. 141-155.

CURTICE R.M. (1984), "An Automated Logical Data Base Design and Structured Analysis Tool", *Database Engg.* , **3**, pp. 221-226.

CZEDO B. & EMBLEY D. (1987), "An Approach to Schema Integration and Query Formulation in Federated Database Systems", *Third Int. Conf. on Data Engg.*, Feb., pp. 477-484.

DATE C.J. (1981), *An Introduction to Database Systems*, (3rd Ed.) Addison-Wesley Pub. Co.

DATE C.J. (1983), *An Introduction to Database Systems*, (Vol. 2.) Addison-Wesley Pub. Co.

DAVIS C.G., JAJODIA S., NG P.A. & YEH R.T. (Eds.) (1983), *Proc. of the Third Int. Conf. on Entity -Relationship Approach*, Elsevier Science Pub. (North-Holland), ER Institute.

DAWSON K.S. & PARKER L.P (1988), "From Entity-Relationship diagrams to Fourth Normal Form: A Practical aid to analysis", *The Computer Journal*, **31** (3), pp. 258-268.

DAYAL U. & HWANG H. (1984), "View Definition and Generalization for Database Interpretation of a Multidatabase System", *IEEE Trans. on Software Engg.* , **10** (6), Nov., pp. 628-644.

De ANTONELLIS V. & ZONTA B. (1984), "A casual approach to dynamic modelling", *Database Engg.*, **3**.

De REIT V (1986), "Expert Systems in trouble", *Information processing 86*, H.J. Kualet (Ed.), Elsevier Science Publ., North-Holland, IFIP.

de SOUZA J.M. (1986), "SIS - A Schema Integration System" *5th British Nation. Conf. on Database (BNCOD-5)*, Canterbury, England.

DEMURJIAN S.A. & HSIAO D.K. (1988), "Towards a Better Understanding of Data Models Through the Multilingual Database System", *IEEE Trans. on Software Engg.* , **14** (7), pp. 946-958.

DL/1 DOS / VS General Information Manual, GH20 - 1246, *IBM Corporation*, Data Processing Division, White Plains, NY.

ELMASRI R. & NAVATHE S. (1984), "Object Integration in Logical Database Design", *Proc. of the IEEE COMPDEC Conf.*, Los Ang., April, pp. 426-433.

ELMASRI R. & WIEDERHOLD G. (1979), "Data Model Integration Using the Structural Model", *Proc. of ACM SIGMOD Int. Conf.*, pp. 191-202.

ELMASRI R. & WIEDERHOLD G. (1980), "Properties of relationships and their representation", *National Computer Conference*, AFIPS, pp. 319-326.

ELMASRI R., LARSON J.A., NAVATHE S. & SASHIDHAR T. (1984), "Tools for View Integration", *Database Engg.* , **3**, pp. 209-214.

ELMASRI R., WEELDREYER J. & HEVNER A. (1985), "The category concept: An extension to the entity-relationship model", *Data and Knowledge Engg.* , **1**, pp. 75-116.

ELMASRI R.R & NAVATHE S.B. (1989), *Fundamental of Database Systems*, Benjamin Cummings.

FAGIN R. (1977), "Multivalued Dependencies and a New Normal Form for Relational Databases", *ACM TODS*, **3**, pp. 262-278.

FERRARA F.M. & BATINI C. (1984), "GDOC: A Tool for Computerized Design and Documentation of Database Systems, *Database*, **15** (4), pp. 15-20.

FERRARA F.M. (1985a), " An integrated system for the design and documentation of database applications", in: Ferrara (ed.) (1985), pp.109-113.

FERRARA F.M. (ED.) (1985b), *Proc. of the Fourth Int. Conf. on Entity Relationships Approach*, IEEE Computer Society, Silver Spring, Md.

FURTADO A.N. & NEUHOLD E.J. (1986), *Formal Techniques for Data Base Design*, Springer Verlag.

GANE C. & SARSON T. (1979), "Structured Systems Analysis; Tools and Techniques", *Prentice-Hall.*

GARDARINE G. & GLENBE E. (Eds.) (1984), *New Applications of Data Bases*, Academic Press.

GERRITSEN R. (1982), "Tools for the Automation of Database Design", in Yao S.B. *et al* (Eds) (1982).

GILBERG R.F (1985), "A schema methodology for large Entity-Relationship diagrams", in: Ferrara (Eds.) (1985).

GONXALEX-SUSTAETA J. & BUCHMANN A.P. (1986), "An Automated Database Design Tool Using the ELKA Conceptual Model", *23rd ACM/IEEE Design Automation Conf.*, June 29 - July 2, pp. 752-759.

HAWRYSZKIEWYCZ I.T. (1985), "A Computer-Aid for E-R Modelling", in: Ferrara (Ed.) (1985), pp. 64-69.

HOUSEL B.C., WADDLE V.E. & YAO S.B. (1979), "The Functional Dependency Model for Logical Database Design", *Proc. 5th Int Conf. on VLDB,* Rio de Janeiro, Brazil, Oct. 3-5.

HOWE D.R. (1983), *Data Analysis for Data Base Design*, Arnold.

HULL R. & KING R. (1987), "Semantic Database Modelling: Survey, Applications and Research Issues", *ACM Computing Surveys*, **19** (3), pp. 201-260.

IBM Data Base Design Aid (1977), Data Base Design Aid, Designers Guide, GH 20 - 1627, *IBM Corporation*, Data Processing Division, White Plains, NY.

IOSSIPHIDIS J. (1980), "A Translator to Convert the DDL of ERM to the DDL of System 2000", in: Chen P.P. (Ed.) (1980).

JAJODIA S., NG P.A. & SPRINGSTEEL F.N. (1983), "The Problem of Equivalence for Entity-Relationship Diagrams", *IEEE Trans. on Software Engg.* , **9** (5), pp. 617-630.

JARDINE D.A. (1984), "Concepts and Terminology for the Conceptual Schema and the Information Base", *Computers and Standards*, **3**, pp. 3-17.

JIANG T.L. & CHIN Y.H. (1984), "A Friendly Logical Database Design Tool for the Humming-Bird System", *Int. Conf. on Data Engg.*, Los Ang, CA, USA, 24-27 April., pp. 526-533.

JOO T.T., POH T.K. & MOI G.H. (1984), "DATADICT- A Data Analysis and Logical Database Design Tool", *Proc 10th VLDB*, Singapore.

KAUFMANN A. (1975), *Introduction to the Theory of Fuzzy Sets*, Academic Press, New York.

KENT W. (1978), *Data and Reality*, McGraw Hill.

KENT W. (1981), " Data Model Theory Meets a Practical Application", *Proc. of the Seventh Int. Conf. on Very Large Databases*, France, pp. 13-22.

KENT W. (1983), " A Simple Guide to Five Normal Forms in Relational Database Theory", *Comm. of the ACM*, **26** (2), pp. 120-125.

KERSTEN M.L. (1987), "A Conceptual Modelling Expert System", in: Spaccapietra S. (Ed.) (1987).

KING R. & McLEOD D. (1985), "Semantic Data Models", in: YAO (Ed) (1985).

LAENDER A.H.F. (1984), "An Approach to Interactive Definition of Database Views", *Proc. of the 3rd British National Conference on Databases*, pp. 173-183.

LANDERS T.A. & ROSENBERG R.L. (1982), "An Overview of Multibase", *Proc. of the 2nd Int. Symposium on Distributed Databases*, Berlin.

LEUNG C.M.R. & NIJSSEN G.M. (1988), "Relational database design using the NIAM conceptual schema", *Inform. Systems*, **13** (2), pp. 219-227.

LIEN Y.E (1980), "On the semantics of the Entity-Relationship model", in: Chen (Ed.) (1980).

LING T-W. (1985a), "A Normal Form for Entity-Relationship Diagrams", *Proc. of the Fourth Int. Conf. on the Entity-Relationship Approach*, IEEE, pp. 24-35.

LING T-W. (1985b), "An analysis of multivalued and join dependencies based on the entity-relationship approach", *Data & Knowledge Engg.* , **1**, pp. 253-271.

LUM V.Y., GHOSH S.P., SCHKOLNICK M., TAYLOR R.W., JEFFERSON D., SU S., FRY J.P. TEORY T.J., YAO B., RUND D.S., KAHN B., NAVATHE S., SMITH D., AGUILAR L., BARR W.J. & JONES P.E. (1979), "1978 New Orleans Database Design Workshop Report", *5th Int. Conf. on Very Large Data Bases*, Rio de Janeiro, Brazil, Oct. 3-5.

MAIER D. (1983), *The Theory of Relational Databases*, Pitman.

MAKOWSKY J.A., MAKOWITZ V.M. & ROTICS N. (1986), "Entity-Relationship consistency for Relational Schemas", *Int. Conf. on Database Theory*, Roma, Italy, pp. 306-322.

MANNINO M. (1983), "A Methodology for Global Schema Design", Ph.D. Dissertation, Dept. of Management Information Systems, University of Arizona, June.

MANNINO M.V. & EFELSBERG W. (1984), "Matching Techniques in Global Schema Design", *Int. Conf. on Data Engg.* , CA, USA, 24-27 Apr., pp. 418-425.

MARINOS L. & PAPAZOHLOU M.P. (1988), "An Approach to Heterogeneous Data Integration", in: I*nformation Technology for Organizational Systems,* H.J. Bullring *et al* (Eds.), Elsevier Science Publishers B.V., North-Holland.

MARINOS L., PAPAZOHLOV M.P. & NORRIE M., (1988), "Towards the design of an integrated environment for a distributed database", in: *Parallel processing and applications*, Chiricozzi E. & D'Amico (Eds.), North Holland.

MARK L. & ROUSSOPOLOUS N. (1983), "Integration of Data, Schema and Meta-Schema in the Context of Self-Documenting Data Models", in: DAVIS C.G. *et al* (Eds.) (1983).

MARKOWITZ V.M. & RAZ Y. (1984), "An Entity-Relationship algebra and its semantic capabilities", *The Journal of Systems and Software*, **4**, pp. 147-162.

MASSIMO F. & BATINI C. (1984), "GDOC: A Tool for computerised design and documentation of database systems", *Database* (USA), **15** (4), pp. 15-20.

McFADDEN F.R.M. & HOFFER J.A. (1985), *Data Base Management*, Benjamin Publ. Co.

MELKANOFF M.A. & ZANIOLO C. (1980), "Decomposition of relations and Synthesis of Entity-Relationship Diagrams", in: Chen P.P. (Ed.) (1980).

MEYER K. & DOUGHTY J. (1984), "Automatic Normalization and Entity-Relationship generation through Attributes and Roles", *British Gas*, work paper.

MOKOWSKY J., MARKOWITZ V.M. & ROTICS N. (1986), "Entity-Relationship consistency for relational schemas", *Int. Conf. on Database Theory*, Roma, Italy, Sep. 8-10, pp. 306-322.

MOTRO A. & BUNEMAN P. (1980), "Automatically Merging Databases", *Proc. of COMPSON 80*, Conf. on Distributed Computing, 21st, Fall, 1980, Washington, DC, pp. 279-286.

MOTRO A. & BUNEMAN P. (1981) "Constructing Superviews", *Proc. of ACM - SIGMOD Int. Conf. on Management of Data*, Ann Arbor, Michigan, pp. 56-64

MOTRO A. (1983), "Interrogating Superviews", *Proc. ICOD - 2, Second Int. Conf. on Databases,* Cambridge, England, Aug. 30 - Sept. 3, pp. 107-126.

MOTRO A. (1987), "Superviews: Virtual Integration of Multiple Databases", *IEEE Trans. on Software Engg.* , **13** (7), July, pp. 785-798.

MYLOPOULOS J. (1978), "Relationship Between and Among Models", *Proc. of the ACM SIGMOD Int. Conf.* , Austin, Texas, June, pp. 77-82.

MYLOPOULOS J., BERNSTEIN P.A. & WONG H.K. (1980), "A Language Facility for Designing Database Intensive Applications", *ACM Trans. on Database Systems*, **5** (2), June, pp. 185-207.

NAVATHE S., ELMASRI R. & LARSON J. (1986), "Integrating User Views In Database Design", *Computer*, Jan., pp. 50-62.

NAVATHE S.B. & GADGIL S.G. (1982), "A Methodology for View Integration in Logical Database Design", *Proc. of the Eighth Int. Conf. on Very Large Data Bases*, Mexico City, Sep., pp. 142-164.

NAVATHE S.B. & SCHKOLNICK M. (1978), "View Representation in Logical Database Design", *Proc. of the ACM - SIGMOD 1st. Conf.*, Austin, TX, June, ACM, NY.

NAVATHE S.B. (1985), "Important Issues in Database Design Methodologies and Tools", in: Albano *et al* (Eds.) (1985).

NAVATHE S.B., SASHIDHAR T. & ELMASRI R. (1984), "Relationship Merging in Schema Integration", *Proc. of the Tenth Int. Conf. on Very Large Data Bases*, Singapore, August, pp. 78-90.

NEGOITA C.V. & RALESCU D.A. (1975), *Application of Fuzzy Sets to Systems Analysis*, Birkhauser Verlag, Brazil.

NG P.A. & PAUL J.A. (1980), "A Formal definition of Entity-Relationship models", in: Chen (Ed.) (1980).

OLLE T.W., SOL H.G. & VERGIN-STUART A.A. (1982), "Information Systems Design Methodologies: A Comparative Review, *Proc. of IFIP 8 Working Conf. on Comparative Review of Information Systems Design Methodologies*, Noordwijkerhout, Netherlands, May, Elsevier North Holland, Amsterdam.

RAVER N. & HUBBARD G.U. (1975), "Automated Logical File Design", *First Int. Conference on Very Large Data Bases*, Framingham, MA, Sep.

RAVER N. & HUBBARD G.U. (1977), "Automated Logical Data Base Design and Applications", *IBM System Journal*, No. **3**, pp. 287-312.

REINER D., BRODIE M.L., BROWN G., FRIEDEL M., KRAMLICH D., LEHMAN J. & ROSENTHAL A. (1984), "The Database Design and Evaluation Workbench (DDEW) Project at CCA", *Database Engg.*, **3**, pp. 191-196.

RISSANEN J. (1982), "On Equivalence of Database Schemes", *Proc. of the ACM Symposium on Principles of Database Design*, pp. 23-26.

RODRIGUEZ-ORITZ G. (1981), "The ELKA Model Approach to the Design of Database Conceptual Models", *Ph.D. Thesis*, UCLA.

ROESNER W. (1985), "DESPATH: An ER Manipulation Language", in Ferrara (Ed.) (1985), pp. 72-81.

SAHA A.B. (1983), "A Set Oriented Abstract Conceptual Schema Based on Logic", *Ph.D. Thesis School of Information Systems, University of East Anglia.*

SAKAI H., KONDO H. & KAWASAKI Z. (1983), "A Development of the Conceptual Schema Design Aid in the Entity Relationship Model", in Chen P.P. (Ed) (1983).

SCHENEIDER H.J. & WASSERMAN A.I. (Eds.) (1982), *Automated Tools for Information Systems Design and Development*, North-Holland.

SCHREFL M., TJOA A.M. & WAGNER R.R. (1984), "Comparison Criteria for Semantic Data Models", *Proc. Second Int. Conf. on Data Engg.*, Los Angeles, CA, PP. 120-125.

SHAN W & SHIXUAN (1984), "Normal Entity-Relationship model", *Int. Conf on computers and applications*, IEEE computer Society press, Silver Spring, pp. 108-117.

SHIPMAN D.W. (1981), "The Functional Data Model and the Data Language DAPLEX", *ACM Trans. Database Systems*, **6** (1), pp. 140-173.

SHOVAL P. GUDES E. & GOLDSTEIN M. (1988), "GISD: A Graphical interactive system for conceptual database design", *Inform.Systems*, **13** (1), pp. 81-95.

SIBLEY E.H. & KERSCHBERG L. (1977), "Data architecture and Data Model Considerations", *National Computer Conference*, AFIPS, pp. 85-96.

SMITH J. & SMITH D. (1977), "Database Abstractions: Aggregation and Generalization", *ACM Trans. Database Syst.*, **2** (2), pp. 105-133.

SMITH J.M., BERNSTEIN P.A. DAYAL U., GOODMAN N., LANDERS T., LIN K.W.T. & WONG E. (1981), "Multibase - Integrating Heterogeneous Distributed Database Systems", *National Computer Conference*, AFIPS, pp. 487-499.

SOWA J.F. (1984), *Conceptual Structures: Information Processing in Mind and Machine*, Addison Wesley.

SPACCAPIETRA S. (Ed.) (1987), *Proc. of the Fifth Int. Conf. on Entity Relationships Approach*, Dijon, France, Nov. 1986, Elsevier Science Pub. Co., (North-Holland).

STACHOWITZ R.A. (1985), "A Formal Framework for Describing and Classifying Semantic Data Models", *Inform.Systems*, **10** (1), pp. 77-96.

STOCKER P.M. & CANTIE R. (1983), "A Target Logical Schema: The ACS", *Proc. Int. Conf. on Very Large Data Bases*, Florence, Italy.

TAMASSIA R. (1985), "New layout technique for Entity-Relationship diagrams", in: Ferrara (Ed.) (1985), pp. 305-311.

TAMASSIA R., BATINI C. & TALAMO M. (1983), "An algorithm for automatic layout of Entity-Relationship diagrams", in: DAVIS *et al* (1983), pp. 421-439.

TEORY T. AND FRY J. (1982), *Design of Database Structures*, Englewood Cliffs, NJ: Prentice-Hall.

TSICHRITZ D.C. & KLUG A. (Eds.) (1978), "The ANSI/ X3/ SPARC DBMS FRAMEWORK", Report of the study group on database Management Systems, *Inform. Systems*, **3**.

TSICHRITZIS D.C. & LOCKOVSKY F.H. (1982), *Data Models*, Prentice Hall.

ULLMAN J.D. (1982), *Database Systems*, Computer Science Press.

VAN GRIETHUYSEN J.J. (Ed.) (1982), "Concepts and Terminology for the Conceptual Schema and the Information Base", *ISO / TC97 / SC5 / WG3*, March.

VETTER M. (1977), "Data Base Design by Applied Data Synthesis", *Proc. of the Third Int. Conf. on Very Large Databases*, Japan, Oct., pp. 428-440.

WANG P.P. & CHANG S.K. (Eds.) (1980), *Fuzzy sets: Theory and applications to policy analysis and Information Systems*, Plenum.

WEELDREYER J. (1980), "Structural Aspects of the Entity-Category-Relationship Model", technical report HR-80-250, Honeywell Computer Science Centre, pp. 17-380.

WIEDERHOLD G. & ELMASRI R. (1979), "A Structural Model for Database Systems", Stanford University, Computer Science Dept., Technical Report CS-79-722, April.

YAO S.B. (Ed) (1985) *Principles of Database Design, Volume 1: Logical Organizations*, Prentice Hall.

YAO S.B., NAVATHE S.B. & WELDON J.L. (1982) "An Integrated Approach to Database Design", in Yao *et al* (Eds) (1982).

YAO S.B., WADDLE V. & HOUSEL B.C. (1985), "An Interactive System for Database Design and Integration", in: Yao (Ed.) (1985).

YAO S.B., WADDLE V.E. & HOUSEL B.C. (1982), "View Modelling and Integration Using the Functional Data Model", *IEEE Trans. on Software Engg.* , **8** (6), Nov. pp. 544-553.

ZADEH L.A. (1965) "Fuzzy sets and Systems", *Proc. of the Symposium on System Theory*, Polytechnic Institute of Brooklyn, N.Y.

ZANIOLO C. & MELKANOFF M.A. (1981), "On the Design of Relational Database Schemes", *ACM Trans. Database Systems*, **6** (1), March, pp. 1-47.